Πρόχειρη έκδοση υπο έκδοση εγχειριδίου ΚΑΛΛΙΠΟΣ

ΕΛΕΥΘΕΡΙΟΣ ΜΩΥΣΙΑΔΗΣ
Αναπληρωτής Καθηγητής, τμήμα Πληροφορικής, ΔΙΠΑΕ

ΧΑΪΡΗ ΚΙΟΥΡΤ
Εντεταλμένος ερευνητής, Ερευνητικό Κέντρο Αθηνά.

Εισαγωγή στην Java

Από το Διαδικαστικό στο Αντικειμενοστρεφές μοντέλο

ΚΑΛΛΙΠΟΣ
ανοικτές ακαδημαϊκές εκδόσεις

Πρόχειρη έκδοση υπο έκδοση εγχειριδίου ΚΑΛΛΙΠΟΣ

Εισαγωγή στην Java

Συγγραφή

Ελευθέριος Μωυσιάδης

Chairi Kiourt

Κριτικός αναγνώστης (style: SintelestesTitle)

Όνομα 1 (Κριτικός αναγνώστης) (style: Sintelestes)

Συντελεστές έκδοσης (style: SintelestesTitle)

Γλωσσική Επιμέλεια: Όνομα (style: Sintelestes)

Γραφιστική Επιμέλεια: Όνομα (style: Sintelestes)

Τεχνική Επεξεργασία: Όνομα (style: Sintelestes)

Πρόχειρη έκδοση υπο έκδοση εγχειριδίου ΚΑΛΛΙΠΟΣ

ΚΑΛΛΙΠΟΣ

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

www.kallipos.gr

ISBN:

Στην γυναίκα μου, Ανθή
και στα παιδιά μου,
Αναστάση και Ειρήνη
Ε.Μ.

Στην κόρη μου, Ναρίν
και στην σύζυγο μου,
Σεβτζάν
Χ.Κ.

Πίνακας περιεχομένων

Πίνακας συντομεύσεων-ακρωνυμίων

Ακρωνύμιο	Περιγραφή
JSE	Java Standard Edition
JEE	Java Enterprise Edition
JME	Java Micro Edition
JVM	Java Virtual Machine
JIT	Just In Time
JRE	Java Runtime Environment
JDK	Java Development Kit
IDE	Integrated Development Environments
API	Application Programming Interface
HTML	HyperText Markup Language
RAM	Random Access Memory
JDB	Java Data Base
RIA	Rich Internet Applications
XML	Extended Markup Language
JAF	JavaBeans Activation Framework
JPEG	Joint Photographic Experts Group
OCR	Optical Character Recognition

Πρόλογος

Κατά την βιομηχανική επανάσταση, η ανθρωπότητα γνώρισε ραγδαία ανάπτυξη και πρόοδο. Τότε ο άνθρωπος κατασκεύασε μηχανές που ήταν σε θέση να τον αντικαταστήσουν σε πλήθος χειρωνακτικών εργασιών με δραματικά αποτελέσματα στην αύξηση της παραγωγής. Στην σύγχρονη εποχή όμως, οι ρυθμοί ανάπτυξης ξεπερνούν κατά πολύ αυτούς της περιόδου εκβιομηχάνισης. Τώρα ο άνθρωπος κατασκευάζει μηχανές ικανές να υπολογίζουν, να βλέπουν, να λαμβάνουν αποφάσεις, ακόμη και να σκέπτονται ή να μιλούν.

Σήμερα, περισσότερο από κάθε άλλη εποχή της ανθρώπινης Ιστορίας, η ισχύς των νέων τεχνολογιών, θέτει μια σειρά από ερωτήματα, γεννάει ελπίδες και εγκυμονεί κινδύνους. Θα καταφέρουμε άραγε να αξιοποιήσουμε τις νέες τεχνολογίες αποκλειστικά για το καλό της ανθρωπότητας; Μήπως θα φτάσουμε σε αυτόν τον βαθμό τεχνητής ευφυίας που η υποδούλωσή μας στις μηχανές θα καταστεί αναπόφευκτη; Μήπως οι ισχυροί του πλανήτη αξιοποιώντας την τεράστια ισχύ της τεχνολογίας καταφέρουν να ελέγξουν τους λαούς;

Τίποτα άλλο δεν μπορεί να εγγυηθεί την έκβαση της επανάστασης της Πληροφορικής παρά η πλατιά συμμετοχή όσων το δυνατό περισσότερων πολιτών στην νέα γνώση. Όσο περισσότεροι είμαστε κοινωνοί των μυστικών της σύγχρονης τεχνολογίας τόσο ισχυρότερη είναι η άμυνά μας και τόσο μεγαλύτερη η ασφάλειά μας απέναντι σε τυχόν κακόβουλη αξιοποίηση της.

Αυτή η σκέψη αποτέλεσε το κύριο κίνητρο συγγραφής αυτού του βιβλίου. Πρόκειται για μια προσπάθεια διάχυσης της γνώσης στον νευραλγικό τομέα της Πληροφορικής, τον Προγραμματισμό. Ευελπιστούμε πως έτσι πραγματοποιούμε μια ελάχιστη συμβολή στην πρόοδο.

Αν οι ελπίδες μας έχουν κάποια βάση ή όχι, ο μόνος κατάλληλος για να το πει είστε εσείς, οι αναγνώστες αυτού του βιβλίου. Ανεξάρτητα από την ετυμηγορία σας, θέλουμε να σας ευχαριστήσουμε γιατί χωρίς εσάς δεν θα είχε κανένα απολύτως νόημα, η ύπαρξη του.

Εισαγωγή

Είναι κοινή αντίληψη πως η ανάπτυξη της πληροφορικής ακολουθεί πρωτόγνωρα ραγδαίους ρυθμούς. Εχθές δεν υπήρχαν κινητά τηλέφωνα. Σήμερα τα κινητά είναι ολοκληρωμένοι υπολογιστές που αναγνωρίζουν το πρόσωπο του ιδιοκτήτη τους και του ανοίγουν μια πύλη εισόδου στο παγκόσμιο δίκτυο. Πέραν της ανάπτυξης της ίδιας της πληροφορικής, πληθώρα διεπιστημονικών πεδίων κάνουν την εμφάνισή τους. Η Βιοπληροφορική, η Μηχατρονική, οι Τηλεπικοινωνίες, οι Τεχνολογίες Πληροφορικής και Επικοινωνιών στην Εκπαίδευση, η Τεχνογλωσία που συνδυάζει Γλωσσολογία και Πληροφορική, η Νανοτεχνολογία, η Νομική Πληροφορική, η Ρομποτική και άλλοι κλάδοι στους οποίους η Πληροφορική διαδραματίζει σημαίνοντα ρόλο.

Ακόμη και επιστήμες που εκ' πρώτης όψεως δεν σχετίζονται με την Πληροφορική, έχουν ωφεληθεί από τις εφαρμογές της. Στους ερευνητές όλων των ειδικοτήτων, προσφέρει άμεση πρόσβαση στο σύνολο σχεδόν της ανθρώπινης γνώσης. Στην ιατρική, προγραμματιζόμενα ρομπότ συμμετέχουν σε κρίσιμες επεμβάσεις και έμπειρα συστήματα βοηθούν σε έγκυρες διαγνώσεις. Στην νομική, βάσεις δεδομένων εξυπηρετούν την πρόσβαση σε μεγάλες βάσεις πληροφοριών. Το ηλεκτρονικό εμπόριο επιταχύνει τις συναλλαγές αυξάνοντας το παραγόμενο προϊόν. Η Ωκεανογραφία εξοπλίζεται με νέα συστήματα που διευκολύνουν την μελέτη των ωκεανών και των βυθών.

Ήδη με την ανάπτυξη της Τεχνητής νοημοσύνης, τα πρώτα συστήματα που έχουν την ικανότητα να μαθαίνουν έχουν τεθεί σε λειτουργία. Ήδη ο άνθρωπος είναι κατώτερος σκακιστής σε σχέση με τον Υπολογιστή. Φαίνεται πως η Ανολοκλήρωτη Επανάσταση [1] γίνεται πραγματικότητα. Οι σύγχρονοι Υπολογιστές αναγνωρίζουν τον ιδιοκτήτη τους από το πρόσωπό του, αναγνωρίζουν τα δαχτυλικά του αποτυπώματα, οι χρήστες μπορούν και επικοινωνούν με τους υπολογιστές στην δική τους ανθρώπινη γλώσσα.

Το μέλλον διαγράφεται ακόμη πιο δυναμικό. Σύμφωνα με μια δημοσίευση [2] της Kaspersky, μεγάλης εταιρείας Πληροφορικής, σε τρεις δεκαετίες, όλες οι βαριές δουλειές θα γίνονται από ρομπότ, τα σπίτια πλήρως αυτοματοποιημένα θα διεκπεραιώνουν εργασίες από την αναγκαία μαγειρική μέχρι την διαχείριση των προμηθειών, τα αυτοκίνητα θα κινούνται αυτόνομα, οι κατασκευές επίσης θα αυτοματοποιηθούν με την βοήθεια 3D εκτυπωτών. Ακόμη και η ίδια η ανθρώπινη φύση θα μεταβληθεί καθώς βαδίζουμε προς τον μηχανικό άνθρωπο, δηλ. τον άνθρωπο που θα διαθέτει τεχνητά μέλη και όργανα.

Στην βάση όλων αυτών των κατακλυσμιαίων αλλαγών βρίσκεται ο προγραμματισμός. Ο προγραμματισμός αποτελεί την βασική οδό επικοινωνίας του ανθρώπου με τον υπολογιστή, τον τρόπο με τον οποίον ο άνθρωπος διαμορφώνει τον "νου" του άψυχου υλικού.

Η ανάπτυξη και η εξέλιξη του προγραμματισμού έχει επίσης γνωρίσει δραματικούς ρυθμούς. Πριν μόλις τριάντα χρόνια, μία γλώσσα προγραμματισμού συμπυκνωνότανε σε ένα βιβλίο περιορισμένου όγκου και ένας επαγγελματίας του χώρου μπορούσε να καυχηθεί ότι γνωρίζει όλες τις λεπτομέρειες της γλώσσας. Σήμερα, οι περισσότερες γλώσσες προγραμματισμού έχουν μετατραπεί σε πλαίσια ανάπτυξης. Τα πλαίσια αυτά πέραν του πυρήνα μιας γλώσσας ενσωματώνουν βιβλιοθήκες ικανές να υποστηρίξουν πάρα πολλές λειτουργίες, από διαχείριση βάσεων δεδομένων μέχρι προγραμματισμό ενσωματωμένων συστημάτων, από την διεθνοποίηση των προγραμμάτων μέχρι τις κανονικές εκφράσεις, από υπηρεσίες εξουσιοδότησης και αναγνώρισης μέχρι την προσαρμοσμένη δικτύωση. Έτσι οι λεπτομέρειες ενός πλαισίου, όπως της Java για παράδειγμα, είναι αδύνατο, λόγω όγκου, να συμπεριληφθούν σε ένα βιβλίο και ο καλύτερος επαγγελματίας δεν γνωρίζει με λεπτομέρεια παρά μόνο ένα ποσοστό από τις διαθέσιμες δυνατότητες.

Ο τεράστιος αυτός όγκος συνιστά μια πρόκληση και στον εκπαιδευτικό τομέα. Σε τι επιμέρους αντικείμενα μπορεί να διαχωριστεί η αναγκαία ύλη; Με ποια σειρά θα πρέπει να δοθούν τα επιμέρους αντικείμενα στον εκπαιδευόμενο; Πως θα πρέπει να διαμορφωθεί η ύλη σε κάθε επιμέρους αντικείμενο; Αυτά είναι κάποια από τα θεμελιώδη ερωτήματα στα οποία οφείλει να απαντήσει κάθε πρόγραμμα σπουδών εφόσον περιλαμβάνει προγραμματισμό. Για να δώσω μια εικόνα στον αναγνώστη αναφέρω ως παράδειγμα ότι στο τμήμα Πληροφορικής του ΔΙΠΑΕ έχουμε 10 μαθήματα εκμάθησης αποκλειστικά του Προγραμματισμού. Πιο συγκεκριμένα, Εισαγωγή στον προγραμματισμό με C++, Εισαγωγή στην Java, Αντικειμενοστρεφής Προγραμματισμός, Προγραμματισμός Διεπαφής Χρήστη, Προηγμένα Θέματα Προγραμματισμού, Προγραμματισμός του Παγκόσμιου Ιστού, Ανάπτυξη Προηγμένων Εφαρμογών Κινητών Συσκευών, Λογική και Λογικός Προγραμματισμός, Προγραμματισμός Δικτύων και τέλος Σχεδιαστικά Πρότυπα. Πέραν αυτών ένα μεγάλο πλήθος μαθημάτων, όπως η Τεχνητή Νοημοσύνη, οι Βάσεις Δεδομένων, η Τεχνητή Όραση, τα Αυτοκινούμενα Ρομπότ, τα Νοήμονα Ρομπότ, τα Λειτουργικά Συστήματα, οι Αλγόριθμοι Βελτιστοποίησης, οι Αλγόριθμοι Βιοπληροφορικής, κα, συμπεριλαμβάνουν τον προγραμματισμό όχι απλώς ως εφαρμογή της

διδακτέας ύλης των μαθημάτων που αφορούν αποκλειστικά τον Προγραμματισμό αλλά διδάσκοντας νέα προγραμματιστικά εργαλεία απαραίτητα για κάθε ένα από αυτά τα αντικείμενα.

Μέσα σε αυτήν την πληθώρα θεμάτων, το εγχειρίδιο αυτό έχει ως στόχο να καλύψει το Διαδικαστικό και το Αντικειμενοστρεφές μοντέλο με την Java. Σε αυτό το επίπεδο, κρίσιμο είναι το ερώτημα της διάρθρωσης της ύλης. Η αλήθεια είναι ότι κυκλοφορούν αρκετά βιβλία που καλύπτουν την συγκεκριμένη θεματική περιοχή. Στην μεγάλη πλειοψηφία τους παρουσιάζουν την ύλη παράλληλα εμπλέκοντας έννοιες του διαδικαστικού με του αντικειμενοστρεφούς μοντέλου. Δεν μπορεί κανείς να θεωρήσει ότι πρόκειται για λανθασμένη επιλογή. Πολλοί συγγραφείς θεωρούν ότι κάποιες έννοιες πρέπει να διδαχθούν από την αρχή καθώς έτσι διευκολύνεται η έγκαιρη εμπέδωσή τους. Ωστόσο, η δική μας προσέγγιση διαφέρει. Ιστορικά, πρώτα αναπτύχθηκε το διαδικαστικό μοντέλο και στην συνέχεια το αντικειμενοστρεφές. Αυτό δεν σημαίνει όμως πως το διαδικαστικό μοντέλο είναι παρωχημένο. Αντίθετα, εμπεριέχεται και αποτελεί την βάση και του αντικειμενοστρεφούς αλλά και σχεδόν όλων των αντικειμένων προγραμματισμού που αναφέρθηκαν προηγουμένως. Επιπλέον, το διαδικαστικό μοντέλο είναι αυτοδύναμο, μπορεί να διδαχθεί αυτόνομα και σε γενικές γραμμές, οι έννοιες του είναι απλούστερες και άρα πιο προσιτές στον εκπαιδευόμενο προγραμματιστή. Βέβαια, η Java είναι μια καθαρά αντικειμενοστρεφής γλώσσα. Αυτό όμως δεν σημαίνει πως δεν συμπεριλαμβάνει το διαδικαστικό μοντέλο. Αξιολογώντας αυτά τα δεδομένα, αποφασίσαμε να οργανώσουμε την ύλη κατά τρόπο ώστε να εξαντλήσουμε πρώτα την μελέτη όλων των διαδικαστικών δομών και εννοιών και στην συνέχεια να προχωρήσουμε στο αντικειμενοστρεφές μοντέλο. Ευελπιστούμε πως αυτή την διάρθρωση διευκολύνει την πρόσληψη της ύλης και την κατανόηση των εννοιών. Σε αυτό το πλαίσιο προσπαθήσαμε να δώσουμε στον αναγνώστη όλες τις αναγκαίες πληροφορίες με την σωστή σειρά αποφεύγοντας την φλυαρία που κατά την άποψή μας κάνει την μελέτη κουραστική.

Πέρα από αυτήν την βασική επιλογή, θα θέλαμε να επιστήσουμε την προσοχή του αναγνώστη στις ακόλουθες κατευθυντήριες γραμμές μελέτης.

Ορολογία. Δίνουμε μεγάλη σημασία στην γνώση της ορολογίας του προγραμματισμού. Με λίγο φιλοσοφική διάθεση μπορούμε να πούμε ότι γλώσσα και σκέψη δεν είναι παρά οι δύο όψεις του ίδιου νομίσματος. Άραγε μπορούμε να επεξεργαστούμε σύνθετες σκέψεις χωρίς να τις οργανώσουμε λεκτικά; Μπορούμε να σκεφτούμε κάτι έστω και απλό χωρίς να το αναπαραστήσουμε νοητικά με λέξεις; Πως θα κατανοήσουμε μια έννοια στον Προγραμματισμό που περιέχει στον ορισμό της άλλες απλούστερες έννοιες αν δεν τις έχουμε ήδη κάνει κτήμα μας;

Πράξη. Η πράξη είναι εξαιρετικά σημαντική στην εκμάθηση του Προγραμματισμού. Όσα βιβλία και να διαβάσω, όση αποφασιστικότητα και να διαθέτω για να εντρυφήσω στις έννοιες του προγραμματισμού, ελάχιστα θα καταφέρω αν δεν προγραμματίζω. Προγραμματίζοντας έχω την ευκαιρία να βλέπω τα λάθη και παραλείψεις μου, να αναπτύσσω την απαραίτητη αλγοριθμική σκέψη και να εξοικειώνομαι με τις αναγκαίες έννοιες στην πράξη. Όμως η πράξη δεν θα πρέπει να είναι τυχαία και άναρχη, αντίθετα θα πρέπει να διακρίνεται από συστηματικό χαρακτήρα. Θα δώσω ένα παράδειγμα για να εξηγήσω σαφώς τι εννοώ. Συχνά βρίσκομαι μάρτυρας του εξής περιστατικού. Το πρόγραμμα κάποιου εκπαιδευόμενου παρουσιάζει κάποιο πρόβλημα. Ο εκπαιδευόμενος αρχίζει να δοκιμάζει τυχαίες αλλαγές στον κώδικα με σκοπό να εξαλείψει το πρόβλημα. Κάποια στιγμή, το πρόβλημα δεν εμφανίζεται πλέον και ο εκπαιδευόμενος θεωρεί ότι το πρόβλημα λύθηκε. Πρόκειται για λανθασμένη εφαρμογή της μεθόδου ‘προσπάθησε και κάνε λάθος’ (try and error). Με αυτόν τον τρόπο είναι πιθανό πως το πρόβλημα δεν λύθηκε καν αλλά απλώς δεν εμφανίζεται κάτω από συγκεκριμένες προϋποθέσεις. Αν αυτές αλλάξουν το πρόβλημα ενδέχεται να εμφανιστεί εκ’ νέου. Ακόμη όμως και αν το πρόβλημα λυθεί, ο εκπαιδευόμενος δεν κατανόησε τι ακριβώς δημιούργησε το πρόβλημα. Σε πρώτη ευκαιρία θα το επαναλάβει σε κάποιο άλλο σημείο του κώδικα. Αντί αυτής της πρακτικής, μελετήστε με προσοχή τα μηνύματα που αναφέρονται στο πρόβλημα και ψάξτε συστηματικά στον κώδικά σας μέχρι να καταλάβετε ποιο ακριβώς είναι το θέμα. Μόνο τότε προχωρήστε σε επέμβαση στον κώδικα όταν είστε σίγουροι πως έχετε καταλάβει σωστά. Με αυτόν τον τρόπο και θα παρέχετε πιο αξιόπιστη λύση και θα ενσωματώσετε μια γνώση που θα αποδειχθεί σίγουρα χρήσιμη.

Θεωρία. Η πράξη είναι απαραίτητη. Το ίδιο απαραίτητη είναι και η θεωρία. Πολλοί αρχάριοι προγραμματιστές δεν δίνουν την πρέπουσα σημασία στην θεωρία. Η αλήθεια είναι πως στον προγραμματισμό, γνωρίζοντας μερικά βασικά στοιχεία μπορεί να επιχειρήσει κανείς να κάνει αρκετά πράγματα. Αυτό το χαρακτηριστικό, συχνά δίνει την εντύπωση πως δεν απαιτούνται και ιδιαίτερες γνώσεις για να προγραμματίσουμε. Πρόκειται όμως για εντελώς λανθασμένη αντίληψη. Δεν αρκεί να μπορώ να κάνω κάτι αν δεν γνωρίζω έναν σχετικά οικονομικό τρόπο για να το πετύχω. Δεν χρησιμεύει σε τίποτα να μπορώ να πάω από την Αθήνα στην Θεσσαλονίκη αν ο μόνος δρόμος που γνωρίζω περνάει από τον Βόρειο Πόλο. Η βαθιά γνώση

της θεωρίας είναι αυτή που θα διαφοροποιήσει τον αυριανό επαγγελματία προγραμματιστή από τον αιώνιο ερασιτέχνη.

Αγγλικά. Ένα άλλο σημείο στο οποίο θα ήθελα να επιστήσω την προσοχή στον επίδοξο εκπαιδευόμενο προγραμματιστή είναι η αναγκαιότητα γνώσης των Αγγλικών. Τυπικά, οι νέες εξελίξεις στον χώρο ανακοινώνονται στα Αγγλικά. Στην συντριπτική πλειοψηφία τους χώροι του διαδικτύου στους οποίους μπορείτε να λύσετε απορίες σας και να ενημερωθείτε σχετικά για ποικίλα θέματα Προγραμματισμού, χρησιμοποιούν τα Αγγλικά. Γνωρίζω ότι πολλοί σπουδαστές αντιμετωπίζουν δυσκολίες με τα Αγγλικά. Δυστυχώς όμως, στον τομέα του Προγραμματισμού, η γνώση των Αγγλικών είναι αναγκαία. Όμως αυτό δεν θα πρέπει να αποθαρρύνει κανέναν. Δεν απαιτείται να γνωρίζουμε Αγγλικά στο επίπεδο του να διαβάζουμε ή να γράφουμε περίπλοκα λογοτεχνικά ή φιλοσοφικά δοκίμια. Η ορολογία του χώρου είναι σχετικά περιορισμένη. Αν επιμείνετε να διαβάζετε προγραμματισμό στα Αγγλικά, σύντομα θα νοιώθετε άνετα να κατανοείτε τις σχετικές έννοιες. Εξάλλου, σήμερα, χάρι στην ανάπτυξη της Πληροφορικής, έχετε στην διάθεσή σας διάφορα γλωσσικά εργαλεία που μπορούν να σας βοηθήσουν σε αυτήν την προσπάθεια, π.χ. ο μεταφραστής της Google (Google Translator).

Στην συνέχεια, αυτό το εγχειρίδιο οργανώνεται ως ακολούθως. Τα κεφάλαια 1 έως και 9 μας εισάγουν στον διαδικαστικό προγραμματισμό με Java. Τα υπόλοιπα κεφάλαια αφορούν το αντικειμενοστρεφές μοντέλο. Αμέσως μετά το κεφάλαιο 9, παρατίθεται μια σειρά από κριτήρια αξιολόγησης. Παρόμοια, κριτήρια αξιολόγησης δίνονται και στο τέλος του βιβλίου. Τα κριτήρια αξιολόγησης έχουν αναπτυχθεί ώστε να στοχεύουν αφενός να πληροφορήσουν τον αναγνώστη κατά πόσο έχει κάνει κτήμα του την αντίστοιχη ύλη αφετέρου να του επισημάνουν πως στον Προγραμματισμό υπάρχουν λεπτομέρειες που είναι αναγκαίο να γνωρίζουμε.

Με αυτές τις σκέψεις, κλείνω την εισαγωγή και σας καλωσορίζω στον δημιουργικό κόσμο του Προγραμματισμού!

Κεφάλαιο 1

Σύνοψη
Παρουσιάζονται καταρχάς ιστορικά στοιχεία σχετικά με την δημιουργία, την εξέλιξη, τους στόχους και τις βασικές σχεδιαστικές επιλογές της Java. Στην συνέχεια, εξηγούνται οι βασικές έννοιες: Γλώσσα μηχανής (Machine language), Assembly, Μεταγλωττιστής (Compiler), Διερμηνευτής (Interpreter), Διασυνδέτης (Linker). Παρουσιάζεται η αρχιτεκτονική της Java και εξηγείται η λειτουργία των συστατικών της, της Εικονικής Μηχανής (Java Virtual Machine), του Περιβάλλοντος Εκτέλεσης (Java Runtime Environment), της Εργαλειοθήκης Ανάπτυξης (Java Development Kit), του μεταγλωττιστή Just In Time. Επισημαίνεται ο διαπλατφορμικός χαρακτήρας της Java. Επιπλέον, παρουσιάζονται οι βασικές αρχές σχεδίασης και οι τεχνολογίες που υποστηρίζονται στο βασικό πλαίσιο της Java. Πιο συγκεκριμένα, αναφέρονται το πλαίσιο σύνδεσης με βάσεις δεδομένων, η ενσωματωμένη βάση δεδομένων, το πλαίσιο ανάπτυξης για ενσωματωμένα συστήματα (Java SE Embedded), τα πλαίσια ασφάλειας (Security), τα πλαίσια διεθνοποίησης (Internationalization), ανάπτυξης εφαρμογών δικτύου (Custom Networking), διανομής (Deployment), κανονικών εκφράσεων (Regular Expressions), τα πλαίσια διαδικτυακών εφαρμογών (Java Servlet API, JavaServer Pages Technology, JavaServer Pages Standard Tag Library, JavaServer Faces Technology, Java Message Service API, JavaMail API and the JavaBeans Activation Framework), το πλαίσιο διαχείρισης XML αρχείων (Java API for XML Processing), τα πλαίσια ανάπτυξης εφαρμογών με παραθυρική διεπαφή (Swing και JavaFx) και εμπλουτισμένων εφαρμογών διαδικτύου (Rich Internet Applications), η δυνατότητα για ανάπτυξη εφαρμογών κινητών και διαπλατφορμικών εφαρμογών.

Προαπαιτούμενη γνώση
Δεν απαιτούνται ειδικές γνώσεις για αυτήν την ενότητα.

Λέξεις κλειδιά
Γλώσσα μηχανής, Μεταγλωττιστής, Εικονική Μηχανή

1 Ιστορικά στοιχεία και βασικές έννοιες

Δύο είναι τα βασικά συστατικά μέρη ενός υπολογιστικού συστήματος, το υλικό (hardware) και το λογισμικό (software). Το υλικό, όπως φανερώνει και το όνομά του περιλαμβάνει εκείνο το τμήμα που έχει υλική υπόσταση. Με άλλα λόγια το υλικό είναι μια μηχανή. Ωστόσο, πρόκειται για μια ιδιαίτερη μηχανή, θεμελιωδώς διαφορετική από όλες τις μη υπολογιστικές μηχανές. Όλες οι άλλες μηχανές σχεδιάζονται για να εκτελούν μια προκαθορισμένη εργασία. Αντίθετα, το υλικό του υπολογιστή από μόνο του δεν εκτελεί καμία εργασία. Μπορεί όμως να εκτελέσει οποιαδήποτε εργασία το λογισμικό θα του υποδείξει. Τι είναι όμως το λογισμικό; Ένας απλός ορισμός λέει πως το λογισμικό ενός υπολογιστή είναι το σύνολο των προγραμμάτων που τρέχουν σε αυτόν. Τι είναι όμως ένα πρόγραμμα; Ένα πρόγραμμα στην απλούστερη μορφή του, δηλ. εκφρασμένο στην γλώσσα της μηχανής, είναι ένα σύνολο εντολών τέτοιων ώστε κάθε εντολή είναι εκτελέσιμη από το υλικό. Πρόκειται για εντολές που αντιστοιχούν σε στοιχειώδεις λειτουργίες. Ωστόσο, οι εντολές αυτές μπορούν να οργανωθούν σε αναρίθμητους συνδυασμούς έτσι ώστε κάθε συνδυασμός να παράγει ένα εντελώς διαφορετικό αποτέλεσμα. Τελικά, η μηχανή καταφέρνει και εκτελεί πολλές και διαφορετικές μεταξύ τους εργασίες.

Έτσι λοιπόν με την βοήθεια του λογισμικού, το υλικό αποκτά χαρακτηριστικά που αντίστοιχά τους συναντά κανείς μόνο στην ανθρώπινη σκέψη, όπως για παράδειγμα, η λήψη αποφάσεων ανάλογα με τα διαθέσιμα δεδομένα. Από αυτό το σημείο ξεκινάει η μεγάλη περιπέτεια της σύγχρονης πληροφορικής η οποία από την γέννησή της μέχρι σήμερα έχει γνωρίσει μια άνευ προηγουμένου ανάπτυξη ενώ ταυτόχρονα συμβάλει τα μέγιστα στην ταχύτατη εξέλιξη και των υπόλοιπων επιστημών.

Όπως όμως σε όλα τα μεγάλα επιτεύγματα της ανθρώπινης επινόησης, έτσι και στον τομέα της ανάπτυξης λογισμικού δεν έλλειψαν τα προβλήματα. Σύντομα, μετά την συγγραφή των πρώτων προγραμμάτων, έγινε αντιληπτή η ανάγκη να αναπτυχθούν γλώσσες με τις οποίες να μπορούν οι άνθρωποι να προγραμματίζουν εύκολα τους υπολογιστές. Μια από αυτές τις γλώσσες είναι και η Java.

Στην συνέχεια αυτού του κεφαλαίου, αφού δώσουμε συνοπτικά τα κύρια ιστορικά στοιχεία για την γέννηση και εξέλιξη της Java, προχωρούμε στην παρουσίαση βασικών εννοιών αναγκαίων για την κατανόηση του προγραμματισμού.

1.1 Ιστορικά στοιχεία

Το 1991, ο James Gosling στην Sun Microsystems, ανέλαβε τον σχεδιασμό μιας νέας γλώσσας, προδρόμου της Java, με το όνομα Oak [3]. Στόχος του ήταν να αξιοποιηθεί η Oak για τον προγραμματισμό οικιακών συσκευών. Προέκυψε λοιπόν η ανάγκη, ένα λογισμικό που θα γραφόταν για συγκεκριμένη συσκευή, π.χ. τηλεόραση ενός κατασκευαστή, να τρέχει και σε οποιοδήποτε άλλο μοντέλο του ίδιου ή άλλου κατασκευαστή. Από την ανάγκη αυτή πηγάζει ο διαπλατφορμικός χαρακτήρας της Java. Ήδη το 1994, η Oak είχε μετονομαστεί σε Java και άρχισε να διαμορφώνεται κατάλληλα ώστε να υποστηρίζει διαδικτυακές εφαρμογές. Σύντομα, οι κατασκευαστές λογισμικού πλοήγησης στο διαδίκτυο με πρώτη την Netscape Communications άρχισαν να υποστηρίζουν την νέα γλώσσα [4].

Το 1996, η Sun Microsystems παρουσίασε την πρώτη δημόσια έκδοση της γλώσσας. Στο τέλος του 1998, η ίδια εταιρεία, παρουσίασε την Java 2 που περιλάμβανε τρεις διαφορετικές διαμορφώσεις, την βασική, την διαμόρφωση για εταιρείες και την διαμόρφωση για προγραμματισμό συσκευών συμπεριλαμβανομένων των κινητών τηλεφώνων. Οι εκδόσεις αυτές εξελίχθηκαν και ονομάστηκαν το 2006 από την Sun ως Java Standard Edition (JSE), Java Enterprise Edition (JEE) και Java Micro Edition (JME), αντίστοιχα [5]. Το 2007 η Java υιοθετείται από την Google ως βασική γλώσσα για ανάπτυξη εφαρμογών Android [6]. Την ίδια εποχή περίπου, ο Chris Oliver στην SeeBeyond υλοποιεί ένα πλαίσιο ανάπτυξης παραθυρικών διεπαφών που ονόμασε F3. Το 2008, η Sun αγοράζει την SeeBeyond και προσθέτει στην οικογένεια της Java την εξέλιξη της F3 που ονόμασε JavaFx [7].

Από την γέννησή της έως σήμερα η Java βελτιώνεται και επαυξάνεται διαρκώς. Η JSE κατά τον χρόνο συγγραφής αυτού του κειμένου βρίσκεται στην έκδοση 17. Οι κώδικες που παρουσιάζονται σε αυτό το βιβλίο μεταγλωττίζονται με επιτυχία από την έκδοση 8 ή μεταγενέστερη.

1.2 Βασικές Έννοιες

Γλώσσα Μηχανής: Η θεμελιώδης γλώσσα προγραμματισμού είναι η λεγόμενη Γλώσσα Μηχανής (Machine Language). Το “αλφάβητο” αυτής της γλώσσας είναι οι χαρακτήρες 0 και 1, δηλ. τα δυαδικά ψηφία (binary digits ή bits). Με συνδυασμό των bits σχηματίζεται το λεξιλόγιο της μηχανής γνωστό ως σύνολο εντολών (Instruction Set). Επομένως, τα προγράμματα που είναι γραμμένα σε γλώσσα μηχανής είναι ακολουθίες από bits πράγμα που καθιστά την συγγραφή αλλά και την ανάγνωσή τους ιδιαίτερα δύσκολη. Επιπλέον, το σύνολο των εντολών διαφέρει ανάλογα με την αρχιτεκτονική της μηχανής. Αυτό σημαίνει ότι η ίδια λειτουργικότητα είναι αναγκαίο να υλοποιηθεί ξεχωριστά για κάθε διαφορετική μηχανή. Τέλος, τα προγράμματα σε γλώσσα μηχανής είναι απευθείας εκτελέσιμα (executable) από τον υπολογιστή.

Assembly: Προκειμένου να διευκολυνθεί ο άνθρωπος στην επικοινωνία του με τον υπολογιστή δημιουργήθηκε η Assembly. Η Assembly είναι κωδικοποιημένη γλώσσα μηχανής, δηλ. υποστηρίζει το ίδιο σύνολο εντολών με την γλώσσα μηχανής. Οι εντολές της όμως δεν εκφράζονται σαν ακολουθίες δυαδικών ψηφίων αλλά παρουσιάζουν κάποια ομοιότητα με τις λέξεις της φυσικής γλώσσας. Για παράδειγμα, η ακολουθία 1011000001100001 συνιστά ορθή εντολή σε γλώσσα μηχανής. Η ίδια εντολή σε Assembly εκφράζεται ως mov al, 061h που σημαίνει μετακίνησε την δεκαεξαδική τιμή 61 (97 στο δεκαδικό σύστημα) στον καταχωρητή με το όνομα "al". Είναι προφανές πως ο άνθρωπος ευκολότερα ανακαλεί εντολές τύπου Assembly παρά εντολές σε γλώσσα μηχανής.

Ο κώδικας σε Assembly δεν εκτελείται κατευθείαν από την μηχανή. Πρέπει προηγουμένως να μετατραπεί σε γλώσσα μηχανής. Η μετατροπή αυτή γίνεται από κατάλληλο συμβολομεταφραστή που ονομάζεται Assembler.

Γλώσσες υψηλού επιπέδου: Η Assembly διευκολύνει στην αναγνωσιμότητα των προγραμμάτων αλλά διατηρεί το μειονέκτημα πως περιορίζεται από το σύνολο εντολών της μηχανής. Οι εντολές αυτές αντιστοιχούν σε στοιχειώδεις λειτουργίες της μηχανής με αποτέλεσμα και το πιο απλό πρόγραμμα σε Assembly να παρουσιάζει αυξημένο μέγεθος και πολυπλοκότητα και να απαιτεί υψηλό φόρτο εργασίας. Αυτός είναι και ο

λόγος που η Assembly χαρακτηρίζεται σαν γλώσσα χαμηλού επιπέδου (low level). Για παράδειγμα, ο κώδικας που ακολουθεί είναι σε Assembly x86-64 και τυπώνει στην οθόνη τον χαρακτήρα '!'.

```
push    $0x21       # '!'
mov     $1, %rax    # sys_write call number
mov     $1, %rdi    # write to stdout (fd=1)
mov     %rsp, %rsi  # use char on stack
mov     $1, %rdx    # write 1 char
syscall
add     $8, %rsp    # restore sp
```

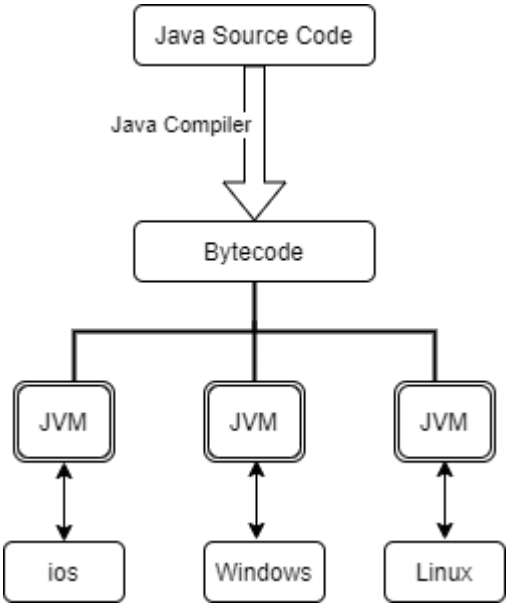
Έτσι προέκυψε η ανάγκη για τις γλώσσες υψηλού επιπέδου (high level programming languages). Ήδη το 1957 κατασκευάστηκε η πρώτη γλώσσα υψηλού επιπέδου, η Fortran της IBM. Οι γλώσσες αυτές είναι ακόμη πιο κοντά στον άνθρωπο σε σχέση με την Assembly και υποστηρίζουν περισσότερες και πιο σύνθετες εντολές. Για παράδειγμα, σε σύγκριση με την Assembly, ο κώδικας Java που εκτυπώνει τον χαρακτήρα '!' είναι System.out.print('!'). Οι γλώσσες υψηλού επιπέδου αυξάνουν δραματικά την παραγωγικότητα κατά την ανάπτυξη εφαρμογών καθώς ο κώδικας είναι απλούστερος σε σύγκριση με γλώσσα μηχανής και Assembly ενώ δεν απαιτούν να γραφεί διαφορετικό πρόγραμμα για κάθε διαφορετική μηχανή. Ο κώδικας που εκφράζεται σε γλώσσα υψηλού επιπέδου ή Assembly ονομάζεται πηγαίος κώδικας (source code). Σήμερα υπάρχει μεγάλο πλήθος γλωσσών υψηλού επιπέδου με κυριότερους αντιπροσώπους τις C, C++, C#, C objective, Java, Python, Javascript, PHP.

Μεταγλωττιστής: Είναι ευνόητο πως ο πηγαίος κώδικας δεν είναι εκτελέσιμος από την μηχανή. Ο μεταγλωττιστής (compiler) λαμβάνει ως είσοδο πηγαίο κώδικα και παράγει κώδικα σε γλώσσα μηχανής. Με αυτόν τον τρόπο κωδικοποιείται μια λειτουργία μια φορά σε γλώσσα υψηλού επιπέδου και από αυτήν την κωδικοποίηση μπορεί να παραχθεί εκτελέσιμος κώδικας για κάθε μηχανή που διαθέτει τον αντίστοιχο μεταγλωττιστή. Ένα από τα πλεονεκτήματα του μεταγλωττιστή είναι πως διευκολύνει την προστασία των πνευματικών δικαιωμάτων του κώδικα καθώς ο κατασκευαστής του λογισμικού δεν διαθέτει στον πελάτη τον πηγαίο αλλά μόνο τον εκτελέσιμο κώδικα. Επιπλέον, διευκολύνει τον προγραμματιστή καθώς εντοπίζει μια σειρά από συντακτικά κυρίως λάθη κατά την φάση της μεταγλώττισης (compile time).

Διασυνδέτης: Συχνά, ο πηγαίος κώδικας μίας εφαρμογής αποτελείται από περισσότερα από ένα αρχεία. Κάθε ένα από αυτά τα αρχεία μεταγλωττίζεται χωριστά και για κάθε ένα παράγεται ένα χωριστό αρχείο. Προκειμένου όμως η εφαρμογή να τρέξει, προκύπτει η ανάγκη τα αρχεία που παράγονται από τον μεταγλωττιστή να διασυνδεθούν μεταξύ τους. Την εργασία αυτή αναλαμβάνει ο διασυνδέτης (linker).

Διερμηνευτής: Ο διερμηνευτής (Interpreter) επίσης μεταφράζει τον πηγαίο κώδικα σε εκτελέσιμο αλλά το κάνει κατά τον χρόνο εκτέλεσης (run time), δηλ. ο διερμηνευτής κατά τον χρόνο εκτέλεσης απαιτεί διαθέσιμο τον πηγαίο κώδικα που μεταφράζει και εκτελεί εντολή προς εντολή. Επομένως ο διερμηνευτής δεν εντοπίζει τα συντακτικά λάθη παρά μόνο κατά τον χρόνο εκτέλεσης του προγράμματος ενώ δεν παρέχει προστασία των πνευματικών δικαιωμάτων του πηγαίου κώδικα. Επιπλέον, ένα πρόγραμμα σε γλώσσα διερμηνευτή απαιτεί μεγαλύτερο χρόνο εκτέλεσης από το ίδιο πρόγραμμα μεταγλωττισμένο.

Εικονική μηχανή: Όπως αναφέραμε προηγουμένως, ο μεταγλωττιστής μεταφράζει τον πηγαίο κώδικα σε εκτελέσιμο. Πρόκειται για περίπλοκο λογισμικό και η υλοποίησή του απαιτεί αρκετή εργασία και υψηλό κόστος. Όταν μάλιστα θέλουμε τα προγράμματα που αναπτύσσουμε σε μια γλώσσα να είναι εκτελέσιμα σε πολλές πλατφόρμες, είμαστε αναγκασμένοι να υλοποιήσουμε ένα διαφορετικό μεταγλωττιστή για κάθε πλατφόρμα. Μια εναλλακτική λύση είναι η αξιοποίηση εικονικής μηχανής . Στην περίπτωση αυτή έχουμε συνδυασμό ενός μεταγλωττιστή με μια εικονική μηχανή. Ο μεταγλωττιστής όμως δεν παράγει κώδικα εκτελέσιμο απευθείας από την μηχανή. Αντίθετα, παράγει έναν ενδιάμεσο κώδικα, γνωστό ως bytecode ο οποίος αποτελεί είσοδο για την εικονική μηχανή που με την σειρά της παράγει τον εκτελέσιμο κώδικα. Επιπλέον, ο ενδιάμεσος κώδικας δεν διασυνδέεται από έναν τυπικό διασυνδέτη αλλά από τον επονομαζόμενο φορτωτή κλάσης (class loader). Η τεχνολογία αυτή έχει υιοθετηθεί από την Java με την υλοποίηση της Java Virtual Machine, ή JVM για συντομία.



Σχήμα 1.1: *Η λειτουργία της εικονικής μηχανής της Java*

Όπως φαίνεται στο Σχήμα 1.1, ο πηγαίος κώδικας Java μεταφράζεται σε Bytecode. Ο κώδικας Bytecode είναι ανεξάρτητος από την φυσική μηχανή στην οποία θέλουμε να τρέξει. Στην συνέχεια, ο ενδιάμεσος κώδικας τρέχει σε διάφορα λειτουργικά μέσω της εικονικής μηχανής. Επομένως δεν χρειαζόμαστε έναν μεταγλωττιστή για κάθε λειτουργικό σύστημα. Αντίθετα, ένας μεταγλωττιστής για όλα τα λειτουργικά μαζί με μια εικονική μηχανή για κάθε ένα λειτουργικό αρκούν. Η εικονική μηχανή όμως είναι πολύ απλούστερη σε σχέση με τον μεταγλωττιστή και έτσι η ανάπτυξή της για κάθε λειτουργικό είναι πολύ οικονομικότερη από την ανάπτυξη μεταγλωττιστή.

Ο μεταγλωττιστής Just In Time (JIT): Ο μεταγλωττιστής JIT εκτελείται από την εικονική μηχανή και η δουλειά του είναι η μεταγλώττιση του bytecode σε εκτελέσιμο. Ο μεταγλωττιστής JIT δεν ενδιαφέρεται για συντακτικά λάθη καθώς αυτά έχουν εντοπιστεί και διορθωθεί κατά την μεταγλώττιση του πηγαίου κώδικα. Επιπλέον, δεν μεταφράζει άμεσα όλον τον bytecode σε εκτελέσιμο. Αντίθετα, μεταφράζει μόνο τα τμήματα που είναι αναγκαία ανάλογα με τις κλήσεις που περιλαμβάνει ο κώδικας. Έτσι η εκτέλεση και η μετάφραση του κώδικα γίνονται παράλληλα. Το αποτέλεσμα είναι πως η ταχύτητα εκτέλεσης του bytecode προσεγγίζει την ταχύτητα εκτέλεσης του κώδικα μηχανής. Ας σημειωθεί πως στην βιβλιογραφία, ο JIT συχνά αναφέρεται ως μεταγλωττιστής ενώ κάποιες φορές αναφέρεται ως διερμηνευτής (Java Interpreter). Στην πράξη πρόκειται για κάτι ενδιάμεσο. Ο JIT μετατρέπει τον bytecode σε εκτελέσιμο κατά τον χρόνο εκτέλεσης που είναι χαρακτηριστικό του διερμηνευτή. Από την άλλη πλευρά όμως, ένα τμήμα κώδικα που έχει ήδη μεταφραστεί δεν απαιτείται να μεταφραστεί και πάλι όταν χρησιμοποιείται κατά την τρέχουσα εκτέλεση του προγράμματος για δεύτερη φορά.

JRE και JDK: Το Java Runtime Environment (JRE) περιλαμβάνει την JVM και επιπλέον κάποιες βιβλιοθήκες της Java που είναι απαραίτητες για να εκτελεστεί μια εφαρμογή Java. Το Java Development Kit (JDK) περιλαμβάνει την JRE και μια σειρά από εργαλεία απαραίτητα για την ανάπτυξη πηγαίου κώδικα σε Java, π.χ. ο μεταγλωττιστής που μεταφράζει από πηγαίο κώδικα σε bytecode.

Απολαθοποιητής (Debugger): Ο απολαθοποιητής είναι ένα λογισμικό που μας βοηθάει να εντοπίζουμε λάθη στον κώδικα. Ο απολαθοποιητής μας δίνει την δυνατότητα να εκτελούμε το πρόγραμμά μας ή κάποιο τμήμα του, βήμα προς βήμα και σε κάθε βήμα να ελέγχουμε το περιεχόμενο των μεταβλητών. Έτσι διευκολυνόμαστε να εντοπίσουμε σε ποιο σημείο ακριβώς συμβαίνει το λάθος και να το διορθώσουμε.

Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης: Τα Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης (Integrated Development Environment ή IDE για συντομία) είναι λογισμικά που σκοπό έχουν να μας βοηθήσουν στην ανάπτυξη εφαρμογών. Σήμερα είναι διαθέσιμα πολλά τέτοια περιβάλλοντα για ποικίλες γλώσσες.

Παραδείγματα αποτελούν το NetBeans, το Eclipse, το Visual Studio, το IntelliJ, το PyCharm, κα. Κατά την συγγραφή αυτού του βιβλίου χρησιμοποιήθηκε το NetBeans, έκδοση 12.5.

Διεπαφή Προγραμματισμού Εφαρμογών: Η Διεπαφή Προγραμματισμού Εφαρμογών (Application Programming Interface ή API για συντομία) είναι μια βιβλιοθήκη κώδικα που μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών.

1.3 Βασικές αρχές σχεδίασης της Java

Η Java σχεδιάστηκε στην βάση των ακόλουθων αρχών [8]

Απλότητα, Αντικειμενοστρέφεια και Οικειότητα
Η Java σχεδιάστηκε να είναι απλή στην χρήση της ώστε οι προγραμματιστές να γίνουν σύντομα παραγωγικοί. Πιθανότατα σε αυτήν την αρχή βασίστηκε, το απλό μοντέλο που υιοθετεί η Java στην διαχείριση της μνήμης. Επιπλέον είναι χρήσιμο, οι προγραμματιστές να είναι ήδη εξοικειωμένοι με τις δομές της Java. Έτσι αποφασίστηκε, οι δομές της Java να ομοιάζουν με αυτές της C++. Επίσης, η Java να υποστηρίζει ολοκληρωμένα το αντικειμενοστρεφές μοντέλο.

Αξιοπιστία και Ασφάλεια
Η Java σχεδιάστηκε για να υλοποιεί εφαρμογές υψηλής αξιοπιστίας. Τα χαρακτηριστικά της γλώσσας καθοδηγούν τους προγραμματιστές να υιοθετούν πρακτικές που καθιστούν τους κώδικες αξιόπιστους και σταθερούς. Η ασφάλεια αποτέλεσε κρίσιμο ζητούμενο κατά τον σχεδιασμό. Ενσωματωμένα στην γλώσσα χαρακτηριστικά καθιστούν τις εφαρμογές της Java δύσκολο στόχο στις επιθέσεις από κακόβουλο λογισμικό.

Φορητότητα
Οι εφαρμογές σε Java θα πρέπει να εκτελούνται χωρίς πρόβλημα σε οποιαδήποτε πλατφόρμα. Η αρχή αυτή επιτυγχάνεται με την υιοθέτηση της αρχιτεκτονικής της εικονικής μηχανής. Σε οποιαδήποτε πλατφόρμα υπάρχει η JVM, οι εφαρμογές Java εκτελούνται χωρίς πρόβλημα.

Ταχύτητα
Η Java επιτυγχάνει πολύ ψηλές ταχύτητες. Σε αυτό συμβάλλουν πολλά χαρακτηριστικά της γλώσσας όπως η αυτόματη διαχείριση της μνήμης (ενότητα 3.8.1), η έξυπνη μετάφραση του κώδικα από τον JIT αλλά και ο καθορισμός των τύπων των μεταβλητών κατά τον χρόνο μεταγλώττισης (ενότητα 3.2).

Πολυνηματικός και δυναμικός χαρακτήρας και διερμήνευση
Ο διερμηνευτής JIT της Java μπορεί να εκτελέσει κώδικα σε οποιαδήποτε μηχανή είναι εγκαταστημένη η εικονική μηχανή. Η Java σχεδιάστηκε εξαρχής ώστε να υποστηρίζει πολυνηματισμό, δηλ. την δυνατότητα, τμήματα του κώδικα να εκτελούνται παράλληλα από τον ίδιο επεξεργαστή. Τέλος ο δυναμικός χαρακτήρας της Java αναφέρεται στην δυναμική σύνδεση που γίνεται κατά τον χρόνο εκτέλεσης. Σε αυτήν την φάση ενδέχεται να συνδεθούν ανάλογα με τις ανάγκες, κώδικες ακόμη και από απομακρυσμένες τοποθεσίες του διαδικτύου.

1.4 Τεχνολογίες της Java

Οι τυπικές γλώσσες προγραμματισμού παρέχουν στον προγραμματιστή μια σειρά από βασικές λειτουργίες όπως είναι η δυνατότητα ανάπτυξης επαναληπτικών δομών, οι δομές επιλογής, η δημιουργία αντικειμένων, η διαχείριση της μνήμης, κλπ. Τις δυνατότητες αυτές προσφέρει και η C++. Όμως κατά την δημιουργία των σύγχρονων εφαρμογών απαιτούνται και άλλες πέραν των βασικών αυτών δυνατοτήτων, π.χ. ο προγραμματισμός της διεπαφής χρήστη (user interface). Έτσι αν αναπτύσσουμε μια εφαρμογή με C++ και χρειαζόμαστε παραθυρική διεπαφή χρήστη θα πρέπει να αξιοποιήσουμε κατάλληλες βιβλιοθήκες τρίτων που υποστηρίζουν την ανάπτυξη τέτοιων διεπαφών καθώς η ίδια η C++ δεν παρέχει αυτήν την δυνατότητα. Αντίθετα η Java υποστηρίζει στο βασικό πλαίσιο και τον προγραμματισμό της διεπαφής χρήστη και πολλές άλλες δυνατότητες όπως θα δούμε αμέσως παρακάτω. Το σύνολο των βασικών δυνατοτήτων στην Java θεωρείται ότι συνιστά τον πυρήνα της γλώσσας (Java core) πάνω στον οποίο αναπτύσσονται μια σειρά από εξειδικευμένες τεχνολογίες κατάλληλες για την ανάπτυξη ποικίλων εφαρμογών.

Βάσεις Δεδομένων: Η σύνδεση και επικοινωνία με βάση δεδομένων γίνεται με βιβλιοθήκες ενσωματωμένες στο βασικό πλαίσιο της Java. Ακόμη περαιτέρω, η Java διαθέτει την δική της Βάση Δεδομένων (Java DB).

Ενσωματωμένα Συστήματα: Με το πλαίσιο Java SE Embedded [9], η Java υποστηρίζει την ανάπτυξη αξιόπιστων και φορητών εφαρμογών υψηλής λειτουργικότητας για τα περισσότερα από τα σύγχρονα ενσωματωμένα συστήματα.

Υπηρεσίες αναγνώρισης και εξουσιοδότησης: Το πλαίσιο αναγνώρισης και εξουσιοδότησης (Java Authentication and Authorization Services) χρησιμοποιείται για να προσδιορίζεται ο χρήστης της εφαρμογής ώστε να εξουσιοδοτούνται να εκτελούν συγκεκριμένες εργασίες μόνο όσοι διαθέτουν κατάλληλα δικαιώματα [10, 11].

Διεθνοποίηση: Με το πλαίσιο διεθνοποίησης (Internationalization) [12] μπορούμε εύκολα να αλλάξουμε την γλώσσα και άλλα τοπικά χαρακτηριστικά μιας εφαρμογής, π.χ. το ημερολόγιο.

Προσαρμοσμένη δικτύωση: Με την υποστήριξη της προσαρμοσμένης δικτύωσης (Custom Networking) [13], η Java μας δίνει την δυνατότητα ανάπτυξης εφαρμογών που χρησιμοποιούν και αλληλεπιδρούν με πόρους στο Διαδίκτυο και στον Παγκόσμιο Ιστό.

Διανομή εφαρμογών: Με την διανομή εφαρμογών (Deployment), η Java μας δίνει την δυνατότητα να παράγουμε και να διανέμουμε μέσω διαδικτύου εμπλουτισμένες εφαρμογές (Rich Internet Applications ή RIA), δηλ. εφαρμογές που διαθέτουν τα ισχυρά χαρακτηριστικά των εφαρμογών επιτραπέζιου υπολογιστή αλλά διανέμονται μέσω διαδικτύου [14].

Κανονικές εκφράσεις: Η Java υποστηρίζει πλήρως τις κανονικές εκφράσεις (Regular Expressions). Πρόκειται για συμβολοσειρές που καθορίζουν ένα μοτίβο αναζήτησης [15, 16].

Διεπαφή προγραμματισμού εφαρμογών επεξεργασίας XML εγγράφων: Η διεπαφή αυτή (API for XML Processing) παρέχει στις εφαρμογές της Java την δυνατότητα εύκολης επεξεργασίας XML εγγράφων.

Διεπαφή προγραμματισμού εφαρμογών email και υπηρεσία διαχείρισης μηνυμάτων: Οι διεπαφές αυτές (JavaMail API και Java Message Service API) παρέχουν την δυνατότητα ανάπτυξης εφαρμογών διαχείρισης και ανταλλαγής μηνυμάτων.

Πλαίσιο ενεργοποίησης JavaBeans: Το πλαίσιο αυτό (JavaBeans Activation Framework, JAF) παρέχει την δυνατότητα αναγνώρισης του τύπου των δεδομένων. Για παράδειγμα, ένα πρόγραμμα περιήγησης στο διαδίκτυο, λαμβάνει μια σειρά από δεδομένα που συνιστούν μια εικόνα JPEG. Με το πλαίσιο ενεργοποίησης μπορούμε να αναγνωρίσουμε ότι πρόκειται για εικόνα JPEG και να την διαχειριστούμε κατάλληλα.

Διεπαφή Servlet: Παρέχει λειτουργίες που συνεργάζονται με έναν εξυπηρετητή διαδικτύου (Web Server) και τον βοηθούν στην διαχείριση των αιτημάτων των πελατών (client) [17].

Σελίδες Εξυπηρετητή: Η τεχνολογία αυτή (JavaServer Pages Technology) παρέχει την δυνατότητα δυναμικής διαμόρφωσης ιστοσελίδων για τις εφαρμογές του Παγκόσμιου Ιστού (Web Applications).

Βιβλιοθήκη ετικετών: Η βιβλιοθήκη ετικετών (JavaServer Pages Standard Tag Library) παρέχει με την μορφή απλών ετικετών (tags) βασική λειτουργικότητα για εφαρμογές του Παγκόσμιου Ιστού, π.χ. ετικέτες για επαναληπτικές δομές, για δομές επιλογής, για διαχείριση XML εγγράφων [18].

Τεχνολογία JavaServer Faces Technology: Η τεχνολογία αυτή απλοποιεί την ενσωμάτωση διεπαφών χρήστη σε εφαρμογές του Παγκόσμιου Ιστού [19].

JavaFx: Το πλαίσιο αυτό είναι κατάλληλο για την ανάπτυξη εμπλουτισμένων εφαρμογών διαδικτύου. Ενσωματώνει την δυνατότητα ανάπτυξης επιτραπέζιων εφαρμογών με παραθυρική διεπαφή [20]. Ας σημειωθεί

ότι είναι το νεότερο από τα τρία πλαίσια που διαθέτει η Java για ανάπτυξη παραθυρικών εφαρμογών. Τα άλλα δύο είναι το Abstract ToolKit και το Swing.

Άλλα χαρακτηριστικά της Java: Η Java είναι η βασική γλώσσα ανάπτυξης εφαρμογών Android. Επίσης, η JavaFx σε συνεργασία με την τεχνολογία Gluon μπορεί να κτίσει παραθυρικές εφαρμογές και να τις μεταφέρει στις πλατφόρμες κινητών Android και iOS. Ένα άλλο σημαντικό χαρακτηριστικό της Java είναι η συμβατότητα μεταξύ των εκδόσεων (Backwards compatibility). Πρόκειται για αναγκαίο χαρακτηριστικό προκειμένου οι επιχειρήσεις να επενδύουν για ανάπτυξη σε μια γλώσσα. Επιπλέον, η Java είναι ανοιχτό λογισμικό (Open Source). Αυτό διευκολύνει την ανάπτυξη βιβλιοθηκών τρίτων σε μορφή ανοιχτού κώδικα. Για παράδειγμα, αν κάποιος θελήσει να αναπτύξει μια εφαρμογή οπτικής αναγνώρισης χαρακτήρων (Optical Character Recognition, OCR), μπορεί εύκολα κατάλληλες βιβλιοθήκες σε Java. Αντίστοιχες βιβλιοθήκες σε άλλες γλώσσες κοστίζουν συνήθως μερικές χιλιάδες Ευρώ.

Βιβλιογραφία

[1] Μ. Δερτούζος, Η ΑΝΟΛΟΚΛΗΡΩΤΗ ΕΠΑΝΑΣΤΑΣΗ. Λιβάνης, 2001. Accessed: Sep. 14, 2021. [Online]. Available: https://www.politeianet.gr/books/9789601403755-dertouzos-michalis-libanis-i-anolokliroti-epanastasi-23070

[2] Y. Ilyin, "Digital realities in 2045: future technologies and security issues," Kaspersky daily, 2015. https://www.kaspersky.com/blog/digital-realities-in-2045-future-technologies-and-security-issues/15047/ (accessed Sep. 14, 2021).

[3] S. Walter and M. Kenrick, Java μια εισαγωγή στην επίλυση προβλημάτων και στον Προγραμματισμό, 7η έκδοση. Εκδόσεις Τζιόλα, 2016.

[4] "Java (software platform)," Wikipedia. Apr. 24, 2021. Accessed: Jun. 13, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Java_(software_platform)&oldid=1019658740

[5] "Java (programming language)," Wikipedia. Jun. 10, 2021. Accessed: Jun. 13, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Java_(programming_language)&oldid=1027807155

[6] S. J. Vaughan-Nichols, "A Google Android and Java history lesson," ZDNet. https://www.zdnet.com/article/a-google-android-and-java-history-lesson/ (accessed Jun. 13, 2021).

[7] Taniani, "What Is JavaFX?," Social net: Databases Oracle, MySQL, programming SQL, Java, APEX, administration, Jul. 03, 2018. https://oracle-patches.com/en/coding/what-is-javafx (accessed Jun. 13, 2021).

[8] "The Java Language Environment." https://www.oracle.com/java/technologies/introduction-to-java.html (accessed Jun. 13, 2021).

[9] "Oracle Java SE Embedded Overview." https://www.oracle.com/java/technologies/javase-embedded/javase-embedded.html (accessed Jun. 13, 2021).

[10] "Java Platform, Standard Edition Security Developer's Guide." https://docs.oracle.com/javase/9/security/java-authentication-and-authorization-service-jaas1.htm#JSSEC-GUID-76141D9C-03F1-40D8-93D8-DFFC2143A2CA (accessed Jun. 13, 2021).

[11] "JAAS Reference Guide." https://docs.oracle.com/javase/7/docs/technotes/guides/security/jaas/JAASRefGuide.html#Introduction (accessed Jun. 13, 2021).

[12] "Java SE 8 Internationalization (I18n) Developer Guides." https://docs.oracle.com/javase/8/docs/technotes/guides/intl/index.html (accessed Jun. 13, 2021).

[13] "Trail: Custom Networking (The Java™ Tutorials)." https://docs.oracle.com/javase/tutorial/networking/index.html (accessed Jun. 13, 2021).

[14] "Trail: Deployment (The Java™ Tutorials)." https://docs.oracle.com/javase/tutorial/deployment/index.html (accessed Jun. 13, 2021).

[15] "Java Regular Expressions." https://www.w3schools.com/java/java_regex.asp (accessed Jun. 13, 2021).

[16] A. Srivastava, Java 9 Regular Expressions: A hands-on guide to implement zero-length assertions, back-references, quantifiers, and many more. Packt Publishing, 2017.

[17] "Introduction to Java Servlets," GeeksforGeeks, May 28, 2018. https://www.geeksforgeeks.org/introduction-java-servlets/ (accessed Jun. 14, 2021).

[18] "JavaServer Pages Standard Tag Library." https://www.oracle.com/java/technologies/java-server-tag-library.html (accessed Jun. 14, 2021).

[19] “JavaServer Faces (JSF) Tutorial - Tutorialspoint.” https://www.tutorialspoint.com/jsf/index.htm (accessed Jun. 14, 2021).

[20] The Definitive Guide to Modern Java Clients with JavaFX - Cross-Platform Mobile and Cloud Development | Stephen Chin | Apress. Apress, 2019. Accessed: Jun. 14, 2021. [Online]. Available: https://www.apress.com/gp/book/9781484249253

Κεφάλαιο 2

Σύνοψη
Σε αυτήν την ενότητα μαθαίνουμε καταρχάς πώς να εγκαθιστούμε την Java και το περιβάλλον ανάπτυξης NetBeans. Στην συνέχεια παρουσιάζουμε και αναλύουμε ένα απλό εισαγωγικό πρόγραμμα σε Java. Στο πλαίσιο αυτό γίνεται εισαγωγική παρουσίαση των εννοιών: πακέτο (package), η main ως κύρια είσοδος στο πρόγραμμα, το υποπρόγραμμα ως void συνάρτηση χωρίς παραμέτρους και η βασική έξοδος στην οθόνη. Επιπλέον, παρουσιάζονται οι βασικές συμβάσεις ονοματολογίας.

Προαπαιτούμενη γνώση
Δεν απαιτούνται ειδικές γνώσεις για αυτήν την ενότητα.

Λέξεις κλειδιά
Περιβάλλον ανάπτυξης, Κλάση, Συνάρτηση, Ονοματολογία, Βασική Έξοδος.

2. Εγκατάσταση και το πρώτο πρόγραμμα

Όπως στις περισσότερες γλώσσες προγραμματισμού, έτσι και στην Java, ο πηγαίος κώδικας έχει την μορφή απλού κειμένου. Επομένως μπορεί κανείς να γράψει πρόγραμμα χρησιμοποιώντας οποιονδήποτε απλό επεξεργαστή κειμένου (text editor). Στην συνέχεια, εφόσον έχει εγκαταστήσει την Java, μπορεί να καλέσει τον μεταγλωττιστή της, να μεταγλωττίσει τον πηγαίο κώδικα και στην συνέχεια να καλέσει την εικονική μηχανή για να το τρέξει. Αυτή εν ολίγοις είναι η διαδικασία που ακολουθούνταν μέχρι τις αρχές της δεκαετίας του 2000. Ο κώδικας όμως είναι χρήσιμο να παρουσιάζεται κατά τρόπο που να είναι ευανάγνωστος. Σε αυτό συμβάλει η κατάλληλη στοίχιση των γραμμών των εντολών αλλά και ο κατάλληλος χρωματισμός των διαφορετικών στοιχείων του κώδικα. Η στοίχιση που απαιτείται είναι αρκετά περίπλοκη καθώς οφείλει να ακολουθεί τις πολύπλοκες δομές του κώδικα. Αντίστοιχα, περίπλοκος είναι και ο χρωματισμός του κώδικα. Οι απλοί επεξεργαστές κειμένου δεν παρέχουν ούτε την δυνατότητα αυτόματης στοίχισης ούτε την δυνατότητα αυτόματου χρωματισμού.

Έτσι, προέκυψε η ανάγκη να αναπτυχθούν περιβάλλοντα εξειδικευμένα για συγγραφή και διαχείριση πηγαίου κώδικα. Σύντομα τα περιβάλλοντα, γνωστά ως Ολοκληρωμένα Περιβάλλοντα Ανάπτυξης (Integrated Development Environments ή IDE), ενσωμάτωσαν πλήθος δυνατοτήτων που σκοπό έχουν να διευκολύνουν την ανάπτυξη του λογισμικού.

Τα περισσότερα σύγχρονα IDE υποστηρίζουν πλήθος από γλώσσες προγραμματισμού. Ωστόσο, συνήθως είναι πιο συγγενικά με κάποιες από αυτές. Η εγκατάστασή τους μπορεί να περιλαμβάνει και την αυτόματη εγκατάσταση των συγγενικών γλωσσών προγραμματισμού. Παρέχουν την δυνατότητα οργάνωσης τόσο των αρχείων του κώδικα σε έργα (projects) αλλά και της εσωτερικής οργάνωσης κάθε αρχείου κώδικα ξεχωριστά. Επιπλέον, αυτοματοποιούν τις διαδικασίες που απαιτούνται ώστε ο πηγαίος κώδικας να μετατραπεί σε εκτελέσιμο και να τρέξει ενώ παρέχουν συσκευές εξόδου όπου ο προγραμματιστής μπορεί να δει άμεσα το αποτέλεσμα της εκτέλεσης του κώδικα που έγραψε. Επίσης, τα σύγχρονα IDE διαθέτουν ενσωματωμένους απολαθοποιητές και διευκολύνουν την διαδικασία απολαθοποίησης.

Σήμερα, διατίθεται πλήθος από IDE. Στα πλαίσια αυτού του συγγράμματος όμως χρησιμοποιούμε το NetBeans, έκδοση 12.5, που είναι το συγγενικό IDE της Java.

Στην συνέχεια αυτού του κεφαλαίου αφού εγκαταστήσουμε το NetBeans μαζί με την Java, παρουσιάζουμε και αναλύουμε το πρώτο μας πρόγραμμα. Μετά, εισάγουμε τον αναγνώστη στην έννοια του απλού υποπρογράμματος ή ρουτίνας, επισημαίνουμε βασικές συμβάσεις ονοματολογίας που πρέπει απαραίτητα να ακολουθούμε κατά την συγγραφή των προγραμμάτων και τέλος παρέχουμε αναγκαίες βασικές πληροφορίες σε σχέση με την εμφάνιση μηνυμάτων από το πρόγραμμα προς τον χρήστη.

2.1 Εγκατάσταση NetBeans και JDK

Μεταβείτε στην τοποθεσία https://netbeans.apache.org/download/nb125/nb125.html και κατεβάστε τον κατάλληλο installer ανάλογα με το λειτουργικό σύστημά σας. Τρέξτε τον installer. Δεχτείτε τις προτεινόμενες επιλογές κατά την διάρκεια της εγκατάστασης. Με την ολοκλήρωση της εγκατάστασης θα έχετε διαθέσιμα, το

περιβάλλον ανάπτυξης, την JSE, την JEE, τις HTML και Javascript και την PHP. Θα έχετε έτσι ένα ολοκληρωμένο περιβάλλον ανάπτυξης εφαρμογών. Βρείτε το εικονίδιο του NetBeans στην επιφάνεια εργασίας και τρέξτε το. Σε περίπτωση που έχετε ήδη παλαιότερη έκδοση του NetBeans εγκατεστημένη, θα ερωτηθείτε αν επιθυμείτε να κάνετε εισαγωγή των ρυθμίσεων της στην νέα έκδοση. Απαντήστε κατάλληλα. Εφόσον, οι ρυθμίσεις στην προηγούμενη έκδοση είναι ορθές, η εισαγωγή τους θα σας φανεί πολλή χρήσιμη. Ας σημειωθεί πως αυτός ο installer εγκαθιστά την έκδοση NetBeans 1.25 με το JDK 17 που αποτελούν τις τελευταίες εκδόσεις και για τα δύο προϊόντα κατά τον χρόνο συγγραφής αυτού του κειμένου.

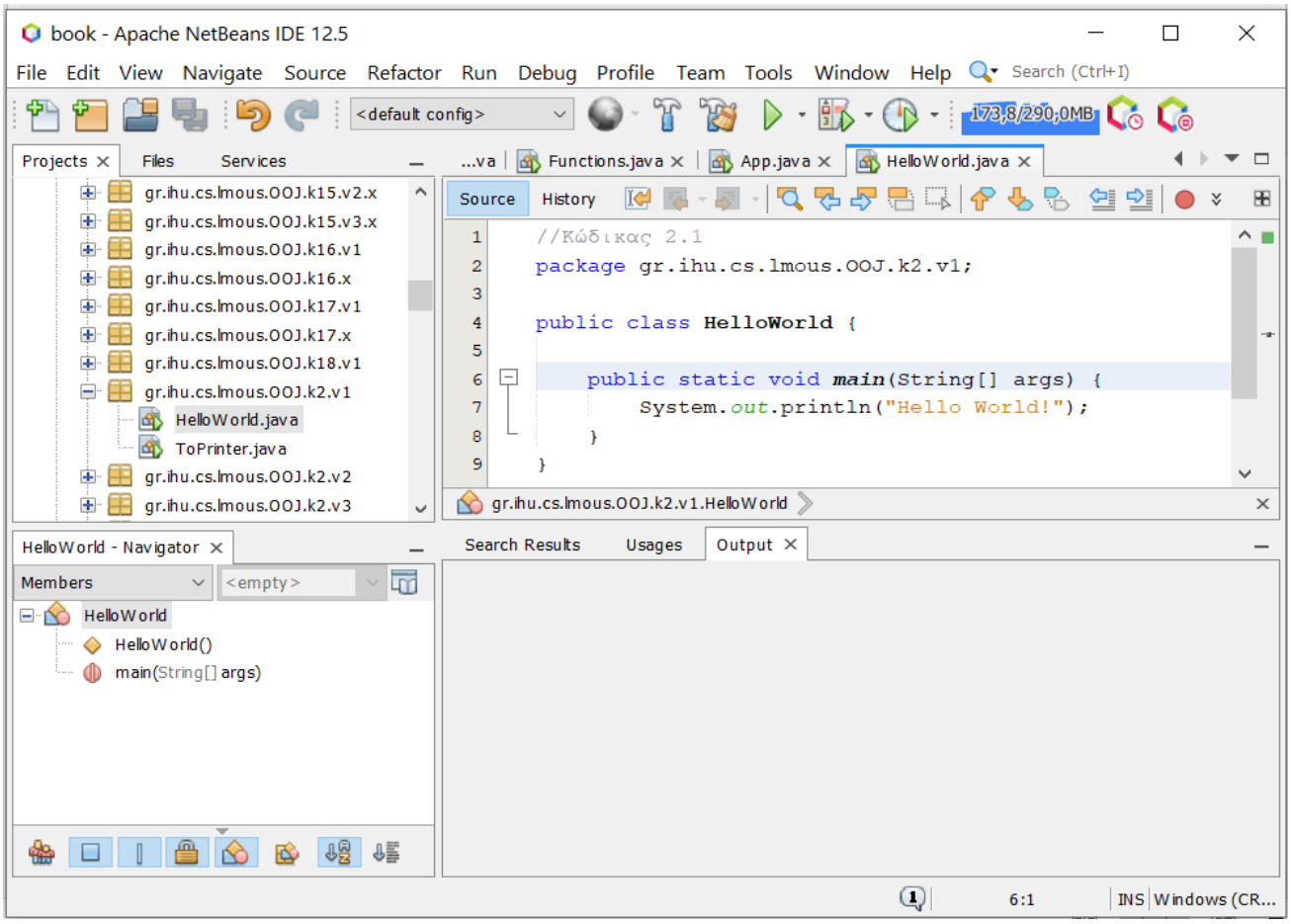
Είστε έτοιμοι να ξεκινήσετε την περιπέτειά σας στον Προγραμματισμό.

2.2 Το πρώτο Πρόγραμμα

Μεταβείτε στο μενού File του NetBeans και επιλέξτε New Project. Στο παράθυρο που θα ανοίξει, επιλέξτε Categories/Java with Ant και Projects/Java Class Library. Πατήστε Next. Στο παράθυρο που θα εμφανιστεί δώστε κατάλληλο όνομα στο project, καθορίστε την τοποθεσία στην οποία θα εμφανιστεί, πατήστε Finish και το project σας έχει δημιουργηθεί. Μπορείτε να καθορίσετε το όνομα της επιλογής σας για το project.

Διαμορφώστε το περιβάλλον εργασίας όπως φαίνεται στην εικόνα 1. Σε αυτήν την διαμόρφωση περιλαμβάνονται τρία τμήματα. Στα αριστερά είναι τοποθετημένο πλαίσιο με τρία παράθυρα, Projects, Files και Services. Δεξιά επάνω, ο editor, δηλ. το παράθυρο στο οποίο γράφουμε τον κώδικά μας και δεξιά κάτω το Output, δηλ. το παράθυρο στο οποίο εμφανίζεται η έξοδος του προγράμματός μας. Αν κάποιο από αυτά τα τμήματα δεν φαίνεται, μεταβείτε στο μενού Window του NetBeans και ανοίξτε το.

Μεταβείτε στο τμήμα Projects, επεκτείνεται το project σας πατώντας στο + που βρίσκεται στα αριστερά του, θα δείτε του φακέλους του project. Όπως βλέπετε στο project περιλαμβάνονται δύο φάκελοι, ο Source Packages και ο Libraries. Ανοίξτε τον φάκελο Source Packages. Θα διαπιστώσετε πως περιλαμβάνει ένα φάκελο που ονομάζεται Default Package. Ο φάκελος αυτός συνιστά έναν χώρο ή ένα πακέτο όπως λέγεται στον οποίον μπορούμε να τοποθετήσουμε πηγαίους κώδικες. Γενικά τα αρχεία με τους κώδικες τα τοποθετούμε σε πακέτα. Ειδικά όμως η χρήση του Default Package συνίσταται να αποφεύγεται. Αντίθετα, θα πρέπει να δημιουργούμε πακέτα ανάλογα με τις ανάγκες κάθε project. Κάντε δεξί κλικ στο όνομα του project. Από το μενού που θα ανοίξει, επιλέξτε New\Java Package. Δώστε ένα όνομα στο πακέτο και πατήστε finish. Θα διαπιστώσετε πως ο φάκελος Default Package έχει εξαφανισθεί ενώ έχει εμφανισθεί ένας φάκελος με το όνομα που δώσατε στο πακέτο σας. Μέσα σε αυτόν θα τοποθετηθεί το πρώτο πρόγραμμά μας.



***Εικόνα 2.1**: Διαμόρφωση πλαισίου εργασίας:*

Κάντε δεξί κλικ στο πακέτο που δημιουργήσατε, επιλέξτε New\Java Class, ονομάστε την κλάση HelloWorld και πατήστε finish. Προσέξτε πως στο πακέτο σας προστέθηκε ένα αρχείο με το όνομα HelloWorld.java. Παράλληλα, το αρχείο άνοιξε στο τμήμα του editor και είναι έτοιμο για επεξεργασία. Ήδη το NetBeans έχει προσθέσει ένα βασικό κώδικα για να μας βοηθήσει. Πιο συγκεκριμένα, έχει προσθέσει, το όνομα του πακέτου στην κορυφή του κώδικα και στην συνέχεια την δήλωση της κλάσης HelloWorld.

Μετατρέψτε το αρχείο HelloWorld.java όπως δείχνει ο κώδικας 2.1. Αντικαταστήστε όμως το gr.ihu.cs.lmous.OOJ.k1.v1 με το όνομα του δικού σας πακέτου.

```
package gr.ihu.cs.lmous.OOJ.k2.v1;

public class HelloWorld {

    public static void main(String[] args) {
        System.out.println("Hello World!");
    }
}
```

***Κώδικας 2.1**: Το πρώτο πρόγραμμα*

Αυτός είναι ο πηγαίος κώδικας του πρώτου μας προγράμματος. Στο τμήμα Projects, κάντε δεξί κλικ πάνω στην κλάση HelloWorld και επιλέξτε Run File. Θα εκτελεστεί το πρόγραμμα με αποτέλεσμα να τυπώσει στο παράθυρο Output την σειρά χαρακτήρων HelloWorld!. Το NetBeans έκανε αυτόματα μεταγλώττιση του κώδικα σε bytecode και ζήτησε από την JVM να τον τρέξει. Αν μεταβείτε στον folder που έχετε αποθηκεύσει το project, θα διαπιστώσετε πως περιέχει μια σειρά από φακέλους. Μεταβείτε στον φάκελο build\classes\ yourPackagePath (Αν χρησιμοποιήσατε το ίδιο όνομα πακέτου με τον κώδικα 2.1, μεταβείτε στον φάκελο

build\classes\gr\ihu\cs\lmous\OOJ\k1\v1), θα δείτε ότι περιέχει ένα αρχείο HelloWorld.class, πρόκειται για το bytecode της κλάσης μας.

Για να τρέξετε την κλάση σας εκτός παραθυρικού περιβάλλοντος, ανοίξτε ένα κέλυφος του λειτουργικού συστήματος, π.χ. στα Windows ανοίξτε ένα powershell ή ένα command prompt. Μεταβείτε στον φάκελο build\classes, πληκτρολογήστε Java yourPackagePath.HelloWorld και Enter, θα δείτε την έξοδο του προγράμματός σας, δηλ. στο κέλυφος θα εκτυπωθεί η σειρά χαρακτήρων HelloWorld!.

Αυτός είναι σε γενικές γραμμές, ο τρόπος με τον οποίο θα δημιουργείτε projects, πακέτα και κλάσεις. Τις κλάσεις βέβαια θα τις τρέχετε κυρίως μέσα από το NetBeans καθώς είναι πολύ απλούστερη διαδικασία. Αργότερα, σε αυτό το βιβλίο, θα δούμε πώς να δημιουργούμε εφαρμογές που συνδυάζουν πολλές κλάσεις οπότε θα δούμε και έναν διαφορετικό τρόπο για να εκτελούμε τις εφαρμογές.

Ας προχωρήσουμε όμως στην ανάλυση αυτού του βασικού κώδικα. Καταρχάς, κάθε κλάση ανήκει σε ένα πακέτο (package). Το όνομα του πακέτου πρέπει να δηλώνεται απαραιτήτως μαζί με την δεσμευμένη λέξη package στην πρώτη γραμμή της κλάσης. Λεπτομέρειες για τα πακέτα, την χρησιμότητά τους και την ονοματολογία τους, μπορείτε να βρείτε στην ενότητα 7.8. Στην συνέχεια ακολουθεί η κλάση. Αυτή ορίζεται με χρήση της λέξης-κλειδί class ακολουθούμενη από το όνομα της κλάσης. Στην συνέχεια ανάμεσα σε μια αγκύλη που ανοίγει και μια που κλείνει, τοποθετούμε τον κώδικα της κλάσης. Προσέξτε πως πριν την λέξη-κλειδί class, έχει τοποθετηθεί η λέξη-κλειδί public. Η λέξη αυτή καθιστά την κλάση δημόσια, δηλ. οποιοσδήποτε διαθέτει το αρχείο HelloWorld.class μπορεί να χρησιμοποιήσει την κλάση. Η λέξη public είναι ένας προσδιοριστής προσπέλασης (access specifier). Θα μιλήσουμε αναλυτικότερα για τους προσδιοριστές προσπέλασης στην ενότητα 7.7. Θα πρέπει όμως να πούμε πως σε ένα αρχείο .java μπορεί να τοποθετηθεί μια μόνο public κλάση. Είναι υποχρεωτικό μάλιστα, αρχείο και κλάση να έχουν το ίδιο ακριβώς όνομα.

Μέσα στην κλάση ορίζεται μια συνάρτηση που ονομάζεται main. Για την ώρα, μπορούμε να θεωρήσουμε πως μια συνάρτηση είναι ένα υποπρόγραμμα ή αλλιώς μια ρουτίνα [1] που εκτελείται όταν το ζητήσουμε. Όπως θα δούμε αμέσως παρακάτω σε αυτήν την ενότητα, μπορούμε να ορίσουμε πολλές συναρτήσεις μέσα σε μια κλάση οι οποίες θα εκτελούνται μόνο όταν το ζητήσουμε σαφώς. Η συνάρτηση main διαφέρει. Αποτελεί την κύρια είσοδο στο πρόγραμμά μας. Έτσι όταν ζητάμε από την Java να τρέξει μια κλάση, αυτή ψάχνει να βρει την συνάρτηση main. Αν η κλάση μας δεν περιλαμβάνει main, τότε δεν είναι εκτελέσιμη. Έτσι είτε τρέξουμε την κλάση μας στο NetBeans είτε την τρέξουμε σε κέλυφος του λειτουργικού, η συνάρτηση εκκίνησης είναι η main. Περισσότερα για την main στην ενότητα 7.3.4. Προσέξτε πως η λέξη main ακολουθείται από την σειρά χαρακτήρων (String[] args). Πρόκειται για την λίστα παραμέτρων (parameter list) της main. Δεν θα επεκταθούμε περισσότερο σε αυτό για την ώρα. Εκείνο που πρέπει να γνωρίζουμε είναι ότι η main που αποτελεί είσοδο στο πρόγραμμα μας υποχρεωτικά ακολουθείται από την συγκεκριμένη λίστα παραμέτρων. Περισσότερα για τις λίστες παραμέτρων των συναρτήσεων γενικά στην ενότητα 7.3 και της main ειδικά στην 7.3.4. Μπροστά από την main έχουμε τοποθετήσει τις λέξεις public static void. Το public έχει επίσης την έννοια πως οποιοσδήποτε διαθέτει το bytecode της κλάσης μπορεί να τρέξει την συνάρτηση. Το static θα το τοποθετούμε μπροστά από όλες τις συναρτήσεις που θα γράφουμε έως ότου εισαχθούμε στο αντικειμενοστρεφές μοντέλο. Τότε θα δούμε μη στατικές συναρτήσεις, δηλ. συναρτήσεις που δεν χαρακτηρίζουμε ως static και θα διαφοροποιήσουμε με λεπτομέρεια μεταξύ των δύο. Τέλος, το void είναι ο τύπος της τιμής επιστροφής της συνάρτησης. Περισσότερα για τις τιμές επιστροφής των συναρτήσεων στην ενότητα 7.1.

Ο κώδικας που περιέχεται στην main είναι αυτός που στέλνει στην τυπική έξοδο (standard output), δηλ. στην οθόνη, την σειρά HelloWorld!. Επομένως, αν θέλουμε να τυπώσουμε μια σειρά χαρακτήρων στην οθόνη, μπορούμε να χρησιμοποιούμε την προκαθορισμένη στην Java συνάρτηση System.out.println ακολουθούμενη από παρενθέσεις μέσα στις οποίες τοποθετούμε την σειρά χαρακτήρων περικλείοντάς την σε διπλά εισαγωγικά. Γενικά, οποτεδήποτε αναφερόμαστε μέσα στο πρόγραμμά μας σε σταθερή σειρά χαρακτήρων θα πρέπει να την εσωκλείουμε σε διπλά εισαγωγικά. Βεβαίως, η System.out.println χρησιμοποιείται για την εκτύπωση και άλλου τύπου δεδομένων και όχι μόνο για σειρές χαρακτήρων.

2.3 Οι συναρτήσεις ως απλά υποπρογράμματα

Παρουσιάζουμε εδώ μια εναλλακτική προσέγγιση για την εκτύπωση της σειράς “HelloWorld”. Θα φτιάξουμε μια νέα κλάση που θα ονομάσουμε ToPrinter. Μέσα στην ToPrinter, θα υλοποιήσουμε μια συνάρτηση που θα ονομάσουμε helloWorld. Στην συνέχεια στην main θα καλέσουμε την helloWorld.

Με δεξί κλικ στο πακέτο σας, δημιουργήστε την κλάση ToPrinter. Διαμορφώστε το περιεχόμενο της ToPrinter όπως φαίνεται στον κώδικα 2.2. Σημειώστε πως στο εξής, η ονομασία του πακέτου θα παραλείπεται από τους κώδικες σε αυτό το βιβλίο. Ωστόσο, για να είναι οι κώδικες μεταγλωττίσιμοι, θα πρέπει να προσθέτετε εσείς την δήλωση package συνοδευόμενη από το όνομα του δικού σας πακέτου.

```
package gr.ihu.cs.lmous.OOJ.k2.v1;

public class ToPrinter {

    static void helloWorld() {
        System.out.println("HelloWorld!");
    }

    public static void main(String[] args) {
        helloWorld();
    }
}
```

***Κώδικας 2.2**: Η πρώτη έκδοση της κλάσης ToPrinter*

Αν τρέξουμε την κλάση, θα λάβουμε το ίδιο αποτέλεσμα με την εκτέλεση της κλάσης HelloWorld. Όπως βλέπουμε στον κώδικα 2.2, η main καλεί την helloWorld() η οποία με την σειρά της καλεί την System.out.println που τυπώνει την σειρά "HelloWorld!". Οι συναρτήσεις είναι υποπρογράμματα ή αλλιώς ρουτίνες και αποτελούν θεμέλιο λίθο τόσο του διαδικαστικού όσο και του αντικειμενοστρεφούς μοντέλου και θα τις χρησιμοποιούμε συστηματικά κατά την ανάπτυξη των κλάσεων. Βεβαίως, η συνάρτηση helloWorld αποτελεί την απλούστερη μορφή συνάρτησης. Είναι μια στατική void συνάρτηση χωρίς παραμέτρους. Εκτεταμένα για τις συναρτήσεις μιλάμε στην ενότητα 7. Για την ώρα, μας αρκεί αυτή η απλή μορφή.

Αν υποθέσουμε τώρα πως θέλουμε να εκτυπώσουμε την διεύθυνσή μας χρησιμοποιώντας την κλάση ToPrinter, μπορούμε να προσθέσουμε μια κατάλληλη συνάρτηση και να την καλέσουμε στην κλάση όπως δείχνει ο κώδικας 2.3.

```
public class ToPrinter {

    static void helloWorld() {
        System.out.println("HelloWorld!");
    }

    static void address() {
        System.out.println("Ιωάννης Νάκος");
        System.out.println("Εσπερίδων 75");
        System.out.println("Καλλιθέα 65303");
        System.out.println("Αθήνα, Ελλάδα");
    }

    public static void main(String[] args) {
        helloWorld();
        address();
    }
}
```

***Κώδικας 2.3**: Η δεύτερη έκδοση της κλάσης ToPrinter*

Προσέξτε όμως θα πρέπει να τοποθετήσετε την δεύτερη έκδοση της ToPrinter σε διαφορετικό πακέτο από την πρώτη ή διαφορετικά να αλλάξετε την ονομασία της κλάσης. Τρέχοντας τον κώδικα 2.3, διαπιστώνουμε πως τυπώνεται πρώτα η σειρά "HelloWorld" και στην συνέχεια η διεύθυνση. Ένας εύκολος τρόπος για να τυπωθεί μόνο η διεύθυνση είναι να σχολιάσουμε την κλήση της HelloWorld, δηλ. να μετατρέψουμε την main όπως δείχνει ο κώδικας 2.4.

```
    public static void main(String[] args) {
        //helloWorld();
        address();
    }
```

***Κώδικας 2.4**: Σχόλια γραμμής*

Όπως βλέπουμε στον κώδικα 2.4, από την κλήση helloWorld() έχουν προηγηθεί δύο ανάστροφοι κάθετοι (backslashes). Με αυτόν τον τρόπο πληροφορούμε τον μεταγλωττιστή πως αυτή η γραμμή είναι γραμμή σχολίων και όχι κώδικα. Τα σχόλια αγνοούνται από τον μεταγλωττιστή. Έτσι, στον bytecode που παράγει ο μεταγλωττιστής δεν συμπεριλαμβάνεται η κλήση helloWorld. Αν εκτελέσουμε τώρα την κλάση ToPrinter, θα διαπιστώσουμε ότι τυπώνει μόνο την διεύθυνση.

Τα σχόλια είναι γενικά απαραίτητα στον κώδικά μας. Όχι μόνο για να αποκλείουμε προσωρινά την κλήση συναρτήσεων αλλά κυρίως για να τοποθετούμε επεξηγήσεις μέσα στον κώδικα ώστε να γίνεται εύκολα κατανοητός από εμάς σε μεταγενέστερο χρόνο αλλά και από συνεργάτες μας που ενδέχεται να δουλέψουν με τον κώδικά μας. Επιπλέον, όπως θα δούμε στην ενότητα 8.6, τα σχόλια αξιοποιούνται και για την παραγωγή της τεκμηρίωσης των κλάσεων. Σε περίπτωση που θέλουμε να εισάγουμε σχόλια που καταλαμβάνουν περισσότερες από μία γραμμές, μπορούμε να τα εσωκλείσουμε μεταξύ των χαρακτήρων ανάστροφη κάθετος και αστερίσκος στην αρχή και αστερίσκος και ανάστροφη κάθετος στο τέλος όπως φαίνεται στον κώδικα 2.5.

```
/*
Στατική συνάρτηση χωρίς παραμέτρους
Τιμή επιστροφής: void
Τυπώνει συγκεκριμένη διεύθυνση
*/
    static void address() {
        System.out.println("Ιωάννης Νάκος");
        System.out.println("Εσπερίδων 75");
        System.out.println("Καλλιθέα 65303");
        System.out.println("Αθήνα, Ελλάδα");
    }
```

***Κώδικας 2.5**: Σχόλια πολλαπλών γραμμών*

Θα προσέξατε πως ο προσδιοριστής προσπέλασης public λείπει από τις δηλώσεις των συναρτήσεων helloWorld και address. Όταν στην δήλωση μιας συνάρτησης δεν τοποθετείται κανένας προσδιοριστής προσπέλασης, η συνάρτηση διατηρεί την εξ' ορισμού προσπέλαση (default access). Η εξ' ορισμού προσπέλαση ονομάζεται και προσπέλαση πακέτου (package access) ή και ιδιωτική προσπέλαση πακέτου (package-private) καθώς αυτή επιτρέπει την προσπέλαση μόνο από κλάσεις του ιδίου πακέτου. Στο πακέτο που τοποθετήσατε την δεύτερη έκδοση της ToPrinter, δημιουργήστε μια κλάση CallFromAnotherClass. Η κλάση CallFromAnotherClass έχει δικαίωμα προσπέλασης στις συναρτήσεις της ToPrinter. Αντίθετα κλάσεις από άλλα πακέτα δεν έχουν αυτό το δικαίωμα. Για να κληθεί όμως μια συνάρτηση της ToPrinter από την CallFromAnotherClass θα πρέπει κατά την κλήση να προσδιορίσουμε πέραν του ονόματος της συνάρτησης και την κλάση στην οποία αυτή ανήκει.
Ο κώδικας 2. 6. παρουσιάζει την κλάση CallFromAnotherClass

```
public class CallFromAnotherClass {
    public static void main(String[] args) {
        ToPrinter.helloWorld();
        ToPrinter.address();
    }
}
```

***Κώδικας 2.6**: Κλήση στατικής συνάρτησης με πρόσβαση πακέτου από άλλη κλάση του ίδιου πακέτου*

Τρέξτε την CallFromAnotherClass και θα διαπιστώσετε πως θα εκτελεστούν οι συναρτήσεις address και helloWorld της κλάσης ToPrinter. Αν θέλουμε να δώσουμε δικαίωμα κλήσης αυτών των συναρτήσεων από κλάσεις έξω από το πακέτο που ανήκουν θα πρέπει να χαρακτηρίσουμε τις συναρτήσεις ως public.

2.4 Συμβάσεις ονοματολογίας

Η Java διαφοροποιεί μεταξύ πεζών και κεφαλαίων (case sensitive). Αν για παράδειγμα προσπαθήσετε να καλέσετε την συνάρτηση helloWorld και την γράψετε ως helloworld, η Java δεν θα την αναγνωρίσει. Για την ονομασία των κλάσεων, των συναρτήσεων αλλά και άλλων αντικειμένων έχουν καθιερωθεί συμβάσεις ονοματολογίας με παγκόσμια ισχύ που θα πρέπει να ακολουθούνται πιστά. Ο μεταγλωττιστής δεν θα μας ενοχλήσει αν παραβούμε μια τέτοια σύμβαση. Ωστόσο, η τήρησή τους είναι αναγκαία ώστε οι κώδικές μας να είναι αναγνώσιμοι από εμάς ή από τρίτους. Πιο συγκεκριμένα, τα ονόματα των κλάσεων πρέπει να αρχίζουν με κεφαλαίο. Αντίθετα, τα ονόματα των συναρτήσεων πρέπει να αρχίζουν από πεζό. Και στις δύο περιπτώσεις, αν

το όνομα είναι σύνθετο, π.χ. helloWorld, κάθε επόμενο συνθετικό πρέπει να αρχίζει από κεφαλαίο. Πιο εκτεταμένα για το θέμα αυτό μιλάμε στην ενότητα 3.4 όπου παρουσιάζουμε τους χαρακτήρες που είναι διαθέσιμοι προς χρήση για την διαμόρφωση των ονομάτων κλάσεων, συναρτήσεων και άλλων αναγνωριστικών (identifier) που χρησιμοποιούμε στα προγράμματά μας.

2.5 Βασική Έξοδος

Είδαμε προηγουμένως πως με την System.out.println μπορούμε να τυπώσουμε μια σειρά χαρακτήρων στην οθόνη. Θα προσέξατε πως κάθε γραμμή της διεύθυνσης τυπώνεται σε διαφορετική γραμμή της οθόνης. Αυτό οφείλεται στην println που αφού εμφανίζει την σειρά χαρακτήρων που της έχουμε δώσει, προετοιμάζει την έξοδο έτσι ώστε αυτή να συνεχίσει σε νέα γραμμή. Οι χαρακτήρες ln στο τέλος της println αυτό τον σκοπό εξυπηρετούν. Αν αυτό είναι κάτι που δεν θέλουμε, τότε αντί για println θα χρησιμοποιήσουμε την συνάρτηση print.

Προσθέστε την συνάρτηση prtABC (κώδικας 2.7) στην ToPrinter, καλέστε την στην main και τρέξτε την κλάση. Η νέα έκδοση της κλάσης φαίνεται στον κώδικα 2.7.

```
public class ToPrinter {

    static void helloWorld() {
        System.out.println("HelloWorld!");
    }

    static void address() {
        System.out.println("Ιωάννης Νάκος");
        System.out.println("Εσπερίδων 75");
        System.out.println("Καλλιθέα 65303");
        System.out.println("Αθήνα, Ελλάδα");
    }

    static void prtABC() {
        System.out.print("A");
        System.out.print("B");
        System.out.println("C");
    }

    public static void main(String[] args) {
        prtABC();
        helloWorld();
    }
}
```

***Κώδικας 2.7**: print και println*

Παρατηρήστε πως στην τελευταία γραμμή της prtABC έχουμε χρησιμοποιήσει println και όχι print. Επιλέξαμε αυτήν την προσέγγιση ώστε οποιαδήποτε έξοδος μετά την κλήση της prtABC να εμφανιστεί σε νέα γραμμή. Έτσι ο κώδικας 2.7 εμφανίζει το HelloWorld! κάτω από την σειρά ABC. Αν δεν είχαμε βάλει println στην τελευταία print της prtABC τότε ο κώδικας θα εμφάνιζε ABCHelloWorld!. Περισσότερα για την έξοδο και είσοδο στις ενότητες 4.2 και 4.3.

2.6 Ασκήσεις

Δημιουργήστε μια κλάση X01. Σε αυτήν προσθέστε τις συναρτήσεις:

prtRomvos: Τυπώνει έναν ρόμβο που σχηματίζεται από αστερίσκους όπως φαίνεται παρακάτω

```
    ***
  **********
**********
```

```
**********
  **********
    ***
```

prtVellos: Τυπώνει ένα βέλος όπως φαίνεται παρακάτω

```
          *****
            *****
              *****
                *****
**********************************
**********************************
**********************************
                *****
              *****
            *****
          *****
```

Τρέξτε τις συναρτήσεις και διαπιστώστε πως λειτουργούν σωστά.

Βιβλιογραφία

[1] “Subroutine,” Wikipedia. Sep. 12, 2021. Accessed: Sep. 14, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Subroutine&oldid=1043969790

Κεφάλαιο 3

Σύνοψη
Σε αυτήν την ενότητα εξετάζεται διεξοδικά η έννοια της τοπικής μεταβλητής και αναλύονται σε βάθος οι θεμελιώδεις τύποι δεδομένων της Java. Περιλαμβάνεται παρουσίαση των ακολουθιών ελέγχου, των συμβάσεων ονοματολογίας, του φαινομένου υπερχείλισης μεταβλητής. Εξηγείται η έκφραση αριθμητικών σταθερών στο δυαδικό, οκταδικό και δεκαεξαδικό σύστημα αρίθμησης καθώς και ο επιστημονικός συμβολισμός τους. Επιπλέον, γίνεται εισαγωγή στον βασικό τύπο διαχείρισης αλφαριθμητικών δεδομένων.

Προαπαιτούμενη γνώση
Η ενότητα 2 αυτού του εγχειριδίου που περιλαμβάνει τις έννοιες: πακέτο (package), η main ως κύρια είσοδος στο πρόγραμμα, το υποπρόγραμμα ως void συνάρτηση χωρίς παραμέτρους, βασική έξοδος στην οθόνη.

Λέξεις κλειδιά
τοπική μεταβλητή (local variable), τοπική σταθερά (local constant), μπλοκ κώδικα (block), εμβέλεια (scope), χρόνος ζωής μεταβλητής (variable life time), αναγνωριστικό (identifier), κυριολεκτική σταθερά (literal), αρχικοποίηση (initialization), χρόνος μεταγλώττισης (Compile time), χρόνος εκτέλεσης (Run time), δεσμευμένη λέξη (reserved word), λέξη κλειδί (keyword), θεμελιώδεις τύποι δεδομένων (Primitive Data Types), κλάση (Class), εκθετική μορφή (exponential Notation) ή επιστημονικός συμβολισμός (scientific notation), αριθμός κινητής υποδιαστολής (floating point number), ακολουθία ελέγχου (control sequence), χαρακτήρας διαφυγής (escape character), υπερχείλιση (overflow), λάθος χρόνου εκτέλεσης (run time error ή bug), λάθος μεταγλώττισης (compile error), αυτόματη μετατροπή τύπου (automatic type conversion) , αλφαριθμητική σειρά (String), σύνδεση αλφαριθμητικών σταθερών (concatenation).

3 Μεταβλητές και Θεμελιώδεις τύποι

Αναφέραμε προηγουμένως πως ένα πρόγραμμα είναι μια σειρά εντολών προς την μηχανή. Αυτό είναι σωστό, ωστόσο πρόκειται για την μισή αλήθεια καθώς οι εντολές συχνά επενεργούν σε δεδομένα. Επομένως, ένα πρόγραμμα για να εκτελεστεί θα πρέπει να φορτωθούν, στην μνήμη του υπολογιστή, όχι μόνο οι εντολές που το απαρτίζουν αλλά και τα δεδομένα στα οποία επενεργούν οι εντολές. Το πρόγραμμα φορτώνει τα δεδομένα σε κατάλληλες θέσεις στην μνήμη προς τις οποίες διατηρεί εύκολη πρόσβαση και από εκεί τα διαχειρίζεται. Η πρόσβαση στις θέσεις μνήμης που αντιστοιχούν σε δεδομένα γίνεται μέσω των μεταβλητών (variables).

Ωστόσο, υπάρχουν δεδομένα διάφορων τύπων. Για παράδειγμα, κάποια δεδομένα μπορεί να είναι ακέραιοι αριθμοί, κάποια άλλα μπορεί να είναι πραγματικοί αριθμοί ή χαρακτήρες. Όμως τα δεδομένα αποθηκεύονται στην μνήμη σαν σειρές από bits. Επομένως, το πρόγραμμα θα πρέπει να γνωρίζει η κάθε σειρά από bits που αντιστοιχεί σε ορισμένο δεδομένο, τι τύπου είναι ώστε να μπορέσει να την διαχειριστεί σωστά. Έτσι οι μεταβλητές χαρακτηρίζονται ως προς τον τύπο των δεδομένων που διαχειρίζονται, δηλ. υπάρχουν μεταβλητές κατάλληλες για ακέραιους, μεταβλητές κατάλληλες για πραγματικούς, κοκ.

Στο κεφάλαιο αυτό, παρουσιάζουμε καταρχάς τις τοπικές μεταβλητές και κάποια βασικά χαρακτηριστικά τους όπως η εμβέλεια και ο χρόνος ζωής, στην συνέχεια παρουσιάζουμε τους θεμελιώδεις τύπους της Java. Μετά εξηγούμε την σημασία και χρήση των ακολουθιών ελέγχου και περιγράφουμε τα αποδεκτά ονόματα μεταβλητών. Ακολουθεί, η επεξήγηση της υπερχείλισης και η έκφραση αριθμητικών σταθερών σε όλα τα συστήματα αρίθμησης που υποστηρίζονται. Τέλος, παρουσιάζεται ο τύπος αλφαριθμητικής σταθεράς ο οποίος δεν είναι μεν θεμελιώδης αλλά το ίδιο βασικός και συχνά αναγκαίος όσο και οι θεμελιώδεις τύποι.

3.1 Τοπικές Μεταβλητές

Ένα πρόγραμμα χρειάζεται να διαχειριστεί κάποια δεδομένα. Η διαχείριση των δεδομένων επιτυγχάνεται με την βοήθεια των μεταβλητών (variables). Η Java υποστηρίζει μια ποικιλία μεταβλητών, τις τοπικές μεταβλητές

(local variables), τις παραμέτρους (parameters), τις στατικές μεταβλητές (class variables ή static fields) και τις μεταβλητές στιγμιότυπου (instance variables)[1] .

Η συζήτηση στην ενότητα αυτή θα περιοριστεί στις τοπικές μεταβλητές. Οι παράμετροι και οι στατικές μεταβλητές παρουσιάζονται στις ενότητες 7.3 και 7.4, αντίστοιχα, ενώ οι μεταβλητές στιγμιότυπου αφορούν το αντικειμενοστρεφές μοντέλο και εξετάζονται στην ενότητα 11.

Για να δηλώσουμε μια τοπική μεταβλητή[2] στην Java, πρέπει να προσδιορίσουμε τον τύπο της και να της δώσουμε ένα αναγνωριστικό όνομα. Για παράδειγμα, ο κώδικας

```
int k;
```

δηλώνει μια ακέραιη μεταβλητή με αναγνωριστικό το k.

Ας δούμε όμως μερικούς αναγκαίους ορισμούς.

Μπλοκ κώδικα: Ο κώδικας μεταξύ μιας αγκύλης που ανοίγει '{' και μιας αγκύλης που κλείνει '}' συνιστά ένα μπλοκ (block) κώδικα.

Κάθε μεταβλητή που δηλώνεται μέσα σε ένα μπλοκ κώδικα είναι τοπική σε αυτό το μπλοκ.

```
void f() {
     int k;
}
```

Στην συνάρτηση f(), έχουμε δηλώσει την τοπική μεταβλητή ακέραιου τύπου k. Ο int είναι ένας θεμελιώδης τύπος της Java. Οι θεμελιώδεις τύποι των μεταβλητών παρουσιάζονται παρακάτω στην ομώνυμη ενότητα. Για την ώρα μας ενδιαφέρουν τα ακόλουθα:

Η εκτέλεση της int k; έχει ως αποτέλεσμα να δεσμευτεί χώρος στην μνήμη που έχει εκχωρηθεί στο πρόγραμμά μας, με μέγεθος ίσο με το μέγεθος μια ακέραιης μεταβλητής. Ο χώρος αυτός συνδέεται με το αναγνωριστικό (identifier) k. Η μεταβλητή k είναι τοπική στο μπλοκ που ορίζει την f. Κάθε φορά που μέσα στην f αναφερόμαστε στην k, στην ουσία προσπελαύνουμε την συγκεκριμένη μνήμη. Ωστόσο, δεν έχουμε δώσει κάποια αρχική τιμή στην k. Με άλλα λόγια, η k είναι αναρχικοποίητη (unitialized)[3]. Προκειμένου να μας προστατεύσει από πιθανά λάθη που προκύπτουν από την χρήση αναρχικοποίητων μεταβλητών, η Java δεν μας επιτρέπει να την προσπελάσουμε παρά μόνο για να την αρχικοποιήσουμε[4]. Έτσι ο παρακάτω κώδικας παράγει ένα λάθος μεταγλώττισης (Compile Error).

```
void f() {
     int k;
     System.out.println(k);
}
```

Αντίθετα, ο κώδικας

```
void f() {
     int k;
```

---

[1] *Επιπλέον, οι μεταβλητές ανάλογα με τον τύπο τους μπορεί να είναι μεταβλητές αναφοράς (reference variable) ή **μεταβλητές τιμής** (value variable).*

[2] *Πριν τον τύπο μπορεί να τοποθετηθούν τροποποιητές (modifiers). Στους πιθανούς τροποποιητές περιλαμβάνονται οι **προσδιοριστές προσπέλασης** (access specifer), η δεσμευμένη λέξη **static** και η δεσμευμένη λέξη **final**. Από αυτούς τους τροποποιητές μόνο ο **final** συντάσσεται με δήλωση τοπικών μεταβλητών*

[3] *Η Java δεν αρχικοποιεί αυτόματα τις τοπικές μεταβλητές. Δεν ισχύει το ίδιο για τους υπόλοιπους τύπους μεταβλητών.*

[4] *Μπορούμε ωστόσο να προσπελάσουμε μια μεταβλητή που ενδεχομένως είναι αναρχικοποίητη εφόσον την περικλείσουμε σε try {} catch μπλοκ. Περισσότερα επ'αυτού κατά την μελέτη των εξαιρέσεων, ενότητα ?*

```
        k = 0;
        System.out.println(k);
}
```

περνάει μεταγλώττιση και όταν εκτελεστεί τυπώνει στην οθόνη το περιεχόμενο της k, δηλ. το 0.

Εμβέλεια: Εμβέλεια (Scope) μιας μεταβλητής ονομάζεται η περιοχή εκείνη του κώδικα που η μεταβλητή είναι αναγνωρίσιμη από τον μεταγλωττιστή.

Η εμβέλεια των τοπικών μεταβλητών αρχίζει από την γραμμή στην οποία δηλώθηκαν και επεκτείνεται σε όλο το μπλοκ μέσα στο οποίο δηλώθηκαν. Στον κώδικα που ακολουθεί, ο μεταγλωττιστής θα παραγάγει λάθος καθώς προσπαθούμε να προσπελάσουμε την μεταβλητή i έξω από το μπλοκ στο οποίο δηλώθηκε.

```
void f() {
        int k = 0;
        System.out.println(k);
        {
                int i = 1;
        }
        System.out.println(i);
}
```

Προσέξτε πως σε αυτό το παράδειγμα, έχουμε εμφωλιασμένα (nested) μπλοκ, δηλ. έχουμε το μπλοκ της f και μέσα σε αυτό, ένα εσωτερικό μπλοκ μέσα στο οποίο δηλώνεται η i. Η εμβέλεια επομένως της i περιορίζεται στο εσωτερικό μπλοκ. Γενικά, η εμβέλεια των τοπικών μεταβλητών περιορίζεται μέσα στο μπλοκ που δηλώθηκαν. Αυτός είναι ένας λόγος στον οποίο οφείλεται ο χαρακτηρισμός τους ως τοπικές. Ένας δεύτερος λόγος σχετίζεται με τον χρόνο ζωής τους.

Χρόνος ζωής: Ο χρόνος ζωής (variable life time) μιας μεταβλητής είναι το διάστημα από την δημιουργία της μεταβλητής με την δέσμευση της κατάλληλης μνήμης μέχρι την καταστροφή της με την αποδέσμευση της μνήμης.

Για παράδειγμα, κατά την εκτέλεση της πρώτης γραμμής της f, δεσμεύεται χώρος στην μνήμη και έχουμε την έναρξη της ζωής της μεταβλητής. Με το τέλος εκτέλεσης της f(), ο χώρος για την k αποδεσμεύεται και έχουμε την λήξη της ζωής της. Στο παράδειγμα αυτό η εμβέλεια μοιάζει να είναι παρόμοια έννοια με τον χρόνο ζωής. Ωστόσο, εμβέλεια και χρόνος ζωής συνιστούν δύο εντελώς διαφορετικές έννοιες. Η μεν εμβέλεια ορίζεται στο πλαίσιο του χρόνου μεταγλώττισης (Compile time), ο δε χρόνος ζωής ορίζεται στο πλαίσιο του χρόνου εκτέλεσης (Run time). Στην ενότητα 11.5 διαφοροποιείται πληρέστερα η εμβέλεια από τον χρόνο ζωής.

Η Java δεν επιτρέπει την δήλωση μεταβλητής σε εσωτερικό μπλοκ με αναγνωριστικό που έχει χρησιμοποιηθεί σε αντίστοιχο εξωτερικό. Για παράδειγμα, ο κώδικας

```
void f() {
        int k = 0;
        {
                int k = 1;
        }
}
```

θα παραγάγει λάθος μεταγλώττισης.
Ας σημειωθεί ότι δήλωση και αρχικοποίηση μιας ή περισσότερων μεταβλητών μπορεί να γίνει στην ίδια γραμμή κώδικα, π.χ.

```
int k=0;
int k=1, z=2;
int k=2, z;
```

Στην πρώτη περίπτωση δηλώνουμε και αρχικοποιούμε την k, στην δεύτερη δηλώνουμε και αρχικοποιούμε τις k και z και στην τρίτη δηλώνουμε τις k και z, αρχικοποιούμε όμως μόνο την k.

3.1.2 Τοπικές Σταθερές

Σε πολλές περιπτώσεις χρειαζόμαστε μεταβλητές που η τιμή τους δεν μπορεί να αλλάξει μετά την αρχική εκχώρηση. Οι μεταβλητές αυτού του τύπου ονομάζονται σταθερές (constant variables). Μια τοπική μεταβλητή μετατρέπεται σε σταθερά αν κατά την δήλωσή της χρησιμοποιηθεί η δεσμευμένη λέξη (reserved word) final. Για παράδειγμα,

```
final int K=1;
```

Ο κώδικας αυτός μέσα σε ένα μπλοκ δηλώνει μια τοπική σταθερά. Σε μια τοπική σταθερά μπορεί να εκχωρηθεί τιμή αυστηρά μία φορά, είτε μαζί με την δήλωσή της είτε αργότερα. Επομένως, ο κώδικας

```
final int K;
K=1;
```

είναι έγκυρος.
Αντίθετα, ο κώδικας

```
final int K=1;
K=2;
```

παράγει λάθος μεταγλώττισης.

Όπως φαίνεται από τα παραδείγματα αυτής της ενότητας, οι συμβάσεις ονοματολογίας για τις σταθερές διαφέρουν από τις αντίστοιχες συμβάσεις για μεταβλητές. Οι σταθερές ονομάζονται αποκλειστικά με χρήση κεφαλαίων χαρακτήρων.

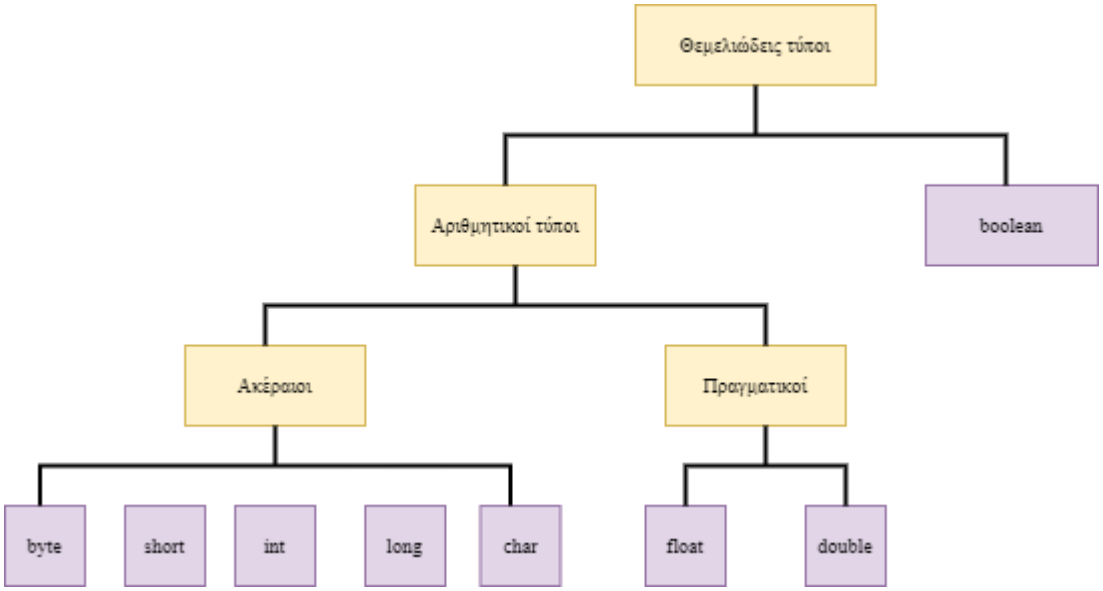
3.2 Θεμελιώδεις τύποι δεδομένων

Όπως εξηγήσαμε στην προηγούμενη ενότητα, τα προγράμματα διαχειρίζονται τα δεδομένα τους με την βοήθεια των μεταβλητών. Τα δεδομένα είναι διαφόρων τύπων, π.χ. ακέραιοι αριθμοί, πραγματικοί, χαρακτήρες, κλπ. Αντίστοιχα, και οι μεταβλητές που χρησιμοποιούνται τυποποιούνται ως ακέραιες, πραγματικές, μεταβλητές χαρακτήρων, κλπ. Η Java θέτει αυστηρούς κανόνες σχετικά με τον προσδιορισμό τύπου μιας μεταβλητής. Πιο συγκεκριμένα, απαιτεί ο προσδιορισμός τύπου κάθε μεταβλητής να είναι δεδομένος κατά τον χρόνο μεταγλώττισης και να παραμένει αμετάβλητος για όλη την ζωή της μεταβλητής. Γλώσσες με παρόμοια συμπεριφορά χαρακτηρίζονται ως γλώσσες ισχυρού τύπου (Strong typed languages), σε αντίθεση με τις γλώσσες που δεν θέτουν αυτές τις απαιτήσεις και χαρακτηρίζονται ως γλώσσες αδύναμου τύπου (Weak typed languages).

Ένα επίσης σημαντικό χαρακτηριστικό της Java είναι η πρώιμη διασύνδεση (early binding). Όπως αναφέραμε μια μεταβλητή μπορεί να φιλοξενεί μια ακέραιη τιμή ή έναν χαρακτήρα. Και στις δύο περιπτώσεις όμως τα δεδομένα στην μνήμη αποθηκεύονται ως μια σειρά από δυαδικά ψηφία. Για να είναι δυνατή η ορθή ερμηνεία αυτών των δεδομένων είναι αναγκαίο να γνωρίζουμε τον τύπο τις μεταβλητής. Με άλλα λόγια, όταν προσπελαύνουμε μια θέση μνήμης, δηλ. μια σειρά από bits, πρέπει να είναι γνωστό αν η σειρά αυτή αναπαριστά έναν ακέραιο ή έναν χαρακτήρα. Η Java χαρακτηρίζει κάθε μεταβλητή με συγκεκριμένο τύπο, γνωστό κατά τον χρόνο μεταγλώττισης και σταθερό καθόλη την εκτέλεση του προγράμματος. Έτσι όταν προσπελαύνει μια μεταβλητή γνωρίζοντας την διεύθυνσή της στην μνήμη και τον τύπο της μπορεί άμεσα να την αξιοποιήσει. Στον αντίποδα, οι γλώσσες καθυστερημένης διασύνδεσης (late binding), επιτρέπουν κατά την διάρκεια εκτέλεσης του προγράμματος την εκχώρηση στην ίδια μεταβλητή δεδομένων ποικίλων τύπων. Για παράδειγμα, η Javascript, επιτρέπει την εκχώρηση σε μεταβλητή μιας ακέραιης τιμής και παρακάτω στον ίδιο κώδικα την εκχώρηση μια συμβολοσειράς. Επομένως, ο τύπος της μεταβλητής δεν μπορεί να καθοριστεί παρά μόνο κατά τον χρόνο εκτέλεσης. Έτσι όταν η Javascript προσπελαύνει μια μεταβλητή θα πρέπει να υπολογίσει ποιος είναι

ο τύπος της σε ακριβώς εκείνο το σημείο της εκτέλεσης. Όπως είναι φανερό, η πρώιμη διασύνδεση, βελτιώνει την ταχύτητα εκτέλεσης.

Οι μεταβλητές επομένως στην Java έχουν προκαθορισμένο τύπο. Οι διαθέσιμοι τύποι είναι πάρα πολλοί. Επιπλέον, ο χρήστης έχει την δυνατότητα να ορίσει και δικούς του τύπους (User Defined Types). Σε αυτήν την ενότητα όμως παρουσιάζουμε ειδικά τους θεμελιώδεις τύπους (primitive types) της Java. Οι θεμελιώδεις τύποι είναι οκτώ [1], είναι προκαθορισμένοι (predefined) και τα ονόματά τους αποτελούν δεσμευμένες λέξεις (reserved words). Πλήρη λίστα με τις λέξεις κλειδιά της Java μπορείτε να βρείτε στο παράρτημα 2. Όλοι οι υπόλοιποι τύποι προκύπτουν από συνδυασμό ή επανορισμό των θεμελιωδών τύπων. Στο σχήμα 3.1, παρουσιάζεται η κατηγοριοποίηση των θεμελιωδών τύπων.



***Σχήμα 3.1**: Κατηγοριοποίηση των θεμελιωδών τύπων*

byte Ο τύπος byte είναι προσημασμένος ακέραιος μήκους οκτώ δυαδικών ψηφίων (bits). Αν η τιμή μιας μεταβλητής τύπου byte είναι θετική, τότε αναπαρίσταται στην μνήμη κωδικοποιημένη με την μορφή πρόσημο και μέτρο (sign and magnitude), δηλ. το πρώτο bit έχει την τιμή 0 για να δείξει πως ο αριθμός είναι θετικός και τα υπόλοιπα bits διατηρούν την τιμή του αριθμού εκφρασμένη στο δυαδικό σύστημα. Αν η τιμή μιας μεταβλητής τύπου byte είναι αρνητική, αυτή αναπαρίσταται στην μνήμη με την μορφή συμπληρώματος ως προς 2 (two's complement) (παράρτημα 1). Σε αυτήν την περίπτωση το πρώτο bit έχει την τιμή 1 που δείχνει πως ο αριθμός είναι αρνητικός. Επομένως, τόσο για τους θετικούς όσο και για τους αρνητικούς, εξαιρώντας το πρώτο bit που ονομάζεται και bit πρόσημου, διαπιστώνουμε πως για την αναπαράσταση της τιμής απομένουν 7 bits. Συνεπώς, αναπαρίστανται συνολικά 27 αρνητικές τιμές και 27 μη αρνητικές τιμές. Πιο συγκεκριμένα, αναπαρίστανται οι αρνητικές τιμές -1..-27 και οι μη αρνητικές τιμές 0..27-1. Προσέξτε πως ο η μικρότερη αρνητική τιμή, δηλ. το -27, είναι κατ' απόλυτη τιμή μεγαλύτερη κατά μια μονάδα από την μεγαλύτερη μη αρνητική, δηλ. το 27-1. Αυτό συμβαίνει γιατί στις μη αρνητικές τιμές συμπεριλαμβάνεται και το 0.

Η Java για κάθε θεμελιώδη τύπο διαθέτει και μια κλάση (Class) που παρέχει λειτουργίες σχετικές με τον θεμελιώδη τύπο. Την έννοια της κλάσης θα αναλύσουμε με λεπτομέρεια κατά την μελέτη του αντικειμενοστρεφούς μοντέλου. Για την ώρα, θα θεωρούμε την κλάση ως ένα σύνθετο τύπο ο οποίος εκτός από δεδομένα ενσωματώνει και λειτουργίες (συναρτήσεις). Σε γενικές γραμμές, θεμελιώδεις τύποι και αντίστοιχες κλάσεις έχουν την ίδια ονομασία με μόνη διαφορά ότι το όνομα της κλάσης ξεκινάει με κεφαλαίο ενώ του τύπου με πεζό. Έτσι ο τύπος byte συνοδεύεται από την κλάση Byte.

```
static void byteDemo() {
        System.out.println("Size of byte is " + Byte.SIZE + " bits");
        byte bMin = Byte.MIN_VALUE, bMax = Byte.MAX_VALUE, bLiteral = 123;
        System.out.println("bMin equals to " + bMin);
        System.out.println("bMax equals to " + bMax);
        System.out.println("bLiteral equals to " + bLiteral);
}
```

***Κώδικας 3.1**: Παράδειγμα χρήσης μεταβλητής τύπου byte*

Στον κώδικα 3.1, καταρχάς, εμφανίζουμε το μέγεθος του τύπου (Byte.SIZE) μετρημένο σε δυαδικά ψηφία (bits). Μετά δηλώνουμε τις μεταβλητές bMin, bMax και bLiteral. Την πρώτη αρχικοποιούμε στην ελάχιστη τιμή του τύπου (Byte.MIN_VALUE), την δεύτερη στην μέγιστη τιμή του τύπου (Byte.MAX_VALUE) και την τρίτη σε μια ενδιάμεση τιμή. Την ελάχιστη και μέγιστη τιμή μας παρέχει η κλάση Byte που τις περιλαμβάνει ως σταθερές μεταβλητές (variable constants). Για την ενδιάμεση τιμή χρησιμοποιούμε μια κυριολεκτική σταθερά (literal). Στην συνέχεια, στέλνουμε στην συσκευή εξόδου, δηλ. στην τυπική περίπτωση στην οθόνη, το περιεχόμενο των bMin, bMax και bLiteral.
Παρατίθεται η έξοδος του κώδικα 3.1.

```
Size of byte is 8 bits
bMin equals to -128
bMax equals to 127
bLiteral equals to 123
```

short Ο τύπος short είναι προσημασμένος ακέραιος. Η κωδικοποίησή του είναι ανάλογη με αυτήν του τύπου byte, δηλ. πρόσημο και μέτρο για τους θετικούς και συμπλήρωμα ως προς 2 για τους αρνητικούς. Το μήκος του όμως16 δυαδικών ψηφίων. Επομένως, οι τιμές που δέχονται οι μεταβλητές τύπου short είναι 0..215-1 και -1..-215.
Στον κώδικα 3.2, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου short.

```
static void shortDemo() { //Κώδικας 3.2
        short s = Short.MIN_VALUE, s1 = Short.MAX_VALUE, s2 = 399;
        System.out.println("short s equals to " + s);
        System.out.println("short s1 equals to " + s1);
        System.out.println("short s2 equals to " + s2);
        System.out.println("Size of short is " + Short.SIZE + " bits");
}
```

***Κώδικας 3.2**: Παράδειγμα χρήσης μεταβλητών τύπου short*

Ακολουθεί η έξοδος του κώδικα 3.2

```
short s equals to -32768
short s1 equals to 32767
short s2 equals to 399
Size of short is 16 bits
```

int Ο τύπος int είναι προσημασμένος ακέραιος μήκους 32 δυαδικών ψηφίων. Η κωδικοποίησή του είναι ανάλογη με αυτήν του τύπου byte, δηλ. πρόσημο και μέτρο για τους θετικούς και συμπλήρωμα ως προς 2 για τους αρνητικούς. Επομένως, οι τιμές που δέχονται οι μεταβλητές τύπου int είναι 0..231-1 και -1..-231.
Στον κώδικα 3.3, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου int.

```
    static void intDemo() { //Κώδικας 3.3
        int iMin = Integer.MIN_VALUE,
                iMax = Integer.MAX_VALUE, iLiteral = 43127;
        System.out.println("int iMin equals to " + iMin);
        System.out.println("int iMax equals to " + iMax);
        System.out.println("int iLiteral equals to " + iLiteral);
        System.out.println("Size of int is " + Integer.SIZE + " bits");
    }
```

***Κώδικας 3.3**: Δήλωση και αρχικοποίηση μεταβλητών τύπου int*

Ακολουθεί η έξοδος του κώδικα 3.3.

```
int iMin equals to -2147483648
```

```
int iMax equals to 2147483647
int iLiteral equals to 43127
Size of int is 32 bits
```

Προσέξτε πως κατ' εξαίρεση, το όνομα της κλάσης που μας παρέχει πληροφορίες σχετικά με τον τύπο int, δεν είναι Int αλλά Integer.

long Ο τύπος long είναι προσημασμένος ακέραιος μήκους 64 δυαδικών ψηφίων. Η κωδικοποίησή του είναι ανάλογη με αυτήν του τύπου byte, δηλ. πρόσημο και μέτρο για τους θετικούς και συμπλήρωμα ως προς 2 για τους αρνητικούς. Επομένως, οι τιμές που δέχονται οι μεταβλητές τύπου long είναι 0..263-1 και -1..-263.
Στον κώδικα 3.4, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου long.

```
static void longDemo() { //Κώδικας 3.4
    long lMin = Long.MIN_VALUE,
            lMax = Long.MAX_VALUE, lLiteral = 2_147_483_699L;
    System.out.println("long lMin equals to " + lMin);
    System.out.println("long lMax equals to " + lMax);
    System.out.println("long lLiteral equals to " + lLiteral);
    System.out.println("Size of long is " + Long.SIZE + " bits");
}
```

***Κώδικας 3.4**: Δήλωση και αρχικοποίηση μεταβλητών τύπου long*

Ακολουθεί η έξοδος του κώδικα 3.4.

```
long lMin equals to -9223372036854775808
long lMax equals to 9223372036854775807
long lLiteral equals to 2147483699
Size of long is 64
```

Προσέξτε πως στην αρχικοποίηση της lLiteral, η κυριολεκτική σταθερά ακολουθείται από τον χαρακτήρα L. Οι ακέραιες κυριολεκτικές σταθερές που δεν ακολουθούνται από το L έχουν τύπο int. Αν θέλουμε να χρησιμοποιήσουμε κυριολεκτικές σταθερές με τύπο long, θα πρέπει αυτές να ακολουθούνται από το L. Στον κώδικα 3.4, αν αφαιρέσουμε το L, ο μεταγλωττιστής θα παραγάγει error καθώς η τιμή 2147483699 είναι μεγαλύτερη από την μέγιστη τιμή τύπου int. Ωστόσο σταθερές τύπου int (μέσα στα όρια του τύπου int) μπορούν να εκχωρηθούν σε μεταβλητές τύπου long καθώς η Java κάνει αυτόματες μετατροπές μεταξύ συμβατών τύπων. Εναλλακτικά, αντί για κεφαλαίο L μπορεί να χρησιμοποιηθεί το πεζό l. Ωστόσο, συνίσταται η χρήση του κεφαλαίου καθώς το πεζό διακρίνεται με δυσκολία από τον αριθμό 1.

Προσέξτε επίσης πως ανάμεσα στα ψηφία έχουμε τοποθετήσει χαρακτήρες '_' (underscore). Από την έκδοση 7 και μετά, η Java επιτρέπει οποιοδήποτε αριθμό από underscores σε αριθμητικές σταθερές ώστε να διευκολύνεται η αναγνωσιμότητα των προγραμμάτων. Ωστόσο, δεν επιτρέπεται ο χαρακτήρας underscore, στην αρχή ή στο τέλος ενός αριθμού.

float Ο τύπος float χρησιμοποιείται για διαχείριση πραγματικών αριθμών. Έχει μήκος 32 δυαδικά ψηφία. Η κωδικοποίηση στην μνήμη των τιμών αυτού του τύπου ακολουθεί ένα πρότυπο γνωστό ως IEEE 754 καθώς αναπτύχθηκε από το Ινστιτούτο Ηλεκτρολόγων και Ηλεκτρονικών Μηχανικών (Institute of Electrical and Electronics Engineers) που εδρεύει στην Νέα Υόρκη και στο οποίο συνήθως αναφερόμαστε με την αγγλική συντομογραφία IEEE. Η ανάλυση του προτύπου IEEE 754 είναι έξω από τα όρια αυτού του εγχειριδίου. Ωστόσο, είναι χρήσιμο να γνωρίζει κανείς πως οι πραγματικοί αριθμοί αναπαρίστανται σε εκθετική μορφή (Exponential Notation) και εκφράζονται με επιστημονικό συμβολισμό (scientific notation), mxEn, όπου m ένας πραγματικός αριθμός, n ακέραιος και E=10. Οι float αριθμοί χαρακτηρίζονται ως αριθμοί κινητής υποδιαστολής απλής ακρίβειας (single-precision floating point numbers).
Στον κώδικα 3.5, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου float.

```
static void floatDemo() { //Κώδικας 3.5
        float fMin = Float.MIN_VALUE, fMax = Float.MAX_VALUE, fLit = 3.15f;
        System.out.println("float fMin equals to " + fMin);
```

```
        System.out.println("float fMax equals to " + fMax);
        System.out.println("float fLit equals to " + fLit);
        System.out.println("Size of float is " + Float.SIZE + " bits");
    }
```

***Κώδικας 3.5**: Μεταβλητές τύπου float*

Ακολουθεί η έξοδος του κώδικα 3.5

```
float fMin equals to 1.4E-45
float fMax equals to 3.4028235E38
float fLit equals to 3.15
Size of float is 32 bits
```

Προσέξτε πως η σταθερά 3.15 ακολουθείται από το f, εναλλακτικά μπορεί να χρησιμοποιηθεί το F, ώστε να καθοριστεί ο τύπος της σταθεράς ως float. Χωρίς το f ή F, ο τύπος της σταθεράς είναι ο double που παρουσιάζεται αμέσως παρακάτω. Όπως βλέπουμε, στην έξοδο του κώδικα 3.5, οι τιμές των fMin και fMax έχουν εκφραστεί με τον επιστημονικό συμβολισμό (scientific notation). Ο επιστημονικός συμβολισμός μπορεί να χρησιμοποιηθεί και για να εκφραστούν κυριολεκτικές σταθερές τύπου float ή double μέσα στον πηγαίο κώδικα. Επομένως, η ελάχιστη πραγματική τιμή που μπορεί να αναπαρασταθεί με τον float είναι 1.4E-45, δηλ. 1.4x10-45 και η μέγιστη είναι 3.4028235E38, δηλ. 3.4028235x1038, ο δε χώρος που καταλαμβάνεται στην μνήμη είναι 32 bits.

Οι μεταβλητές τύπου float δύνανται να λάβουν επιπλέον τις ειδικές τιμές Float.NEGATIVE_INFINITY, Float.POSITIVE_INFINITY και Float.NaN (Not a Number). Επιπλέον, σε αντίθεση με τους ακέραιους τύπους, οι πραγματικοί υποστηρίζουν δύο μηδέν, το +0f και το -0f. Αυτό οφείλεται στην κωδικοποίηση IEEE 754 που είναι συμμετρική μεταξύ θετικών και αρνητικών.

Για παράδειγμα, η τιμή Float.NEGATIVE_INFINITY επιστρέφεται όταν ένας αρνητικός αριθμός τύπου float διαιρείται με το +0f ή αντίστοιχα ένας θετικός float διαιρείται με το -0f. Επίσης, Float.NEGATIVE_INFINITY ή Float.POSITIVE_INFINITY μπορεί να επιστραφεί ως αποτέλεσμα έκφρασης στην οποία συμμετέχουν αυτές οι τιμές. Αν όμως έχουμε διαίρεση με Infinity, το αποτέλεσμα είναι NaN. Η τιμή Float.NaN επιστρέφεται και σε διάφορες άλλες περιπτώσεις, όπως για παράδειγμα όταν ζητηθεί η τετραγωνική ρίζα αρνητικού αριθμού.

Η διαίρεση ακεραίου με το 0 παράγει εξαίρεση[5] του τύπου “διαίρεση με το 0” (division by zero). Για παράδειγμα, ο κώδικας που ακολουθεί παράγει εξαίρεση.

```
int i = 1, j = 0;
System.out.println(i / j);
```

Αντίθετα, η διαίρεση πραγματικού με το 0 επιστρέφει Infinity. Για παράδειγμα, ο κώδικας που ακολουθεί τυπώνει Infinity.

```
float f=1, r=0;
System.out.println(f/r);
```

Αντίθετα, ο κώδικας

```
float nAN=(float)Math.sqrt(-1);
System.out.println(nAN);
```

θα τυπώσει NaN.

Σημειώστε πως η συνάρτηση sqrt της κλάσης Math επιστρέφει την τετραγωνική ρίζα του ορίσματός της. Στην συγκεκριμένη κλήση επιστρέφει την τετραγωνική ρίζα του -1. Όπως είναι γνωστό τέτοιος πραγματικός δεν υπάρχει, σαν αποτέλεσμα η sqrt επιστρέφει NaN. Επίσης, ο τύπος της τιμής που επιστρέφει η sqrt είναι

[5] Οι εξαιρέσεις είναι ο μηχανισμός διαχείρισης απροσδόκητων γεγονότων. Παρουσιάζονται αναλυτικά στην ενότητα 16. Για την ώρα θεωρείστε την εξαίρεση συνώνυμη του λάθους χρόνου εκτέλεσης.

double. Επομένως, στην κλήση που συζητάμε θα επιστρέψει NaN τύπου double (Double.NaN). Για τον λόγο αυτόν αναγκαζόμαστε να προχωρήσουμε σε μετατροπή του τύπου από double σε float προκειμένου να εκχωρήσουμε την επιστρεφόμενη τιμή στην μεταβλητή nAN τύπου float. Όπως φαίνεται στον κώδικα, για να μετατρέψουμε το αποτέλεσμα της sqrt τοποθετήσαμε πριν την κλήση της, το όνομα του νέου τύπου μέσα σε παρενθέσεις. Θα πρέπει να έχουμε υπόψη μας πως μια τέτοια μετατροπή γίνεται με ευθύνη του προγραμματιστή και μπορεί να οδηγήσει σε απώλεια ακρίβειας.

Η κωδικοποίηση IEEE 754 μας δίνει την δυνατότητα στο ίδιο μήκος μνήμης να αναπαραστήσουμε μεγαλύτερους αριθμούς από ότι η κωδικοποίηση συμπληρώματος ως προς 2 που χρησιμοποιείται για την κωδικοποίηση των ακεραίων. Ωστόσο, είναι σημαντικό να κατανοήσουμε ότι οι πραγματικοί αριθμοί αποθηκεύονται στην μνήμη προσεγγιστικά. Σαν αποτέλεσμα συγκρίσεις πραγματικών με χρήση του τελεστή ισότητας (equality operator), ==, ή των υπόλοιπων σχεσιακών τελεστών (>, <, >=, <= και !=, ενότητα 4.1.4) δεν είναι αξιόπιστες και πρέπει να αποφεύγονται.
Για παράδειγμα, ο κώδικας

```
float f=0.1f+0.000000001f;
System.out.println(f==0.1f);
```

θα τυπώσει true.
Προσέξτε τον ακόλουθο κώδικα

```
float d1 = 1, d2=1;
for (int j = 1; j <= 10; j++) {
     d1 = d1+0.00001f;
}
d2=d2+0.00001f * (10);
System.out.println(d1==d2);
```

Οι μεταβλητές d1 και d2 αρχικοποιούνται και οι δύο στην τιμή 1. Σε κάθε βήμα της επαναληπτικής διαδικασίας for, προστίθεται η τιμή 0.00001f στην d1. Έχουμε ένα σύνολο 10 επαναλήψεων, άρα στην d1 προστίθεται 10 φορές το 0.00001. Στην συνέχεια, προσθέτουμε στην d2, το 10*0.00001. Αναμένουμε επομένως ότι τα d1 και d2 είναι ίσα μεταξύ τους. Ωστόσο, ο κώδικας θα τυπώσει false καθώς η τιμή της d1 έχει υπολογιστεί ως 1.0001001 και της d2 ως 1.0001. Για αυτόν τον λόγο, ανάλογα με τις ανάγκες τις εφαρμογής μας, η σύγκριση μεταξύ πραγματικών αριθμών μπορεί να γίνεται προσεγγιστικά.

```
static boolean approximateEquals(float f1, float f2, float
epsilon) {
     return Math.abs(f2 - f1) < epsilon;
}
```

Χρησιμοποιήστε την συνάρτηση approximateEquals για να συγκρίνετε αριθμούς float. Για να δείτε το αποτέλεσμά της, καλέστε την ως εξής:

```
System.out.println(approximateEquals(d1, d2, 0.0001));
```

Η τρίτη παράμετρος, συνιστά μια οριακή τιμή. Αν η απόλυτη τιμή της διαφοράς των δύο float είναι μικρότερη από αυτήν την οριακή τιμή, οι αριθμοί θεωρούνται ίσοι. Αν για παράδειγμα, θέλουμε να συγκρίνουμε τα μήκη δύο οδών και το μήκος της πρώτης το μετρήσουμε στα 2 km και της δεύτερης στα 2 km και 1 χιλιοστό του εκατοστού, μπορεί να τις θεωρήσουμε ισομήκεις.

Γενικότερα, ο τύπος float θα πρέπει να αποφεύγεται για διαχείριση τιμών που απαιτούν ακρίβεια. Για τέτοιου τύπου πληροφορίες συνίσταται η χρήση της κλάσης Java.math.BigDecimal. Το ίδιο ισχύει και για τον τύπο double που παρουσιάζεται αμέσως παρακάτω.

double Πρόκειται για τύπο αποθήκευσης πραγματικών αριθμών, επίσης κωδικοποιημένο με το IEEE 754. Η διαφορά του από τον float είναι πως ο double έχει μήκος 64 bits. Πρόκειται για τύπο κινητής υποδιαστολής διπλής ακρίβειας (double-precision 64-bit IEEE 754 floating point). Εξαιτίας του μεγαλύτερου μεγέθους του

μπορεί να φιλοξενήσει πραγματικές τιμές με μεγαλύτερη ακρίβεια από τον float. Στην τυπική περίπτωση είναι ο τύπος που θα πρέπει να χρησιμοποιούμε όταν θέλουμε να διαχειριστούμε πραγματικούς αριθμούς. Ο float θα πρέπει να χρησιμοποιείται μόνο στις περιπτώσεις που έχουμε πολλά δεδομένα από πραγματικούς αριθμούς, χρειάζεται να κάνουμε οικονομία στην μνήμη και είμαστε σίγουροι πως η ακρίβεια που προσφέρει, μας αρκεί. Ο τύπος double διαθέτει επίσης τιμές Double.PositiveInfinity, Double.NegativeInfinity, Double.NaN, αρνητικό και θετικό μηδέν με παρόμοια χαρακτηριστικά με τις αντίστοιχες τιμές του float.
Στον κώδικα 3.6, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου double.

```
    static void doubleDemo() { //Κώδικας 3.6
        double dMin = Double.MIN_VALUE, dMax = Double.MAX_VALUE, dLit =
3.4028235E38;
        System.out.println("double dMin equals to " + dMin);
        System.out.println("double dMax equals to " + dMax);
        System.out.println("double dLit equals to " + dLit);
        System.out.println("Size of double is " + Double.SIZE + "bits");
    }
```

***Κώδικας 3.6**: Δήλωση και αρχικοποίηση μεταβλητών τύπου double*

Ακολουθεί η έξοδος του κώδικα 3.6.

```
double dMin equals to 4.9E-324
double dMax equals to 1.7976931348623157E308
double dLit equals to 3.4028235E38
Size of double is 64bits
```

Επομένως, η ελάχιστη τιμή του double είναι 4.9x10-324, η μέγιστη 1.7976931348623157x10308 και το μέγεθός του 64 bits. Παρατηρήστε πως στην αρχικοποίηση της dLit έχει χρησιμοποιηθεί επιστημονικός συμβολισμός. Οι σταθερές με δεκαδικά που δεν έχουν χαρακτηριστεί ως float, θεωρούνται εξ' ορισμού πως είναι τύπου double. Μπορεί να συνοδευτεί μια τέτοια σταθερά από τον χαρακτήρα D ή d προκειμένου να δηλωθεί ο τύπος της σαφώς, ωστόσο αυτό αποφεύγεται από σύμβαση.

Παρότι ο double προσφέρει μεγαλύτερη ακρίβεια από τον float δεν παύει να αντιπροσωπεύει προσεγγιστικά τους πραγματικούς αριθμούς και η χρήση του θα πρέπει να αποφεύγεται όπου μας ενδιαφέρει η ακρίβεια και στην θέση του να χρησιμοποιείται η κλάση Java.math.BigDecimal. Για προσεγγιστική ισότητα χρησιμοποιείστε την συνάρτηση

```
static boolean approximateEquals(double d1, double d2, double
epsilon) {
     return Math.abs(d2 - d1) < epsilon;
}
```

Ίσως να προσέξατε πως οι συναρτήσεις MIN_VALUE επιστρέφουν αρνητικές τιμές για τους ακέραιους τύπους και θετικές τιμές για τους float και double. Πράγματι, για τους πραγματικούς οι συναρτήσεις αυτές επιστρέφουν την πλησιέστερη τιμή στο 0. Αν χρειάζεστε την ελάχιστη αρνητική τιμή ενός πραγματικού τύπου χρησιμοποιείστε –Float.MAX_VALUE και –Double.MAX_Value, αντίστοιχα.

boolean Οι μεταβλητές του τύπου boolean λαμβάνουν αυστηρά 2 δυνατές τιμές, true ή false. Ένα δυαδικό ψηφίο αρκεί για την αποθήκευση των τιμών αυτού του τύπου. Ωστόσο, οι προδιαγραφές της Java δεν καθορίζουν το μέγεθος του τύπου. Επομένως, το μέγεθος του τύπου boolean εξαρτάται από την συγκεκριμένη υλοποίηση. Σε κάποιες γλώσσες είναι δυνατή η μετατροπή μεταξύ boolean και ακέραιων τύπων, στην Java δεν είναι επιτρεπτή τέτοια μετατροπή τύπου [1].
Στον κώδικα 3.7, παρουσιάζουμε παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου boolean.

```
static void booleans() { //Κώδικας 3.7
    boolean b1 = true, b2 = false;
    System.out.println("b1 equals to " + b1);
    System.out.println("b2 equals to " + b2);
```

```
    }
```

***Κώδικας 3.7**: Δήλωση και αρχικοποίηση boolean μεταβλητών*

Ακολουθεί η έξοδος του κώδικα 3.7.

```
b1 equals to true
b2 equals to false
```

char Ο τύπος char χρησιμοποιείται για την αποθήκευση χαρακτήρων. Υποστηρίζει το σύνολο Unicode, δηλ. τους χαρακτήρες από τις περισσότερες γλώσσες συν μια ποικιλία χαρακτήρων που δεν συνιστούν γράμματα αλφάβητου και διάφορους χαρακτήρες ελέγχου (control characters) που είναι συνήθως μη εκτυπώσιμοι, όπως για παράδειγμα, ο χαρακτήρας που σηματοδοτεί το τέλος της τρέχουσας γραμμής (End Of Line). Στον κώδικα 3.8, δίνουμε παράδειγμα χρήσης μεταβλητών του τύπου char.

```
1 static void charDemo() { //Κώδικας 3.8
2         char cMin = Character.MIN_VALUE, cMax = Character.MAX_VALUE, cLit = 'A',
c = '\u0043';
3         System.out.println(cMin + " " + (int) cMin);
4         System.out.println(cMax + " " + (int) cMax);
5         System.out.println(c + " " + (int) c);
6         System.out.println(cLit + " " + (int) cLit);
7         System.out.println((char) (cLit + 1));
8     }
```

***Κώδικας 3.8**: Παράδειγμα χρήσης του τύπου char*

Ο κώδικας 3.8, έχει την ακόλουθη έξοδο

```
 0
□ 65535
C 67
A 65
B
```

Προσέξτε πως η κλάση που αντιστοιχεί στον τύπο char ονομάζεται Character. Σε κάθε μια από τις γραμμές 2-5, τυπώνουμε τον χαρακτήρα και στην συνέχεια μια τιμή που προκύπτει από την μετατροπή του χαρακτήρα σε int. Η μετατροπή αυτή είναι ορθή γιατί οι χαρακτήρες κωδικοποιούνται στην μνήμη σαν 16-bit μη προσημασμένοι ακέραιοι. Επομένως, ο char είναι και αυτός ένας ακέραιος τύπος. Πιο συγκεκριμένα, μια μεταβλητή τύπου char κρατάει την ακέραιη τιμή που συνιστά τον κωδικό του χαρακτήρα στο Unicode. Η έξοδος της γραμμής 2 φαίνεται σαν ένα διάστημα ακολουθούμενο από το 0. Αυτό συμβαίνει γιατί ο cMin είναι ο NULL χαρακτήρας ο οποίος όταν στέλνεται σε συσκευή εξόδου δεν κάνει τίποτα. Επομένως τυπώνεται το διάστημα που στέλνουμε στην γραμμή 2 και μετά ο κωδικός Unicode του NULL, δηλ. το 0. Η γραμμή 3 τυπώνει ένα τετραγωνάκι ακολουθούμενο από τον κωδικό 65535 που στο δεκαεξαδικό σύστημα είναι το FFFF. Σημειώστε πως οι θέσεις του Unicode από FFF0 έως και FFFF χρησιμοποιούνται για ειδικές περιπτώσεις [3] που δεν ενδιαφέρουν στα πλαίσια αυτού του εγχειριδίου.

Προσέξτε πως η μεταβλητή c έχει αρχικοποιηθεί με την σταθερά ‘\u0043’. Είναι εναλλακτικός τρόπος για έκφραση σταθερών τύπου char. Σύμφωνα με τον αυτόν τον τρόπο, ένας χαρακτήρας μπορεί να αναπαρασταθεί με \u ακολουθούμενο από τον δεκαεξαδικό κωδικό του. Θα δούμε περισσότερα σε σχέση με αυτό, αμέσως παρακάτω, στους χαρακτήρες διαφυγής. Η γραμμή 5 τυπώνει τον χαρακτήρα ‘A’ με τον κωδικό του. Στην γραμμή 6, προσθέτουμε το cLit που περιέχει το A με το 1. Η πρόσθεση αυτή είναι δυνατή γιατί όπως εξηγήσαμε, στην μνήμη ένας char διατηρεί μια ακέραιη τιμή που στην συγκεκριμένη περίπτωση είναι η τιμή 65. Επομένως, η τιμή της έκφραση cLit+1 είναι ίση με 66. Όταν το 66 μετατρέπεται σε char, στην ίδια γραμμή, τότε το αποτέλεσμα είναι ο χαρακτήρας B.

3.3 Ακολουθίες ελέγχου (Control Sequences)

Ο χαρακτήρας \ (backslash) ακολουθούμενος από έναν ειδικό χαρακτήρα ονομάζεται ακολουθία ελέγχου (control sequence) ή χαρακτήρας διαφυγής (escape character).
Στον πίνακα 3.1 παρουσιάζονται οι ακολουθίες ελέγχου της Java.

\t	tab	Αφήνει ένα διάστημα
\b	backspace	Μεταφέρει την επόμενη έξοδο ένα χαρακτήρα πίσω
\n	newline	Μεταφέρει την επόμενη έξοδο στην επόμενη γραμμή
\r	carriage return	Μεταφέρει την επόμενη έξοδο στην αρχή της τρέχουσας γραμμής
\f	form feed	Αλλαγή σελίδας στον εκτυπωτή
\'	single quote	Ο ειδικός χαρακτήρας ', λαμβάνεται ως κανονικός
\"	double quote	Ο ειδικός χαρακτήρας ", λαμβάνεται ως κανονικός
\\	backslash	Ο ειδικός χαρακτήρας \, λαμβάνεται ως κανονικός
\u	unicode character	Ακολουθούμενο από αριθμητική σταθερά αναπαριστά τον αντίστοιχο Unicode χαρακτήρα

Πίνακας 3.1 *Ακολουθίες Ελέγχου*

Ακολουθεί παράδειγμα χρήσης των ακολουθιών ελέγχου

```
9      static void controlSequences() { //Κώδικας 3.9
10          System.out.println("slash_t" + '\t' + "second ");
11          System.out.println("slash_b" + '\b' + "second ");
12          System.out.println("slash_n" + '\n' + "second ");
13          System.out.println("slash_r" + '\r' + "second ");
14          System.out.println("slash_singlequote" + '\'' + "second ");
15          System.out.println("slash_doublequote" + '\"' + "second ");
16          System.out.println("slash_backslash" + '\\' + "second ");
17          System.out.println("slash_unicode" + '\u0041' + "second ");
18      }
```

Κώδικας 3.9*: Παραδείγματα χρήσης ακολουθιών ελέγχου*

Η έξοδος του κώδικα έχει ως εξής:

```
slash_t         second
slash_second
slash_n
second
second
slash_singlequote'second
slash_doublequote"second
slash_backslash\second
slash_unicodeAsecond
```

Η γραμμή 10 του κώδικα 3.9, τυπώνει “slash_t”, μετά αφήνει ένα tab και τέλος τυπώνει την σειρά “second”.
Στην γραμμή 11, ο χαρακτήρας ‘\b’ ΄φέρνει την σειρά “second” έναν χαρακτήρα πιο πίσω με αποτέλεσμα την εξαφάνιση του b από την “slash_b”.
Στην γραμμή 12, ο χαρακτήρας newline έχει ως αποτέλεσμα να εκτυπωθεί η “slash_n” και η “second” να εκτυπωθεί στην επόμενη γραμμή.
Στην γραμμή 13, το carriage return φέρνει την έξοδο στην αρχή της γραμμής με αποτέλεσμα να εξαφανίζεται η σειρά “slash_r”.
Η γραμμή 14 τυπώνει ένα quote ανάμεσα στην “slash_singlequote” και στην “second”.
Η γραμμή 15 τυπώνει ένα double quote ανάμεσα στην “slash_doublequote” και στην “second”.
Η γραμμή 16 τυπώνει ένα backslash ανάμεσα στην “slash_backslash” και στην “second”.
Η γραμμή 17 τυπώνει τον χαρακτήρα ‘A’ ανάμεσα στην “slash_unicode” και στην “second”.

3.4 Αναγνωριστικά

Είναι προφανές από την μέχρι τώρα παρουσίαση πως κάθε τοπική μεταβλητή λαμβάνει ένα όνομα που αποτελεί το αναγνωριστικό (identifier) της. Η σύνθεση των αναγνωριστικών υπακούει σε κανόνες. Πιο συγκεκριμένα, ο πρώτος χαρακτήρας υποχρεωτικά πρέπει να είναι ένας χαρακτήρας του Αγγλικού αλφάβητου, ο χαρακτήρας $ (dollar) ή ο χαρακτήρας _ (underscore). Στην συνέχεια, επιπλέον των προαναφερόμενων χαρακτήρων μπορούν να χρησιμοποιηθούν και ψηφία.
Στον πίνακα 3.2, δίνονται παραδείγματα από έγκυρα και μη έγκυρα αναγνωριστικά

Έγκυρα Αναγνωριστικά	**Μη έγκυρα**
rVal	!color
costPerUnit	1start
event23	*profit
net_weight	start-up
$score	last/digit

Πίνακας 3.2 *Παραδείγματα έγκυρων και άκυρων αναγνωριστικών*

Εκτός από αυτούς τους κανόνες που είναι υποχρεωτικοί καθώς επιβάλλονται από τον μεταγλωττιστή, υπάρχει και μια σειρά από συμβάσεις που διέπουν την ονοματολογία των αναγνωριστικών των μεταβλητών. Οι συμβάσεις δεν επιβάλλονται από τον μεταγλωττιστή, έχουν ωστόσο καθολική αποδοχή και θα πρέπει να ακολουθούνται συστηματικά.

Από σύμβαση, τα αναγνωριστικά των μεταβλητών πρέπει να αρχίζουν με πεζό χαρακτήρα του Αγγλικού αλφάβητου. Επομένως, οι χαρακτήρες dollar και underscore πρέπει να αποφεύγονται από την πρώτη θέση. Ειδικότερα ο χαρακτήρας dollar πρέπει να αποφεύγεται και από τις υπόλοιπες θέσεις. Θα πρέπει να χρησιμοποιούνται πλήρεις λέξεις ώστε ο κώδικας να είναι αναγνώσιμος και αυτοτεκμηριωμένος (self-documented). Για παράδειγμα, το αναγνωριστικό totalCost είναι πολύ σαφέστερο σε σχέση με την σύντμηση tC. Η Java διαφοροποιεί μεταξύ πεζών και κεφαλαίων (case sensitive). Το αναγνωριστικό totalCost είναι διαφορετικό από το αναγνωριστικό totalcost. Αν ένα αναγνωριστικό μεταβλητής αποτελείται από μια λέξη, τότε όλα τα γράμματα στην ονομασία του πρέπει να είναι πεζοί χαρακτήρες. Αν όμως αποτελείται από περισσότερες από μία λέξεις, όλες οι επόμενες πρέπει να αρχίζουν από κεφαλαίο, π.χ. counter, customerWeight, first, firstOccurance.

Οι κανόνες και οι συμβάσεις για τα αναγνωριστικά των μεταβλητών ισχύουν και για τις παραμέτρους. Αντίθετα, όλοι οι χαρακτήρες στα αναγνωριστικά των σταθερών μεταβλητών πρέπει να είναι κεφαλαίοι.

3.5 Υπερχείλιση

Κατά την επιλογή ενός αριθμητικού τύπου, θα πρέπει να είμαστε σίγουροι πως η χωρητικότητά του επαρκεί για τα δεδομένα που θέλουμε να διαχειριστούμε. Σε περίπτωση που επιχειρήσουμε να εκχωρήσουμε τιμή έξω από τα όρια του τύπου, το αποτέλεσμα θα είναι να υπερχειλίσει (overflow) η μνήμη της μεταβλητής.
Στον κώδικα 3.10, παρουσιάζουμε ένα παραδείγματα υπερχείλισης

```
static void overflow() { //Κώδικας 3.10
    int i = Integer.MAX_VALUE;
    i = i + 1;
    System.out.println(i);
    System.out.println(Integer.MIN_VALUE - 1);
}
```

Κώδικας 3.10*: Υπερχείλιση ακεραίων*

Η έξοδος της overflow είναι

```
-2147483648
2147483647
```

Η έξοδος αυτή είναι το αποτέλεσμα της υπερχείλισης. Καθώς τα αθροίσματα υπολογίζονται κατά τον χρόνο εκτέλεσης, ο μεταγλωττιστής δεν είναι σε θέση να μας προειδοποιήσει. Σαν αποτέλεσμα έχουμε την εισαγωγή ενός λάθους χρόνου εκτέλεσης (run-time error) στο πρόγραμμά μας. Τα λάθη χρόνου εκτέλεσης είναι πολύ πιο σοβαρά από τα λάθη μεταγλώττισης ακριβώς γιατί δεν εντοπίζονται κατά την μεταγλώττιση και ταξιδεύουν ως τον χρήστη του λογισμικού μας όπου μπορεί να προξενήσουν σημαντική ζημιά. Φανταστείτε τις πιθανές επιπτώσεις ενός τέτοιου λάθους σε ένα λογισμικό πλοήγησης αεροσκάφους.

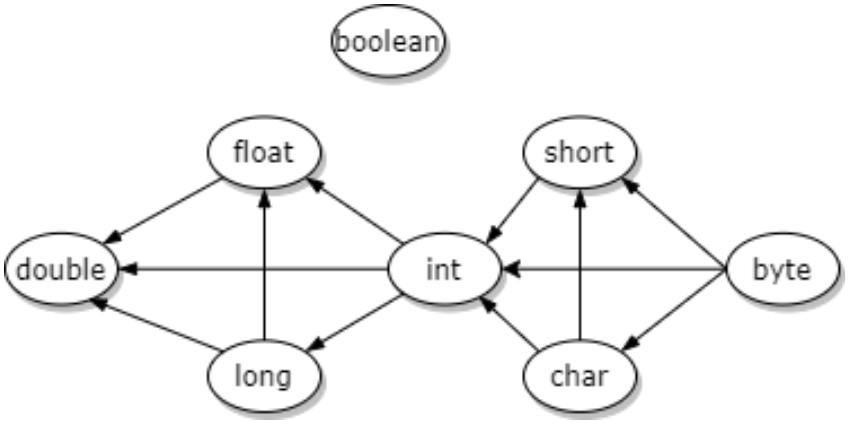
Αν παρόλα αυτά θέλετε να προσθέσετε δύο int και δεν είστε βέβαιοι ότι το άθροισμά τους είναι στα όρια του τύπου, μπορείτε να κάνετε μετατροπή τύπου τουλάχιστον στον ένα από τους δύο και στην συνέχεια να εκχωρήσετε το αποτέλεσμα σε έναν long. Στον κώδικα 3.11, δίνεται σχετικό παράδειγμα.

```
static void notOverflow() { //Κώδικας 3.11
    int i = Integer.MAX_VALUE;
    long k = (long) i + 1;
    System.out.println(k);
}
```

***Κώδικας 3.11**: Άθροισμα ακεραίων εκτός ορίων του τύπου*

Η μετατροπή του τύπου του ενός ορίσματος σε long, έχει ως αποτέλεσμα, ο τύπος του αθροίσματος να είναι long. Επομένως, αν επιχειρήσουμε να εκχωρήσουμε το αποτέλεσμα αυτής της έκφρασης σε int, ο μεταγλωττιστής θα μας αποτρέψει. Αν όμως, όπως στο παράδειγμα, εκχωρήσουμε το άθροισμα σε μεταβλητή τύπου long, τότε θα λάβουμε το σωστό αποτέλεσμα, δηλ. στο παράδειγμα τον αριθμό 2147483648.

Στην Java, ο τύπος των εκφράσεων στις οποίες συμμετέχουν οι αριθμητικοί τύποι byte και short είναι int. Επομένως, όταν προσθέτουμε 2 μεταβλητές τύπου short ή δύο τύπου byte ή μια short με μια byte, δεν χρειάζεται να μετατρέψουμε κανέναν τύπο καθώς το αποτέλεσμα είναι int. Αυτό σημαίνει πως το αποτέλεσμα δεν μπορεί παρά να εκχωρηθεί σε μεταβλητή συμβατού τύπου, μεγαλύτερης ή ίσης χωρητικότητας από τον int. Μπορούμε ωστόσο να εκχωρήσουμε μια μεταβλητή ενός τύπου σε μεταβλητή άλλου τύπου με την προϋπόθεση πως οι τύποι είναι συμβατοί μεταξύ τους και ο τύπος της μεταβλητής που τοποθετείται δεξιά από τον τελεστή εκχώρησης, =, έχει μικρότερη χωρητικότητα από τον τύπο της μεταβλητής που βρίσκεται αριστερά. Σε αυτές τις περιπτώσεις, η Java προχωράει σε αυτόματη μετατροπή τύπου (automatic type conversion). Στο σχήμα 2, απεικονίζονται οι αυτόματες μετατροπές μεταξύ των θεμελιωδών τύπων.



***Σχήμα 3.2**: Αυτόματες μετατροπές μεταξύ θεμελιωδών τύπων*

Όπως φαίνεται στο σχήμα 3.2, σε μια μεταβλητή τύπου double μπορεί να εκχωρηθεί τιμή οποιουδήποτε αριθμητικού τύπου, σε float οποιουδήποτε πλην του double, σε long οποιουδήποτε εκτός από double και float, σε int μπορούν να εκχωρηθούν short, char και byte και φυσικά int, σε short μπορούν να εκχωρηθούν char, byte και short, σε char μπορούν να εκχωρηθούν byte και char και σε byte μόνο byte. Ο τύπος boolean δεν εκχωρείται παρά μόνο σε boolean και κανένας άλλος τύπος δεν μετατρέπεται σε boolean. Περισσότερες πληροφορίες για τις μετατροπές τύπων μπορείτε να βρείτε στις προδιαγραφές της Java [4], §5.1.2, 5.1.3, 5.1.4.

Η υπερχείλιση ισχύει και για τους πραγματικούς τύπους, float και double. Ωστόσο, η συμπεριφορά εδώ είναι διαφορετική. Για παράδειγμα, η έκφραση Double.MAX_VALUE+1d επιστρέφει Double.MAX_VALUE. Μάλιστα, το ίδιο αποτέλεσμα έχουμε αν προσθέσουμε στον Double.MAX_VALUE και μεγαλύτερες τιμές. Μόνο αν προσθέσουμε τιμή μεγαλύτερη ή ίση του 2970, το αποτέλεσμα της έκφρασης θα γίνει Infinity. Επίσης,

η έκφραση Double.MIN_VALUE-1 επιστρέφει -1. Ο Double.MIN_VALUE είναι ένας πάρα πολύ μικρός αριθμός, τόσο μικρός ώστε μπορούμε να θεωρήσουμε ότι προσεγγίζει το μηδέν. Επομένως, αν από το 0 αφαιρέσουμε το 1 θα λάβουμε -1. Αντίστοιχα, αν διαιρέσουμε το Double.MIN_VALUE με το 2 θα λάβουμε ως αποτέλεσμα 0. Η περίεργη εκ πρώτης όψεως συμπεριφορά των πραγματικών μεταβλητών οφείλεται στον τρόπο αναπαράστασής τους. Προσέξτε πως μεταξύ δύο διαδοχικών πραγματικών αριθμών, x1 και x2, που μπορούν να αναπαρασταθούν με την κωδικοποίηση της IEEE, μεσολαβεί άπειρο πλήθος πραγματικών τιμών. Όλες αυτές οι τιμές κωδικοποιούνται σαν x1 ή σαν x2. Ο προσεγγιστικός χαρακτήρας των πραγματικών αριθμών δεν οφείλεται ειδικά στην κωδικοποίηση IEEE. Ένα διαφορετικό σύστημα κωδικοποίησης θα μπορούσε να παρέχει μεγαλύτερη ακρίβεια αλλά ο προσεγγιστικός χαρακτήρας παραμένει. Εξάλλου μεταξύ 2 οποιονδήποτε πραγματικών αριθμών, μεσολαβεί πάντα άπειρο πλήθος πραγματικών τιμών.

3.6 Αριθμητικές σταθερές

Για την έκφραση των κυριολεκτικών αριθμητικών σταθερών μπορούν να χρησιμοποιηθούν, πέραν του δεκαδικού συστήματος και το δυαδικό, το οκταδικό και δεκαεξαδικό σύστημα αρίθμησης. Όταν μια αριθμητική σταθερά αρχίζει από 0b και ακολουθείται από δυαδικά ψηφία, τότε ερμηνεύεται στο δυαδικό σύστημα. Ο κώδικας

```
int i = 0b10, j = 0b11;
System.out.println(i + j);
```

έχει ως αποτέλεσμα να εκτυπωθεί 5, καθώς προσθέτει το 10 με βάση το 2 που είναι ίσο με το 2 στο δεκαδικό σύστημα, με το 11 του δυαδικού που είναι ίσο με το 3 του δεκαδικού.
Όταν μια αριθμητική σταθερά αρχίζει από 0 και ακολουθείται από τα ψηφία του οκταδικού συστήματος ερμηνεύεται σαν οκταδικός αριθμός. Ο κώδικας

```
int i=010, j=011;
System.out.println(i+j);
```

έχει ως αποτέλεσμα να εκτυπωθεί 17, καθώς προσθέτει το 10 με βάση το 8 που είναι ίσο με το 8 στο δεκαδικό σύστημα με το 11 του οκταδικού που είναι ίσο με το 9 του δεκαδικού.
Όταν μια αριθμητική σταθερά αρχίζει από 0x και ακολουθείται από τα ψηφία του δεκαεξαδικού συστήματος ερμηνεύεται σαν δεκαεξαδικός αριθμός. Ο κώδικας

```
int i=0x10, j=0x11;
System.out.println(i+j);
```

έχει ως αποτέλεσμα να εκτυπωθεί 33, καθώς προσθέτει το 10 με βάση το 16 που είναι ίσο με το 16 στο δεκαδικό σύστημα, με το 11 του δεκαεξαδικού που είναι ίσο με το 17 του δεκαδικού.

3.7 Αλφαριθμητικές Σειρές

Σε πολλές περιπτώσεις χρειάζεται να διαχειριστούμε δεδομένα όπως επωνυμίες και μηνύματα, δηλ. σειρές από αλφαριθμητικούς χαρακτήρες. Για αυτές τις περιπτώσεις, η Java παρέχει τους τύπους String, StringBuffer και StringBuilder.

Το String δεν είναι θεμελιώδης τύπος στην Java, λόγω όμως της εκτεταμένης χρήσης του και της ανάγκης να αξιοποιείται σε παραδείγματα από τα πρώτα βήματα στην μελέτη της γλώσσας, παρουσιάζονται εδώ κάποιες αναγκαίες πληροφορίες. Ήδη στο πρώτο μας πρόγραμμα στην ενότητα 3, χρησιμοποιήσαμε σταθερές τύπου String. Ο κώδικας

```
System.out.println("Hello World!");
```

χρησιμοποιεί την αλφαριθμητική σταθερά "Hello World!" για να τυπώσει το κατάλληλο μήνυμα στην οθόνη.

Οι σταθερές τύπου String εσωκλείονται σε διπλά εισαγωγικά (double quotes), σε αντίθεση με τις σταθερές τύπου char που εσωκλείονται σε μονά εισαγωγικά. Μια πράξη που γίνεται πολύ συχνά ανάμεσα σε String είναι η σύνδεση των String (concatenation). Η σύνδεση των String γίνεται με την βοήθεια του τελεστή +.

```
String s1 = "Hello";
String s2 = "World";
System.out.println(s1 + " " + s2);
```

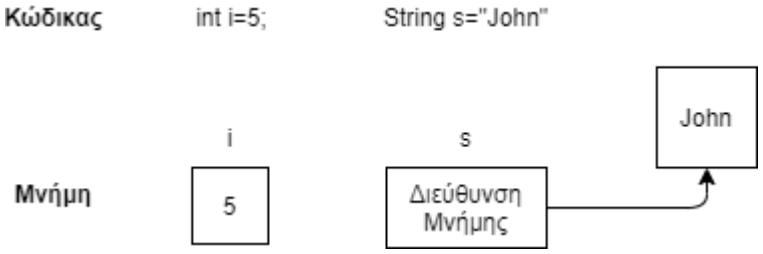
Κώδικας 3.11*: Παράδειγμα μεταβλητών τύπου String*

Στον κώδικα 3.11, στην πρώτη γραμμή, δηλώνουμε και αρχικοποιούμε την μεταβλητή s1 τύπου String, στην δεύτερη γραμμή, δηλώνουμε και αρχικοποιούμε την μεταβλητή s2. Τέλος, στην τρίτη γραμμή, συνδέουμε το s1 με μια σταθερά τύπου String που αποτελείται μόνο από τον χαρακτήρα διάστημα (space) και το αποτέλεσμα αυτής της σύνδεσης το συνδέουμε με το String s2. Σαν αποτέλεσμα, τυπώνεται Hello World.

Προσέξτε πως η ονομασία του τύπου String ξεκινά με κεφαλαίο χαρακτήρα. Αυτό οφείλεται στο ότι ο τύπος δεν είναι θεμελιώδης αλλά συνιστά μια κλάση και όπως έχουμε επισημάνει, από σύμβαση τα ονόματα των κλάσεων ξεκινούν με κεφαλαίο.

3.8 Μεταβλητές τιμής και αναφοράς

Μια άλλη πολύ ουσιαστική διαφορά μεταξύ String και θεμελιωδών τύπων είναι πως οι μεταβλητές τύπου String είναι μεταβλητές αναφοράς (reference variables) σε αντίθεση με τις μεταβλητές των θεμελιωδών τύπων που είναι μεταβλητές τιμής (value variables). Όταν σε μία μεταβλητή π.χ. τύπου int εκχωρούμε μια τιμή, τότε στην μνήμη της μεταβλητής εκχωρείται αυτή καθαυτή η τιμή. Αντίθετα, όταν σε μεταβλητή τύπου String εκχωρούμε μια τιμή, τότε δεσμεύεται χώρος στην μνήμη δυναμικά, στον χώρο αυτόν αποθηκεύεται η τιμή ενώ στην μεταβλητή αποθηκεύεται η διεύθυνση της τιμής, όπως δείχνει το σχήμα 3.3.



Σχήμα 3.3 *Μεταβλητές τιμής και αναφοράς*

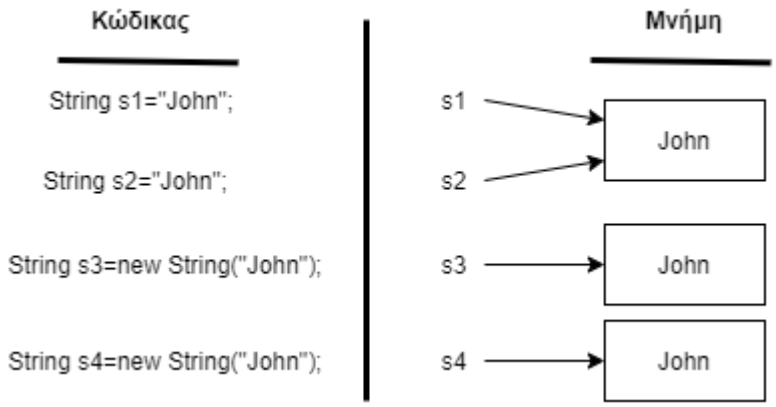
Στην τυπική περίπτωση, οι μεταβλητές αναφοράς δημιουργούνται με την χρήση της λέξης κλειδί new. Για παράδειγμα, η δημιουργία ενός String γίνεται ως εξής:

```
String s=new String("John");
```

Ειδικά για τον τύπο String και για τις κλάσεις των θεμελιωδών τύπων παρέχεται η δυνατότητα δημιουργίας μεταβλητών χωρίς χρήση της λέξης new. Για παράδειγμα,

```
String s="John";
Integer i=125;
```

Όμως οι δύο τρόποι έχουν ουσιαστική διαφορά. Κάθε φορά που χρησιμοποιείται η new για δημιουργία μιας μεταβλητής δεσμεύεται νέος χώρος για την καταχώριση της τιμής της μεταβλητής. Όταν όμως η δημιουργία των μεταβλητών γίνεται χωρίς το new τότε η ίδια τιμή αποθηκεύεται στην ίδια διεύθυνση μνήμης.



Σχήμα 3.4 *Δημιουργία μεταβλητών με την new*

Όπως δείχνει το σχήμα 3.4, οι μεταβλητές s1 και s2 δημιουργήθηκαν χωρίς χρήση της λέξης κλειδί new. Εφόσον έχουν την ίδια τιμή, κατανέμεται ο ίδιος χώρος στην μνήμη. Αντίθετα, οι μεταβλητές s3 και s4 δημιουργήθηκαν με χρήση της new. Σε αυτήν την περίπτωση κατανέμεται διαφορετικός χώρος στην μνήμη ακόμη και αν τα περιεχόμενα των μεταβλητών είναι ίδια. Επομένως, η s1 και η s2 περιέχουν την ίδια διεύθυνση μνήμης και άρα ισχύει s1==s2, δηλ. οι μεταβλητές συγκρινόμενες με τον τελεστή ισότητας (ενότητα 4.1.4.1) θεωρούνται ίσες. Οι μεταβλητές s3 και s4 αναφέρονται στο ίδιο περιεχόμενο που είναι κατανεμημένο όμως σε διαφορετικές θέσεις στην μνήμη. Επομένως, στην s3 αποθηκεύεται διαφορετική διεύθυνση από ότι στην s4 και συνεπώς δεν ισχύει η ισότητα μεταξύ s3 και s4.
Αυτή η διαχείριση έχει ως στόχο την βελτίωση της αποτελεσματικότητας του κώδικα και βασίζεται στο γεγονός ότι οι τιμές του τύπου String και των κλάσεων των θεμελιωδών τύπων είναι αμετάβλητες (immutable), δηλ. από την στιγμή που θα δημιουργηθεί μια τιμή ενός τέτοιου τύπου διασφαλίζεται πως δεν θα μεταβληθεί ως το τέλος του προγράμματος. Οι αμετάβλητοι τύποι εξυπηρετούν στον πολυνηματικό προγραμματισμό και χαρακτηρίζονται ως νηματικά ασφαλείς (thread safe).

3.8.1 Συλλέκτης απορριμμάτων

Εδώ όμως προκύπτει ένα άλλο θέμα. Το θέμα της απελευθέρωσης της μνήμης από τιμές μεταβλητών αναφοράς που δεν είναι χρήσιμες πλέον. Οι τοπικές μεταβλητές όπως είδαμε καταστρέφονται στο τέλος του μπλοκ στο οποίο δηλώθηκαν. Αυτό ισχύει είτε είναι μεταβλητές τιμής είτε μεταβλητές αναφοράς. Τι γίνεται όμως με τις τιμές των μεταβλητών αναφοράς που δεσμεύουν χώρο στην μνήμη δυναμικά; Είναι ένα θέμα που στην τυπική περίπτωση δεν απασχολεί ιδιαίτερα τον προγραμματιστή της Java καθώς η διαχείριση της δυναμικής μνήμης στην Java γίνεται από τον συλλέκτη απορριμμάτων (garbage collector). Ο συλλέκτης απορριμμάτων θεωρεί τις τιμές αναφοράς ως επιλέξιμες για αποδέσμευση μνήμης όταν δεν υπάρχει προς αυτές καμιά αναφορά μέσα στο πρόγραμμα.

```
static void garbageCollection() { //Κώδικας 3.4
    String s = new String("John");
    s = new String("Mary");
    System.out.println(s);
}
```

Κώδικας 3.4 Επιλέξιμες τιμές αναφοράς για αποδέσμευση μνήμης

Στην τελευταία γραμμή της garbageCollection στον κώδικα 3.4, το String John είναι ήδη επιλέξιμο προς αποδέσμευση. Η μόνη αναφορά στο John είναι η s έτσι όπως δημιουργείται στην πρώτη γραμμή. Στην δεύτερη γραμμή όμως, η αναφορά s ανακατευθύνεται σε άλλη τιμή. Έτσι το String John μένει χωρίς καμία αναφορά, δηλ. είναι αδύνατο να το προσπελάσουμε μέσα από το πρόγραμμά μας καθώς το πρόγραμμά μας δεν γνωρίζει την διεύθυνσή του. Με το τέλος της εκτέλεσης της garbageCollection, η τοπική μεταβλητή s καταστρέφεται οπότε μένει και το String Mary χωρίς αναφορά και άρα καθίσταται και αυτό επιλέξιμο προς αποδέσμευση.

Το πότε ακριβώς θα πραγματοποιηθεί η αποδέσμευση της μνήμης εξαρτάται από τον συλλέκτη απορριμμάτων ο οποίος έχει δικούς του αλγόριθμους με στόχο την αποδοτική διαχείριση της μνήμης.

3.9 Ασκήσεις

1. Περνάει με επιτυχία μεταγλώττιση η x1;

```
static void x1() {
     int k = 1, K = 2;
     System.out.println(k + " " + K);
}
```

Παραβιάζει κάποια από τις συμβάσεις ονοματολογίας; Εξηγήστε την απάντησή σας.

2. Περνάει με επιτυχία μεταγλώττιση η x2; Εξηγήστε την απάντησή σας.

```
static void x2() {
     int k = 1, j;
     j = j + 1;
     System.out.println(k + " " + j);
}
```

3. Περνάει με επιτυχία μεταγλώττιση η x3; Εξηγήστε την απάντησή σας.

```
static void x3() {
     int k = 1;
     final int j = k;
     j = j + 1;
     System.out.println(k + " " + j);
}
```

4. Περνάει με επιτυχία μεταγλώττιση η x4; Εξηγήστε την απάντησή σας.

```
static void x4() {
     int k = 1;
     {
          k = 2;
     }
     System.out.println(k);
}
```

5. Περνάει με επιτυχία μεταγλώττιση η x5; Εξηγήστε την απάντησή σας.

```
static void x5() {
     int k = 1;
     {
          int k = 2;
     }
     System.out.println(k);
}
```

6. Ποια είναι η έξοδος της x6;

```
static void x6() {
     int k = 010;
     System.out.println(k);
}
```

7. Ποια είναι η έξοδος της x7;

```
static void x7() {
     int k = 010, j = 10;
     System.out.println(k + j);
}
```

8. Ποια είναι η έξοδος της x8;

```
static void x8() {
     int k = 010, j = 0x10;
     System.out.println(k + j);
}
```

9. Ποια από τα παρακάτω ονόματα μεταβλητών παραβιάζουν τους κανόνες και ποια τις συμβάσεις ονοματολογίας;

```
int sum; char Start; final char END; boolean ExitLoop;
int finalsum; double 3oAthroisma; byte day; float float_10;
float _1_;
```

10. Υπάρχει διαφορά μεταξύ των κυριολεκτικών σταθερών 8 και 8.0;

Υποδείξεις: Προσπαθήστε να απαντήσετε χωρίς την βοήθεια του μεταγλωττιστή. Χρησιμοποιήστε τον μεταγλωττιστή για να ελέγξετε τις απαντήσεις σας.

Βιβλιογραφία

[1] “Primitive Data Types (The Java™ Tutorials > Learning the Java Language > Language Basics).” https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html (accessed Sep. 14, 2021).

[2] “Java All-in-One For Dummies, 6th Edition | Wiley,” Wiley.com. https://www.wiley.com/en-bb/Java+All+in+One+For+Dummies%2C+6th+Edition-p-9781119680451 (accessed Sep. 14, 2021).

[3] “Unicode 13.0.0.” https://unicode.org/versions/Unicode13.0.0/ (accessed Sep. 14, 2021).

[4] “The Java® Language Specification, Chapter 5. Conversions and Contexts, §5.1.2, §5.1.3, §5.1.4.” https://docs.oracle.com/javase/specs/jls/se10/html/index.html (accessed Sep. 14, 2021).

Κεφάλαιο 4

Σύνοψη
Σε αυτήν την ενότητα παρουσιάζονται οι τελεστές της Java και αναλύεται η συμπεριφορά τους. Πιο συγκεκριμένα, αναλύονται ο τελεστής εκχώρησης (assignment), οι αριθμητικοί τελεστές (arithmetic operators), ο τελεστής σύνδεσης συμβολοσειρών (concatenation), οι μοναδιαίοι (unary) τελεστές, ο τελεστής ισότητας (equality), οι σχεσιακοί (relational) τελεστές, ο τριαδικός τελεστής συνθήκης (ternary), ο τελεστής ελέγχου τύπου (instanceof) καθώς και οι τελεστές ψηφίου προς ψηφίο (Bitwise) και κύλισης ψηφίου (shift operators). Επιπλέον, παρουσιάζονται βασικοί τρόποι διαμόρφωσης της εξόδου στην οθόνη και εισαγωγής δεδομένων από το πληκτρολόγιο.

Προαπαιτούμενη γνώση
Προαπαιτείται να έχετε εγκατεστημένο το NetBeans με Java έκδοσης μεγαλύτερης ή ίσης της 8 και να γνωρίζετε τους θεμελιώδεις τύπους της Java, να χρησιμοποιείτε τον τύπο String, τις τοπικές μεταβλητές και σταθερές.

Λέξεις κλειδιά
Τελεστής, Τελεστέος, Εκχώρηση.

4 Τελεστές και βασική Είσοδος και Έξοδος

Η Java ενσωματώνει μια σειρά από συναρτήσεις που χρησιμεύουν ώστε να μπορούμε να κάνουμε διάφορες πράξεις (operations). Κάθε πράξη περιλαμβάνει τον τελεστή (operator) και τους τελεστέους (operands). Για παράδειγμα, στην πράξη 5+3, τελεστής είναι το σύμβολο της πρόσθεσης, + και τελεστέοι τα 5 και 3. Προκειμένου να διευκολυνθεί η έκφραση στο πρόγραμμα τέτοιων πράξεων, η Java παρέχει ειδικό συντακτικό κλήσης των τελεστών έτσι ώστε οι ανάλογες εκφράσεις να μην ακολουθούν το τυπικό συντακτικό κλήσης συναρτήσεων αλλά να διατυπώνονται κατά τρόπο που ομοιάζει με την εκτεταμένη χρήση τους στα μαθηματικά και άλλους τομείς.

Ένα άλλο ζήτημα, αναγκαίο για τα προγράμματά μας είναι η βασική είσοδος και έξοδος. Πρόκειται για τις διαδικασίες που επιτρέπουν την επικοινωνία και αλληλεπίδραση μεταξύ χρήστη και προγράμματος. Πως δηλ. θα πάρω δεδομένα από τον χρήστη και πως θα παρουσιάσω σε αυτόν τις πληροφορίες που θέλω;

Σε αυτό το κεφάλαιο παρουσιάζουμε τους βασικούς τελεστές που υποστηρίζει η Java [1] ταξινομώντας τους σε κατηγορίες ανάλογα με τα χαρακτηριστικά τους καθώς και τις βασικές λειτουργίες εισόδου από το πληκτρολόγιο και εξόδου στην οθόνη.

4.1 Τελεστές

Οι τελεστές (operators) στην Java οργανώνονται ανάλογα με τον αριθμό τελεστέων (operands) που δέχονται. Έτσι, έχουμε τους μοναδιαίους (unary) τελεστές, δηλ. αυτούς που δέχονται ένα τελεστέο, τους δυαδικούς (binary), δηλ. αυτούς με δύο τελεστέους και έναν τριαδικό (ternary) τελεστή. Επίσης, οι τελεστές κατηγοριοποιούνται σε αριθμητικούς (arithmetic), υποθετικούς (conditional) ή λογικούς (logical), σχεσιακούς (relational), τελεστές ψηφίου (bitwise) και τους τελεστές εκχώρησης (assignment).
Σε μια αλγεβρική έκφραση είναι δυνατό να συνδυάζονται περισσότεροι από ένα τελεστές. Για παράδειγμα, στην αλγεβρική έκφραση 2+3x5, συνδυάζονται οι τελεστές πρόσθεσης και πολλαπλασιασμού. Όπως γνωρίζουμε από την άλγεβρα, ο πολλαπλασιασμός έχει προτεραιότητα σε σχέση με την πρόσθεση. Επομένως, 2+3x5=17. Παρόμοια και στις εκφράσεις (expressions) της Java είναι δυνατό να συνδυάζονται πολλοί τελεστές. Επομένως, για τον συνεπή υπολογισμό των εκφράσεων έχουν καθορισθεί προτεραιότητες και για τους τελεστές της Java. Στον πίνακα 4.1, παρουσιάζονται οι τελεστές διατεταγμένοι από υψηλότερη σε χαμηλότερη προτεραιότητα. Τελεστές στην ίδια γραμμή του πίνακα 4.1, έχουν ίση προτεραιότητα.

Τύπος	**Λειτουργία**	**Προτεραιότητα**
Μοναδιαίος	Μεταθεματική αύξηση/μείωση	++, --
	Προθεματική αύξηση/μείωση, συμπλήρωμα, λογικό NOT	++, --, ~, !

Αριθμητικοί	Πολλαπλασιασμός	*, /, %
	Πρόσθεση	+, -
Διολίσθησης	Διολίσθηση	<<, >>, >>>
Σχεσιακοί	Σύγκριση	<, >, <=, >=, instanceof
	Έλεγχος ισότητας	==, !=
Ψηφίου	AND ψηφίο προς ψηφίο	&
	XOR ψηφίο προς ψηφίο	^
	OR ψηφίο προς ψηφίο	|
Λογικοί	Λογικό AND	&&
	Λογικό OR	||
Τριαδικός		? :
Εκχώρησης	Εκχώρησης	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=

***Πίνακας 4.1**: Προτεραιότητα τελεστών*

Ας υποθέσουμε την αλγεβρική έκφραση x=5+3-2. Η πρόσθεση και η αφαίρεση έχουν ίδια προτεραιότητα μεταξύ τους και ψηλότερη από την εκχώρηση. Επομένως, πρώτα θα υπολογισθεί η έκφραση 5+3-2 και το αποτέλεσμά της θα εκχωρηθεί στην μεταβλητή x. Πως όμως θα αξιολογηθεί το τμήμα 5+3-2; Ο κανόνας εδώ είναι πως η αξιολόγηση γίνεται από αριστερά προς τα δεξιά, δηλ. γίνεται πρώτα η πρόσθεση και το αποτέλεσμά της συνιστά τελεστέο για την αφαίρεση. Το ίδιο ισχύει για τις περισσότερες εκφράσεις της Java. Οι υπολογισμοί γίνονται από αριστερά προς τα δεξιά για όλους τους τελεστές πλην της εκχώρησης όπου η αξιολόγηση γίνεται από δεξιά προς τα αριστερά.

4.1.1 Ο τελεστής εκχώρησης

Η εκχώρηση είναι δυαδική πράξη και ο βασικός τελεστής της συμβολίζεται με τον χαρακτήρα '='. Στην γραμμή 2 του κώδικα 4.1, δηλώνονται τρείς ακέραιες μεταβλητές και αρχικοποιούνται οι δύο από αυτές. Στην γραμμή 3, εκχωρείται η τιμή 8 στην μεταβλητή i. Συνηθίζεται η πρώτη εκχώρηση σε μεταβλητή να αποκαλείται αρχικοποίηση (initialization). Η έκφραση (expression) στην γραμμή 3, επομένως, μπορούμε να πούμε πως αρχικοποιεί την i. Η αξιολόγηση γίνεται από την δεξιά προς την αριστερή πλευρά της έκφρασης. Αυτό σημαίνει πως πρώτα θα υπολογιστεί ότι υπάρχει στα δεξιά του τελεστή εκχώρησης και στην συνέχεια η τιμή που θα προκύψει θα εκχωρηθεί στην i. Βέβαια, στην συγκεκριμένη περίπτωση, δεν απαιτούνται ιδιαίτεροι υπολογισμοί δεδομένου ότι στα δεξιά του τελεστή υπάρχει μια αριθμητική σταθερά. Θα μπορούσε όμως να υπάρχει και μια έκφραση που το αποτέλεσμα της αξιολόγησής της θα ήταν ακέραιος. Στην συνέχεια αυτής της ενότητας θα δούμε αρκετά τέτοια παραδείγματα.

```
1      static void assignmentDemo() { //Κώδικας 4.1
2          int i, j = 5, k = 9;
3          i = 8;
4          System.out.println(j = i);
5          j = k = i;
6          System.out.println(i + " " + j + " " + k);
7      }
```

***Κώδικας 4.1**: Ο τελεστής εκχώρησης*

Εκείνο όμως που έχει ιδιαίτερο ενδιαφέρον, είναι πως η εκχώρηση επιστρέφει στο περιβάλλον κλήσης της μια τιμή και συγκεκριμένα την τιμή της i μετά την εκχώρηση. Αυτή η τιμή δεν αξιοποιείται στην γραμμή 3. Αξιοποιείται όμως στην γραμμή 4 που τυπώνει την έκφραση j=i. Αν τρέξετε τον κώδικα 4.1, θα διαπιστώσετε πως η τιμή που τυπώνεται είναι 8. Η εκχώρηση, αφού αξιολόγησε την τιμή του i, την εκχώρησε στην j και επέστρεψε στην println την τιμή του j μετά την εκχώρηση. Σε αυτό το χαρακτηριστικό βασίζεται και η υποστήριξη της πολλαπλής εκχώρησης (γραμμή 5).

Όπως φαίνεται στον πίνακα 4.1, είναι διαθέσιμοι και άλλοι τελεστές εκχώρησης. Η παρουσίασή τους γίνεται πιο κάτω σε αυτήν την ενότητα.

4.1.2 Αριθμητικοί τελεστές

Η Java παρέχει τους δυαδικούς τελεστές πρόσθεση, αφαίρεση, πολλαπλασιασμό, διαίρεση, υπόλοιπο διαίρεσης. Όλοι αυτοί οι τελεστές είναι δυαδικοί.

4.1.2.1 Πρόσθεση

Ο τελεστής πρόσθεσης συμβολίζεται με τον χαρακτήρα '+'. Στον κώδικα 4.2, προσθέτουμε το i με το j. Το αποτέλεσμα στέλνεται στην έξοδο όπου και τυπώνεται η τιμή 5. Στην γραμμή 20 όμως, το αποτέλεσμα ίσως φανεί λίγο παράδοξο. Πράγματι, η έξοδος του κώδικα της γραμμής 4 είναι i+j=32. Ο χαρακτήρας '+' χρησιμοποιείται τόσο για την πρόσθεση μεταξύ αριθμών, όσο και για την σύνδεση String (concatenation) μεταξύ τους. Το αν θα ληφθεί ως πρόσθεση αριθμών ή ως σύνδεση String εξαρτάται από τους τελεστέους που συντάσσονται μαζί του. Αν έστω και ένας από τους τελεστέους είναι String, τότε εκτελείται η σύνδεση Strings. Στην γραμμή 4, η αξιολόγηση γίνεται από αριστερά προς τα δεξιά. Έτσι, στην έκφραση "i+j="+i εντοπίζεται πρώτα το String "i+j=", οπότε λαμβάνεται η απόφαση πως πρόκειται για σύνδεση. Έτσι ο επόμενος ακέραιος τελεστέος μετατρέπεται αυτόματα σε String, έτσι το αποτέλεσμα "i+j="+i είναι "i+j=3" και ο τύπος του είναι String. Στην συνέχεια μένει να συνδεθεί αυτό το String με την ακέραιη μεταβλητή j. Με παρόμοια διαδικασία, το τελικό αποτέλεσμα είναι "i+j=32".

```
1     static void additionDemo() { //Κώδικας 4.2
2         int i = 3, j = 2;
3         System.out.println(i + j);
4         System.out.println("i+j= " + i + j); //concatenation
5         System.out.println("i+j= " + (i + j));
6         String h = "hello", w = "World";
7         System.out.println(h + " " + w);
8         double k = 1;
9         float m = 1f;
10        k = k + m;
11        char a = 'A';
12        System.out.println(a + 1);
13        System.out.println((char) (a + 1));
14    }
```

***Κώδικας 4.2**: Ο τελεστής πρόσθεσης*

Αν εκείνο που θέλουμε να πετύχουμε είναι να προστεθούν πρώτα το i με το j και το αποτέλεσμα να συνδεθεί με το "i+j= " ώστε να εκτυπωθούν μαζί, τότε θα πρέπει να δώσουμε προτεραιότητα στην πρόσθεση των ακεραίων, όπως δείχνει η γραμμή 5. Ισχύει και εδώ, όπως και στην άλγεβρα πως οι εκφράσεις μέσα σε παρενθέσεις αξιολογούνται κατά προτεραιότητα. Στην συνέχεια, οι γραμμές 22 και 23, παρουσιάζουν μια περίπτωση σύνδεσης String. Οι γραμμές, 8, 9 και 10 επιδεικνύουν πως ο τελεστής πρόσθεσης συντάσσεται με όλους τους αριθμητικούς τύπους. Ωστόσο, θα πρέπει να έχουμε υπόψη πως όταν συντάσσεται με δύο διαφορετικούς αριθμητικούς τύπους, ο τύπος του αποτελέσματος αποκτάται από τον το τελεστέο που έχει το μεγαλύτερο μέγεθος. Στο παράδειγμά μας, η έκφραση k+m επιστρέφει double. Αν επιχειρούσαμε να εκχωρήσουμε αυτό το αποτέλεσμα σε μεταβλητή τύπου float, ο μεταγλωττιστής θα παρήγαγε λάθος. Όπως έχουμε αναφέρει στην ενότητα 3.2, το αποτέλεσμα των εκφράσεων με byte ή short είναι αυτόματα int.
Ο τελεστής της πρόσθεσης δέχεται και τελεστέους τύπου char. Στην περίπτωση αυτή προστίθεται ο κωδικός των χαρακτήρων και το αποτέλεσμα της πράξης είναι τύπου int. Έτσι η γραμμή 12 θα προσθέσει στο 65 που είναι ο κωδικός του 'A' το 1 και θα τυπώσει το 66 που είναι ο κωδικός του 'B'. Μπορούμε να δούμε τον χαρακτήρα 'B' αν μετατρέψουμε τον τύπο του αποτελέσματος (type casting) όπως δείχνει η γραμμή 13.

4.1.2.2 Αφαίρεση

Ο τελεστής αφαίρεσης συμβολίζεται με τον χαρακτήρα '-'. Η γραμμή 3 του κώδικα 4.3 παράγει λάθος μεταγλώττισης εφόσον αποσχολιαστεί. Πράγματι, ανάλογα με όσα είπαμε στην περιγραφή του τελεστή πρόσθεσης, η έκφραση "i-j="+i επιστρέφει String. Στην συνέχεια η αφαίρεση ενός ακεραίου από ένα String δεν ορίζεται. Επομένως, σε αυτήν την περίπτωση, είμαστε αναγκασμένοι να δώσουμε προτεραιότητα στην αφαίρεση όπως φαίνεται στην γραμμή 4. Η αφαίρεση συντάσσεται και με δεδομένα τύπου char. Η συμπεριφορά

της σε αυτήν την περίπτωση είναι ανάλογη με της πρόσθεσης. Επίσης, ανάλογη με την πρόσθεση συμπεριφορά παρουσιάζει η αφαίρεση στην περίπτωση τελεστέων διαφορετικών αριθμητικών τύπων.

```
1     static void subtractionDemo() { //Κώδικας 4.3
2         int i = 3, j = 2;
3         //System.out.println("i-j= "+i-j);
4         System.out.println("i-j= " + (i - j));
5         char bita = 'B';
6         System.out.println(bita - 1);
7     }
```

***Κώδικας 4.3**: Ο τελεστής αφαίρεσης*

4.1.2.3 Πολλαπλασιασμός

Ο τελεστής του πολλαπλασιασμού συμβολίζεται με το χαρακτήρα αστερίσκο, '*'. Το αποτέλεσμα της γραμμής 4 του κώδικα 4.4 είναι το αναμενόμενο. Αυτό οφείλεται στο γεγονός ότι ο πολλαπλασιασμός έχει προτεραιότητα έναντι της πρόσθεσης. Επομένως, στη γραμμή 4 υπολογίζεται πρώτα το γινόμενο i*j και στη συνέχεια συνδέεται το String "i*j=" με το αποτέλεσμα του πολλαπλασιασμού. Κατά συνέπεια οι παρενθέσεις που περικλείουν το γινόμενο i*j, στη γραμμή 5, είναι περιττές χωρίς να οδηγούν σε λάθος. Ο πολλαπλασιασμός δέχεται και τελεστέους τύπου char. Το αποτέλεσμα είναι το γινόμενο των κωδικών των χαρακτήρων. Έτσι, η γραμμή 6 θα τυπώσει 132, δηλ. τον κωδικό του B επί 2.

```
1     static void multiplicationDemo() { //Κώδικας 4.4
2        int i = 3, j = 2;
3        char bita = 'B';
4        System.out.println("i*j= " + i * j);
5        System.out.println("i*j= " + (i * j));
6        System.out.println("bita*2=" + bita * 2);
7     }
```

***Κώδικας 4.4**: Ο τελεστής του πολλαπλασιασμού*

4.1.2.4 Διαίρεση

Ο τελεστής της διαίρεσης συμβολίζεται με τον χαρακτήρα πλάγια κάθετος, '/'. Ότι αναφέρθηκε για τον πολλαπλασιασμό, ισχύει κατ' αντιστοιχία και για την διαίρεση. Επιπλέον, η διαίρεση διακρίνεται σε διαίρεση μεταξύ πραγματικών και διαίρεση μεταξύ ακέραιων. Αν οι τελεσταίοι είναι ακέραιοι τότε έχουμε ακέραιη διαίρεση. Αν ένας τουλάχιστον από τους τελεσταίους είναι πραγματικός τότε έχουμε διαίρεση.

```
1     static void divisionDemo() { //Κώδικας 4.5
2        int i = 3;
3        int j = 2;
4        System.out.println("i/j= " + i / j);
5        System.out.println("i/j= " + (i / j));
6        System.out.println();
7        double d = 3, b = 2;
8        System.out.println("d/b= " + (d / b));
9        System.out.println("(double)i/j= " + (double) i / j);
10    }
```

***Κώδικας 4.5**: Ο τελεστής της διαίρεσης*

Η έξοδος της γραμμής 4 του κώδικα 4.5 είναι i/j=1. Και οι δύο τελεσταίοι είναι ακέραιοι, οπότε η Java εκτελεί διαίρεση ακεραίων. Αντίθετα στην γραμμή 8, η έξοδος είναι d/b=1.5. Αν επιθυμούμε να κάνουμε διαίρεση πραγματικών μεταξύ ακέραιων μεταβλητών, θα πρέπει να μετατρέψουμε τον τύπο τουλάχιστον σε έναν από τους τελεστέους όπως φαίνεται στην γραμμή 9.

4.1.2.4 Υπόλοιπο διαίρεσης

Το υπόλοιπο συμβολίζεται με το χαρακτήρα επί τοις εκατό , '%'. Εδώ έχουμε διαφοροποίηση μεταξύ υπολοίπου που προκύπτει από διαίρεση πραγματικών και υπολοίπου διαίρεσης ακεραίων.

```
1     static void moduloDemo() { //Κώδικας 4.6
2        int i = 3, j = 2; //1
3        System.out.println("i%j= " + i % j);
4        System.out.println("i%j= " + (i % j));
5        double d = 7.5, b = 3.5;
6        System.out.println("d%b= " + d % b);
7        System.out.println("d%b= " + (d % b));
8     }
```

***Κώδικας 4.6**: Το υπόλοιπο διαίρεσης*

Η έξοδος της γραμμής 3 του κώδικα 4.6 είναι i%j=1. Ο τύπος της έκφρασης i%j είναι int. Στην γραμμή 6 εκτελείται διαίρεση μεταξύ πραγματικών και η έξοδος είναι d%b=0.5. Ο τύπος της έκφρασης d%b είναι double.

4.1.3 Μοναδιαίοι τελεστές

Οι μοναδιαίοι τελεστές που υποστηρίζονται είναι το θετικό πρόσημο, το αρνητικό πρόσημο, οι τελεστές προσαύξησης, οι τελεστές μείωσης και ο λογικός τελεστής αντιστροφής.

4.1.3.1 Οι τελεστές πρόσημου

Ο μοναδιαίος τελεστής μείον (unary minus operator) συμβολίζεται με τον χαρακτήρα '-' και επιστρέφει τον αντίθετο του τελεσταίου. Στην γραμμή 73, ο κώδικας 4.7, τυπώνει -1. Ο τελεστής μείον, δέχεται ως τελεστέους και πραγματικούς αλλά και χαρακτήρες. Ανάλογη είναι η λειτουργία του μοναδιαίου τελεστή συν (unary plus operator), '+'. Ωστόσο, ο τελεστής συν, ισχύει εξορισμού και έτσι δεν αναφέρεται σχεδόν ποτέ σε κώδικες Java.

```
1     static void unaryOps() { //Κώδικας 4.7
2        int i = 1;
3        System.out.println(i);
4        System.out.println(+i);
5        System.out.println(-i);
6        System.out.println();
7
8        System.out.println("Postfix increment");
9        System.out.println(i++);
10       System.out.println(i);
1        System.out.println();
12
13        System.out.println("Prefix increment");
14        i = 1;
15        System.out.println(++i);
16        System.out.println(i);
17        System.out.println();
18
19        System.out.println("Postfix decrement");
20        i = 1;
21        System.out.println(i--);
22        System.out.println(i);
23        System.out.println();
24
25        System.out.println("Prefix decrement");
26        i = 1;
27        System.out.println(--i);
28        System.out.println(i);
29        System.out.println();
30
31        boolean b = false;
```

```
32          System.out.println("b= " + b + ", !b= " + !b);
33      }
```

***Κώδικας 4.7**: Μοναδιαίοι τελεστές*

4.1.3.2 Τελεστές προσαύξησης

Ο τελεστής προσαύξησης συμβολίζεται με τους χαρακτήρες "++" και παρέχεται σε δύο μορφές, στον προθεματικό τελεστή προσαύξησης (prefix increment operator) και στον μεταθεματικό τελεστή προσαύξησης (postfix increment operator). Όταν ο τελεστής προηγείται του τελεσταίου, έχουμε τον προθεματικό τελεστή. Αντίθετα, έχουμε τον μεταθεματικό όταν ο τελεστέος προηγείται του τελεστή. Η λειτουργία των δύο τελεστών διαφέρει. Ο προθεματικός τελεστής αυξάνει κατά 1 την τιμή του τελεσταίου και επιστρέφει την αυξημένη τιμή. Αντίθετα, ο μεταθεματικός τελεστής αυξάνει την τιμή του τελεσταίου κατά 1, επιστρέφει όμως την τιμή που είχε ο τελεσταίος πριν την αύξηση. Ο κώδικας στην γραμμή 9 θα τυπώσει 1. Ο μεταθεματικός τελεστής αύξησε την τιμή του i κατά 1 αλλά επέστρεψε την τιμή που είχε το i πριν την αύξηση. Στην συνέχεια στη γραμμή 10 θα τυπωθεί η αυξημένη τιμή, δηλ. το 2. Αντίθετα, στην γραμμή 15 θα τυπωθεί 2. Επίσης, 2 θα τυπωθεί στην γραμμή 16.

4.1.3.3 Τελεστές μείωσης

Παρόμοια είναι η λειτουργία των τελεστών μείωσης. Ο μεταθεματικός τελεστής μείωσης μειώνει την τιμή του τελεσταίου κατά 1, επιστρέφει όμως την τιμή πριν την μείωση. Αντίθετα, ο προθεματικός τελεστής μείωσης μειώνει την τιμή του τελεσταίου κατά 1 και επιστρέφει την τιμή μετά τη μείωση.
Στην γραμμή 21 του κώδικα 4.7, θα τυπωθεί 1 ενώ στη γραμμή 22 θα τυπωθεί 0. Αντίθετα, στις γραμμές 27 και 28 θα τυπωθεί 0.

4.1.3.4 Ο λογικός τελεστής αντιστροφής

Ο λογικός τελεστής αντιστροφής (logical complement operator) συμβολίζεται με τον χαρακτήρα '!' και συντάσσεται με τύπο boolean. Το αποτέλεσμα του τελεστή είναι η αντιστροφή της τιμής του τελεσταίου. Έτσι, ο κώδικας στη γραμμή 32, θα τυπώσει b= false, !b= true.

4.1.4 Σχεσιακοί τελεστές

Υποστηρίζονται οι ακόλουθοι σχεσιακοί τελεστές: ο τελεστής ελέγχου ισότητας, ο τελεστής ελέγχου ανισότητας, ο τελεστής μεγαλύτερο από, ο τελεστής μεγαλύτερο ή ίσο, ο τελεστής μικρότερο από και ο τελεστής μικρότερο ή ίσο. Όλοι οι σχεσιακοί τελεστές είναι δυαδικοί και επιστρέφουν boolean τιμή.

4.1.4.1 Ο τελεστής ελέγχου ισότητας

Ο τελεστής ελέγχου ισότητας (equality operator) συμβολίζεται με "==" και επιστρέφει true αν και μόνο αν οι τελεστέοι του έχουν την ίδια τιμή. Στην περίπτωση που οι τελεστέοι είναι ακέραιοι τύποι ή boolean, η λειτουργία του τελεστή είναι απλή. Η γραμμή 3 του κώδικα 4.8, θα τυπώσει i==j is true.

Στην περίπτωση όμως πραγματικών τελεστέων δεν ισχύει το ίδιο. Στην γραμμή 4, ορίζουμε τρείς μεταβλητές τύπου double. Η μεταβλητή d αρχικοποιείται στην τιμή 1+2*k. Στην συνέχεια, στις γραμμές 107 και 108 προσθέτουμε στην b την k, δηλ. προσθέτουμε δύο φορές το k. Σύμφωνα με τα μαθηματικά αναμένουμε πως το d είναι ίσο με το b. Ωστόσο λόγω του προσεγγιστικού χαρακτήρα και της διαφορετικής σειράς που γίνονται οι πράξεις κατά τον υπολογισμό των d και b, το d δεν είναι ίσο με το b. Η έξοδος της γραμμής 7 είναι b=1.0000200000000001 and d=1.00002. Λογική συνέπεια είναι πως η γραμμή 8, τυπώνει false. Ο γενικός κανόνας είναι πως η χρήση του τελεστή ισότητας με πραγματικούς τελεστέους πρέπει να αποφεύγεται. Αντίθετα, για την σύγκριση πραγματικών αριθμών θα πρέπει να χρησιμοποιείται κατάλληλη συνάρτηση προσεγγιστικής ισότητας που παρουσιάστηκε στην ενότητα 3.2.

Η συμπεριφορά του τελεστή με τύπους αναφοράς φαίνεται διαφορετική καθώς στην περίπτωση αυτή συγκρίνονται διευθύνσεις και όχι περιεχόμενα. Οι αναφορές τύπου String s1 και s2 δείχνουν στο ίδιο String στην μνήμη το περιεχόμενο του οποίου είναι "John". Επομένως, η γραμμή 11 τυπώνει s1==s2 is true. Αντίθετα, η s3 αναφέρεται σε ένα String που παρότι το περιεχόμενό του είναι "John" βρίσκεται σε άλλη θέση της μνήμης. Επομένως, η γραμμή 12 τυπώνει s1==s3 is false. Αν θέλουμε να συγκρίνουμε για ισότητα δύο Strings ως προς

το περιεχόμενό τους, μπορούμε να χρησιμοποιήσουμε την συνάρτηση equals όπως φαίνεται στην γραμμή 13. Η έξοδος της γραμμής 13 είναι s1.equals(s3) is true. Κατά αντίστοιχο τρόπο λειτουργεί ο τελεστής ισότητας για όλους τους μη θεμελιώδεις τύπους. Κάθε μεταβλητή αναφοράς που έχει λάβει την τιμή της από την επιστροφή της new δεν μπορεί να συγκριθεί με τον τελεστή ισότητας αλλά με την συνάρτηση equals.

```
1     static void equalityOp() { //Κώδικας 4.8
2         int i = 1, j = 1;
3         System.out.println("i==j is " + (i == j));
4         double k = 0.00001, d = 1 + 2 * k, b = 1;
5         b = b + k;
6         b = b + k;
7         System.out.println("b=" + b + " and d=" + d);
8         System.out.println(b == d);
9         System.out.println(Double.compare(b,d));
10        String s1 = "John", s2 = "John", s3 = new String("John");
11        System.out.println("s1==s2 is " + s1 == s2);
12        System.out.println("s1==s3 is " + s1 == s3);
13        System.out.println("s1.equals(s3) is " + s1.equals(s3));
14    }
```

***Κώδικας 4.8**: Ο τελεστής ελέγχου ισότητας*

4.1.4.2 Ο τελεστής ανισότητας

Ο τελεστής ανισότητας (inequality operator) συμβολίζεται με "!=". Ο τελεστής λειτουργεί με ανάλογο τρόπο και περιορισμούς με τον τελεστή ισότητας. Ισχύει !(a==b) == (a!=b), δηλ. αν αντιστρέψουμε τον έλεγχο ισότητας μεταξύ δύο τελεστέων θα λάβουμε το ίδιο αποτέλεσμα με αυτό που θα λάβουμε εφαρμόζοντας τον τελεστή ανισότητας στους ίδιους τελεστέους. Επομένως, οι παρατηρήσεις σχετικά με τους πραγματικούς αριθμούς και τις μεταβλητές αναφοράς που επισημάνθηκαν στον τελεστή ισότητας ισχύουν κατ' αναλογία και εδώ.

4.1.4.3 Οι υπόλοιποι σχεσιακοί τελεστές

Πρόκειται για τον τελεστή μεγαλύτερο από (greater than) που συμβολίζεται με τον χαρακτήρα '>', τον τελεστή μεγαλύτερο ή ίσο από (greater than or equal to) που συμβολίζεται με τον χαρακτήρα '>=', τον τελεστή μικρότερο από (less than) που συμβολίζεται με τον χαρακτήρα '<' τον τελεστή μικρότερο ή ίσο από (less than or equal to) που συμβολίζεται με τον χαρακτήρα '<='. Οι τελεστές αυτοί δέχονται ως τελεστέους όλους τους θεμελιώδεις τύπους πλην του τύπου boolean. Δεν συντάσσονται με μη θεμελιώδεις τύπους. Με πραγματικούς τελεσταίους, θα πρέπει να χρησιμοποιούνται με την επιφύλαξη που προκύπτει από τον προσεγγιστικό χαρακτήρα των πραγματικών.

```
1     static void relationalOps() { //Κώδικας 4.9
2        int i = 5, j = 2;
3        char a = 'A', b = 'B';
4        System.out.println(i > j);
5        System.out.println(b > a);
6     }
```

***Κώδικας 4.9**: Ο τελεστής μεγαλύτερο από*

Η έξοδος της γραμμής 4 του κώδικα 4.9 εμφανίζει true. Στην γραμμή 5, συγκρίνονται δύο χαρακτήρες. Η έξοδος της γραμμής είναι επίσης true καθώς εκείνο που συμβαίνει είναι πως συγκρίνονται οι κωδικοί των χαρακτήρων. Παρότι, οι τελεστές αυτοί δεν συντάσσονται με String, αξίζει να σημειωθεί πως γενικότερα, η σύγκριση μεταξύ αριθμών είναι αριθμητική ενώ η σύγκριση μεταξύ σειρών χαρακτήρων είναι λεξικογραφική. Σύμφωνα με την αριθμητική σύγκριση το 10 είναι μεγαλύτερο από το 2, ενώ σύμφωνα με την λεξικογραφική το "10" είναι μικρότερο από το "2".

4.1.5 Λογικοί τελεστές

Περιλαμβάνονται οι λογικοί τελεστές and και or, ο τριαδικός τελεστής και ο τελεστής instanceof. Ο τελεστής instanceof σχετίζεται με το αντικειμενοστρεφές μοντέλο και η παρουσίασή του γίνεται στην ενότητα 13.2.1.

4.1.5.1 Λογικό and

Το λογικό and συμβολίζεται ως "&&", είναι δυαδικός τελεστής και συντάσσεται αποκλειστικά με τελεστέους τύπου boolean. Η τιμή επιστροφής του υπολογίζεται όπως δείχνει ο πίνακας 4.2.

Τελεστέος 1	**Τελεστέος 2**	**Αποτέλεσμα**
TRUE	TRUE	TRUE
TRUE	FALSE	FALSE
FALSE	TRUE	FALSE
FALSE	FALSE	FALSE

Πίνακας 4.2 *Πίνακας αληθείας λογικού and*

Η υλοποίηση του λογικού and ακολουθεί την λογική βραχέως κυκλώματος (short circuit). Η αξιολόγηση γίνεται από αριστερά προς τα δεξιά. Αν ο πρώτος τελεστέος αρκεί για τον υπολογισμό του αποτελέσματος, ο δεύτερος τελεστέος δεν αξιολογείται. Σύμφωνα με τον πίνακα 2, όταν ο τελεστέος 1, έχει την τιμή false, τότε το αποτέλεσμα της έκφρασης είναι false ανεξάρτητα από την τιμή του δεύτερου τελεσταίου. Αντίθετα, όταν ο πρώτος τελεστέος έχει την τιμή true, τότε το αποτέλεσμα της έκφρασης εξαρτάται από την τιμή του δεύτερου τελεσταίου.

```
1    static void shortCircuitAnd() { //Κώδικας 4.10
2        int i = 1, j = 2;
3        System.out.println("i++>1 && j--<3 is " + (i++ > 1 && j-- < 3));
4        System.out.println("i= " + i + " j= " + j);
5        i = 1;
6        j = 2;
7        System.out.println("++i>1 && j--<3 is " + (++i > 1 && j-- < 3));
8        System.out.println("i= " + i + " j= " + j);
9    }
```

Κώδικας 4.10*: Short circuit and*

Η γραμμή 3 του κώδικα 4.10, τυπώνει i++>1 && j--<3 is false. Ο πρώτος τελεστέος είναι η λογική έκφραση i++>1, η τιμή του είναι false. Επομένως, ο δεύτερος, δηλ. το j--<3 δεν θα αξιολογηθεί και άρα δεν θα μειωθεί η τιμή του j με αποτέλεσμα η έξοδος της γραμμής 4 είναι i= 2 j= 2. Αντίθετα, στην γραμμή 7, ο πρώτος τελεστέος του λογικού and είναι ++i>1 και η τιμή του είναι true. Επομένως θα αξιολογηθεί και ο δεύτερος τελεστέος με αποτέλεσμα να μειωθεί το j. Καθώς και ο δεύτερος τελεστέος είναι true, το αποτέλεσμα του λογικού and είναι επίσης true. Επομένως, η γραμμή 7 θα τυπώσει ++i>1 && j--<3 is true και η γραμμή 8 θα τυπώσει i= 2 j= 1.

4.1.5.2 Λογικό or

Το λογικό or συμβολίζεται ως "||", είναι δυαδικός τελεστής και συντάσσεται αποκλειστικά με τελεστέους τύπου boolean. Η τιμή επιστροφής του υπολογίζεται όπως δείχνει ο πίνακας 4.3.

Τελεστέος 1	**Τελεστέος 2**	**Αποτέλεσμα**
TRUE	TRUE	TRUE
TRUE	FALSE	TRUE
FALSE	TRUE	TRUE
FALSE	FALSE	FALSE

Πίνακας 4.3 *Πίνακας αληθείας λογικού or*

Η υλοποίηση του λογικού or ακολουθεί επίσης την λογική βραχέως κυκλώματος (short circuit). Η αξιολόγηση γίνεται από αριστερά προς τα δεξιά. Αν ο πρώτος τελεστέος αρκεί για τον υπολογισμό του αποτελέσματος, ο δεύτερος τελεστέος δεν αξιολογείται. Σύμφωνα με τον πίνακα 4.3, όταν ο τελεστέος 1, έχει την τιμή true, τότε το αποτέλεσμα της έκφρασης είναι true ανεξάρτητα από την τιμή του δεύτερου τελεσταίου. Αντίθετα, όταν ο πρώτος τελεστέος έχει την τιμή false, τότε το αποτέλεσμα της έκφρασης εξαρτάται από την τιμή του δεύτερου τελεσταίου.

```
1     static void shortCircuitOr() { //Κώδικας 4.11
2        int i = 1, j = 2;
3        System.out.println("i++>=1 || j--<3 is " + (i++ >= 1 || j-- < 3));
4        System.out.println("i= " + i + " j= " + j);
5        i = 1;
6        j = 2;
7        System.out.println("i++>=1 || j--<3 is " + (i++ > 1 || j-- < 3));
8       System.out.println("i= " + i + " j= " + j);
9     }
```

***Κώδικας 4.11**: Short circuit or*

Η γραμμή 3 του κώδικα 4.11, τυπώνει i++>=1 || j--<3 is true. Ο πρώτος τελεστέος είναι η λογική έκφραση i++>=1, η τιμή του είναι true. Επομένως, ο δεύτερος, δηλ. το j--<3 δεν θα αξιολογηθεί και άρα δεν θα μειωθεί η τιμή του j με αποτέλεσμα η έξοδος της γραμμής 4 ναι είναι i= 2 j= 2. Αντίθετα, στην γραμμή 7, ο πρώτος τελεστέος του λογικού or είναι ++i>1 και η τιμή του είναι false. Επομένως θα αξιολογηθεί και ο δεύτερος τελεστέος με αποτέλεσμα να μειωθεί το j. Καθώς ο δεύτερος τελεστέος είναι true, το αποτέλεσμα του λογικού or είναι επίσης true. Επομένως, η γραμμή 7 θα τυπώσει ++i>1 && j--<3 is true και η γραμμή 8 θα τυπώσει i= 2 j= 1.

4.1.5.3 Ο τριαδικός τελεστής

Ο τριαδικός τελεστής (ternary operator) συμβολίζεται ως “?:”. Όπως φανερώνει το όνομά του, συντάσσεται με τρείς τελεστέους. Ο πρώτος είναι μια έκφραση τύπου boolean, π.χ. i<j, ο δεύτερος η τιμή που επιστρέφει ο τελεστής αν ο πρώτος τελεστέος έχει τιμή true και ο τρίτος τελεστέος δηλώνει την τιμή επιστροφής αν ο πρώτος τελεστέος έχει την τιμή false. Στον κώδικα 4.12 δίνουμε ένα παράδειγμα χρήσης του τριαδικού τελεστή. Στην γραμμή 3, ο πρώτος τελεστέος είναι η λογική έκφραση i<j. Στο συγκεκριμένο παράδειγμα, η έκφραση αυτή έχει τιμή true, επομένως επιστρέφεται η τιμή μετά το ?, δηλ. η τιμή του i. Αν όμως, η τιμή i<j ήταν false, τότε θα επιστρεφόταν ο τρίτος τελεστέος, δηλ. το j, η τιμή μετά τον χαρακτήρα ‘:’.

```
1     static void ternaryOp() { //Κώδικας 4.12
2         int i = 10, j = 2;
3         int minIJ = i < j ? i : j;
4         System.out.println(minIJ);
5         double d = 5.2;
6         double max = i > d ? i : d;
7         System.out.println(max);
8     }
```

***Κώδικας 4.12**: Ο τριαδικός τελεστής*

Ο τριαδικός τελεστής μπορεί να επιστρέψει δεδομένα οποιουδήποτε τύπου με τον περιορισμό πως οι τύποι του δεύτερου και τρίτου τελεσταίου είναι συμβατοί μεταξύ τους. Αν οι τύποι αυτοί είναι ίδιοι μεταξύ τους, τότε ο τύπος επιστροφής του τελεστή είναι ίδιος με τον τύπο του δεύτερου και τρίτου τελεσταίου. Αν δεν είναι ίδιοι, τότε ο τύπος επιστροφής είναι ο τύπος με την μεγαλύτερη χωρητικότητα. Για παράδειγμα, στην γραμμή 6 του κώδικα 4.12, ο τελεστής επιστρέφει το i, τύπου int, εφόσον i>d και το d, τύπου double, εφόσον !(i>d). Ανεξάρτητα, από ποια τιμή θα επιστραφεί, ο τύπος της θα είναι double.

4.1.6 Τελεστές ψηφίου

Υποστηρίζονται τέσσερις τελεστές ψηφίου, ο τελεστής ψηφίου AND, ο τελεστής ψηφίου OR, ο τελεστής ψηφίου XOR (Exclusive OR) και ο τελεστής συμπληρώματος. Οι τρείς πρώτοι είναι δυαδικοί ενώ ο τελεστής

συμπληρώματος είναι μοναδιαίος. Οι τελεστές ψηφίου δέχονται ως τελεστέους όλους τους ακέραιους τύπους και τον τύπο boolean. Οι ακέραιοι τύποι μπορούν να συνδυαστούν μεταξύ τους με τους τελεστές ψηφίου, όχι όμως και με τους boolean. Οι τελεστές ψηφίου όταν συντάσσονται με boolean λειτουργούν ως λογικοί τελεστές πλήρους αξιολόγησης. Οι τελεστές ψηφίου αντιμετωπίζουν τους τελεστέους τους σαν ψηφιοχάρτες (bitmaps) και κάνουν υπολογισμούς ψηφίο προς ψηφίο. Όταν συντάσσονται με ακέραιους τύπους, αν έστω και ένας από τους τελεστέους είναι long, τότε επιστρέφουν long αλλιώς int. Όταν συντάσσονται με boolean, επιστρέφουν boolean.

4.1.6.1 Τελεστής ψηφίου AND

Ο τελεστής ψηφίου AND (bitwise AND) συμβολίζεται με τον χαρακτήρα '&'. Ο τελεστής ακολουθεί τον πίνακα αληθείας του AND, δηλ. αν και τα δύο ψηφία έχουν την τιμή 1, τότε το αντίστοιχο ψηφίο του αποτελέσματος έχει την τιμή 1. Αν έστω και ένα από τα ψηφία έχει την τιμή 0, τότε το αντίστοιχο ψηφίο του αποτελέσματος έχει την τιμή 0. Η γραμμή 4 του κώδικα 4.13 τυπώνει 0. Ο ψηφιοχάρτης του i αποτελείται μόνο από μηδενικά ψηφία. Τα μηδενικά αυτά ψηφία συνδυάζονται με τα αντίστοιχα ψηφία του j. Επομένως, όλα τα ψηφία του αποτελέσματος έχουν την τιμή 0.

```
1   static void bitwiseAnd() { //Κώδικας 4.13
2       int i = 0, j = 1232;
3       int k = i & j;
4       System.out.println("i&j=" + k);
5       int i1 = 0b00000000000000000000000000001010;
6       int j1 = 0b00000000000000001100001100001111;
7       System.out.println("i1=" + i1 + " " + "j1=" + j1);
8       System.out.println("i1&j1=" + (i1 & j1));
9       char d1 = 'A';
10      int m = i & d1;
11      System.out.println("i&d=" + m);
12      long b1 = 0;
13      long rslt = b1 & i;
14      System.out.println(rslt);
15  }
```

***Κώδικας 4.13**: Ο τελεστής ψηφίου AND*

Ένας άλλος τρόπος για να καταλάβουμε καλύτερα τους τελεστές ψηφίου είναι να ορίσουμε ακέραιες μεταβλητές σε δυαδική μορφή όπως έχουμε κάνει στις γραμμές 159 και 160. Στην γραμμή 7, τυπώνουμε τα i1 και j1 και βλέπουμε πως i1=10 και j1=100111; Το αποτέλεσμα του τελεστή μεταξύ των δύο αυτών μεταβλητών είναι ένας ψηφιοχάρτης που έχει 1 μόνο στις θέσεις που έχει 1 και η i1 και η j1. Όλες οι υπόλοιπες θέσεις καταλαμβάνονται από το 0. Οι θέσεις στις οποίες έχουν 1 και η i1 και η j1, ταυτίζονται με τις θέσεις που έχει 1 η i1. Επομένως, i1&j1==i1. Πράγματι, η γραμμή 8 τυπώνει i1&j1=10.

Όπως έχουμε ήδη αναφέρει, όλοι οι ακέραιοι τύποι είναι δεκτοί ως τελεστέοι. Στην γραμμή 10, εφαρμόζεται ο τελεστής μεταξύ μιας μεταβλητής τύπου int και μίας τύπου char. Το αποτέλεσμα είναι τύπου int και η τιμή του 0. Στην γραμμή 13, εφαρμόζεται ο τελεστής μεταξύ μιας μεταβλητής τύπου int και μιας τύπου long. Το αποτέλεσμα είναι τύπου long.

4.1.6.2 Τελεστής ψηφίου OR

Ο τελεστής ψηφίου OR (bitwise OR) συμβολίζεται με τον χαρακτήρα '|'. Ισχύουν και εδώ κατ' αναλογία, όσα παρατίθενται για τον τελεστή ψηφίου AND. Η διαφορά έγκειται στο ότι εδώ ακολουθείται ο πίνακας αληθείας του OR, δηλ. αν ένα τουλάχιστον ψηφίο έχει την τιμή 1, τότε το αποτέλεσμα είναι 1, διαφορετικά είναι 0.

4.1.6.3 Τελεστής ψηφίου XOR:

Ο τελεστής ψηφίου XOR (bitwise XOR) συμβολίζεται με τον χαρακτήρα '^'. Ισχύουν και εδώ κατ' αναλογία, όσα παρατίθενται για τον τελεστή ψηφίου AND. Η διαφορά έγκειται στο ότι εδώ ακολουθείται ο πίνακας αληθείας του XOR που παρατίθεται αμέσως από κάτω στον πίνακα 4.4, δηλ. αν ένα μόνο ψηφίο έχει την τιμή 1, τότε το αποτέλεσμα είναι 1, διαφορετικά είναι 0.

bit 1	bit 2	bit1^bit2
1	1	0
1	0	1
0	1	1
0	0	0

Πίνακας 4.4 *Πίνακας XOR*

4.1.6.4 Τελεστής ψηφίου NOT

Ο τελεστής ψηφίου NOT (bitwise NOT) ή τελεστής συμπληρώματος (complement operator) είναι μοναδιαίος, συμβολίζεται με τον χαρακτήρα '~' και αντιστρέφει όλα τα bits του τελεσταίου του. Ο τελεστής δεν συντάσσεται με μεταβλητές τύπου boolean.

4.1.7 Τελεστές διολίσθησης

Υποστηρίζονται οι τελεστές, διολίσθηση δεξιά, διολίσθηση αριστερά και μη προσημασμένη διολίσθηση δεξιά.

4.1.7.1 Διολίσθηση δεξιά

Ο τελεστής δεξιάς διολίσθησης (right shift) είναι δυαδικός, συμβολίζεται ως "**>>**"και συντάσσεται με ακέραιους τύπους. Εφόσον ο πρώτος τελεστέος του είναι long επιστρέφει long, διαφορετικά επιστρέφει int. Η εφαρμογή του τελεστή ενεργεί ως εξής: Καταρχάς δεν επηρεάζει τους τελεστέους αλλά δημιουργεί ένα αντίγραφο του ψηφιοχάρτη του πρώτου τελεσταίου. Στην συνέχεια, προκαλεί διολίσθηση προς τα δεξιά στο αντίγραφο του πρώτου τελεσταίου κατά τον αριθμό των bits που καθορίζει η τιμή του δεύτερου τελεσταίου. Έτσι, με την διολίσθηση χάνεται ένας αριθμός από bits από την δεξιά πλευρά του αντιγράφου. Τα bits που χάνονται αντικαθιστούνται στην αριστερή πλευρά του αντιγράφου με τιμή την τιμή του πρώτου bit του, δηλ. την τιμή του πρόσημου. Ας σημειωθεί εδώ πως η Java χρησιμοποιεί για την κωδικοποίηση ακεραίων την παράσταση μέτρο και πρόσημο για τις θετικές τιμές και την παράσταση συμπληρώματος ως προς δύο (two's complement) για τις αρνητικές. Επομένως, η τιμή του πρόσημου είναι 0 για τους θετικούς ακέραιους και 1 για τους αρνητικούς. Για λεπτομέρειες σχετικά με την κωδικοποίηση των ακεραίων σε μορφή συμπληρώματος ως προς δύο, ανατρέξτε στο παράρτημα 1.

```
1   static void rightShift() { //Κώδικας 4.14
2       int i = 3;
3       int j = -3;
4       System.out.println("Integer.toBinaryString(" + i + ")=" +
Integer.toBinaryString(i));
5       System.out.println("Integer.toBinaryString(" + i + ">>1)=" +
Integer.toBinaryString(i >> 1));
6       System.out.println("Integer.toBinaryString(" + j + ")=" +
Integer.toBinaryString(j));
7       System.out.println("Integer.toBinaryString(" + j + ">>1)=" +
Integer.toBinaryString(j >> 1));
8   }
```

Κώδικας 4.14*: Ο τελεστής δεξιάς διολίσθησης*

Στον κώδικα 4.14, έχουμε ορίσει τις ακέραιες μεταβλητές i και j. Στην γραμμή 4, εκτυπώνεται ο ψηφιοχάρτης της μεταβλητής i με την βοήθεια της συνάρτησης toBinaryString. Η έξοδος αυτής της γραμμής είναι Integer.toBinaryString(3)=11. Πράγματι, η αναπαράσταση του 3 με την μορφή συμπληρώματος του δύο είναι 11. Βέβαια, δεδομένου, ότι ο int στην Java έχει μέγεθος 32 bit, η πραγματική αναπαράσταση στην μνήμη είναι 00000000000000000000000000000011. Απλώς, η Java δεν τυπώνει τυχόν μηδενικά στην αρχή ενός αριθμού (leading zeroes) δεδομένου ότι αυτά δεν επηρεάζουν την τιμή του. Η γραμμή 5 εκτυπώνει Integer.toBinaryString(3>>1)=1. Στην μεταβλητή i έγινε διολίσθηση δεξιά κατά 1 bit, επομένως από το

αποτέλεσμα χάθηκε το τελευταίο 1 ενώ παράλληλα προστέθηκε στην αρχή ένα μηδέν. Ο πλήρης ψηφιοχάρτης του αποτελέσματος είναι 00000000000000000000000000000001.

Στην γραμμή 6 τυπώνεται ο ψηφιοχάρτης του -3. Πιο συγκεκριμένα, η έξοδος της γραμμής 6 είναι Integer.toBinaryString(-3)=11111111111111111111111111111101. Μετά την διολίσθηση δεξιά κατά 1, το τελευταίο 1 χάνεται και προστίθεται στην αρχή του αποτελέσματος το ψηφίο 1. Έτσι, η έξοδος της γραμμής 7 είναι Integer.toBinaryString(-3>>1)=11111111111111111111111111111110. Ο ψηφιοχάρτης αυτός αντιστοιχεί στην δεκαδική τιμή -2.

4.1.7.2 Διολίσθηση αριστερά

Ο τελεστής αριστερής διολίσθησης (left shift) είναι δυαδικός, συμβολίζεται ως "<<"και συντάσσεται με ακέραιους τύπους. Η διαφορά του από την διολίσθηση δεξιά είναι πως τα bits χάνονται από αριστερά και προστίθενται δεξιά. Επιπλέον, τα bits που προστίθενται έχουν την τιμή μηδέν ανεξάρτητα από το πρόσημο του πρώτου τελεσταίου. Κατά τα λοιπά, ισχύουν κατ' αναλογία όσα ισχύουν και για τον τελεστή δεξιάς διολίσθησης.

```
    static void leftShift() { //Κώδικας 4.15
        int i = 3;
        int j = -3;
        System.out.println("Integer.toBinaryString(" + i + ")=       " +
Integer.toBinaryString(i));
        System.out.println("Integer.toBinaryString(" + i + "<<1)=" +
Integer.toBinaryString(i << 1));
        System.out.println("Integer.toBinaryString(" + j + ")=" +
Integer.toBinaryString(j));
        System.out.println("Integer.toBinaryString(" + j + "<<1)=" +
Integer.toBinaryString(j << 1));
        System.out.println("value of j<<1=" + (j << 1));
    }
```

***Κώδικας 4.15**: Τελεστής αριστερής διολίσθησης*

Η έξοδος του κώδικα 4.15 είναι η ακόλουθη

```
Integer.toBinaryString(3)=11
Integer.toBinaryString(3<<1)= 110
Integer.toBinaryString(-3)=11111111111111111111111111111101
Integer.toBinaryString(-3<<1)= 11111111111111111111111111111010
value of j<<1=-6
```

Προσέξτε τον ψηφιοχάρτη του j, δηλ. του -3. Μετά την διολίσθηση έχει χαθεί ένα ψηφίο με τιμή 1 ενώ στο τέλος του ψηφιοχάρτη έχει προστεθεί ένα ψηφίο με τιμή 0. Η τιμή του j μετά την διολίσθηση έχει γίνει -6.

4.1.7.3 Μη προσημασμένη διολίσθηση δεξιά

Ο τελεστής μη προσημασμένης δεξιάς διολίσθησης (unsigned right shift), συμβολίζεται με >>> και διαφέρει από τον τελεστή δεξιάς διολίσθησης κατά το ότι τα ψηφία που αντικαθιστούν όσα χάνονται δεν εξαρτώνται από το πρόσημο του πρώτου τελεσταίου αλλά έχουν πάντα την τιμή 0.

Ας σημειωθεί πως στην Java δεν υπάρχει τελεστής μη προσημασμένης αριστερής διολίσθησης καθώς αυτός θα ήταν πανομοιότυπος με την αριστερή διολίσθηση.

4.1.8 Σύνθετη εκχώρηση

Υποστηρίζεται μια σειρά δυαδικών τελεστών σύνθετης εκχώρησης (compound assignment), δηλ. τελεστών που τελεσταίου συνδυάζουν την εκχώρηση με μία άλλη πράξη. Στον πίνακα 4.5, παρουσιάζονται οι τελεστές σύνθετης εκχώρησης μαζί με την ισοδύναμη έκφραση που αντιστοιχεί στον κάθε ένα.

Τελεστής	Παράδειγμα Χρήσης	Ισοδύναμη έκφραση
+=	i+=5	i=i+5
-=	i-=k	i=i-k
*=	i*=k	i=i*k
/=	i/=k	i=i/k
%=	i%=k	i=i%k
&=	i&=k	i=i&k
|=	i|=k	i=i|k
<<=	i<<=k	i=i<<k
>>=	i>>=k	i=i>>k
>>>=	i>>>=k	i=i>>>k

Πίνακας 4.5 *Τελεστές Σύνθετης Εκχώρησης*

4.2. Βασική Διαμόρφωση εξόδου

Συχνά χρειάζεται να διαμορφώσουμε τα δεδομένα πριν τα εμφανίσουμε. Για παράδειγμα, σε μια εφαρμογή μπορεί να επιθυμούμε να εμφανίσουμε τους πραγματικούς αριθμούς με δύο δεκαδικά ψηφία. Σε αυτήν την ενότητα παρουσιάζονται δύο εναλλακτικοί τρόποι για την διαμόρφωση της εξόδου.

4.2.1 Η printf

Η printf αποτελεί τον παραδοσιακό τρόπο διαμόρφωσης της εξόδου με την έννοια ότι υποστηρίζεται και στην C [2]. Ας ξεκινήσουμε την παρουσίαση με ένα παράδειγμα

```
1   static void stringF() { ////Κώδικας 4.16
2       System.out.printf("%s%n", "Hello");
3       System.out.printf("%-10s%10S%n", "Hello", "Hello");
4       System.out.printf("%10.3S%n", "Hello");
5   }
```

Κώδικας 4.16*: Διαμόρφωση εξόδου του τύπου String*

Η πρώτη παράμετρος σε όλες τις κλήσεις της printf στον κώδικα 4.16 ονομάζεται προσδιοριστής διαμόρφωσης (format specifier). Ο προσδιοριστής διαμόρφωσης περιλαμβάνει μια σειρά από κανόνες διαμόρφωσης (formatting rules). Κάθε κανόνας σηματοδοτεί την αρχή του με τον χαρακτήρα %. Στην γραμμή 2, ο προσδιοριστής διαμόρφωσης έχει την τιμή %s%n. Επομένως, περιλαμβάνει δύο κανόνες. Ο πρώτος κανόνας εκφράζεται με τον χαρακτήρα s και μας λέει πως η διαμόρφωση αφορά αλφαριθμητική σειρά. Ο χαρακτήρας s σε αυτό το πλαίσιο ονομάζεται χαρακτήρας μετατροπής (conversion character). Άλλοι χαρακτήρες μετατροπής είναι ο d για διαμόρφωση ακεραίων, ο f για διαμόρφωση πραγματικών αριθμών, ο c για διαμόρφωση χαρακτήρων και ο b για διαμόρφωση τιμών boolean. Ωστόσο, οι χαρακτήρες μετατροπής c και b χρησιμοποιούνται σπάνια. Ο δεύτερος κανόνας περιλαμβάνει τον χαρακτήρα n που προκαλεί αλλαγή γραμμής.

Η δεύτερη παράμετρος στην γραμμή 2 είναι το String που επιθυμούμε να τυπώσουμε. Καθότι ο κανόνας διαμόρφωσης στην πρώτη γραμμή δεν καθορίζει τίποτα περισσότερο παρά μόνο ότι θα εμφανίσουμε μια αλφαριθμητική σειρά και θα προβούμε σε αλλαγή γραμμής, εκείνο που θα δούμε ως έξοδο είναι απλώς η σειρά Hello.

Στην γραμμή 3, μετά τον προσδιοριστή διαμόρφωσης έχουμε μια σειρά από String. Είναι γενικό χαρακτηριστικό της printf ότι μετά τον προσδιοριστή διαμόρφωσης μπορεί να δέχεται πολλές παραμέτρους. Για περισσότερες πληροφορίες επάνω σε αυτό, δείτε την ενότητα 7.3.3. Κάθε μία από αυτές τις παραμέτρους προορίζεται για να εμφανισθεί. Την διαμόρφωσή τους ρυθμίζει ο προσδιοριστής διαμόρφωσης, δηλ. η πρώτη παράμετρος. Στην γραμμή 3 λοιπόν έχουμε αυτόν τον προσδιοριστή προσπέλασης %-10s%10S%n. Περιλαμβάνει τρεις κανόνες. Ο τρίτος όπως εξηγήθηκε παραπάνω προκαλεί αλλαγή γραμμής της εξόδου. Ο πρώτος αφορά την δεύτερη παράμετρο της printf και ο δεύτερος την τρίτη. Οι δύο πρώτοι κανόνες περιλαμβάνουν το 10. Αυτό προσδιορίζει το πλάτος στο οποίο θα εμφανιστεί η αντίστοιχη παράμετρος. Τα

δεδομένα στοιχίζονται σε αυτό το πλάτος στην δεξιά πλευρά, δηλ. το Hello σε πλάτος 10 θα εμφανισθεί ως ␣ ␣␣␣␣Hello, όπου ␣ συμβολίζει ένα διάστημα (space). Αν ωστόσο επιθυμούμε τα δεδομένα μας να στοιχηθούν στα αριστερά του πλάτους τότε τοποθετούμε τον χαρακτήρα – (μείον) όπως έχουμε κάνει στον πρώτο κανόνα. Σημειώστε πως σε αυτό το πλαίσιο ο χαρακτήρας μείον ονομάζεται σηματοφόρος (flag). Έτσι το πρώτο Hello θα εμφανισθεί ως Hello␣␣␣␣␣. Αν επιθυμούμε η σειρά να εμφανισθεί με κεφαλαία, τότε ως χαρακτήρα διαμόρφωσης δίνουμε το S αντί του s, όπως κάνουμε στον δεύτερο κανόνα. Στην γραμμή 4, το πλάτος ακολουθείται από την ακρίβεια (precision). Η ακρίβεια δηλώνεται με μια τελεία ακολουθούμενη από έναν ακέραιο.
Η έξοδος του κώδικα 4.16 έχει ως εξής:

```
Hello
Hello          HELLO
       HEL
```

Το γενικό συντακτικό ενός κανόνα διαμόρφωσης έχει ως εξής:

```
%[σηματοφορείς][πλάτος][.ακρίβεια]χαρακτήρας μετατροπής
```

Από αυτά τα στοιχεία, μόνο ο χαρακτήρας % που δηλώνει την αρχή του κανόνα και ο χαρακτήρας μετατροπής που δηλώνει τον τύπο των δεδομένων ή αλλαγή γραμμής είναι υποχρεωτικά. Οι σηματοφορείς, το πλάτος και η ακρίβεια είναι προαιρετικά στοιχεία. Ωστόσο, η σειρά με την οποία τα στοιχεία τοποθετούνται μέσα σε ένα κανόνα είναι υποχρεωτικά αυτή που δείχνει το γενικό συντακτικό του κανόνα.
Ας δούμε και μερικά άλλα χρήσιμα παραδείγματα.

```
1   static void intF() { //Κώδικας 4.17
2       System.out.printf("ακέραιος: %d%n", 1000000L);
3       System.out.printf("ακέραιος: %,d%n", 1000000L);
4       System.out.printf(Locale.ENGLISH,"ακέραιος: %,d%n", 1000000L);
5   }
```

***Κώδικας 4.17**: Διαμόρφωση εξόδου ακεραίων τύπων*

Όπως φαίνεται στον κώδικα 4.17, ένας προσδιοριστής διαμόρφωσης μπορεί να περιλαμβάνει και απλές αλφαριθμητικές σειρές οι οποίες απλώς αποτελούν μέρος της εξόδου. Στην γραμμή 2, δεν δίνουμε κάποια ιδιαίτερη διαμόρφωση, οπότε η έξοδος αυτής της γραμμής είναι απλά 1000000. Στην γραμμή 3 όμως, έχουμε τοποθετήσει τον σηματοφόρο, (κόμμα). Με αυτόν το σηματοφόρο ζητάμε ο αριθμός να εμφανισθεί με διαχωριστικά χιλιάδων, οπότε η έξοδος εδώ θα είναι 1.000.000. Επειδή όμως τα διαχωριστικά χιλιάδων δεν είναι ίδια για όλες τις χώρες, μας δίνεται η δυνατότητα να χρησιμοποιήσουμε μια άλλη έκδοση της printf που μας επιτρέπει να καθορίσουμε ποιο ακριβώς διαχωριστικό θα χρησιμοποιήσουμε. Έτσι, η γραμμή 4, εμφανίζει 1,000,000.

```
1     static void realF() { //Κώδικας 4.18
2       System.out.printf("%f%n", 5.1473);
3       System.out.printf("%(010.2f%n", -5.1473);
4       System.out.printf("%-10.2f%n", 5.1473);
5   }
```

***Κώδικας 4.18**: Διαμόρφωση εξόδου πραγματικών τύπων*

Η έξοδος του κώδικα 4.18 έχει ως εξής:

```
5,147300
(00005,15)
5,15
```

Στην γραμμή 3, ο σηματοφόρος ‘(‘ (αριστερή παρένθεση) έχει ως αποτέλεσμα να εμφανίζονται οι αρνητικοί αριθμοί εντός παρενθέσεων. Το δε 0 που ακολουθεί καλύπτει το πλάτος με μηδενικά στα αριστερά. Ο

σηματοφόρος 0 δεν συντάσσεται με στοίχιση στα αριστερά. Κατά τα λοιπά, με όσα έχουμε συζητήσει μέχρι εδώ είναι εύκολο να ερμηνεύσετε την έξοδο του κώδικα 4.18.

4.2.2 Η DecimalFormat

Ένας εναλλακτικός τρόπος για διαμόρφωση δεκαδικών αριθμών για έξοδο βασίζεται στην κλάση DecimalFormat. Η DecimalFormat παρέχει ευκολίες στην διαμόρφωση των αριθμών για έξοδο, εισάγει ωστόσο κάποιον βαθμό πολυπλοκότητας στον κώδικά μας.

```
  1     static void outputFormat() { //Κώδικας 4.19
  2         DecimalFormat f = new DecimalFormat("###,###.##");
  3         double d1 = 9.566, d2 = 9.273;
  4         System.out.println(f.format(d1) + "+" + f.format(d2) + " = " +
f.format(d1 + d2));
  5
  6         d1 = 9.565;
  7         d2 = 9.275;
  8         System.out.println(f.format(d1) + "+" + f.format(d2) + " = " +
f.format(d1 + d2));
  9
  10         d1 = 0.275;
  11         d2 = 0.385;
  12         System.out.println(f.format(d1) + "+" + f.format(d2) + " = " +
f.format(d1 + d2));
  13
  14         d1 = 9_856_327.563;
  15         d2 = 5627869.289;
  16         System.out.println(f.format(d1) + "+" + f.format(d2) + " = " +
f.format(d1 + d2));
  17
  18         f.applyPattern("###,###.###");
  19         System.out.println(f.format(d1) + "+" + f.format(d2) + " = " +
f.format(d1 + d2));
  20
  21         f.applyPattern("€###,###.###");
  22         System.out.println(f.format(d1));

  23         f.applyPattern("(€###,###.###)");
  24         System.out.println(f.format(d1));
  25         int k = 128;
  26         System.out.println(f.format(k));
  27
  28         char c = 'A';
  29         System.out.println(f.format(c));
  30
  31         f.applyPattern("000,000,000.0000");
  32         System.out.println(f.format(d1));
  33     }
```

***Κώδικας 4.19**: Διαμόρφωση εξόδου*

Η έξοδος της outputFormat είναι η εξής:

```
9,57+9,27 = 18,84
9,56+9,28 = 18,84
0,28+0,39 = 0,66
9.856.327,56+5.627.869,29 = 15.484.196,85
9.856.327,563+5.627.869,289 = 15.484.196,852
€9.856.327,563
(€9.856.327,563)
```

```
(€128)
(€65)
009.856.327,5630
```

Η διαμόρφωση βασίζεται στην προκαθορισμένη κλάση DecimalFormat [2]. Για να την χρησιμοποιήσετε προσθέστε στην κλάση σας την import Java.text.DecimalFormat.

Στην γραμμή 2 του κώδικα 4.19, δημιουργούμε ένα αντικείμενο της κλάσης DecimalFormat. Η μεταβλητή f είναι μεταβλητή αναφοράς. Κατά την δημιουργία της συνδέεται με το μοτίβο (pattern), "###,###.##". Στην γραμμή 4, στέλνουμε προς εμφάνιση στην οθόνη τρείς πραγματικές τιμές, χρησιμοποιώντας την συνάρτηση format της DecimalFormat. Η συνάρτηση format επιστρέφει μια αλφαριθμητική σειρά. Η σειρά που επιστρέφει η format υπολογίζεται με βάση την τιμή που τις δίνουμε κατά την κλήση της και το μοτίβο που ορίσαμε κατά την δημιουργία της f. Στο μοτίβο αυτό ο χαρακτήρας ',' συμβολίζει τον διαχωριστή χιλιάδων, ο χαρακτήρας '.' συμβολίζει τον διαχωριστή δεκαδικών και ο χαρακτήρας '#' συμβολίζει το ψηφίο. Επομένως, σύμφωνα με το μοτίβο, ζητάμε μια διαμόρφωση στην οποία να εμφανίζονται τα ψηφία του αριθμού κατά τρόπο που οι χιλιάδες να διαχωρίζονται μεταξύ τους, επίσης να διαχωρίζεται το δεκαδικό από το ακέραιο μέρος και το δεκαδικό μέρος να εμφανίζεται με δύο ψηφία. Η έξοδος της γραμμής 4 είναι 9.57+9.27 = 18.84. Προσέξτε καταρχάς πως έχουμε στρογγυλοποίηση του δεκαδικού μέρους. Η μεταβλητή d1 εμφανίστηκε ως 9.57 αντί του 9.566. Αντίστοιχα η d2 εμφανίζεται ως 9.27 αντί του 9.273. Η format στρογγυλοποιεί εξορισμού τα δεκαδικά προς την πλησιέστερη τιμή. Το 0.57 είναι πλησιέστερο στο 0.566 σε σχέση με το 0.56. Παρόμοια, το 0.27 είναι πλησιέστερο στο 0.273 από ότι το 0.28.

Όταν όμως δεν υπάρχει πλησιέστερη τιμή, τότε η στρογγυλοποίηση γίνεται προς τον άρτιο γείτονα. Η έξοδος της γραμμής 8 είναι 9.56+9.28 = 18.84. Προσέξτε πως η τιμή του d1 από 9.565 μετατράπηκε σε 9.56. Πράγματι, η τιμή 9.565 απέχει εξίσου από το 9.56 και το 9.57. Σε αυτήν την περίπτωση, η format στρογγυλοποίησε προς τον άρτιο γείτονα, δηλ. το 6 και όχι τον περιττό, δηλ. το 7. Αντίστοιχα, η d2 με τιμή 9.275, εμφανίζεται ως 9.28. Επομένως, η στρογγυλοποίηση έγινε προς τον άρτιο γείτονα, το 8, και όχι τον περιττό, δηλ. το 7.

Η έξοδος της γραμμής 12, μετά την μεταβολή των τιμών των d1 και d2 που έγινε στην γραμμή 10 και 11, είναι 0.28+0.39 = 0.66. Επομένως, το d1 με τιμή 0.275 εμφανίζεται ως 0.28 και το d2 με τιμή 0.385 εμφανίζεται ως 0.39. Εκείνο όμως που χρήζει ιδιαίτερης προσοχής είναι πως το άθροισμα 0.28+0.39 ισούται με 0.67 και όχι με 0.66. Η ασυμφωνία αυτή προκύπτει καθώς υπολογίζουμε πρώτα το άθροισμα d1+d2 και στην συνέχεια το στρογγυλοποιούμε. Επομένως, 0.275+0.385=0.610. Η στρογγυλοποίηση στα δύο δεκαδικά του 0.610 είναι προφανώς το 0.61. Είναι κατανοητό από αυτό το παράδειγμα πως η στρογγυλοποίηση κατά την εμφάνιση πραγματικών αριθμών πρέπει να γίνεται με ιδιαίτερη προσοχή.

Στην γραμμή 16 το μοτίβο "###,###.##" εφαρμόζεται σε αριθμούς με περισσότερα ψηφία από τους χαρακτήρες του χωρίς να δημιουργείται κάποιο πρόβλημα.

Στην γραμμή 18 αλλάζουμε το μοτίβο της f. Το νέο μοτίβο υποστηρίζει τρία δεκαδικά ψηφία. Έτσι η έξοδος της γραμμής 19 είναι 9,856,327.563+5,627,869.289 = 15,484,196.852.

Είναι εφικτό κατά την διαμόρφωση των αριθμών να προσθέτουμε σταθερούς χαρακτήρες. Έτσι στην γραμμή 21 έχουμε προσθέσει το σύμβολο του Ευρώ ώστε να προηγείται της αριθμητικής τιμής. Πράγματι η έξοδος της γραμμής 22 είναι €9,856,327.563.

Στην γραμμή 23, τροποποιούμε πάλι το μοτίβο της f ώστε να περικλείεται ο αριθμός και το σύμβολο του Ευρώ από παρενθέσεις. Έτσι, η έξοδος της γραμμής 24 είναι (€9,856,327.563).

Στην γραμμή 26 επιδεικνύεται πως το μοτίβο μπορεί να εφαρμοστεί και σε ακέραιους. Η έξοδος της γραμμής είναι (€128). Παρότι το μοτίβο προβλέπει δεκαδικά ψηφία, η format τυπώνει σωστά τους ακέραιους τύπους χωρίς δεκαδικά.

Στην γραμμή 29 εκτυπώνεται ο χαρακτήρας 'A' μέσω της format. Η έξοδος της γραμμής είναι (€65), που σημαίνει πως στην θέση του χαρακτήρα εμφανίζεται ο κωδικός του.

Σε κάποιες εφαρμογές, θέλουμε να εμφανίζουμε αριθμούς σε σταθερό πλάτος συμπληρώνοντας τα στοιχεία που λείπουν με μηδενικά. Σε αυτήν την περίπτωση, προκειμένου να μην αλλοιωθεί η τιμή του αριθμού, μηδενικά μπορούν να προστεθούν μόνο στην αρχή του ακέραιου μέρους ή στο τέλος του δεκαδικού μέρους ενός αριθμού. Στην γραμμή 31 μεταβάλλουμε το μοτίβο, έτσι ώστε ο αριθμός να τοποθετείται σε πλάτος 13 ψηφίων από τα οποία τα 9 ανήκουν στο ακέραιο μέρος και τα 4 στο δεκαδικό. Η έξοδος της γραμμής 32 είναι 009,856,327.5630.

Ας σημειωθεί πως η κλάση DecimalFormat παρέχει πολλές επιπλέον δυνατότητες. Για παράδειγμα, την δυνατότητα να καθορίσουμε ως διαχωριστικό χιλιάδων τον χαρακτήρα '.' και ως διαχωριστικό δεκαδικών τον

χαρακτήρα ‘,’ ή να αλλάξουμε τον τρόπο στρογγυλοποίησης. Περισσότερες πληροφορίες για την DecimalFormat μπορείτε να βρείτε στην τεκμηρίωση της Oracle [1].

Επιπλέον, υπάρχουν και άλλοι τρόποι για την διαμόρφωση της εξόδου όπως η συνάρτηση format της κλάσης String [3].

4.3 Είσοδος από το πληκτρολόγιο

Για είσοδο από το πληκτρολόγιο θα χρησιμοποιούμε την προκαθορισμένη κλάση Scanner. Για να χρησιμοποιηθεί πρέπει να την εισάγετε στον κώδικά σας προσθέτοντας την import Java.util.Scanner. Με την Scanner μπορούμε να διαβάσουμε δεδομένα και από αρχεία. Ωστόσο σε αυτήν την ενότητα, περιοριζόμαστε στην χρήση της Scanner για απλή είσοδο από το πληκτρολόγιο. Η ανάγνωση αρχείων παρουσιάζεται αναλυτικά στην ενότητα 15.

Η Scanner για κάθε θεμελιώδη τύπο διαθέτει μια συνάρτηση με την οποία μπορούμε να διαβάζουμε αντίστοιχα δεδομένα. Ο πίνακας 4.6 παρουσιάζει τις συναρτήσεις που μας ενδιαφέρουν ώστε να μπορούμε να υποστηρίζουμε στα προγράμματά μας είσοδο από το πληκτρολόγιο.

Τύπος επιστροφής	**Συνάρτηση**	**Χρήση**
String	next()	Είσοδος δεδομένων τύπου String
boolean	nextBoolean()	Είσοδος δεδομένων τύπου boolean
byte	nextByte()	Είσοδος δεδομένων τύπου byte
double	nextDouble()	Είσοδος δεδομένων τύπου double
float	nextFloat()	Είσοδος δεδομένων τύπου float
int	nextInt()	Είσοδος δεδομένων τύπου int
String	nextLine()	Είσοδος δεδομένων τύπου String
long	nextLong()	Είσοδος δεδομένων τύπου long
short	nextShort()	Είσοδος δεδομένων τύπου short

Πίνακας 4.6 *Συναρτήσεις της Scanner για είσοδο δεδομένων*

Σε κάθε κλήση τους, κάθε μια από τις συναρτήσεις του πίνακα 4.6, διαβάζει μια ολοκληρωμένη σειρά δεδομένων (token). Εξορισμού, μια σειρά δεδομένων θεωρείται ότι τερματίζεται από την παρουσία ενός διαστήματος (whitespace). Εξαίρεση αποτελεί η συνάρτηση nextLine που διαβάζει τα δεδομένα μιας γραμμής, θεωρώντας το διάστημα ως μέρος των δεδομένων.

Στον κώδικα 4.20, επιδεικνύεται ενδεικτική χρήση των συναρτήσεων εισόδου. Στην γραμμή 2, ορίζουμε το αντικείμενο in της κλάσης Scanner μέσα από το οποίο έχουμε την δυνατότητα να καλούμε τις συναρτήσεις εισόδου. Κατά τον ορισμό του in, το συνδέουμε με την System.in. Με αυτόν τον τρόπο καθορίζεται πως η είσοδος θα γίνεται από το πληκτρολόγιο.

```
1     static void keyboardInput() { //Κώδικας 4.20
2        Scanner in = new Scanner(System.in);
3        System.out.print("Δώστε όνομα : ");
4        String name = in.nextLine();
5
6        System.out.print("Δώστε αριθμό Μητρώου: ");
7        int aem = in.nextInt();
8
9        System.out.print("Εισόδημα έτους 2021 : ");
10       double eisodima = in.nextDouble();
11       System.out.println();
12
13       System.out.println("Όνομα: " + name);
14       System.out.println("ΑΜ : " + aem);
15       System.out.println("Εισόδημα 2021: " + eisodima);
16      // in.close();
17    }
```

***Κώδικας 4.20**: Είσοδος από το πληκτρολόγιο*

Στην γραμμή 4, διαβάζουμε μια γραμμή δεδομένων. Θα μπορούσαμε να χρησιμοποιήσουμε την συνάρτηση next() μια και το όνομα είναι τύπου String. Ωστόσο, αν ο χρήστης καταχωρίσει όνομα και επώνυμο διαχωρισμένα με διάστημα, η next() θα διαβάσει μόνο μέχρι το διάστημα, δηλ. μόνο το όνομα. Ακόμη χειρότερα, η επόμενη είσοδος που επιχειρείται με την nextInt στην γραμμή 7, θα βρει ως δεδομένα εισόδου, το επώνυμο του χρήστη. Θα βρει δηλ. ένα String ενώ περιμένει int οπότε θα παραχθεί λάθος χρόνου εκτέλεσης.

Εφόσον διαβάσουμε σωστά το ονοματεπώνυμο του χρήστη, προχωράμε στην είσοδο του αριθμού μητρώου. Αν εδώ ο χρήστης δεν δώσει ακέραιη τιμή, επίσης θα παραχθεί λάθος χρόνου εκτέλεσης. Στην συνέχεια, στην γραμμή 10, διαβάζουμε έναν αριθμό τύπου double.

Το παράδειγμα κλείνει με την in.close() σχολιασμένη. Προσοχή, μην κλείσετε ποτέ ένα αντικείμενο Scanner που είναι σχετισμένο με την System.in. Αν το κάνετε, η εφαρμογή σας δεν θα είναι σε θέση να ανοίξει ροή επικοινωνίας με το πληκτρολόγιο παρά μόνο αν επανεκκινηθεί [3].

Βέβαια, η σωστή χρήση της Scanner απαιτεί την διαχείριση των λαθών που ενδέχεται να παραχθούν, ζήτημα με το οποίο ασχολούμαστε στην ενότητα 16.

4.4 Ασκήσεις

1. Ποια είναι η έξοδος του κώδικα που ακολουθεί;

```
int i = 0, j = 0;
System.out.print(i++ + " " + (++j) + " ");
System.out.println(i == j);
```

2. Ποια είναι η έξοδος του κώδικα που ακολουθεί; Εξηγήστε την απάντησή σας.

```
int k = 2;
System.out.println((k += 2) == (k *= 2));
```

3. Ποια είναι η έξοδος του κώδικα:

```
int i,j,k;
i = j = k = 0;
System.out.println((i++ == 0 || j++ == 0) && k++ == 0);
System.out.println(i + " " + j + " " + k);
```

4. Ποια είναι η έξοδος του κώδικα

```
int i,j,k;
i = j = k = 0;
System.out.println(i++ == 0 || j++ == 0 && k++ == 0);
System.out.println("i++ == 0 || j++ == 0 && k++ == 0");
System.out.println(i + " " + j + " " + k);
```

5. Ποια είναι η έξοδος του κώδικα

```
int i,j,k;
i = j = k = 0;
System.out.println(i++ == 0 && j++ == 0 || k++ == 0);
System.out.println("i++ == 0 && j++ == 0 || k++ == 0");
System.out.println(i + " " + j + " " + k);
```

6. Λάβετε από το πληκτρολόγια 2 πραγματικούς αριθμούς και εμφανίστε στην οθόνη το γινόμενό τους. Εμφανίστε τα αποτελέσματα κατά τρόπο ώστε να προηγείται η περιγραφή τους και με 2 δεκαδικά ψηφία κατά μέγιστο.

```
Παράδειγμα
Δώσε αριθμό: 3.58
Δώσε αριθμό: 2.99
Το γινόμενο 3.58 x 2.99 = 10.7
```

7. Χρησιμοποιήστε μια μόνο println για να τυπώστε την διεύθυνσή σας με την ακόλουθη μορφή

```
Ονοματεπώνυμο
Οδός Αριθμός
Πόλη, ΤΚ
Χώρα
```

Υπόδειξη: Θυμηθείτε τους χαρακτήρες διαφυγής στην ενότητα 3.3.

8. Ο Γιώργος λαμβάνει μηνιαίο μισθό 1000 Ευρώ. Επίσης, λαμβάνει ενοίκιο από το διαμέρισμα στην Καλλιθέα 450 τον μήνα και το διπλάσιο ενοίκιο από το διαμέρισμα στου Παπάγου. Ο Γιώργος έχει ένα επιπλέον μηνιαίο εισόδημα. Τα έξοδα του μηνός Απριλίου έχουν ως εξής: Για τρόφιμα ξόδεψε τα 12/100 του συνολικού του εισοδήματος, τα καύσιμα κοστίζουν 0.2 Ευρώ το χιλιόμετρο, το Ηλεκτρικό ρεύμα 200 Ευρώ και το κόστος νερού 120 Ευρώ. Να αναπτυχθεί αλγόριθμος που ζητά από τον Γιώργο το επιπλέον μηνιαίο εισόδημα και τα χιλιόμετρα του Απριλίου και εμφανίζει κατάσταση εσόδων-εξόδων του Απριλίου σύμφωνα με το παρακάτω υπόδειγμα.

```
--------Κατάσταση Εσόδων-Εξόδων Απριλίου-----------------
Μισθός 1000
Ενοίκιο Καλλιθέας 450
Ενοίκιο Παπάγου 900
Επιπλέον μηνιαίο εισόδημα 100
-----------------------------------------------------------
Συνολικό Εισόδημα 2450

Κόστος Τροφίμων 294.00
Κόστος Καυσίμων 160.00
Κόστος Ηλεκτρικού Ρεύματος 160.00
Κόστος Νερού 120
-----------------------------------------------------------
Συνολικό Κόστος 774.00
***********************************************************
Διαφορά 1676.00
```

9. Ο Γιάννης είναι καπνιστής. Κάθε μέρα καταναλώνει ένα πακέτο τσιγάρα που κοστίζει 4 Ευρώ +20% ΦΠΑ. Επίσης, κάθε δύο μήνες χρειάζεται ένα αναπτήρα. Ρωτήστε τον χρήστη πόσο κοστίζει ένας αναπτήρας και υπολογίστε πόσο κοστίζει στον Γιάννη το κάπνισμα κάθε ημέρα και πόσο κάθε μήνα. Ποιο είναι το ετήσιο κόστος; Θεωρήστε πως όλοι οι μήνες αποτελούνται από 30 μέρες. Για να υπολογίσετε το ημερήσιο κόστος επιμερίστε το κόστος του αναπτήρα στις ημέρες χρήσης του. Αν το ετήσιο εισόδημα του Γιάννη είναι 8.000 Ευρώ, πόσο τοις εκατό του εισοδήματός του ξοδεύει για κάπνισμα. Αν τα υπόλοιπα μηνιαία έξοδα του Γιάννη είναι 500 Ευρώ, τι ποσό από το εισόδημά του αποταμιεύει κάθε χρόνο.

Βιβλιογραφία

[1] “Operators (The Java™ Tutorials > Learning the Java Language > Language Basics).” https://docs.oracle.com/javase/tutorial/java/nutsandbolts/operators.html (accessed Sep. 14, 2021).

[2] “Formatting Numeric Print Output (The Java™ Tutorials > Learning the Java Language > Numbers and Strings).” https://docs.oracle.com/javase/tutorial/java/data/numberformat.html (accessed Sep. 14, 2021).

[3] K. Fogel, “If You’re Using Java’s Scanner Class for Keyboard Input, You’re Doing it Wrong - DZone Java,” DZone. https://dzone.com/articles/if-youre-using-javas-scanner-class-for-keyboard-in (accessed Sep. 14, 2021).

Κεφάλαιο 5

Σύνοψη
Στην ενότητα αυτή παρουσιάζονται οι δομές επιλογής και οι επαναληπτικές δομές της Java. Πιο συγκεκριμένα, παρουσιάζονται οι δομές επιλογής, if, if…else, switch και ο τριαδικός τελεστής. Επίσης, παρουσιάζονται οι επαναληπτικές δομές for, while, do-while και οι λέξεις-κλειδιά break και continue. Περιλαμβάνονται λυμένες ασκήσεις και ασκήσεις προς λύση.

Προαπαιτούμενη γνώση
Οι θεμελιώδεις τύποι και βασική γνώση του τύπου String. Βασική είσοδος και έξοδος. Η λειτουργία των τελεστών.

Λέξεις κλειδιά
Δομή επιλογής, επαναληπτική διαδικασία, βρόχος, τριαδικός τελεστής

5 Έλεγχος Ροής

Ο έλεγχος της ροής του προγράμματος πραγματοποιείται με τις δομές επιλογής και τις επαναληπτικές δομές. Οι δομές επιλογής μας δίνουν την δυνατότητα να δημιουργούμε πολλαπλές διαδρομές μέσα στον κώδικά μας και να επιλέγουμε κάθε φορά την καταλληλότερη ανάλογα με τις συνθήκες που επικρατούν κατά την εκτέλεση του προγράμματος. Με τις επαναληπτικές δομές ρυθμίζουμε την επανάληψη τμημάτων του κώδικά μας, επίσης, ανάλογα με συγκεκριμένες συνθήκες που επικρατούν κατά την εκτέλεση του προγράμματος. Τόσο οι δομές επιλογής όσο και οι επαναληπτικές δομές αποτελούν βασικές δομές προγραμματισμού όχι μόνο στην Java αλλά σε κάθε προγραμματιστικό περιβάλλον.
Επιπλέον, τόσο οι δομές επιλογής όσο και οι επαναληπτικές δομές βασίζονται στον έλεγχο μιας συνθήκης (condition). Η συνθήκη είναι μία λογική έκφραση, δηλ. μία έκφραση τύπου boolean. Για παράδειγμα, η έκφραση i>5 && j<0 είναι λογική έκφραση καθώς λαμβάνει την τιμή true ή false ανάλογα με την τιμή των i και j.

5.1 Δομές επιλογής

Οι δομές επιλογής μας δίνουν την δυνατότητα να καθορίζουμε διαφορετικές διαδρομές που ακολουθεί ο κώδικάς μας ανάλογα με τις συνθήκες που επικρατούν στο πρόγραμμα μας κατά τον χρόνο εκτέλεσης. Πρόκειται για θεμελιώδεις δομές που συναντώνται με την μία ή την άλλη μορφή σε κάθε γλώσσα προγραμματισμού. Για να αντιληφθεί κανείς την σπουδαιότητά τους αρκεί να αναλογιστεί πως χωρίς αυτές τα προγράμματα θα ήταν μια συλλογή εντολών που θα εκτελούνταν αναγκαστικά σειριακά. Με άλλα λόγια η λήψη αποφάσεων κατά την εκτέλεση του προγράμματος θα ήταν αδύνατη. Για παράδειγμα, αν ένα πρόγραμμα διέθετε ως δεδομένα μια σειρά από ονόματα και κάθε όνομα συνδεόταν με χαρακτηρισμό φύλλου, δεν θα ήταν σε θέση να προσφωνήσει τους άντρες ως κυρίους και τις γυναίκες ως κυρίες.

Η Java υποστηρίζει τις δομές επιλογής if, if…else, switch και τον τριαδικό τελεστή. Είναι όμως εφικτό να συνδυαστούν οι δομές if και if…else διαμορφώνοντας έτσι μια ιδιαίτερη δομή γνωστή ως εμφωλευμένη (nested) if. Στην συνέχεια, παρουσιάζονται μια προς μια οι δομές επιλογής της Java.

5.1.1 Η επιλογή if

Η γενική μορφή της if δίνεται ακολούθως:

```
if (condition) {
     statement(s);
}
```

Η δεσμευμένη λέξη if ακολουθείται από την συνθήκη ελέγχου μέσα σε παρενθέσεις. Αν η συνθήκη είναι αληθής, η ροή του προγράμματος μπαίνει στο μπλοκ που ακολουθεί και εκτελεί τα statements. Αν η συνθήκη είναι ψευδής, η ροή του προγράμματος μεταπηδά στην έξοδο από το μπλοκ της if.

Μέσα στο μπλοκ της if, μπορούμε να έχουμε ένα ή περισσότερα statements. Αν έχουμε μόνο ένα, οι αγκύλες είναι προαιρετικές, ωστόσο, συνηθίζεται να τοποθετούνται.

```
static void simpleIf() { //Κώδικας 5.1
    int i = 1, j = -1;
    if (i > 0 && j < 1) {
        System.out.println("i μεγαλύτερο από το 0");
        System.out.println("j μικρότερο από το 1");
    }
    System.out.println("Μετά τον έλεγχο της έκφρασης \"i>0 && j<1\"");
}
```

Κώδικας 5.1 *Παράδειγμα απλής if*

Στον κώδικα 5.1, η τιμή της συνθήκης ελέγχου εξαρτάται από τις τιμές των i και j. Αν για παράδειγμα, i==1 και j==-1, η συνθήκη είναι αληθής οπότε οι προτάσεις "i μεγαλύτερο από το 0" και "j μικρότερο από το 1" θα εμφανισθούν. Αν όμως το i είναι μικρότερο ή ίσο με το 0 ή το j είναι μεγαλύτερο ή ίσο με το 1, η συνθήκη είναι ψευδής οπότε αυτές οι προτάσεις δεν θα εμφανιστούν. Σε κάθε περίπτωση, θα τυπωθεί η πρόταση "Μετά τον έλεγχο της έκφρασης "i>0 && j<1"".

```
static void IfWithSimpleStatement() { //Κώδικας 5.2
    int i = 0;
    if (i > 0) {
        System.out.println("Το i είναι μεγαλύτερο από το 0");
    }
    System.out.println("Μετά τον έλεγχο της έκφρασης \"i>0\"");
}
```

Κώδικας 5.2 *Παράδειγμα απλής if με ένα statement*

Ο κώδικας 5.2 παρουσιάζει μια if που εφόσον η συνθήκη της αληθεύει εκτελεί μια μόνο πρόταση. Σε αυτήν την περίπτωση, δεν είναι υποχρεωτικό να χρησιμοποιηθούν αγκύλες. Ωστόσο, στην πράξη, συχνά τοποθετούνται αγκύλες από πολλούς επαγγελματίες αλλά και από πολλά IDE, ακόμη και στην περίπτωση μίας πρότασης.
Η πρόταση που ακολουθεί μια if, μπορεί να είναι και η ίδια μια if. Έτσι προκύπτει μια δομή γνωστή ως εμφωλευμένη if (nested if).

```
static void nestedIf() { //Κώδικας 5.3
    int i = 1, j = -1;
    if (i > 0) {
        if (j < 1) {
            System.out.println("i>0 και j<1");
        }
    }
}
```

Κώδικας 5.3 *Παράδειγμα εμφωλευμένης if*

Ο κώδικας 5.3 ελέγχει καταρχάς αν το i είναι μεγαλύτερο του μηδενός. Αν η συνθήκη αληθεύει, προχωρά στην εκτέλεση της επόμενης πρότασης που είναι επίσης μια if. Η δεύτερη if ελέγχει αν το j είναι μικρότερο από το 1. Εφόσον η συνθήκη αυτή αληθεύει, τότε τυπώνει την σειρά χαρακτήρων "i>0 και j<1".

5.1.2 Η επιλογή if…else

Η γενική μορφή της if…else έχει ως εξής:

```
if (condition) {
    statement(s);
}
else {
```

```
            statement(s);
        }
```

Στην περίπτωση που η συνθήκη αληθεύει εκτελούνται οι προτάσεις μέσα στο μπλοκ της if. Αν η συνθήκη είναι ψευδής, εκτελούνται οι προτάσεις μέσα στο μπλοκ της else. Τόσο οι προτάσεις μέσα στο μπλοκ της if όσο και οι προτάσεις στο μπλοκ της else μπορεί να αρχίζουν επίσης με μια if.

```
static void ifElseDemo() { //Κώδικας 5.4
    int i = 1, j = 2;
    if (i > 0) {
        if (j < 2 && j > 0) {
            System.out.println("j ίσο με 1");
        } else if (j < 2) {
            System.out.println("j<=0");
        }
        System.out.println("i>0");
    } else {
        System.out.println("i<=0");
    }
}
```

Κώδικας 5.4 *Παράδειγμα if…else*

Στον κώδικα 5.4 ελέγχεται καταρχάς η τιμή του i. Αν αυτή είναι θετική, η ροή συνεχίζει στο μπλοκ της εξωτερικής if, διαφορετικά, η ροή διακλαδώνεται στον όρο else στο τέλος του κώδικα όπου τυπώνεται η σειρά "i<=0". Στην περίπτωση που η ροή εισέλθει στο μπλοκ της εξωτερικής if, ελέγχεται το j. Εφόσον η μεταβλητή j είναι ακέραιου τύπου και το j είναι μικρότερο του 2 και μεγαλύτερο του 0, δεν μπορεί παρά να είναι 1, οπότε τυπώνεται η σειρά "j ίσο με 1". Αν όμως η συνθήκη j < 2 && j>0 είναι ψευδής, δηλ. το j δεν είναι 1, τότε ελέγχεται αν το j είναι μικρότερο του 2. Εφόσον το j είναι μικρότερο του 2 και δεν είναι 1, σημαίνει πως είναι μικρότερο ή ίσο του 0, οπότε τυπώνεται η σειρά "j<=0". Σε κάθε περίπτωση, μέσα στο μπλοκ της εξωτερικής if εκτυπώνεται η σειρά "i>0" ενώ στο όρο else της εξωτερικής if τυπώνεται η σειρά "i<=0"

5.1.3 Η switch

Σε μερικές περιπτώσεις, ο έλεγχος της ροής του προγράμματος καθορίζεται από πολλαπλές συνθήκες. Όλες αυτές οι περιπτώσεις μπορούν να κωδικοποιηθούν με χρήση εμφωλευμένων if. Ωστόσο, ανάλογα με το πλήθος των συνθηκών, η χρήση εμφωλευμένων if αυξάνει την πολυπλοκότητα και καθιστά τον κώδικα δυσανάγνωστο. Σε κάποιες από αυτές τις περιπτώσεις, ο κώδικας μπορεί να απλοποιηθεί με χρήση της switch.
Η γενική μορφή της switch έχει ως εξής:

```
switch(expression) {
     case constant1: statement(s); break;
     case constant2: statement(s); break;
     case constant3: statement(s); break;
     …
     …
     default: statement(s);
}
```

Η expression της switch πρέπει να είναι τύπου char, byte, short, int ή String. Οι τιμές ελέγχου στους όρους case πρέπει να είναι σταθερές, δηλ. literals ή final μεταβλητές. Τα statement(s) όπως και στην if αντιπροσωπεύουν ένα ή περισσότερα statements. Ωστόσο εδώ στην περίπτωση που έχουμε περισσότερα από ένα statements δεν είναι υποχρεωτικό να τα εσωκλείσουμε σε αγκύλες. Η break ακολουθεί τα statements μίας case. Ωστόσο, η παρουσία της είναι προαιρετική ανάλογα με το αποτέλεσμα που θέλουμε να πετύχουμε. Πάντως, η break έχει ως αποτέλεσμα την έξοδο από την switch. Η default είναι επίσης προαιρετική.

Πως δουλεύει όμως η switch; Καταρχάς αξιολογείται η expression. Ανάλογα με την τιμή της, ο έλεγχος της ροής κατευθύνεται στην κατάλληλη case. Αν σε καμία case δεν προβλέπεται η τιμή της expression, ο

έλεγχος της ροής κατευθύνεται στην default εφόσον υπάρχει αλλιώς έξω από την switch. Στην περίπτωση, που ταιριάζει η τιμή της expression με την τιμή ελέγχου μιας case, εκτελούνται τόσο τα statements της αντιστοιχιζόμενης case όσο και τα statements των επόμενων case ή και της default έως ότου εκτελεστεί ένα break.

Η default δεν τοποθετείται υποχρεωτικά ως τελευταίος όρος της switch. Αντίθετα μπορεί να τοποθετηθεί ως πρώτος όρος ή οπουδήποτε ανάμεσα στις case. Χρειάζεται όμως προσοχή ως προς το που θα τοποθετήσουμε μια default πρόταση γιατί σε συνδυασμό με τα break, μπορεί να έχουμε εντελώς διαφορετικά αποτελέσματα. Αποσαφηνίζουμε παρακάτω με παραδείγματα χρήσης της switch.

```
static void switchDemo() { //Κώδικας 5.5
    int cV = 2;
    final int con = 3;
    switch (cV) {
        case 1:
            System.out.print("1");
            System.out.println("-case 1");
            break;
        case 2:
            System.out.println("2");
            break;
        case con:
            System.out.println("three");
            break;
        default:
            System.out.println("default");
    }
}
```

Κώδικας 5.5 *Τυπικό παράδειγμα χρήσης της switch*

Έστω στον κώδικα 5.5, η μεταβλητή cV είναι ακέραιου τύπου. Αν η τιμή της είναι 1, η έξοδος του κώδικα θα είναι 1-case 1, δηλ. η τιμή της μεταβλητής αντιστοιχίζεται στην case 1 και επομένως εκτελούνται οι τρείς statements της case 1. Το πρώτο statement τυπώνει 1, το δεύτερο -case 1 ενώ το τρίτο statement είναι ένα break που τερματίζει την switch. Αν η cV έχει την τιμή 2, τότε τυπώνεται το 2. Αν η τιμή της cV είναι 3, τότε τυπώνεται η σειρά "three". Προσέξτε πως αν η μεταβλητή con δεν είχε δηλωθεί ως final, ο μεταγλωττιστής θα παρήγαγε κατάλληλο λάθος.
Τέλος αν η τιμή της cV είναι διαφορετική και από το 1 και από το 2 και από το 3, τότε εκτελείται η default και εκτυπώνεται η σειρά “default”. Σημειώστε πως η default δεν ακολουθείται από break. Πράγματι, η τοποθέτηση ενός break στον τελευταίο όρο μιας switch δεν μεταβάλλει κατά κανένα τρόπο το αποτέλεσμα της switch καθώς μετά τον τελευταίο όρο ακολουθεί έτσι κι αλλιώς έξοδος από την switch.

```
static void switchNoBreak() { //Κώδικας 5.6
    int cV = 1;
    final int con = 3;
    switch (cV) {
        case 1:
            System.out.print("1");
        case 2:
            System.out.print("2");
        case con:
            System.out.print("3");
        default:
            System.out.println("default");
    }

}
```

Κώδικας 5.6 *Παράδειγμα switch χωρίς break*

Αν η τιμή της cV είναι 1, η έξοδος του κώδικα 5.6 είναι 123default. Αν η τιμή της είναι 2, η έξοδος του κώδικα είναι 23default. Αν η τιμή της είναι 3, η έξοδος είναι 3default. Τέλος, αν η τιμή της cV είναι διάφορη και από το 1 και από το 2 και από το 3, η έξοδος είναι default. Επομένως, αν η break απουσιάζει, τότε η switch κατευθύνεται στην κατάλληλη case και από εκτελεί όλα τα statements έως ότου βρει break ή έως ότου τερματίσει η switch.
Όπως αναφέραμε προηγουμένως, η default δεν τοποθετείται υποχρεωτικά ως τελευταίος όρος μιας switch. Ωστόσο, η συμπεριφορά της switch δεν είναι ανεξάρτητη από την θέση της default.

```
static void switchDefault() { //Κώδικας 5.7
    int cV = 1;
    final int con = 3;
    switch (cV) {
        default:
            System.out.print("default");
        case 1:
            System.out.print("1");
        case 2:
            System.out.print("2");
        case con:
            System.out.println("3");
    }
}
```

Κώδικας 5.7 *Παράδειγμα switch με την default ως πρώτο όρο*

Αν η τιμή της cV είναι 1, η έξοδος του κώδικα 5.7 είναι 123. Αν η τιμή της είναι 2, η έξοδος του κώδικα είναι 23. Αν η τιμή της είναι 3, η έξοδος είναι 3. Τέλος, αν η τιμή της cV είναι διάφορη και από το 1 και από το 2 και από το 3, η έξοδος είναι default123.
Ας δούμε και ένα παράδειγμα όπου η expression είναι τύπου String.

```
static void switchString() { //Κώδικας 5.8
    String monthLabel = "Jan";
    String monthAA = null;
    switch (monthLabel) {
        case "Jan":
            monthAA = "first";
            break;
        case "Feb":
            monthAA = "second";
            break;
        case "Mar":
            monthAA = "third";
            break;
        case "Apr":
            monthAA = "fourth";
            break;
        case "May":
            monthAA = "fifth";
            break;
        case "Jun":
            monthAA = "sixth";
            break;
        case "Jul":
            monthAA = "seventh";
            break;
        case "Aug":
            monthAA = "eighth";
            break;
        case "Sep":
            monthAA = "ninth";
```

```
                break;
            case "Oct":
                monthAA = "tenth";
                break;
            case "Nov":
                monthAA = "eleventh";
                break;
            case "Dec":
                monthAA = "twelfth";
        }
        if (monthAA == null) {
            System.out.println("Unkown Month Name: " + monthLabel);
        } else {
            System.out.println(monthLabel + " is the " + monthAA + " month of the
year");
        }
    }
```

Κώδικας 5.8 *Παράδειγμα switch με έκφραση ελέγχου τύπου String*

5.1.4 Ο τριαδικός τελεστής

Ο τριαδικός τελεστής που είδαμε ήδη στην ενότητα 4.1.5.3, αποτελεί μια συνοπτική δομή επιλογής. Παρουσιάζει μεγάλη ομοιότητα με την δομή επιλογής if…else. Για παράδειγμα, ο κώδικας 5.9, επιτυγχάνει δύο φορές το ίδιο αποτέλεσμα, την πρώτη χρησιμοποιεί τον τριαδικό τελεστή και την δεύτερη την δομή if…else.

```
    static void ternaryOp() { //Κώδικας 5.9
        int i = 1, j = 2, max;

        max = i > j ? i : j;

        if (i > j) {
            max = i;
        } else {
            max = j;
        }
    }
```

Κώδικας 5.9 *Παράδειγμα αναλογίας τριαδικού τελεστή και if…else*

5.2 Επαναληπτικές δομές

Οι επαναληπτικές δομές αποτελούν παρόμοια με τις δομές επιλογής θεμελιώδεις δομές κάθε γλώσσας προγραμματισμού. Αναλογιστείτε πως χωρίς αυτές αν επιθυμούσαμε να τυπώσουμε τους ακέραιους από το 1 έως το 100.000, θα έπρεπε να πληκτρολογήσουμε 100.000 εντολές. Στην ουσία, ο αποτελεσματικός προγραμματισμός των Ηλεκτρονικών Υπολογιστών θα ήταν αδύνατος. Στα Αγγλικά, η επαναληπτική δομή αναφέρεται με τον όρο loop που αποδίδεται στα Ελληνικά ως βρόχος. Οι επαναληπτικές δομές που υποστηρίζει η Java είναι η while, η do-while και η for.

5.2.1 Η επαναληπτική δομή while

Η γενική μορφή της επαναληπτικής δομής while έχει ως εξής:

```
    while(condition) {
        statement(s);
    }
```

Η while εξετάζει την συνθήκη condition και όσο αυτή είναι αληθής εκτελεί μια ή περισσότερες statements. Παρόμοια με την if, αν έχουμε μια μόνο statement, οι αγκύλες είναι προαιρετικές.

```
static void whileDemo() { //Κώδικας 5.10
    int i = 10;
    while (i > 0) {
        System.out.print("i>0 ");
        i--;
    }
    System.out.println("i=" + i);
}
```

Κώδικας 5.10 *Παράδειγμα απλής while*

Στον κώδικα 5.10, η ακέραιη μεταβλητή i αρχικοποιείται στην τιμή 10. Στην συνέχεια η while εξετάζει αν το i είναι μεγαλύτερο από το 0. Μέσα στο μπλοκ της while, τυπώνεται το μήνυμα "i>0" και ακολούθως μειώνεται η τιμή του i κατά 1. Η διαδικασία επαναλαμβάνεται έως ότου το i πάψει να είναι μεγαλύτερο από το 0. Δεδομένου ότι το i αρχικοποιήθηκε στο 10 και η τιμή του μειώνεται κατά 1 σε κάθε επαναληπτικό βήμα, το i θα γίνει 0 ακριβώς μετά από 10 επαναλήψεις. Επομένως, ο κώδικας 5.10, τυπώνει 10 φορές την σειρά "i>0", στην συνέχεια το i γίνεται 0, η συνθήκη i>0 καθίσταται ψευδής και ο κώδικας εξέρχεται από τον βρόχο while οπότε εκτελεί την επόμενη πρόταση και τυπώνει i=0.

Προσέξτε πως αν στον κώδικα 5.10, η μεταβλητή i είχε αρχικοποιηθεί σε τιμή μικρότερη ή ίση με το 0, η συνθήκη ελέγχου θα ήταν εξαρχής ψευδής και ο κώδικας δεν θα εισερχόταν ποτέ στο μπλοκ της while. Αντίθετα, αν η μεταβλητή i δεν μειωνότανε μέσα στην while, τότε η ροή του κώδικα δεν θα εξερχόταν από τον βρόχο. Σε αυτήν την περίπτωση λέμε ότι έχουμε έναν ατέρμονο βρόχο (endless loop). Προφανώς, η παρουσία ατέρμονου βρόχου μέσα σε ένα πρόγραμμα αποτελεί σοβαρό προγραμματιστικό λάθος.

Τόσο η while όσο και οι υπόλοιπες επαναληπτικές δομές, μπορούν να αξιοποιήσουν τις λέξεις-κλειδιά break και continue. Η μεν break μεταφέρει την ροή του προγράμματος στην πρώτη πρόταση έξω από το μπλοκ της while, η δε continue μεταφέρει την ροή στην συνθήκη ελέγχου.

Ας υποθέσουμε πως θέλουμε να τυπώσουμε 10 το πολύ τυχαίες ακέραιες τιμές στην κλίμακα 1 έως 10. Αν όμως κάποια από τις τιμές αυτές είναι μικρότερη από το 2, τότε να την τυπώσουμε και να σταματήσουμε ακόμη και αν δεν έχει συμπληρωθεί ο αριθμός των 10 εκτυπωμένων τιμών.
Ένας τρόπος για να πετύχουμε αυτήν την απαίτηση δίνεται στον κώδικα 5.11.

```
static void prtRandoms() {
    int i = 0, rI;
    Random generator = new Random();
    while (i < 10) {
        i++;
        rI = generator.nextInt(10) + 1;
        if (rI >= 2) {
            System.out.println(rI);
        } else {
            System.out.println(rI);
            break;
        }
    }
}
```

Κώδικας 5.11 *Παράδειγμα while με break*

Καταρχάς στον κώδικα 5.11, δηλώνουμε και αρχικοποιούμε την μεταβλητή i στο 0. Παράλληλα δηλώνουμε την μεταβλητή rI. Στην συνέχεια, δημιουργούμε το αντικείμενο generator. Πρόκειται για μια γεννήτρια τυχαίων αριθμών. Σημειώστε πως για να χρησιμοποιήσετε ένα τέτοιο αντικείμενο θα πρέπει να εισάγετε στο πρόγραμμά σας την βιβλιοθήκη Java.util.Random. Αυτό επιτυγχάνετε αν εισάγετε αμέσως μετά την δήλωση του πακέτου την σειρά import Java.util.Random; Στην συνέχεια αναπτύσσουμε μια επαναληπτική διαδικασία while. Η συνθήκη ελέγχου είναι i<10, δηλ. η επαναληπτική διαδικασία θα επαναλαμβάνεται όσο το i θα είναι μικρότερο από το 10. Δεδομένου ότι το i αρχικοποιήθηκε στο 0 και σε κάθε επανάληψη αυξάνεται κατά 1, θα εκτελεστούν 10 το πολύ επαναλήψεις. Αυτό είναι συμβατό με την απαίτηση να τυπώσουμε 10 το πολύ τυχαίους ακεραίους.

Στην συνέχεια χρησιμοποιούμε τον generator για να παράγουμε έναν τυχαίο αριθμό. Η συνάρτηση nextInt παράγει έναν τυχαίο ακέραιο από το 0 έως την τιμή του ορίσματός της μείον 1, δηλ. από το 0 έως το 9. Σε αυτόν τον αριθμό προσθέτουμε το 1, οπότε ο ακέραιος που εκχωρείται στην μεταβλητή rI είναι ένας τυχαίος από το 1 έως το 10. Μετά, ελέγχουμε την τιμή της rI και αν είναι μεγαλύτερη ή ίση με το 2 απλώς την τυπώνουμε διαφορετικά ισχύει η απαίτηση σύμφωνα με την οποία αν κάποια από τις τυχαίες τιμές είναι μικρότερη από το 2, τότε θα πρέπει να την τυπώσουμε και να σταματήσουμε περαιτέρω εκτύπωση τυχαίων αριθμών. Αυτό ακριβώς κάνει ο κώδικας 5.11. Στην συνέχεια τυπώνουμε την τιμή που είναι μικρότερη από το 2 και αμέσως μετά εκτελείται η break που διακόπτει την επαναληπτική διαδικασία και μεταφέρει την ροή έξω από την επαναληπτική διαδικασία όπου ακολουθεί και η έξοδος της συνάρτησης prtRandoms.
Δύο σχόλια σχετικά με τον κώδικα 5.11. Γενικά είναι καλό στα προγράμματά μας να αποφεύγουμε την φλυαρία, δηλ. όπου είναι εφικτό να προτιμούμε τον βραχύτερο έναντι του μακρύτερου κώδικα. Προσέξτε πως ο κώδικας System.out.println(rI) εκτελείται και στις δύο διακλαδώσεις της if, δηλ. είτε ισχύει η συνθήκη rI>=2 είτε όχι. Ένας τέτοιος κώδικας μπορεί να βγει πάντα έξω από την if έτσι ώστε μια παρουσία του να είναι αρκετή.

Ένα άλλο θέμα που θα μπορούσαμε να βελτιώσουμε, σχετίζεται με την αύξηση του i. Αφού ελέγξουμε το i στην επόμενη γραμμή το αυξάνουμε κατά 1. Το ίδιο ακριβώς αποτέλεσμα μπορούμε να πετύχουμε αν αυξήσουμε το i στην συνθήκη ελέγχου με χρήση του μεταθεματικού τελεστή προσαύξησης. Προσοχή όμως, αν η αύξηση γίνει όπως γίνεται στον κώδικα 5.11 δεν έχει ιδιαίτερη σημασία αν χρησιμοποιηθεί ο προθεματικός ή ο μεταθεματικός τελεστής προσαύξησης. Αν όμως η αύξηση γίνει μέσα στην συνθήκη τότε η χρήση του μεταθεματικού τελεστή διαφοροποιεί ουσιαστικά την συμπεριφορά του κώδικα. Μελετήστε πως την αλλάζει και γιατί. Ο κώδικας 5.12 παράγει το ίδιο ακριβώς αποτέλεσμα με τον κώδικα 5.11 με το πλεονέκτημα ότι είναι βραχύτερος.

```
static void prtRandoms() { //Κώδικας 5.12
    int i = 0, rI;
    Random generator = new Random();
    while (i++ < 10) {
        rI = generator.nextInt(10) + 1;
        System.out.println(rI);
        if (rI < 2) {
            break;
        }
    }
}
```

Κώδικας 5.12 *Βραχύτερη έκδοση της συνάρτησης prtRandoms*

Ας υποθέσουμε τώρα πως θέλουμε να τυπώσουμε 10 τυχαίες ακέραιες τιμές στην κλίμακα 1 έως 10 εξαιρώντας τις τιμές 3 και 4, δηλ. 10 τιμές που ανήκουν στο σύνολο {1..10}-{3,4}. Η λέξη-κλειδί continue θα μας βοηθήσει στην επίλυση αυτού του προβλήματος.

```
static void prt10Randoms() { //Κώδικας 5.13
    int i = 0, rI;
    Random generator = new Random();
    while (i < 10) {
        rI = generator.nextInt(10) + 1;
        if (rI == 3 || rI == 4) {
            continue;
        }
        System.out.println(rI);
        i++;
    }
}
```

Κώδικας 5.13 *Παράδειγμα while με χρήση της λέξης-κλειδί continue*

Μέσα στον βρόχο while του κώδικα 5.13 παράγουμε καταρχάς έναν τυχαίο αριθμό από 1 έως 10. Στην συνέχεια ελέγχουμε αν πρόκειται για αριθμό που εξαιρείται. Αν όντως πρόκειται για αριθμό που εξαιρείται καλούμε την continue ώστε να μεταφέρουμε την ροή στην αρχή του βρόχου όπου ελέγχεται η συνθήκη i<10. Αν δεν

πρόκειται για αριθμό που εξαιρείται προχωρούμε κανονικά στην εκτύπωσή του και στην συνέχεια αυξάνουμε την μεταβλητή ελέγχου i κατά 1. Προσοχή, εδώ η i θα ήταν λάθος να αυξηθεί μέσα στην συνθήκη ελέγχου. Κάτι τέτοιο θα είχε ως αποτέλεσμα την παραγωγή συνολικά 10 ακεραίων και την εκτύπωση μόνο όσον είναι διάφοροι από το 3 και το 4, δηλ. να εκτυπωθούν πιθανότατα λιγότεροι από 10 ακέραιους που προσδιορίζουν οι απαιτήσεις του προβλήματός μας.

5.2.2 Η επαναληπτική δομή do-while

Η γενική μορφή της do-while έχει ως εξής:

```
do {
     statement(s);
} while (condition);
```

Η διαφορά της από την while είναι πως η συνθήκη ελέγχεται στο τέλος. Επομένως, η do-while εγγυάται πως ο κώδικας μέσα στο σώμα της θα εκτελεστεί τουλάχιστον μία φορά.

Ας υποθέσουμε πως θέλουμε να λάβουμε μια σειρά από ακέραιους από τον χρήστη και να υπολογίσουμε το άθροισμά τους. Επιπλέον, να σταματήσουμε την διαδικασία εισόδου από τον χρήστη όταν αυτός δώσει την τιμή 0.

Σε αυτό το πρόβλημα ο βρόχος do-while θα μας φανεί πιο χρήσιμος από τον βρόχο while καθώς είναι αναγκαίο να λάβουμε μια τουλάχιστον είσοδο από τον χρήστη.

```
static void calcSum() { //Κώδικας 5.14
    int i;
    int sum = 0;
    Scanner in = new Scanner(System.in);
    do {
        System.out.print("Enter int (0 terminates input):");
        i = in.nextInt();
        sum += i;
    } while (i != 0);
    System.out.println("Sum = " + sum);
}
```

Κώδικας 5.14 *Παράδειγμα χρήσης του βρόχου do-while*

Στον κώδικα 5.14, ορίζουμε την μεταβλητή i που θα χρησιμοποιήσουμε για να καταχωρούμε την είσοδο του χρήστη. Την μεταβλητή i δεν είναι αναγκαίο να την αρχικοποιήσουμε καθώς είναι βέβαιο πως πριν χρησιμοποιηθεί θα εκχωρηθεί τιμή σε αυτήν. Στην συνέχεια, ορίζουμε την μεταβλητή sum. Είναι αναγκαίο να αρχικοποιήσουμε την sum στο 0 καθώς αυτή προσαυξάνεται κατά την είσοδο του χρήστη σε κάθε βήμα της επαναληπτικής διαδικασίας do-while. Ακολούθως ορίζουμε ένα αντικείμενο της κλάσης Scanner για να το χρησιμοποιήσουμε για είσοδο από την χρήστη. Στο επόμενο βήμα μπαίνουμε στην επαναληπτική διαδικασία do-while όπου ζητάμε και παίρνουμε έναν ακέραιο από τον χρήστη με τον οποίο ενημερώνουμε τον αθροιστή sum. Στο τέλος της do-while γίνεται ο έλεγχος της συνθήκης. Αν ο χρήστης έδωσε ακέραιο διάφορο του 0, η διαδικασία επαναλαμβάνεται αλλιώς τερματίζεται.

Όλες οι επαναληπτικές διαδικασίες μπορούν να υλοποιηθούν με χρήση οποιασδήποτε επαναληπτικής δομής. Ωστόσο κάποιες δομές παρέχουν διευκολύνσεις ανάλογα με τις απαιτήσεις που πρέπει να υλοποιηθούν. Στον κώδικα 5.15, επιλύουμε το ίδιο πρόβλημα με αυτό του κώδικα 5.14 αλλά χρησιμοποιώντας την δομή while αντί της δομής do-while.

```
static void calcSum() { //Κώδικας 5.15
    int i;
    int sum = 0;
    Scanner in = new Scanner(System.in);
    System.out.print("Enter int (0 terminates input):");
    i = in.nextInt();
    while (i != 0) {
        sum += i;
```

```
            System.out.print("Enter int (0 terminates input):");
            i = in.nextInt();
        }
        System.out.println("Sum = " + sum);
    }
```

Κώδικας 5.15 *Υλοποίηση της calcSum με while αντί do-while*

Προσέξτε πως στον κώδικα 5.15, οι εντολές εισόδου από τον χρήστη εμφανίζονται δύο φορές, μία έξω από τον βρόχο και μια μέσα στον βρόχο μετά την επεξεργασία που στην περίπτωσή μας είναι η πρόσθεση του ακεραίου i στον αθροιστή sum. Πρόκειται για μια τυπική μεταφορά ενός do-while σε while. Ωστόσο η διπλή παρουσία των εντολών εισόδου μπορεί να αποφευχθεί αν οργανώσουμε τον κώδικά μας όπως φαίνεται στον κώδικα 5.16.

```
    static void calcSum() { //Κώδικας 5.16
        int i;
        int sum = 0;
        Scanner in = new Scanner(System.in);
        while (true) {
            System.out.print("Enter int (0 terminates input):");
            i = in.nextInt();
            if (i == 0) {
                break;
            }
            sum += i;
        }
        System.out.println("Sum = " + sum);
    }
```

Κώδικας 5.16 *Προσομοίωση της do-while με while*

Στον κώδικα 5.16 προσομοιώνουμε την λειτουργία της do-while με χρήση της while. Η ροή του κώδικα θα εισέλθει οπωσδήποτε μια τουλάχιστον φορά στον βρόχο while καθώς η συνθήκη ελέγχου είναι πάντα true. Στην ουσία, ο έλεγχος του βρόχου γίνεται από την if η οποία αν εντοπίσει τιμή ίση με το 0 καλεί την break που προκαλεί έξοδο από τον βρόχο.
Συχνά δημιουργείται η αντίληψη ότι η continue κατευθύνει την ροή του προγράμματος στην αρχή της επαναληπτικής διαδικασίας. Η αντίληψη αυτή όμως είναι λανθασμένη. Η continue κατευθύνει την ροή στην συνθήκη ελέγχου του βρόχου. Αν ο βρόχος είναι τύπου while, τότε η αρχή της διαδικασίας συμπίπτει με την συνθήκη ελέγχου. Αν όμως ο βρόχος είναι τύπου do-while τότε η συνθήκη βρίσκεται στο τέλος. Ο κώδικας 5.17 επιδεικνύει την λειτουργία της continue σε ένα do-while βρόχο.

```
    static void checkContinue() {
        int i = 0;
        do {
            i++;
            System.out.println(i);
            if (i == 1) {
                continue;
            }
            System.out.println(i);
        } while (i < 1);
    }
```

Κώδικας 5.17 *Η λειτουργία της continue*

Ο κώδικας 5.17 όταν εκτελείται τυπώνει 1. Το αποτέλεσμα του κώδικα είναι απόδειξη ότι η continue μεταφέρει την ροή στην συνθήκη ελέγχου και όχι απλώς στην αρχή του βρόχου. Με την είσοδο μέσα στον βρόχο, η μεταβλητή i αυξάνεται κατά 1 και γίνεται 1 από 0 που ήταν η τιμή στην οποία αρχικοποιήθηκε στην πρώτη γραμμή της checkContinue. Στην συνέχεια τυπώνεται η τιμή της i, δηλ. το 1. Ακολούθως ελέγχεται η τιμή της i, η συνθήκη i==1 βρίσκεται αληθής και εκτελείται το continue. Αν το continue κατηύθυνε την ροή στην αρχή του βρόχου, τότε το i θα γινόταν 2 και στην συνέχεια θα τυπωνόταν πριν φτάσει στην συνθήκη ελέγχου και

εξέλθει από τον βρόχο. Όμως, η continue κατευθύνει άμεσα την ροή στην συνθήκη ελέγχου η οποία φυσικά για την τιμή του i ίση με το 1 είναι ψευδής και έτσι ο βρόχος διακόπτεται έχοντας εκτυπώσει μόνο το 1.

5.2.3 Η επαναληπτική δομή for

Η for παρουσιάζει μεγάλη ευελιξία και είναι ίσως η συχνότερα χρησιμοποιούμενη επαναληπτική δομή. Η γενική μορφή της έχει ως εξής:

```
for (initialization; condition; statement) {
     statement(s);
}
```

Η for επομένως αποτελείται από τέσσερα μέρη. Το initialization είναι το τμήμα εκείνο στο οποίο τυπικά αρχικοποιείται ή και δηλώνεται η μεταβλητή ελέγχου του βρόχου. Η condition είναι η συνθήκη που ελέγχει αν θα συνεχισθεί η επαναληπτική διαδικασία ή όχι. Το statement μπορεί να είναι μια οποιοδήποτε εντολή ή συνάρτηση. Στην τυπική περίπτωση, στο statement μεταβάλλεται η τιμή της μεταβλητής ελέγχου. Το τέταρτο μέρος είναι το μπλοκ της for στο οποίο μπορούμε να τοποθετήσουμε ένα ή περισσότερα statements.

Το initialization εκτελείται μια φορά κατά την είσοδο στον βρόχο. Στην συνέχεια εκτελείται η condition. Αν βρεθεί αληθής, η ροή κατευθύνεται στο μπλοκ της for. Αφού εκτελεστούν οι statements του μπλοκ, η ροή κατευθύνεται στο statement εντός του κυρίου σώματος της for. Μετά την εκτέλεση του statement εντός του κυρίου σώματος, η ροή κατευθύνεται στην condition για να ακολουθήσει μια κυκλική διαδρομή του τύπου condition-statement(s)-statement έως ότου βέβαια η condition λάβει την τιμή false. Σε αυτήν την περίπτωση, διακόπτεται η for και η ροή του προγράμματος κατευθύνεται στην έξοδο του βρόχου.
Ας δούμε καταρχάς μια απλή εφαρμογή της for. Ο κώδικας 5.18, τυπώνει τους ακεραίους από 0 έως 9.

```
for (int i = 0; i < 10; i++) {
     System.out.println(i);
}
```

Κώδικας 5.18 *Εκτύπωση των ακεραίων 0..9 με την for*

Η πρώτη τιμή που εκτυπώνεται είναι η τιμή 0. Αυτό δείχνει πως η ροή εκτέλεσης, πρώτα εισέρχεται στο μπλοκ της for και στην συνέχεια εκτελεί την προσαύξηση του i στο κυρίως σώμα της for. Η δε τελευταία τιμή είναι το 9. Πράγματι, μόλις τυπωθεί το 9, η ροή κατευθύνεται στην πρόταση προσαύξησης του i που γίνεται 10. Στην συνέχεια, η συνθήκη ελέγχου καθίσταται ψευδής και οδηγεί στην έξοδο του βρόχου.
Ας σημειωθεί πως στο τμήμα αρχικοποίησης της for δεν είναι υποχρεωτικό να δηλώνεται η μεταβλητή ελέγχου. Ωστόσο αποτελεί καλή πρακτική η δήλωσή της εφόσον αυτή δεν είναι χρήσιμη έξω από την for. Κάτι ακόμη που πρέπει να αποσαφηνίσουμε είναι πως κανένα από τα τέσσερα τμήματα της for δεν είναι υποχρεωτικό.

Στην συνέχεια δίνουμε παραδείγματα χρήσης της for μέσα από τα οποία γίνονται σαφείς οι παρατηρήσεις μας σε σχέση με αυτήν. Πιο συγκεκριμένα, υλοποιούμε με for κώδικες που αξιοποιούν την while ή την do-while και έχουν ήδη παρουσιαστεί μέχρι εδώ σε αυτήν την ενότητα. Με αυτόν τον τρόπο θα αποκτήσουμε πληρέστερη εικόνα της σχετικής ευελιξίας της for.
Ο κώδικας 5.10 που επιδεικνύει την χρήση της while, υλοποιείται με for όπως δείχνει ο κώδικας 5.19.

```
static void forDemo() { //Κώδικας 5.19
    int i;
    for (i = 10; i > 0; i--) {
        System.out.print("i>0 ");
    }
    System.out.println("i=" + i);
}
```

Κώδικας 5.19 *Ίδιο αποτέλεσμα με την while του κώδικα 5.10*

Στον κώδικα 5.19, η μεταβλητή ελέγχου i δεν δηλώνεται στο τμήμα αρχικοποίησης της for. Αυτό είναι αναγκαίο καθώς θέλουμε να προσπελάσουμε την μεταβλητή έξω από τα όρια της for. Αν την είχαμε δηλώσει στο τμήμα αρχικοποίησης της for, η εμβέλεια της μεταβλητής θα περιοριζόταν στα τμήματα της for.

Η συνάρτηση prtRandoms που παρουσιάζεται στον κώδικα 5.12 μεταγράφεται με for όπως φαίνεται στον κώδικα 5.20.

```
static void prtRandoms() { //Κώδικας 5.20
    Random generator = new Random();
    int rI;
    for (int i = 0; i++ < 10;) {
        rI = generator.nextInt(10) + 1;
        System.out.println(rI);
        if (rI < 2) {
            break;
        }
    }
}
```

Κώδικας 5.20 *Η prtRandoms του κώδικα 5.12 υλοποιημένη με for*

Προσέξτε πως σε αυτήν την περίπτωση, το τρίτο τμήμα της for δεν χρησιμοποιήθηκε. Όπως αναφέραμε προηγουμένως, όλα τα τμήματα της for είναι προαιρετικά.
Η συνάρτηση prtRandoms που παρουσιάστηκε στον κώδικα 5.13 μεταγράφεται με for όπως δείχνει ο κώδικας 5.21.

```
static void prtRandoms() { //Κώδικας 5.21
    int rI;
    Random generator = new Random();
    for (int i = 0; i < 10; i++) {
        rI = generator.nextInt(10) + 1;
        if (rI == 3 || rI == 4) {
            continue;
        }
        System.out.println(rI);
    }
}
```

Κώδικας 5.21 *Η prtRandoms του κώδικα 5.13 υλοποιημένη με for*

Ο κώδικας 5.22 υλοποιεί την calcSum του κώδικα 5.14 χρησιμοποιώντας for αντί για do-while.

```
static void calcSum() { //Κωδικας 5.22
    int sum = 0;
    Scanner input = new Scanner(System.in);
    for (int i = 1; i != 0;) {
        System.out.print("Enter int (0 terminates input):");
        i = input.nextInt();
        sum += i;
    }
    System.out.println("Sum = " + sum);
}
```

Κώδικας 5.22 *Η calcSum του κώδικα 5.14 υλοποιημένη με for*

Ο κώδικας 5.23 παρέχει μια κάπως διαφορετική υλοποίηση του κώδικα 5.18.

```
 for (int i=0; i<10; System.out.println(i));
```

Κώδικας 5.23 *Εναλλακτική υλοποίηση του κώδικα 5.18*

Συχνά οι αρχάριοι προγραμματιστές έχουν την εντύπωση πως οι μεταβλητές ελέγχου της for είναι υποχρεωτικά τύπου int. Κάτι τέτοιο βέβαια δεν αληθεύει. Παρακάτω στον κώδικα 5.24, εκτυπώνουμε την Αγγλική αλφάβητο με κεφαλαίους χαρακτήρες.

```
static void prtAlpha() { //Κώδικας 5.24
```

```
        for (char c = 'A'; c <= 'Z'; c++) {
            System.out.print(c + " ");
        }
        System.out.println();
    }
```

Κώδικας 5.24 *Εκτύπωση της Αγγλικής Αλφάβητου με for*

Η μεταβλητή που αρχικοποιείται στο τμήμα αρχικοποίησης της for δεν λαμβάνει υποχρεωτικά μέρος στην συνθήκη ελέγχου. Αυτό επιδεικνύεται στον κώδικα 5.25 ο οποίος τυπώνει διαδοχικούς ακέραιους αρχίζοντας από το 0 έως ότου ο χρήστης επιλέξει έξοδο.

```
    static void prtInts() {//Κώδικας 5.25
        int v = 1;
        Scanner input = new Scanner(System.in);
        for (int i = 0; v > 0; i++) {
            System.out.println(i);
            System.out.print("Εισάγετε 0 για έξοδο:");
            v = input.nextInt();
        }
    }
```

Κώδικας 5.25 *Εκτύπωση διαδοχικών ακέραιων μέχρι να επιλέξει ο χρήστης έξοδο*

Όπως στις δομές επιλογής, έτσι και εδώ έχουμε την δυνατότητα εμφωλευμένων επαναληπτικών δομών. Ο κώδικας 5.26, τυπώνει 10 φορές την Αγγλική αλφάβητο.

```
    static void nestedFor() { //Κώδικας 5.26
        for (int i = 0; i < 10; i++) {
            for (char c = 'A'; c <= 'Z'; c++) {
                System.out.print(c + " ");
            }
            System.out.println();
        }
    }
```

Κώδικας 5.26 *Παράδειγμα εμφωλευμένης for*

Ο κώδικας 5.27 έχει σαν σκοπό να δώσει έμφαση στο γεγονός ότι τα διάφορα τμήματα της for είναι προαιρετικά. Τέτοιου είδους χρήση της for υποστηρίζεται μεν από τον μεταγλωττιστή, είναι ωστόσο ασυνήθιστη.

```
    static void emptyFor() { //Κώδικας 5.27
        int i = 0;
        for (;;) {
            System.out.println(i++);
            if (i > 10) {
                break;
            }
        }
    }
```

Κώδικας 5.27 *Παράδειγμα της for χωρίς κανένα από τα τρία κύρια τμήματα*

Σημείωση: Υπάρχει και μια επιπλέον μορφή της for, γνωστή ως ενισχυμένη (enhanced) for. Η ενισχυμένη for σχετίζεται με τις συλλογές, δηλ. με δομές που είναι κατάλληλες για αποθήκευση πλήθους δεδομένων, π.χ. πίνακες. Η ενισχυμένη for παρουσιάζεται στην ενότητα 6.1.2.

5.3. Λυμένες Ασκήσεις

5.3.1 Εκτύπωση πραγματικών με δεκαδικό βήμα

Αναπτύξτε συνάρτηση που τυπώνει τους πραγματικούς αριθμούς από 1 έως 10 με βήμα 0.1.

Λύση
Μια απλή for με μεταβλητή ελέγχου τύπου double αρκεί για την λύση αυτής της άσκησης.

```
static void simpleForwithDouble() { //Κώδικας 5.28
    for (double i = 0; i <= 10; i += 0.1) {
        System.out.println(i);
    }
}
```

Κώδικας 5.28 *Απλή for με μεταβλητή ελέγχου double και βήμα 0.1*

5.3.2 Εκτύπωση ακεραίων υπό συνθήκη

Να γράψετε πρόγραμμα που τυπώνει όλους τους ακεραίους από το 0..100 που διαιρούνται ακριβώς με το 4, το 6 και το 8.

Λύση
Η πρώτη σκέψη που μας έρχεται στο μυαλό είναι να ελέγξουμε κάθε ακέραιο από το 0 έως το 100 και αν διαιρείται ακριβώς με το 4, το 6 και το 8 να το εμφανίσουμε. Ωστόσο, υπάρχει και πιο αποδοτικός τρόπος. Αφού οι αριθμοί που θα τυπώνουμε διαιρούνται με το 4, το 6 και το 8, μπορούμε να ξεκινήσουμε από το 4 και να ελέγξουμε μόνο τα πολλαπλάσια του 4. Ελέγχουμε λοιπόν τα πολλαπλάσια του 4. Αν πληρούν και τις λοιπές προϋποθέσεις, τότε τα τυπώνουμε. Αν θεωρήσουμε πως και το 0 είναι πολλαπλάσιο του 4, τότε θα ξεκινήσουμε από το 0. Με αυτόν τον τρόπο αντί να ελέγξουμε 100 περίπου ακέραιους, ελέγχουμε μόνο 25. Μπορούμε και εδώ να χρησιμοποιήσουμε μια for.

```
static void prtRealsConditional() { //Κώδικας 5.29
    for (int i = 0; i <= 100; i += 4) {
        if (i % 6 == 0 && i % 8 == 0) {
            System.out.println(i);
        }
    }
}
```

Κώδικας 5.29 *Εκτύπωση ακεραίων υπο συνθήκη*

5.3.3 Εκτύπωση και καταμέτρηση ακεραίων υπό συνθήκη

Να γράψετε πρόγραμμα που
Τυπώνει όσους ακεραίους στην περιοχή 0..1000 είναι πολλαπλάσια του 3 και ταυτοχρόνως τέλεια τετράγωνα, δηλ. η τετραγωνική τους ρίζα είναι ακέραιος.
Μετράει και εκτυπώνει το πλήθος των ακεραίων που πληρούν τις περιγραφόμενες προϋποθέσεις.
Σημείωση: Η τετραγωνική ρίζα μιας αριθμητικής μεταβλητής i δίνεται από την συνάρτηση sqrt της κλάσης Math : Math.sqrt(i).

Λύση
Το ερώτημα εδώ είναι ο έλεγχος για να διαπιστώσουμε αν η τετραγωνική ρίζα ενός αριθμού είναι ακέραιος. Καταρχάς την τετραγωνική ρίζα θα την υπολογίσουμε με την συνάρτηση sqrt() της Math. Όπως είναι λογικό, η συνάρτηση αυτή επιστρέφει αριθμό τύπου double. Για να διαπιστώσουμε αν ο αριθμός αυτός είναι ακέραιος θα ελέγξουμε αν το υπόλοιπο της διαίρεσης μεταξύ τετραγωνικής ρίζας και 1 είναι ίσο με το 0. Προκειμένου όμως να ελέγξουμε σωστά θα πρέπει να χρησιμοποιήσουμε προσεγγιστική ισότητα όπως έχουμε επισημάνει στις ενότητες 3.2 και 4.1.4.1.

```
static boolean approximateEquals(double f1, double f2, double epsilon) {
    return Math.abs(f2 - f1) < epsilon;
```

```
    }

    static void search() { //Κώδικας 5.30
        int cnt = 0;
        for (int i = 0; i < 1000; i += 3) {
            if (approximateEquals(Math.sqrt(i) % 1, 0, 0.00001d)) {
                System.out.print(i + " ");
                cnt++;
            }
        }
        System.out.println();
        System.out.println("Ta teleia tetragwna twn pollaplasiwn tou 3 apo
0..1000 einai " + cnt);
    }
```

Κώδικας 5.30 *Αναζήτηση σε πίνακα πραγματικών με προσεγγιστική ισότητα*

5.3.4 Τυχαίοι ακέραιοι και switch

Να γράψετε πρόγραμμα που λαμβάνει 2 ακεραίους από τον χρήστη και ανάλογα με τις τιμές τους, τυπώνει ένα μήνυμα της μορφής "εποχή ημέρα". Η ονομασία ημέρας και εποχής με βάση τις ακόλουθες αντιστοιχίσεις: (1->Κυριακή, 2->Δευτέρα, …, 7 Σάββατο), (1-> Άνοιξη, 2-> Καλοκαίρι, 3-> Φθινόπωρο, 4->Χειμώνας). Σε περίπτωση που η τιμή ενός (ή και των δύο) ακεραίων είναι εκτός των προαναφερόμενων περιοχών τυπώνει unkown value.

Λύση
Εδώ είναι βολικό να υπολογίσουμε πρώτα το μήνυμα και μετά να το τυπώσουμε. Για τον σκοπό αυτό θα χρησιμοποιήσουμε κατάλληλες μεταβλητές τύπου String.

```
    static void decodeDaySeason() { //Κώδικας 5.31
        Scanner in = new Scanner(System.in);
        System.out.print("Εισάγετε ΑΑ ημέρας: ");
        int day = in.nextInt();
        System.out.print("Εισάγετε ΑΑ εποχής: ");
        int season = in.nextInt();
        String message = "";
        if (day < 1 || day > 7 || season < 1 || season > 4) {
            message = "Unkown Value";
        } else {
            switch (day) {
                case 1:
                    message = "Sunday";
                    break;
                case 2:
                    message = "Monday";
                    break;
                case 3:
                    message = "Tuesday";
                    break;
                case 4:
                    message = "Wensday";
                    break;
                case 5:
                    message = "Thursday";
                    break;
                case 6:
                    message = "Friday";
                    break;
                case 7:
                    message = "Saturday";
```

```
                break;
            }
            switch (season) {
                case 1:
                    message += " Spring";
                    break;
                case 2:
                    message += " Summer";
                    break;
                case 3:
                    message += " Autumn";
                    break;
                case 4:
                    message += " Winter";
                    break;
            }
        }
        System.out.println(message);
    }
```

Κώδικας 5.31 *Εκτύπωση ημέρας και εποχής*

Στον κώδικα 5.31, καταρχάς, εισάγουμε από το πληκτρολόγιοστο δύο ακεραίους, ο ένας αντιπροσωπεύει τον αύξοντα αριθμό της ημέρας και ο άλλος της εποχής. Στην συνέχεια, δηλώνουμε και αρχικοποιούμε την μεταβλητή message τύπου String. Η αρχικοποίηση στην συγκεκριμένη περίπτωση είναι υποχρεωτική. Αν δεν αρχικοποιήσουμε την message, ο μεταγλωττιστής θα παράγει σφάλμα σε όλα τα σημεία του κώδικα που επιχειρείται η σύνδεση της message με την περιγραφή της εποχής. Πριν μπούμε στον έλεγχο της ημέρας στην πρώτη switch, ελέγχουμε αν οι αύξοντες αριθμοί είναι στα επιτρεπτά όρια. Αν κάποιος από τους ακέραιους που θα εισάγει ο χρήστης είναι εκτός ορίων τότε το message έχει λάβει τιμή. Επιπλέον, η ροή του κώδικα δεν θα περάσει από τις επίμαχες γραμμές. Αν πάλι κανένας από τους ακεραίους δεν είναι εκτός ορίων, το message θα έχει λάβει τιμή από την πρώτη switch. Επομένως, το message θα έχει λάβει οπωσδήποτε τιμή πριν η ροή κατευθυνθεί στην δεύτερη switch. Ωστόσο, το αποτέλεσμα δεν μπορεί να διαπιστωθεί κατά τον χρόνο μεταγλώττισης. Για τον μεταγλωττιστή είναι πιθανό, η ροή να μπει στην δεύτερη switch χωρίς να έχει αρχικοποιηθεί το message. Για παράδειγμα, αν δεν ελέγξουμε την day για την τιμή 1 αλλά την ελέγξουμε για μια άλλη τιμή, π.χ. 9, τότε το message μπορεί να φτάσει στην δεύτερη switch αναρχικοποίητο.

5.3.5 Έλεγχος αν ακέραιος είναι πρώτος ή όχι

Να γράψετε πρόγραμμα που λαμβάνει έναν ακέραιο από τον χρήστη και εξετάζει αν είναι πρώτος αριθμός ή όχι.
Σημείωση: Πρώτος είναι ένας ακέραιος μεγαλύτερος της μονάδας που διαιρείται ακριβώς μόνο με το 1 και τον εαυτό του.

Λύση
Όλοι οι ακέραιοι, πρώτοι και μη, διαιρούνται ακριβώς με το 1 και τον εαυτό τους. Αν όμως ένας ακέραιος διαιρείται ακριβώς με οποιονδήποτε άλλον πλην του 1 και του εαυτού του τότε δεν είναι πρώτος. Επομένως θα πρέπει να εξετάσουμε αν ο ακέραιος που θα εισάγει ο χρήστης διαιρείται ακριβώς με οποιονδήποτε άλλον ακέραιο. Αν θεωρήσουμε έναν ακέραιο x, ποιοι είναι οι ακέραιοι εκτός του 1 και του x που έχουν πιθανότητα να διαιρούν ακριβώς τον x; Είναι προφανές πως οι μεγαλύτεροι του x δεν έχουν τέτοια πιθανότητα. Επίσης αποκλείονται το 1 και το x. Επομένως μένουν οι ακέραιοι από το 2 έως x-1. Πράγματι αν εξετάσουμε αν το x διαιρείται ακριβώς με οποιονδήποτε ακέραιο από 2 έως x-1 μπορούμε να διαπιστώσουμε αν ο x είναι πρώτος ή όχι. Αν όμως σκεφτούμε λίγο πιο προσεχτικά θα αντιληφθούμε πως οι αριθμοί που είναι μεγαλύτεροι από x/2 επίσης αποκλείεται να διαιρούν ακριβώς τον x καθώς ο μεγαλύτερος πιθανός διαιρέτης του είναι ο x/2. Συνεπώς αρκεί να εξετάσουμε τους ακέραιους από 2 έως και x/2.

```
    static void prwtos() { //Κώδικας 5.32
        Scanner in = new Scanner(System.in);
        System.out.print("Εισάγετε ακέραιο: ");
```

```
        int k = in.nextInt();

        boolean isPrwtos = k >= 2;
        if (isPrwtos) {
            for (int i = 2; i <= k / 2; i++) {
                if (k % i == 0) {
                    isPrwtos = false;
                    break;
                }
            }
        }
        if (isPrwtos) {
            System.out.println(k + " είναι πρώτος");
        } else {
            System.out.println(k + " δεν είναι πρώτος");
        }
    }
```

Κώδικας 5.32 *Έλεγχος αν ένας ακέραιος είναι πρώτος αριθμός*

Καταρχάς, λαμβάνουμε την είσοδο του χρήστη στην ακέραιη μεταβλητή k. Στην συνέχεια, δηλώνουμε και αρχικοποιούμε την μεταβλητή isPrwtos τύπου boolean. Η isPrwtos αρχικοποιείται στην τιμή της έκφρασης k>=2, δηλ. στην τιμή true αν το k είναι μεγαλύτερο ή ίσο του 2 και στην τιμή false αν το k είναι μικρότερο από το 2. Στην δεύτερη αυτή περίπτωση, το k εξορισμού δεν είναι πρώτος αριθμός. Επομένως ο κώδικας δεν θα μπει καθόλου στην if που ακολουθεί αλλά θα μεταβεί κατευθείαν στην else της επόμενης if και θα τυπώσει πως ο αριθμός δεν είναι πρώτος.

Αντίθετα αν το k είναι μεγαλύτερο ή ίσο με το 2, η ροή του κώδικα θα εισέλθει στην πρώτη if. Στην συνέχεια ο έλεγχος περνάει στην επαναληπτική διαδικασία που ακολουθεί. Ο σκοπός αυτού του βρόχου είναι να ελέγξει αν το k διαιρείται ακριβώς με οποιονδήποτε ακέραιο από το 2 έως το k/2. Αν βρεθεί έστω ένας ακέραιος που διαιρεί ακριβώς το k, τότε το k δεν είναι πρώτος οπότε η isPrwtos ενημερώνεται με την τιμή false και η επαναληπτική διαδικασία διακόπτεται. Στην συνέχεια, η ροή κατευθύνεται στην δεύτερη if η οποία ανάλογα με την τιμή της isPrwtos τυπώνει το κατάλληλο μήνυμα.

5.3.6 Εκτύπωση Αγγλικής αλφάβητου

Να γράψετε πρόγραμμα που τυπώνει 100 φορές την Αγγλική αλφάβητο με πεζούς χαρακτήρες αντεστραμμένη, δηλ. από το z στο a. Δώστε 2 λύσεις. Η πρώτη να χρησιμοποιεί μόνο for και η δεύτερη να χρησιμοποιεί συνδυασμό while και do-while.

Λύση
Ακολουθούν οι ζητούμενες υλοποιήσεις

```
    static void printReversedAlphabet() { //Κώδικας 5.33
        for (int i = 0; i < 100; i++) {
            for (char c = 'z'; c >= 'a'; c--) {
                System.out.print(c + " ");
            }
            System.out.println();
        }
    }

    static void printReversedAlphabet2() { //Κώδικας 5.33
        int i = 0;
        while (i++ < 100) {
            char c = 'z';
            do {
                System.out.print(c + " ");
                c--;
            } while (c >= 'a');
```

```
            System.out.println();
        }
    }
```

Κώδικας 5.33 *Εκτύπωση της Αγγλικής αλφαβήτου με αντεστραμμένη διάταξη. Δύο προσεγγίσεις.*

5.3.7 Εκθετική εξίσωση

Γράψτε κατάλληλο κώδικα που υπολογίζει τις τιμές των a,b,c στην 2a*3b*5c=22500, με a<20, b<20 και c<20. Σε περίπτωση που δεν βρεθεί λύση το πρόγραμμα να εμφανίζει το μήνυμα “Δεν βρέθηκε λύση”.

Λύση
Ο πιο άμεσος τρόπος για την λύση αυτής της άσκησης είναι η χρήση εμφωλευμένων for βάθους τριών επιπέδων. Θα πρέπει όμως να προσέξουμε ώστε να χρησιμοποιήσουμε έλεγχο προσεγγιστικής ισότητας για την σύγκριση πραγματικών αριθμών.

```
    static void ekthetikiExiswsi() { //Κώδικας 5.34
        boolean solutionFound = false;

        for (int a = 0; a < 20; a++) {
            for (int b = 0; b < 20; b++) {
                for (int c = 0; c < 20; c++) {
                    double rslt = Math.pow(2, a) * Math.pow(3, b) * Math.pow(5,
c);
                    if (approximateEquals(rslt, 22500, 0.000001)) {
                        solutionFound = true;
                        System.out.println("a=" + a + ", b=" + b + ", c=" + c);
                        break;
                    }
                }
            }
        }
        if (!solutionFound) {
            System.out.println("Δεν βρέθηκε λύση");
        }
    }
```

Κώδικας 5.34 *Λύση εξίσωσης*

Μία σημαντική παρατήρηση εδώ είναι πως η break μεταφέρει την ροή του προγράμματος έξω από τον εξωτερικό βρόχο και όχι απλά έξω από τον εσωτερικό βρόχο.

5.4 Ασκήσεις προς λύση

1. Αναπτύξτε πρόγραμμα που δέχεται έναν ακέραιο ως είσοδο από τον χρήστη και τυπώνει όλους τους διαιρέτες του.

2. Αναπτύξτε πρόγραμμα που λαμβάνει δύο ακεραίους από τον χρήστη και τυπώνει τον Μέγιστο Κοινό Διαιρέτη τους.

3. Αναπτύξτε πρόγραμμα που λαμβάνει έναν ακέραιο x από τον χρήστη και τυπώνει όλους τους πρώτους αριθμούς που περιλαμβάνονται μεταξύ του 2 και του x.

4. Αναπτύξτε πρόγραμμα που παράγει ερωτήσεις πρόσθεσης ή αφαίρεσης με τυχαίους τελεστέους, ζητά τις απαντήσεις από τον χρήστη και παρέχει αξιολόγηση των απαντήσεων στην μορφή:

```
Απάντησες σωστά σε k από m ερωτήσεις πρόσθεσης
Απάντησες σωστά σε r από t ερωτήσεις αφαίρεσης
```

```
Μέσος όρος βαθμολογίας : y
```

5. Αναπτύξτε πρόγραμμα που ζητά από τον χρήστη των αριθμό ωρών που εργάστηκε κάθε ημέρα μιας εβδομάδας, το ωρομίσθιό του και υπολογίζει

1. Ποια ημέρα εργάστηκε τις περισσότερες ώρες;
2. Ποια ημέρα εργάστηκε τις λιγότερες ώρες;
3. Πόσες ώρες κατά μέσο όρο εργάστηκε την κάθε ημέρα της εβδομάδας;
4. Ποια είναι η αμοιβή της εβδομάδας σύμφωνα με τις ώρες που εργάστηκε και το ωρομίσθιό του;

6. Αναπτύξτε πρόγραμμα που λαμβάνει ένα αριθμό από τον χρήστη, έστω n, και υπολογίζει τον μικρότερο ακέραιο k για τον οποίο ισχύει k2>n

7. Αναπτύξτε πρόγραμμα που λαμβάνει είσοδο από τον χρήστη μια ακέραιη τιμή που αναπαριστά ένα έτος και υπολογίζει αν το έτος είναι δίσεκτο. Βρείτε τον ορισμό του δίσεκτου έτους στο διαδίκτυο.

8. Ένας πωλητής λαμβάνει μηνιαίο μισθό 1200 Ευρώ. Επιπλέον, του μισθού λαμβάνει προμήθεια επι των πωλήσεων. Για πωλήσεις από 0 έως 10.000 Ευρώ, η προμήθεια είναι 5%. Για πωλήσεις από 10.000 έως 20.000 Ευρώ, η προμήθεια είναι 10%. Για πωλήσεις από 20.000 Ευρώ και πάνω, η προμήθεια είναι 20%. Επομένως, ο πωλητής για έναν μήνα που πραγματοποίησε πωλήσεις 15.000 Ευρώ θα λάβει προμήθεια ίση με 10.000x5%+5.000x10%. Αναπτύξτε πρόγραμμα που ζητά από τον πωλητή τις πωλήσεις για κάθε μήνα ενός έτους και υπολογίζει και εκτυπώνει τόσο την μηνιαία αμοιβή του όσο και την ετήσια.

9. Η εταιρία Service απασχολεί τρείς κατηγορίες εργαζόμενων, κατόχους απολυτηρίου Λυκείου, κατόχους Πανεπιστημιακού πτυχίου και κατόχους Διδακτορικού τίτλου. Τα ωρομίσθια των εργαζόμενων έχουν ως εξής: 40 Ευρώ για τους κατόχους απολυτηρίου Λυκείου, 60 Ευρώ για τους κατόχους πανεπιστημιακού πτυχίου και 80 Ευρώ για τους κατόχους Διδακτορικού τίτλου. Το τυπικό ωράριο για κάθε εργαζόμενο είναι 40 ώρες την εβδομάδα. Κάθε επιπλέον ώρα εργασίας αμείβεται με προσαύξηση 50% επι του ωρομισθίου του εργαζόμενου. Γράψτε πρόγραμμα που ζητά από τον εργαζόμενο την κατηγορία στην οποία ανήκει, τον αριθμό ωρών που εργάστηκε την προηγούμενη εβδομάδα και υπολογίζει πόσες είναι οι τακτικές αποδοχές της εβδομάδας, πόσες οι αποδοχές λόγω υπερωρίας και ποιο το σύνολο των αποδοχών.

10. Αναπτύξτε συνάρτηση που λαμβάνει έναν ακέραιο από τον χρήστη τον οποίο εκχωρεί σε κατάλληλη μεταβλητή αφού διασφαλίσει πως η εισαγόμενη τιμή βρίσκεται στα όρια 1 έως 10.

11. Μελετήστε τις συναρτήσεις

```
static void chkBoolean() {
     Random r = new Random();
     boolean b = r.nextBoolean();
     if (b == true) {
          System.out.println("Επιτυχία!");
     } else {
          System.out.println("Αποτυχία");
     }
}

static void chkBoolean2() {
     Random r = new Random();
     boolean b = r.nextBoolean();
     if (b) {
          System.out.println("Επιτυχία!");
     } else {
          System.out.println("Αποτυχία");
     }
}
```

Έχουν κάποια διαφορά ως προς το αποτέλεσμα; Ποια θα προτιμούσατε και γιατί;

12. Μελετήστε τον ακόλουθο κώδικα

```
static void bugAlphabet() { Άσκηση 5.20
     for (char c = 'z'; c >= 'a'; c--) {
          System.out.print(c + ' ');
     }
}
```

Ποια είναι η έξοδός του; Τρέξτε τον, να διαπιστώσετε αν προβλέψατε την σωστή έξοδο. Αν όχι, εξηγήστε ποια η διαφορά και γιατί;

14. Να μεταγραφεί η συνάρτηση

```
public static void xEx1() {
     for (int i = 0, j = 1; i < 10 || i < j; i++) {
          j += i;
          if (i % 3 == 0) {
               j = 0;
          }
          System.out.print("(" + i + ", " + j + ") ");
     }
     System.out.println();
}
```

Με αντικατάσταση του for με while

Με αντικατάσταση του for με do while

15. Να μεταγραφεί η συνάρτηση

```
public static void xEx2() {
     int k = 3;
     if (k == 3) {
          System.out.print(3 + " ");
     }
     if (k == 4 || k == 3) {
          System.out.print(4 + " ");
     } else if (k == 5) {
          System.out.print(5 + " ");
     } else {
          System.out.print("unkown");
     }
     System.out.println();
}
```

με χρήση της swich.

Κεφάλαιο 6

Σύνοψη
Στο κεφάλαιο αυτό παρουσιάζεται μια εισαγωγή στους πίνακες. Πιο συγκεκριμένα, περιλαμβάνονται οι δημιουργία μονοδιάστατων και πολυδιάστατων πινάκων, η ενισχυμένη for, η δημιουργία αντιγράφων πινάκων, η ταξινόμηση, η σειριακή και δυαδική αναζήτηση, ο έλεγχος ισότητας, η εμφάνιση πίνακα, ο υπολογισμός αθροίσματος των στοιχείων ενός πίνακα, η εύρεση του μεγαλύτερου και του μικρότερου στοιχείου, η αρχικοποίηση με τυχαίες τιμές. Το κεφάλαιο περιέχει λυμένες ασκήσεις και ασκήσεις προς λύση.

Προαπαιτούμενη γνώση
Οι θεμελιώδεις τύποι και βασική γνώση του τύπου String. Βασική είσοδος και έξοδος. Η λειτουργία των τελεστών. Οι δομές επιλογής και οι επαναληπτικές δομές.

Λέξεις κλειδιά
Μονοδιάστατοι πίνακες, πολυδιάστατοι πίνακες, δείκτης, ταξινόμηση, σειριακή αναζήτηση, δυαδική αναζήτηση.

6 Πίνακες

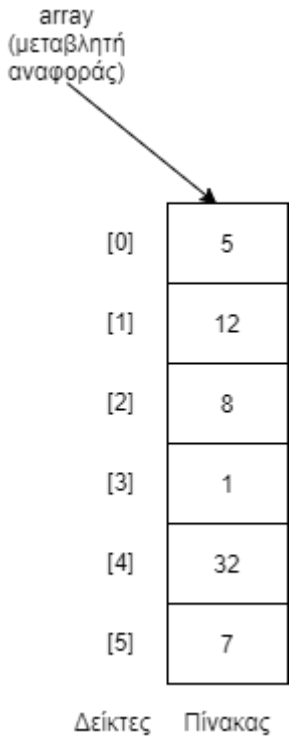
Οι πίνακες είναι συλλογές δεδομένων ίδιου τύπου. Για παράδειγμα, ένας πίνακας ακεραίων αποθηκεύει μια σειρά από τιμές τύπου int ή ένας πίνακας πραγματικών αποθηκεύει μια σειρά από τιμές τύπου double. Όπως εξηγούμε αναλυτικά παρακάτω σε αυτήν την ενότητα, η προσπέλαση στα στοιχεία ενός πίνακα είναι η ταχύτερη δυνατή σε σχέση με την προσπέλαση στα στοιχεία συλλογής οποιουδήποτε άλλου τύπου.

Στο πλαίσιο της Java, ένας πίνακας τα στοιχεία του οποίου δεν είναι πίνακες ονομάζεται μονοδιάστατος πίνακας. Ένας πίνακας τα στοιχεία του οποίου είναι μονοδιάστατοι πίνακες ονομάζεται δισδιάστατος πίνακας. Αντίστοιχα, ένας πίνακας τα στοιχεία του οποίου είναι δυσδιάστατοι πίνακες ονομάζεται τρισδιάστατος πίνακας, κόκ. Συχνά οι μη μονοδιάστατοι πίνακες αναφέρονται ως πολυδιάστατοι.

Σε αυτό το κεφάλαιο, παρουσιάζουμε τις συντακτικές λεπτομέρειες ορισμού των μονοδιάστατων και πολυδιάστατων πινάκων, εξηγούμε βασικά στοιχεία της ενσωματωμένης στην Java, κλάσης Arrays που παρέχει μια σειρά από συναρτήσεις για την διαχείριση πινάκων και παρουσιάζουμε μια σειρά από βασικές επεξεργασίες που αφορούν τους πίνακες.

6.1 Μονοδιάστατοι Πίνακες

Για την διαχείριση ενός πίνακα, χρειάζεται στο πρόγραμμά μας να έχουμε τουλάχιστον μια μεταβλητή μέσω της οποίας μπορούμε να προσπελάσουμε τον πίνακα και τα στοιχεία του. Στην Java, οι μεταβλητές μέσα από τις οποίες προσπελαύνουμε πίνακες είναι μεταβλητές αναφοράς και όχι τιμής, δηλ. αποθηκεύουν την διεύθυνση μνήμης στην οποία είναι κατανεμημένος ο πίνακας. Επιπλέον, τα στοιχεία ενός μονοδιάστατου πίνακα κατανέμονται σε διαδοχικές θέσεις στην μνήμη. Σε αυτό το χαρακτηριστικό οφείλεται η ταχύτατη προσπέλαση των στοιχείων ενός πίνακα.
Στο σχήμα 6.1 παρουσιάζουμε έναν πίνακα ακεραίων.



Σχήμα 6.1 *Αναπαράσταση μονοδιάστατου πίνακα ακεραίων*

Ο πίνακας του σχήματος 6.1 έχει μήκος 6 και περιέχει τους ακεραίους: 5, 12, 8, 1, 32, 7. Η μεταβλητή αναφοράς array περιέχει την διεύθυνση της αρχής του πίνακα. Οι θέσεις του πίνακα είναι αριθμημένες. Στην θέση 0, για παράδειγμα, είναι τοποθετημένη η τιμή 5, στην θέση 1 είναι τοποθετημένη η τιμή 12, κοκ.

Για να προσπελάσουμε ένα στοιχείο του πίνακα χρειαζόμαστε τρία στοιχεία. Την μεταβλητή αναφοράς που δείχνει στην αρχή του πίνακα, έναν δείκτη (index) που δείχνει την θέση μέσα στον πίνακα του στοιχείου που θέλουμε να προσπελάσουμε και τον τελεστή θέσης (index operator) που συμβολίζεται ως []. Σύμφωνα με τα παραπάνω, η έκφραση array[2] προσπελαύνει την θέση με δείκτη 2 του πίνακα array όπου είναι αποθηκευμένη η ακέραιη τιμή 8. Αντίστοιχα, array[3]==1, array[4]==32, κλπ.

Οι δείκτες των πινάκων στην Java αριθμούνται πάντα αρχίζοντας από το μηδέν (zero based). Έτσι, ένας πίνακας με μήκος ν έχει δείκτες από 0 έως ν-1. Ο πίνακας του σχήματος 6.1 για παράδειγμα, έχει μήκος 6 και δείκτες από 0 έως 5. Επομένως, αν επιχειρήσουμε να προσπελάσουμε το στοιχείο array[6] επιχειρούμε στην ουσία να προσπελάσουμε περιοχή έξω από την μνήμη που έχει δεσμευθεί για τον πίνακα. Τέτοια ενέργεια συνιστά λάθος χρόνου εκτέλεσης και παραγάγει κατάλληλη εξαίρεση.

Όπως αναφέραμε στην αρχή αυτής της ενότητας, οι πίνακες είναι οι ταχύτερες συλλογές. Πιο συγκεκριμένα, οι πίνακες είναι δομές τυχαίας προσπέλασης (random access), δηλ. δομές στις οποίες ο χρόνος που απαιτείται για την προσπέλαση οποιουδήποτε στοιχείου είναι σταθερός. Με άλλα λόγια, χρειαζόμαστε τον ίδιο χρόνο για να προσπελάσουμε ένα στοιχείο στην θέση 1000 ενός πίνακα με τον χρόνο που απαιτεί η προσπέλαση στην θέση 0 του πίνακα.

Στον πίνακα του σχήματος 6.1, το στοιχείο στην θέση 0 βρίσκεται στην διεύθυνση μνήμης array+0*Integer.SIZE και το στοιχείο στην θέση 1 βρίσκεται στην διεύθυνση μνήμης array+1*Integer.SIZE. Γενικότερα, το στοιχείο στην θέση k ενός μονοδιάστατου πίνακα ακεραίων βρίσκεται στην διεύθυνση αρχής του πίνακα + k*Integer.SIZE. Επομένως, για να υπολογισθεί η διεύθυνση ενός οποιουδήποτε στοιχείου του πίνακα, χρειαζόμαστε να υπολογίσουμε μόνο μια απλή έκφραση.

Αυτή η ευκολία είναι αποτέλεσμα της διαδοχικής κατανομής στην μνήμη των στοιχείων των μονοδιάστατων πινάκων. Το μειονέκτημα βέβαια της διαδοχικής κατανομής είναι πως μετά την δημιουργία δεν είναι δυνατόν να μεταβληθεί το μέγεθος του πίνακα. Ωστόσο, αυτό το μειονέκτημα, εύκολα μπορούμε να το παρακάμψουμε όπως θα δούμε παρακάτω στην άσκηση 6.5.7.

6.1.1 Δημιουργία μονοδιάστατων πινάκων

Σε αυτήν την ενότητα παρουσιάζονται οι τρόποι βασικής διαχείρισης των μονοδιάστατων πινάκων. Πιο συγκεκριμένα, θα δούμε πως δηλώνουμε έναν μονοδιάστατο πίνακα, πως τον αρχικοποιούμε και πως προσπελαύνουμε τα στοιχεία του.
Στον κώδικα 6.1, δηλώνουμε, δημιουργούμε και αρχικοποιούμε τα στοιχεία ενός μονοδιάστατου πίνακα ακεραίων.

```
1   int[] array;
2   array = new int[3];
3   array[0] = 1;
4   array[1] = 3;
5   array[2] = 5;
```

Κώδικας 6.1 *Δήλωση, δημιουργία και αρχικοποίηση μονοδιάστατου πίνακα ακεραίων*

Στην γραμμή 1 του κώδικα 6.1, δηλώνουμε την μεταβλητή array που ο τύπος της είναι πίνακας ακεραίων. Σε αυτήν την φάση δεν έχει δημιουργηθεί ακόμη ο πίνακας. Έχει απλά δηλωθεί μια μεταβλητή κατάλληλη για την διαχείριση μονοδιάστατων πινάκων ακεραίων. Αν θέλαμε μονοδιάστατους πίνακες πραγματικών αριθμών θα έπρεπε να δηλώσουμε double[] array. Γενικότερα, μπορούμε να δηλώσουμε πίνακες οποιουδήποτε τύπου στην Java.

Στην γραμμή 2, δημιουργείται ο πίνακας. Η δημιουργία του επιτυγχάνεται με την χρήση της λέξης-κλειδί new που συνοδεύεται από τον αριθμό 3 μέσα σε τετράγωνες παρενθέσεις (square brackets). Επομένως, η εντολή new int[3], δεσμεύει στην μνήμη τρεις διαδοχικές θέσεις που στην κάθε μία μπορεί να αποθηκευθεί ένας ακέραιος. Επιπλέον, στην συγκεκριμένη περίπτωση η new επιστρέφει μεταβλητή τύπου int[] την οποία εκχωρούμε στην μεταβλητή array που δηλώσαμε στην γραμμή 1. Στις γραμμές 3 έως 5, τοποθετούμε στις θέσεις 0, 1 και 2 τις τιμές 1, 3, 5, αντίστοιχα.
Στην συνέχεια, μπορούμε να προσπελάσουμε ένα προς ένα τα στοιχεία του πίνακα χρησιμοποιώντας μια απλή for, όπως δείχνει ο κώδικας 6.2.

```
for (int i = 0; i < array.length; i++) {
    System.out.println("Η τιμή στην θέση " + i + " είναι " + array[i]);
}
```

Κώδικας 6.2 *Εμφάνιση των στοιχείων μονοδιάστατου πίνακα ακεραίων*

Όπως φαίνεται στον κώδικα 6.2, προσπελαύνουμε τα στοιχεία του πίνακα στις θέσεις 0 έως array.length-1. Αυτό καταρχάς σημαίνει ότι η δομή τύπου int[] που επιστρέφει η new περιλαμβάνει εκτός από την διεύθυνση μνήμης στην οποία κατανεμήθηκε ο πίνακας και το μήκος του πίνακα. Παρότι ο συγκεκριμένος πίνακας είναι μήκους 3, συνίσταται η χρήση της length καθώς όπως θα δούμε στην ενότητα 7 είναι δυνατόν ένας πίνακας να περάσει παραμετρικά σε έναν κώδικα και η χρήση σταθεράς να μας οδηγήσει σε προβλήματα.
Αντίστοιχα με τους άλλους τύπους έτσι και για τους πίνακες δήλωση και δημιουργία μπορούν να γίνουν στην ίδια γραμμή όπως δείχνει ο κώδικας:

```
int[] array=new int[3];
```

Επίσης, παρέχεται η δυνατότητα, δήλωση, δημιουργία και αρχικοποίηση να πραγματοποιηθούν στην ίδια γραμμή όπως φαίνεται στον κώδικα που ακολουθεί.

```
int[] array=new int[]{1,3,5,7};
```

Τέλος, δήλωση, δημιουργία και αρχικοποίηση να πραγματοποιηθούν στην ίδια γραμμή με την βοήθεια ενός αρχικοποιητή (initializer). Ο κώδικας που ακολουθεί δημιουργεί έναν πίνακα πραγματικών με στοιχεία 2.5, 4.0, 6.8.

```
double[] table={2.5, 4.0, 6.8};
```

Στο παράδειγμα αυτό ο αρχικοποιητή είναι η παράσταση {2.5, 4.0, 6.8}.

6.1.2 Ενισχυμένη for

Όπως αναφέραμε στην ενότητα 5.2.3 περί του βρόχου for, υποστηρίζεται μια διευρυμένη μορφή της for που σχετίζεται με συλλογές όπως οι πίνακες.
Η γενική μορφή της ενισχυμένης (enhanced) for έχει ως εξής:

```
for (dataType item : collection) (
     statement(s)
}
```

Επομένως, στην ενισχυμένη for, ορίζουμε καταρχάς μια μεταβλητή, τοποθετούμε άνω και κάτω τελεία και στην συνέχεια μια μεταβλητή που αντιπροσωπεύει μια συλλογή, π.χ. ένα πίνακα. Για παράδειγμα:

```
for (double currentItem : table) {
     System.out.println(currentItem+" ");
}
```

Τον κώδικα αυτόν μπορούμε να τον διαβάσουμε ως εξής: Για κάθε ένα στοιχείο του πίνακα table που είναι τύπου double και ονομάζουμε currentItem εκτέλεσε τις προτάσεις μέσα στο μπλοκ που ακολουθεί. Σε κάθε επαναληπτικό βήμα, μέσα στο μπλοκ έχουμε πρόσβαση στο τρέχων στοιχείο του table. Εκείνο που πρέπει να προσέξουμε είναι ότι η πρόσβαση που έχουμε στα στοιχεία του table με αυτήν την for είναι μόνο για διάβασμα (read only). Με άλλα λόγια, η μεταβλητή currentItem είναι τοπική μεταβλητή της for. Σε κάθε επαναληπτικό βήμα, αντιγράφεται η τιμή ενός στοιχείου του table στην currentItem. Επομένως μέσα στο μπλοκ δεν προσπελαύνουμε άμεσα το στοιχείο του πίνακα αλλά μόνο ένα αντίγραφό του.
Με την βοήθεια των παραδειγμάτων που ακολουθούν διευκρινίζουμε αυτό το ζήτημα.

```
static void alterArray() { //Κώδικας 6.3
    int[] array = {1, 3, 5};
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
        array[i] += 1;
    }
    for (int i = 0; i < array.length; i++) {
        System.out.println(array[i] + " ");
    }
}
```

Κώδικας 6.3 *Μεταβολή των στοιχείων του array*

Στον κώδικα 6.3, αρχικοποιούμε καταρχάς το array. Στην συνέχεια, σε κάθε επαναληπτικό βήμα μιας κλασικής for, εμφανίζουμε ένα στοιχείο του array και αμέσως μετά προσαυξάνουμε το στοιχείο κατά 1. Έξω από την πρώτη επαναληπτική διαδικασία αναπτύσσεται μια δεύτερη που σκοπό έχει να εμφανίσει τα στοιχεία του array. Πράγματι την πρώτη φορά, εμφανίζονται οι αρχικές τιμές 1, 3, 5 και στην δεύτερη for οι τιμές αυξημένες κατά 1, δηλ. οι τιμές 2, 4, 6.

```
static void noAlteration() { //Κώδικας 6.4
    int[] array = {1, 3, 5};
    for (int cElement : array) {
        System.out.print(cElement + " ");
        cElement += 1;
    }
    System.out.println();
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
    System.out.println();
}
```

Κώδικας 6.4 *Μεταβολή της τοπικής μεταβλητής cElement*

Αντίθετα, ο κώδικας 6.4, εμφανίζει κατά την πρώτη for τις τιμές 1, 3, 5 αλλά εμφανίζει τις ίδιες τιμές και κατά την δεύτερη for. Αυτό συμβαίνει γιατί όπως εξηγήσαμε, η cElement δεν είναι παρά ένα αντίγραφο του τρέχοντος στοιχείου του array. Επομένως, η προσαύξηση κατά 1 που κάνουμε μέσα στην πρώτη for αφορά το αντίγραφο και όχι το στοιχείο του array. Όπως φαίνεται από την έξοδο του κώδικα, τα στοιχεία του πίνακα παραμένουν αμετάβλητα.

6.1.3 Δημιουργία αντιγράφων

Θα συζητήσουμε τώρα πως μπορούμε να παράγουμε αντίγραφα ενός πίνακα ή ενός τμήματος πίνακα. Καταρχάς θα πρέπει να αποσαφηνίσουμε τις επιπτώσεις που προκύπτουν από το γεγονός ότι οι μεταβλητές με τις οποίες διαχειριζόμαστε τους πίνακες είναι μεταβλητές αναφοράς.
Ας δούμε λίγο τον κώδικα 6.5.

```
1     int[] array = {1, 3, 5};
2     int[] table;
3     table = array;
4     array[1] = 9;
5     System.out.println(table[1]);
```

Κώδικας 6.5 *Εκχώρηση μεταξύ πινάκων*

Στον κώδικα 6.5, δημιουργούμε καταρχάς τον πίνακα ακεραίων array. Στην συνέχεια δηλώνουμε έναν πίνακα επίσης ακεραίων που ονομάζουμε table. Στην ουσία, η table είναι μια μεταβλητή αναφοράς τύπου int[]. Στο επόμενο βήμα, εκχωρούμε την array στην table. Εφόσον η array είναι μία δομή που διατηρεί την διεύθυνση και το μήκος του πίνακα που δημιουργήθηκε με την new, αυτές είναι οι πληροφορίες που αντιγράφονται στην table. Επομένως, η table δείχνει στον ίδιο πίνακα με αυτόν που δείχνει η array. Με άλλα λόγια αυτή η εκχώρηση δεν δημιουργεί αντίγραφο του πίνακα αλλά απλώς αντίγραφο της μεταβλητής διαχείρισης του πίνακα. Έτσι, όταν στην γραμμή 4 μεταβάλλουμε σε 9 την τιμή του στοιχείου στην θέση 1 του array και στην συνέχεια εμφανίζουμε το στοιχείο στη θέση 1 του table, βλέπουμε στην οθόνη μας να εμφανίζεται το 9. Είναι αποτέλεσμα του γεγονότος ότι και οι δύο μεταβλητές αναφέρονται στον ίδιο πίνακα στην μνήμη.
Επομένως, η προσέγγιση του κώδικα 6.5 δεν είναι κατάλληλη για την δημιουργία ενός αντιγράφου πίνακα. Ωστόσο είναι εύκολο να κάνουμε ένα κώδικα που δημιουργεί αντίγραφο ενός πίνακα.

```
int[] array = {1, 3, 5};
int[] table = new int[array.length];
for (int i = 0; i < array.length; i++) {
    table[i] = array[i];
}
array[1] = 9;
System.out.println(table[1]);
```

Κώδικας 6.6 *Δημιουργία αντιγράφου μονοδιάστατου πίνακα*

Στον κώδικα 6.6, δημιουργούμε καταρχάς τον μονοδιάστατο πίνακα ακεραίων, array. Αμέσως μετά, δημιουργούμε επίσης έναν μονοδιάστατο πίνακα ακεραίων που ονομάζουμε table και έχει μήκος ίσο με το μήκος του array. Στην συνέχεια, με την βοήθεια μιας for αντιγράφουμε ένα προς ένα τα στοιχεία του array στον table. Οι τελευταίες 2 γραμμές επιδεικνύουν ότι μεταβολές των στοιχείων του array δεν επηρεάζουν τα στοιχεία του table.

Η μέθοδος του κώδικα 6.6 για την δημιουργία πινάκων μας δίνει μεν σωστό αντίγραφο, έχει όμως ένα μειονέκτημα. Για την δημιουργία του αντιγράφου χρειάζεται να προσπελάσουμε ένα προς ένα, όλα τα στοιχεία των δύο πινάκων. Αυτό στην περίπτωση που έχουμε πίνακες με μεγάλο μήκος στο πρόγραμμά μας μπορεί να θεωρηθεί χρονοβόρο. Με δεδομένο ότι τα στοιχεία ενός πίνακα είναι κατανεμημένα σε διαδοχικές θέσεις της μνήμης, θα ήταν πολύ ταχύτερο αν μπορούσαμε να αντιγράψουμε όλο το τμήμα της μνήμης στο οποίο είναι κατανεμημένος ο αρχικός πίνακας. Η Java μας παρέχει δύο συναρτήσεις με τις οποίες μπορούμε να παράγουμε αντίγραφα πινάκων ή τμημάτων πινάκων χωρίς να χρειάζεται να προσπελάσουμε ένα προς ένα τα επιμέρους στοιχεία. Πρόκειται για την arraycopy στατική συνάρτηση της κλάσης System και την copyOfRange στατική συνάρτηση της κλάσης Arrays.
Η γενική μορφή της arraycopy έχει ως εξής:

```
public static void arraycopy(Object source, int posAtSource,
Object destination, int posAtDest, int len)
```

Όπως φαίνεται από την γενική μορφή, η συνάρτηση απαιτεί τις ακόλουθες παραμέτρους:

1. source είναι ο πίνακας του οποίου αντίγραφο θέλουμε να παράγουμε.
2. posAtSource είναι η θέση στον πίνακα source από την οποία επιθυμούμε να αρχίσει η αντιγραφή.
3. destination είναι ο πίνακας-αντίγραφο που θέλουμε να ενημερώσουμε.
4. posAtDest είναι η θέση στον πίνακα destination από όπου επιθυμούμε να αρχίσει η επικόλληση των στοιχείων του source.
5. len είναι το πλήθος των στοιχείων που επιθυμούμε να αντιγράψουμε.

Ο τύπος της πρώτης και της τρίτης παραμέτρου είναι Object. Ο τύπος αυτός παρουσιάζεται εκτεταμένα στην ενότητα 13.1.4. Για την ώρα, αρκεί να γνωρίζουμε ότι κατά την κλήση της arraycopy, μπορούμε να περάσουμε πίνακες στην θέση των παραμέτρων τύπου Object.

```
1   public static void arrayCopyDemo() { //Κώδικας 6.7
2       char[] copyFrom = {'d', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a',
't', 'e', 'd'};
3       char[] copyTo = new char[7];
4        System.arraycopy(copyFrom, 2, copyTo, 0, copyTo.length);
5        System.out.println(new String(copyTo));
   }
```

Κώδικας 6.7 *Παράδειγμα χρήσης της arraycopy*

Στην γραμμή 2 του κώδικα 6.7 ορίζουμε τον πίνακα χαρακτήρων copyFrom. Όπως φανερώνει το όνομά του είναι ο πίνακας που θα αντιγραφεί και πιο συγκεκριμένα μέρος του οποίου θα αντιγράψουμε στον πίνακα copyTo που δηλώνεται και δημιουργείται στην γραμμή 3. Προσέξτε πως το μήκος του copyFrom είναι 13 ενώ του copyTo είναι 7. Η επιλογή αυτή οφείλεται στην πρόθεσή μας να αντιγράψουμε μόνο ένα τμήμα του copyFrom. Αν θέλουμε να αντιγράψουμε ολόκληρο τον πίνακα τότε ο πίνακας προορισμού πρέπει να δημιουργηθεί σύμφωνα με το μήκος του πίνακα πηγής. Στην γραμμή 4, καλούμε την arraycopy για να πραγματοποιήσουμε την αντιγραφή. Στην κλήση αυτή περνάμε σαν πρώτο όρισμα τον copyFrom και σαν τρίτο τον copyTo. Το δεύτερο όρισμα σηματοδοτεί την θέση στον copyFrom στην οποία αρχίζουν τα δεδομένα που θέλουμε να αντιγράψουμε. Εφόσον έχουμε περάσει την τιμή 2, τα δεδομένα θα αρχίσουν να αντιγράφονται από την δεύτερη θέση του copyFrom, δηλ. από τον χαρακτήρα ‘c’. Το τελευταίο όρισμα έχει ως αποτέλεσμα το πλήθος των χαρακτήρων που θα αντιγραφεί να είναι 7. Επομένως, στον copyTo θα αντιγραφούν οι χαρακτήρες 'c', 'a', 'f', 'f', 'e', 'i', 'n'. Τέλος, στην γραμμή 5, χρησιμοποιούμε μια βολική μέθοδο για να εκτυπώσουμε τον πίνακα χαρακτήρων copyTo. Αντί να εκτυπώσουμε έναν-έναν τους χαρακτήρες του, προφανώς με έναν κατάλληλο βρόχο, δημιουργούμε ένα String με όρισμα τον copyTo. Όπως ήδη γνωρίζουμε, η print γνωρίζει πως ακριβώς να εμφανίσει δεδομένα τύπου String.

Σημαντικό είναι να προσέξουμε πως κατά την χρήση της arraycopy, πρέπει να δημιουργήσουμε έναν πίνακα με επαρκές μέγεθος για τα δεδομένα που θα αντιγραφούν. Στην περίπτωση που στην θέση του destination περάσουμε πίνακα μικρότερου μήκους από τα δεδομένα που θα αντιγραφούν θα προκληθεί σφάλμα χρόνου εκτέλεσης.

Ας δούμε τώρα την copyOfRange. Η διεπαφή (interface) της συνάρτησης για πίνακες ακεραίων έχει ως εξής:

```
public static int[] copyOfRange(int[] original, int from, int
to)
```

Το όρισμα int[] σημαίνει πως στην θέση εκείνη μπορούμε να περάσουμε οποιονδήποτε μονοδιάστατο πίνακα ακεραίων. Επίσης, το int[] πριν το όνομα της συνάρτησης σημαίνει πως η συνάρτηση θα επιστρέψει στο περιβάλλον κλήσης έναν μονοδιάστατο πίνακα ακεραίων.

Η συνάρτηση είναι υπερφορτωμένη (overloaded) για όλους τους θεμελιώδεις τύπους. Αυτό σημαίνει πως στην θέση του πίνακα ακεραίων μπορούμε να έχουμε πίνακα οποιουδήποτε θεμελιώδη τύπου. Περισσότερες λεπτομέρειες για την υπερφόρτωση συναρτήσεων στην ενότητα 7.2.

Επίσης, υποστηρίζεται μια γενίκευση (generic) της συνάρτησης η οποία μας επιτρέπει να αντιγράφουμε και πίνακες μη θεμελιωδών τύπων. Περισσότερα για τις γενικεύσεις στην ενότητα 17.

Συνοψίζοντας σημειώνουμε πως στην copyOfRange μπορούμε να περνάμε πίνακες οποιουδήποτε τύπου. Πάντα θα μας επιστρέψει έναν πίνακα ίδιου τύπου με αυτόν που δώσαμε ως όρισμα κατά την κλήση της.

Τα ορίσματα from και το καθορίζουν από ποιο στοιχείο θα αρχίσει η αντιγραφή και σε ποιο στοιχείο θα καταλήξει. Ωστόσο να δοθεί προσοχή στο γεγονός ότι το τελευταίο στοιχείο που θα αντιγραφεί είναι το στοιχείο στην θέση to-1 και όχι το στοιχείο στην θέση to.

```
    public static void copyOfRangeDemo() { //Κώδικας 6.8
        char[] copyFrom = {'d', 'e', 'c', 'a', 'f', 'f', 'e', 'i', 'n', 'a', 't',
'e', 'd'};
        char[] copyTo = Arrays.copyOfRange(copyFrom, 2, 9);
        System.out.println(new String(copyTo));
    }
```

Κώδικας 6.8 *Παράδειγμα χρήσης της copyOfRange*

Ο κώδικας 6.8 παρουσιάζει ένα παράδειγμα χρήσης της copyOfrange. Να υπογραμμίσουμε πως για να χρησιμοποιηθεί πρέπει να γίνει εισαγωγή της κλάσης Java.util.Arrays. Αυτό επιτυγχάνεται με την πρόταση import Java.util.Arrays; αμέσως μετά την δήλωση του πακέτου. Περισσότερα για την διαδικασία import στην ενότητα 7.8.

Αρχικά ορίζουμε τον πίνακα χαρακτήρων copyFrom. Στην συνέχεια καλούμε την copyOfRange με ορίσματα τον copyFrom, το 2 και το 9. Η copyOfRange θα δημιουργήσει και θα επιστρέψει έναν πίνακα χαρακτήρων με δεδομένα από την θέση 2 του copyFrom, δηλ. από τον χαρακτήρα 'c' έως και την θέση 8, δηλ. τον χαρακτήρα 'n'. Επομένως, η copyOfRangeDemo θα εμφανίσει την σειρά "caffein".

Προσέξτε ότι κατά την χρήση της copyOfRange δεν χρειάστηκε να δημιουργήσουμε πίνακα στον οποίον θα αντιγραφόταν τα δεδομένα. Τον πίνακα αυτόν δημιούργησε η copyOfRange και τον επέστρεψε στο πρόγραμμά μας. Επομένως, κατά την χρήση της copyOfRange δεν κινδυνεύουμε να κάνουμε λάθος με το μέγεθος του πίνακα-αντιγράφου όπως στην arraycopy.

6.2 Η κλάση Arrays

Εκτός από την copyOfRange, η κλάση Arrays περιλαμβάνει μια σειρά από στατικές συναρτήσεις που υποστηρίζουν ποικιλία επεξεργασιών των πινάκων. Παρόμοια με την copyOfRange, οι συναρτήσεις που συζητάμε εδώ είναι οργανωμένες κατά τέτοιο τρόπο ώστε στις περισσότερες περιπτώσεις επιτρέπουν την επεξεργασία πίνακα οποιουδήποτε τύπου. Σε αυτήν την ενότητα, θα παρουσιάσουμε μέρος των συναρτήσεων της Arrays. Η διεπαφή των συναρτήσεων παρουσιάζεται για πίνακες ακεραίων αλλά αντίστοιχα ισχύουν και για τους υπόλοιπους τύπους. Όπου υπάρχουν διαφορές επισημαίνονται σαφώς.

6.2.1 Ταξινόμηση

Όλοι οι θεμελιώδεις τύποι πλην του τύπου boolean έχουν στην Java μια φυσική σειρά (natural order), δηλ. για 2 τιμές ενός τύπου είναι προκαθορισμένο ποια είναι μεγαλύτερη και ποια μικρότερη. Οι τιμές boolean θεωρείται πως δεν διαθέτουν φυσική σειρά, δηλ. δεν θεωρείται εξορισμού πως το false είναι μικρότερο από το true ή το αντίθετο.

Η Java υποστηρίζει ταξινόμηση πινάκων όλων των θεμελιωδών, πλην του boolean, τύπων καθώς και των μη θεμελιωδών τύπων. Ωστόσο, για πολλούς μη θεμελιώδεις τύπους δεν υπάρχει φυσική σειρά. Αντίθετα η σειρά των μη θεμελιωδών τύπων πρέπει να καθοριστεί με ευθύνη του προγραμματιστή. Το θέμα αυτό θα συζητήσουμε στην ενότητα 14.1.2. Η συζήτηση εδώ θα περιοριστεί στην ταξινόμηση πινάκων θεμελιωδών τύπων. Η διεπαφή της συνάρτησης ταξινόμησης για πίνακες ακεραίων είναι:

```
public static void sort(int[] a)
```

Επομένως, αν επιθυμούμε να ταξινομήσουμε ένα πίνακα ακεραίων t αρκεί η κλήση

```
Arrays.sort(t);
```

Ίδια είναι η κλήση και για οποιονδήποτε άλλο θεμελιώδη τύπο, πλην του τύπου boolean.

6.2.2 Δυαδική αναζήτηση (Binary Search)

Κατά την αναζήτηση, ελέγχουμε έναν πίνακα για να διαπιστώσουμε αν περιλαμβάνει ή όχι μια τιμή. Αν η τιμή βρεθεί επιστρέφεται η θέση της στον πίνακα. Αν η τιμή δεν βρεθεί επιστρέφεται -1. Ειδικότερα, η δυαδική αναζήτηση εφαρμόζεται μόνο σε ταξινομημένους πίνακες. Η εφαρμογή της σε μη ταξινομημένους έχει απροσδιόριστα αποτελέσματα.
Η διεπαφή της συνάρτησης δυαδικής αναζήτησης για πίνακες ακεραίων έχει ως εξής:

```
public static int binarySearch(int[] a, int key)
```

Η binarySearch λαμβάνει ως παραμέτρους τον πίνακα στον οποίο επιθυμούμε να πραγματοποιήσουμε αναζήτηση και την τιμή που αναζητούμε. Επιστρέφει την θέση στον πίνακα του στοιχείου που αναζητούμε ή -1 σε περίπτωση που το στοιχείο δεν υπάρχει στον πίνακα. Επομένως αν έχουμε έναν μονοδιάστατο ταξινομημένο πίνακα ακεραίων, έστω t και ψάχνουμε για το στοιχείο schElement, η κλήση θα πρέπει να είναι του τύπου

```
int idx=Arrays.binarySearch(t, schElement);
```

6.2.3 Έλεγχος Ισότητας

Δύο πίνακες θεωρούνται ίσοι μεταξύ τους αν περιέχουν τα ίδια στοιχεία και με την ίδια σειρά.
Η διεπαφή της συνάρτησης ισότητας για πίνακες ακεραίων έχει ως εξής:

```
public static boolean equals(int[] a, int[] a2)
```

Επομένως, αν έχουμε τους πίνακες t1 και t2 και θέλουμε να τους συγκρίνουμε για ισότητα αρκεί η κλήση

```
boolean tEquals=Arrays.equals(t1, t2);
```

6.2.4 Αρχικοποίηση

Η αρχικοποίηση πινάκων με την ίδια τιμή για κάθε στοιχείο του πίνακα γίνεται με τις συναρτήσεις fill. Η διεπαφή της fill για πίνακες ακεραίων έχει ως εξής:

```
public static void fill(int[] a, int val)
```

Επομένως για να αρχικοποιήσουμε όλα τα στοιχεία ενός πίνακα ακεραίων, t, στην τιμή 1, αρκεί η κλήση

```
Arrays.fill(t,1);
```

6.2.5 Αναπαράσταση πίνακα ως String

Αν χρειαζόμαστε αναπαράσταση ενός πίνακα σαν αλφαριθμητική σειρά μπορούμε να χρησιμοποιήσουμε την ομάδα συναρτήσεων toString. Η διεπαφή της συνάρτησης για πίνακες ακεραίων έχει ως εξής:

```
public static String toString(int[] a)
```

Για παράδειγμα, αν θέλουμε να τυπώσουμε τον πίνακα ακεραίων t, αρκεί ο ακόλουθος κώδικας:

```
System.out.println(Arrays.toString(t));
```

6.3 Πολυδιάστατοι πίνακες

Όπως αναφέραμε στην αρχή αυτού του κεφαλαίου, η Java υποστηρίζει και πολυδιάστατους πίνακες. Θα παρουσιάσουμε εδώ κυρίως τους δυσδιάστατους πίνακες ενώ θα δώσουμε και παραδείγματα χρήσης πινάκων με περισσότερες διαστάσεις. Πάντως, ότι ισχύει για τους δυσδιάστατους πίνακες, ισχύει αναλογικά και για πίνακες περισσότερων διαστάσεων.
Ένας πίνακας δύο διαστάσεων ενός τύπου στην Java είναι ένας πίνακας κάθε στοιχείο του οποίου είναι ένας μονοδιάστατος πίνακας του ίδιου τύπου.

```
static void twoDimArray() { //Κώδικας 6.9
    int[][] store = {
        {1, 2, 3, 4},
        {5, 6, 7},
        {8, 9, 10, 11, 12}
    };
    System.out.println(Arrays.toString(store[0]));
    System.out.println(Arrays.toString(store[1]));
    System.out.println(Arrays.toString(store[2]));
}
```

Κώδικας 6.9 *Παράδειγμα δήλωσης και αρχικοποίησης πίνακα δύο διαστάσεων με αρχικοποιητή*

Στο παράδειγμα του κώδικα 6.9, δηλώνουμε και αρχικοποιούμε τον δυσδιάστατο πίνακα ακεραίων store. Ο πίνακας έχει δηλωθεί ως δυσδιάστατος καθώς την δήλωση του τύπου int ακολουθούν δύο σύνολα από τετραγωνικές παρενθέσεις. Αντίστοιχα μπορούμε να δηλώσουμε τρισδιάστατους πίνακες χρησιμοποιώντας τρία σύνολα τετραγωνικών παρενθέσεων, τετραδιάστατους με τέσσερα σύνολα, κοκ, χωρίς άνω όριο διαστάσεων. Στο συγκεκριμένο παράδειγμα, ο πίνακας store αρχικοποιήθηκε με την χρήση αρχικοποιητή.
Τα στοιχεία του store είναι τρείς μονοδιάστατοι πίνακες ακεραίων. Επομένως, το μήκος του store είναι τρία, store.length==3. Τα δε στοιχεία του είναι τα εξής: το store[0] είναι ο πίνακας {1, 2, 3, 4}, το store[1] είναι ο πίνακας {5, 6, 7} και το store[2] είναι ο πίνακας {8, 9, 10, 11, 12}. Προσέξτε πως κάθε ένας από τους πίνακες-στοιχεία του store έχει διαφορετικό μέγεθος.

```
static void createTwoDimWithNew() { //Κώδικας 6.10
    int[][] store = new int[3][];
    store[0] = new int[4];
    store[1] = new int[3];
    store[2] = new int[5];
    int data = 0;
    for (int i = 0; i < store.length; i++) {
        for (int j = 0; j < store[i].length; j++) {
            store[i][j] = 1 + data++;
        }
    }
    System.out.println(Arrays.toString(store[0]));
    System.out.println(Arrays.toString(store[1]));
    System.out.println(Arrays.toString(store[2]));
}
```

Κώδικας 6.10 *Παράδειγμα δήλωσης και αρχικοποίησης πίνακα δύο διαστάσεων με χρήση της new*

Στον κώδικα 6.10, δημιουργούμε έναν δυσδιάστατο πίνακα με την βοήθεια της new. Στην περίπτωση αυτή, η μία διάσταση, αυτή που καθορίζει τα στοιχεία του store, υποχρεωτικά δηλώνεται μέσω μιας σταθεράς. Η δεύτερη μπορεί να δηλωθεί ή να μην δηλωθεί. Αν δηλωθεί όμως, τότε υποχρεωτικά όλα τα στοιχεία-πίνακες

του store θα έχουν την διάσταση που δηλώθηκε κατά την εκτέλεση της new. Στο παράδειγμά μας, δεν έχουμε δηλώσει την δεύτερη διάσταση. Έτσι μπορούμε να δημιουργήσουμε στοιχεία-πίνακες του store ποικίλων μεγεθών. Στην περίπτωση αυτή, τα στοιχεία του store πρέπει να δημιουργηθούν ένα προς ένα με χρήση της new. Για τα στοιχεία αυτά δεν είναι δυνατή η χρήση αρχικοποιητή καθώς ο αρχικοποιητής είναι αποδεκτός μόνο στην γραμμή δήλωσης ενός πίνακα.

Στην συνέχεια, στον κώδικα 6.10, προσπελαύνουμε ένα προς ένα τα τελικά στοιχεία του store με μια εμφωλευμένη for προκειμένου να τους καταχωρήσουμε τιμές. Προσέξτε πως η εσωτερική for πηγαίνει μέχρι store[i].length. Συχνά παρατηρείται σε κώδικες αρχαρίων να χρησιμοποιείται το μήκος του δυσδιάστατου πίνακα ως άνω όριο της εσωτερικής for. Αυτό είναι προφανώς λανθασμένη πρακτική καθώς είναι δυνατό ο δυσδιάστατος πίνακας να έχει διαφορετικό μήκος από τα στοιχεία-πίνακες που περιέχει.

Αν όμως όλα τα στοιχεία-πίνακες ενός δυσδιάστατου πίνακα έχουν την ίδια διάσταση, τότε αν την δηλώσουμε, τα στοιχεία-πίνακες δημιουργούνται αυτόματα χωρίς την χρήση της new για καθένα από αυτά.

```
static void createArrayWithCommonLength() { //Κώδικας 6.11
    int[][] s = new int[3][3];
    int data = 0;
    for (int i = 0; i < s.length; i++) {
        for (int j = 0; j < s[i].length; j++) {
            s[i][j] = 1 + data++;
        }
    }
    System.out.println(Arrays.toString(s[0]));
    System.out.println(Arrays.toString(s[1]));
    System.out.println(Arrays.toString(s[2]));
}
```

Κώδικας 6.11 *Δημιουργία δυσδιάστατου πίνακα με σταθερό μήκος στοιχείων*

Στον κώδικα 6.11, τα στοιχεία του δυσδιάστατου πίνακα s έχουν όλα το ίδιο μήκος. Σε αυτήν την περίπτωση κατά την δημιουργία του s δηλώνουμε και την δεύτερη διάσταση οπότε ο s και τα στοιχεία του δημιουργούνται χωρίς να απαιτείται η αρχικοποιητή χρήση της new για κάθε στοιχείο ξεχωριστά.

Θα θεωρήσει κανείς πως σε αυτήν την περίπτωση δεν είναι αναγκαίο η εσωτερική for να αναφέρεται στο μήκος του τρέχοντος στοιχείου. Μια και το μήκος κάθε στοιχείου είναι ίδιο με το μήκος του s μπορεί να χρησιμοποιηθεί το μήκος του s και στην εσωτερική for. Αυτή η προσέγγιση είναι λανθασμένη. Η κατάσταση του πίνακα ανεξάρτητα από το πώς δημιουργήθηκε δεν είναι σταθερή. Είναι πιθανό να έχουν μεσολαβήσει μεταβολές από την δημιουργία του μέχρι την επεξεργασία του. Για παράδειγμα, μετά την δημιουργία του s είναι δυνατό να τρέξει κώδικας της μορφής s[1]=new int[7];. Ο κώδικας αυτός μεταβάλλει τον πίνακα στην θέση 1 του s. Για αυτόν τον λόγο, σε όλες τις περιπτώσεις, θα πρέπει να θυμόμαστε πως το μήκος ενός πίνακα, αξιόπιστα, το γνωρίζει μόνο ο ίδιος ο πίνακας.

Σε ποιες περιπτώσεις μπορούμε να αξιοποιήσουμε ένα πίνακα τριών διαστάσεων; Εξαρτάται από τα δεδομένα του προβλήματος που έχουμε να κωδικοποιήσουμε. Έστω, ότι έχουμε δεδομένα σχετικά με τις ώρες απασχόλησης ενός εργαζόμενου. Ας υποθέσουμε πως τα δεδομένα μας είναι δομημένα σε τέσσερις χρονικές περιόδους κάθε μία από τις οποίες αποτελείται από τρεις μήνες, κάθε ένας από τους οποίους περιλαμβάνει είκοσι δύο εργάσιμες ημέρες. Για κάθε μία από αυτές τις ημέρες, ο εργαζόμενος έχει εργαστεί έναν ορισμένο αριθμό ωρών. Τέτοιου τύπου δεδομένα είναι πολύ βολικό να φορτωθούν σε έναν τρισδιάστατο πίνακα. Για παράδειγμα, έστω ο πίνακας

```
int[][][] timesheet=new int[4][3][22];
```

Η πρώτη διάσταση αφορά τις περιόδους, η δεύτερη τον μήνα μιας περιόδου και η τρίτη, τις εργάσιμες ημέρες ενός μήνα. Επομένως, αν θέλουμε να προσπελάσουμε πόσες ώρες εργάστηκε ο εργαζόμενος την έβδομη ημέρα του δεύτερου μήνα της πρώτης περιόδου αρκεί να προσπελάσουμε το στοιχείο timesheet[0][1][6].

6.4 Βασικές επεξεργασίες

Συχνά απαιτείται να κάνουμε κάποιες βασικές εργασίες με πίνακες όπως να υπολογίσουμε το άθροισμα των στοιχείων ενός πίνακα αριθμών, να εντοπίσουμε το μεγαλύτερο ή το μικρότερο στοιχείο ενός πίνακα, να

πραγματοποιήσουμε σειριακή αναζήτηση ψάχνοντας για μια συγκεκριμένη τιμή σε έναν πίνακα, να ανακατέψουμε τα στοιχεία ενός πίνακα, να γεμίσουμε ένα πίνακα με τυχαίες τιμές. Σε αυτήν την ενότητα, θα δούμε τους κώδικες που μπορούν να αξιοποιηθούν για την υλοποίηση των αναφερόμενων επεξεργασιών.

6.4.1 Σειριακή αναζήτηση

Η σειριακή αναζήτηση σε αντίθεση με την δυαδική μπορεί να εφαρμοστεί σε αταξινόμητους πίνακες. Όπως φανερώνει το όνομά της ελέγχει ένα προς ένα τα στοιχεία του πίνακα συγκρίνοντάς τα με το στοιχείο που ψάχνουμε.
Ο κώδικας της σειριακής αναζήτησης είναι σχετικά απλός.

```
    public static void sequentialSearch() { //Κώδικας 6.12
        final int[] store = {4, 7, 9, 13};
        int schElement = 7;
        int idx = -1;
        for (int i = 0; i < store.length; i++) {
            if (store[i] == schElement) {
                idx = i;
                break;
            }
        }
        if (idx >= 0) {
            System.out.println("Ο ακέραιος " + schElement + " βρέθηκε στην θέση "
+ idx);
        } else {
            System.out.println("Ο ακέραιος " + schElement + " δεν βρέθηκε");
        }
    }
```

Κώδικας 6.12 *Σειριακή αναζήτηση σε μονοδιάστατο πίνακα ακεραίων*

Στον κώδικα 6.12, ορίζουμε καταρχάς τον πίνακα ακεραίων store και το στοιχείο για το οποίο θα ψάξουμε, το schElement. Επίσης, ορίζουμε μια ακέραιη μεταβλητή idx που αρχικοποιούμε στο -1. Αν κατά την αναζήτηση στον πίνακα βρεθεί το schElement, τότε στην idx θα εκχωρήσουμε την θέση του στον πίνακα οπότε η idx θα αποκτήσει τιμή μεγαλύτερη ή ίση με το 0. Έτσι θα γνωρίζουμε πως το στοιχείο βρέθηκε αλλά και που ακριβώς είναι τοποθετημένο. Αν με το τέλος της αναζήτησης το στοιχείο δεν έχει βρεθεί, η idx θα παραμείνει μικρότερη από το μηδέν σηματοδοτώντας την αποτυχία εύρεσης του στοιχείου. Όπως έχουμε ήδη δει, η σύμβαση σύμφωνα με την οποία επιστρέφεται -1 για ένα στοιχείο που δεν βρέθηκε σε ένα πίνακα χρησιμοποιείται και από την συνάρτηση binarySearch της κλάσης Arrays που παρουσιάστηκε στην 6.2.2.

Αφού λοιπόν ορίσουμε την idx μπαίνουμε σε ένα βρόχο for με την βοήθεια του οποίου ελέγχουμε ένα προς ένα τα στοιχεία του store συγκρίνοντάς τα με το schElement. Αν για κάποιο από αυτά επιβεβαιωθεί ισότητα, τότε το schElement βρέθηκε οπότε ενημερώνεται η idx και διακόπτεται η επαναληπτική διαδικασία.

Σε μια τέτοια συνάρτηση θα πρέπει να προσέξουμε ιδιαίτερα τον τύπο του πίνακα. Το πρότυπο του κώδικα 6.12 είναι κατάλληλο για πίνακες θεμελιωδών τύπων όχι όμως και για πίνακες μη θεμελιωδών τύπων όπως ο τύπος String. Στην περίπτωση αυτή ο τελεστής ισότητας == θα πρέπει να αντικατασταθεί με την συνάρτηση equals του τύπου του πίνακα όπως δείχνει ο κώδικας 6.13.

```
    public static void sequentialSearchReference() { //Κώδικας 6.13
        final String[] store = {"John", "Mary", "Jim", "Kelly"};
        String schElement = new String("Jim");
        int idx = -1;
        for (int i = 0; i < store.length; i++) {
            if (store[i].equals(schElement)) {
                idx = i;
                break;
            }
        }
        if (idx >= 0) {
```

```
            System.out.println("Το όνομα " + schElement + " βρέθηκε στην θέση " +
idx);
        } else {
            System.out.println("Το όνομα " + schElement + " δεν βρέθηκε");
        }
    }
```

Κώδικας 6.13 *Σειριακή αναζήτηση σε πίνακα αλφαριθμητικών σειρών*

6.4.2 Μέγιστο ή ελάχιστο στοιχείο

Ένα άλλο πρόβλημα είναι η αναζήτηση του μέγιστου στοιχείου ενός πίνακα. Πρόκειται για απλή επεξεργασία. Παραθέτουμε τον κώδικα υλοποίησης και στην συνέχεια τον σχολιάζουμε:

```
    public static void max() { //Κώδικας 6.14
        int[] t = {2, 6, 1, 9, 15};
        int max = t[0];
        int idx = 0;
        for (int i = 1; i < t.length; i++) {
            if (t[i] > max) {
                max = t[i];
                idx = i;
            }
        }
        System.out.println("Το μεγαλύτερο στοιχείο είναι το " + max + " και
βρίσκεται στην θέση " + idx);
```

Κώδικας 6.14 *Εύρεση του μεγαλύτερου στοιχείου μονοδιάστατου πίνακα*

Στον κώδικα 6.14, καταρχάς ορίζουμε τον πίνακα t. Στην συνέχεια, δύο βοηθητικές μεταβλητές, την max και την idx. Η max αρχικοποιείται στην τιμή του στοιχείου στην θέση μηδέν του πίνακα. Επίσης, η idx αρχικοποιείται ώστε να δείχνει στην θέση μηδέν. Ακολούθως, προσπελαύνουμε τα στοιχεία του πίνακα ένα προς ένα και όποτε βρίσκουμε τιμή μεγαλύτερη από την τρέχουσα τιμή του max ενημερώνουμε την max και την idx. Στο τέλος της διαδικασίας, η max έχει την μεγαλύτερη τιμή του πίνακα και η idx την θέση της.

Προσέξτε πως ο βρόχος for ξεκινάει από 1 και όχι από 0. Πράγματι, στο συγκεκριμένο πρόβλημα δεν υπάρχει λόγος να ξεκινήσει η σύγκριση από το 0. Αντίθετα, αν ξεκινούσε από το 0, κατά το πρώτο επαναληπτικό βήμα θα συγκρινόταν η max με το στοιχείο t[0]. Η max όμως σε αυτό το βήμα έχει τιμή ίση με το t[0] οπότε είναι περιττή η σύγκριση.

Στην περίπτωση που ο πίνακας έχει δύο στοιχεία ίσα μεταξύ τους και μεγαλύτερα από τα υπόλοιπα στοιχεία του πίνακα, ο κώδικας 6.14 εντοπίζει το στοιχείο στην μικρότερη θέση. Αν επιθυμούμε να βρούμε το στοιχείο στην μεγαλύτερη θέση αρκεί να τροποποιήσουμε την συνθήκη της if σε t[i]>=max.

Αντίστοιχα, η εύρεση του μικρότερου στοιχείου, βρίσκεται αν αντικαταστήσουμε την συνθήκη ελέγχου της if με την έκφραση t[i]<min ή t[i]<=min.

6.4.3 Άθροισμα

Ένα άλλο τυπικό πρόβλημα είναι ο υπολογισμός του αθροίσματος των στοιχείων ενός πίνακα. Πρόκειται επίσης για μια σειριακή προσπέλαση των στοιχείων του πίνακα και προσαύξηση ενός αθροιστή κατά την τιμή κάθε στοιχείου. Ο κώδικας 6.15 παρουσιάζει την λύση.

```
    public static void sum() { //Κώδικας 6.15
        int[] t = {2, 6, 1, 9, 15};
        int sum = 0;
        for (int i = 0; i < t.length; i++) {
            sum += t[i];
        }
        System.out.println("Το άθροισμα είναι " + sum);
    }
```

Κώδικας 6.15 *Άθροισμα στοιχείων μονοδιάστατου πίνακα*

6.4.4 Ανακατανομή

Σε πολλές εφαρμογές απαιτείται τυχαία ανακατανομή των στοιχείων ενός πίνακα. Για κάθε στοιχείο του πίνακα, υπολογίζουμε μια τυχαία θέση μέσα στα όρια του πίνακα και αντιμεταθέτουμε το στοιχείο στην τρέχουσα θέση με το στοιχείο στην τυχαία θέση.

```
public static void shuffle() { //Κώδικας 6.16
    int[] t = {2, 6, 1, 9, 15};
    Arrays.sort(t);
    System.out.println(Arrays.toString(t));
    Random r = new Random();

    for (int i = 0; i < t.length; i++) {
        int rIdx = r.nextInt(t.length);
        int tmp = t[i];
        t[i] = t[rIdx];
        t[rIdx] = tmp;
    }
    System.out.println(Arrays.toString(t));
}
```

Κώδικας 6.16 *Ανακατανομή στοιχείων πίνακα*

Στον κώδικα 6.16, δημιουργούμε το πίνακα t, τον ταξινομούμε και τον τυπώνουμε έτσι ώστε να είναι εμφανής η διαφορά από τον ανακατεμένο πίνακα. Στην συνέχεια υλοποιείται ο αλγόριθμος ανακατανομής που περιεγράφηκε αμέσως προηγουμένως.

6.4.5 Αρχικοποίηση με τυχαίες τιμές

Πολλές φορές χρειαζόμαστε να γεμίσουμε έναν πίνακα με τυχαίες τιμές είτε για λόγους ελέγχου είτε επειδή η εφαρμογή μας το απαιτεί. Στον κώδικα 6.17 ο πίνακας γεμίζει με τυχαίες τιμές από το 0 έως 99.

```
public static void fillRandom() { //Κώδικας 6.17
    int[] t = new int[20];
    Random r = new Random();
    for (int i = 0; i < t.length; i++) {
        t[i] = r.nextInt(100);
    }
    System.out.println(Arrays.toString(t));
}
```

Κώδικας 6.17 *Αρχικοποίηση πίνακα με τυχαίες τιμές*

6.4.6 Τελικοί πίνακες

Η δεσμευμένη λέξη final μπορεί να χρησιμοποιηθεί και με μεταβλητές που αναφέρονται σε πίνακες. Η σημασία της είναι ακριβώς ίδια με την σημασία που έχει όταν χρησιμοποιείται με μεταβλητές τιμής. Ωστόσο, επειδή οι μεταβλητές αναφοράς έχουν διαφορετική συμπεριφορά, συχνά παρατηρείται σύγχυση στον αρχάριο προγραμματιστή σε σχέση με τις επιπτώσεις του χαρακτηρισμού ως final ενός πίνακα.

Όπως αναφέρουμε στην ενότητα 3.1.2, ο χαρακτηρισμός ως final μιας μεταβλητής έχει σαν συνέπεια να επιτρέπεται εκχώρηση στην μεταβλητή μόνο μία φορά. Το ίδιο ακριβώς συμβαίνει και με τις μεταβλητές πινάκων. Επομένως, ο κώδικας που ακολουθεί παράγει λάθος χρόνου μεταγλώττισης.

```
final int[] t={1,3,5,7};
t=new int[9];
```

Ωστόσο, η μεταβολή των στοιχείων του πίνακα είναι διαφορετική υπόθεση. Αυτή μπορεί να γίνει χωρίς να μεταβληθεί η αναφορά στον πίνακα.

Επομένως, ο ακόλουθος κώδικας είναι απολύτως σωστός

```
final int[] t={1,3,5,7};
t[0]=10;
```

6.5 Λυμένες Ασκήσεις

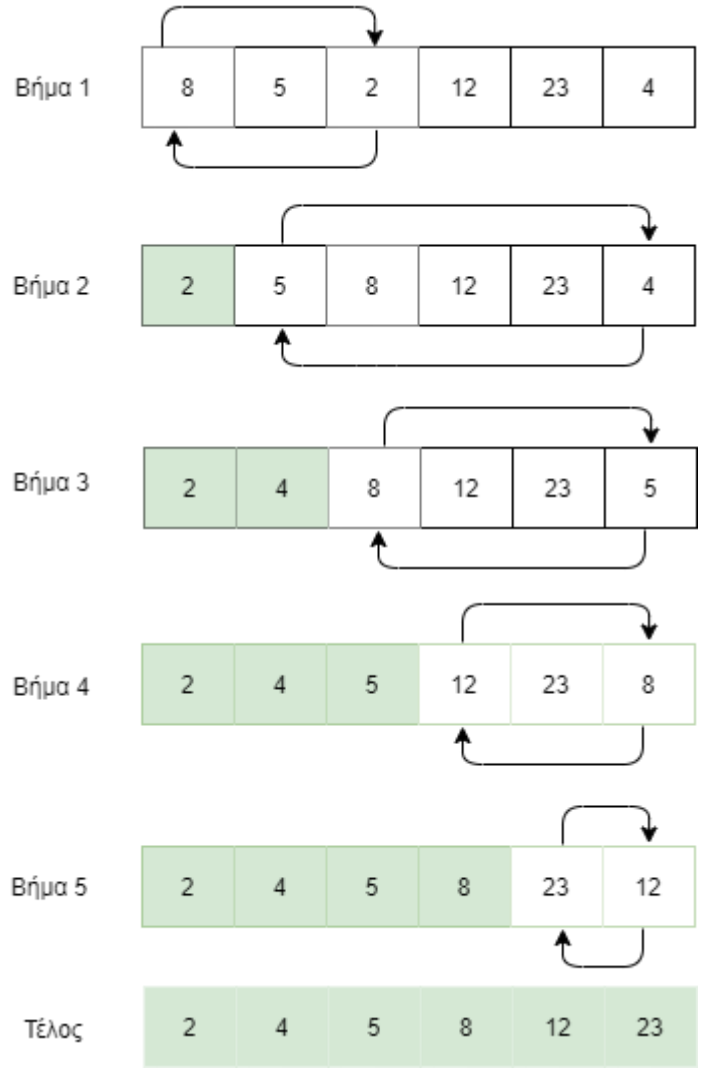
Παρουσιάζουμε εδώ μια σειρά από βασικές επεξεργασίες πινάκων

6.5.1 Ταξινόμηση

Να αναπτύξετε συνάρτηση στην οποία να ορίσετε έναν πίνακα ακεραίων και στην συνέχεια να τον ταξινομήσετε κατ' αύξουσα σειρά χωρίς να χρησιμοποιήσετε συνάρτηση sort της Arrays.

Λύση

Υπάρχουν πολλοί αλγόριθμοι ταξινόμησης δεδομένων. Για την λύση αυτής της άσκησης θα χρησιμοποιήσουμε ίσως τον απλούστερο όλων, γνωστό ως selection sort [1]. Το βασικό βήμα σε αυτήν την ταξινόμηση είναι η ανταλλαγή θέσης μεταξύ μικρότερου στοιχείου και του στοιχείου που είναι στην αρχή του αταξινόμητου μέρους του πίνακα. Το βήμα επαναλαμβάνεται από το πρώτο μέχρι το προτελευταίο στοιχείο του πίνακα. Με το τέλος της επανάληψης ο πίνακας είναι ταξινομημένος. Εξηγούμε λεπτομερέστερα με την βοήθεια του σχήματος 6.2.



Σχήμα 6.2 *Αναπαράσταση του selection sort*

Βήμα 1: Ολόκληρος ο πίνακας είναι αταξινόμητος. Το πρώτο στοιχείο του αταξινόμητου πίνακα βρίσκεται στην θέση 0 και έχει τιμή 8. Το μικρότερο στοιχείο του πίνακα βρίσκεται στην θέση 2 και έχει τιμή 2. Τα στοιχεία θα ανταλλάξουν θέσεις.

Βήμα 2: Μετά την ανταλλαγή θέσεων του βήματος 1, το μικρότερο στοιχείο βρίσκεται στην θέση 0. Το τμήμα του πίνακα που περιλαμβάνει την θέση 0 θεωρείται ταξινομημένο. Το μη ταξινομημένο τμήμα του πίνακα εκτείνεται από την θέση 1 έως το τέλος του πίνακα. Επαναλαμβάνεται, η λογική του βήματος 1 για το αταξινόμητο τμήμα του πίνακα. Το μικρότερο στοιχείο με τιμή 4 θα ανταλλάξει θέση με το στοιχείο με τιμή 5.

Βήμα 3: Με την ίδια λογική το 5 θα ανταλλάξει θέση με το 8.

Βήμα 4: Το 8 θα ανταλλάξει θέση με το 12

Βήμα 5: Το 12 θα ανταλλάξει θέση με το 23

Τέλος: Ο πίνακας είναι ταξινομημένος

Στην συνέχεια δίνουμε την υλοποίηση της selectionSort.

```
1   static void selectionSort(String[] args) { //Κώδικας 6.18
2       int[] tbl = {2, 1, 6, 3, 4, 9, 8, 5, 7, 4};
3       for (int i = 0; i < tbl.length - 1; i++) {
4           int min = tbl[i];
5           int idx = i;
6           for (int j = i + 1; j < tbl.length; j++) {
7               if (tbl[j] < min) {
8                   min = tbl[j];
9                   idx = j;
10              }
11          }
12
13          if (idx != i) {
14             int tmp = tbl[i];
15             tbl[i] = tbl[idx];
16             tbl[idx] = tmp;
17         }
18      }
19      System.out.println(Arrays.toString(tbl));
20  }
```

Κώδικας 6.18 *Υλοποίηση του selection sort*

Στην γραμμή 2 ορίζουμε ένα πίνακα ακεραίων. Στην γραμμή 3 ξεκινάμε την βασική επαναληπτική διαδικασία. Στις γραμμές 4 έως 11 αναζητούμε το μικρότερο στοιχείο του πίνακα από την θέση i και μετά. Αν το μικρότερο στοιχείο βρίσκεται σε θέση διαφορετική από το i, αντιμεταθέτουμε τα δύο στοιχεία (γραμμές 11 έως 15). Ο εξωτερικός βρόχος (γραμμή 3) ελέγχει μέχρι tbl.length-2. Αυτό είναι λογικό καθώς το tbl.length-1 είναι η τελευταία θέση του πίνακα και δεν έχει νόημα να ελέγξουμε αν μετά από αυτήν υπάρχει στοιχείο με μικρότερη τιμή. Στην γραμμή 19 εμφανίζουμε τον ταξινομημένο πίνακα.

6.5.2 Δυαδική αναζήτηση

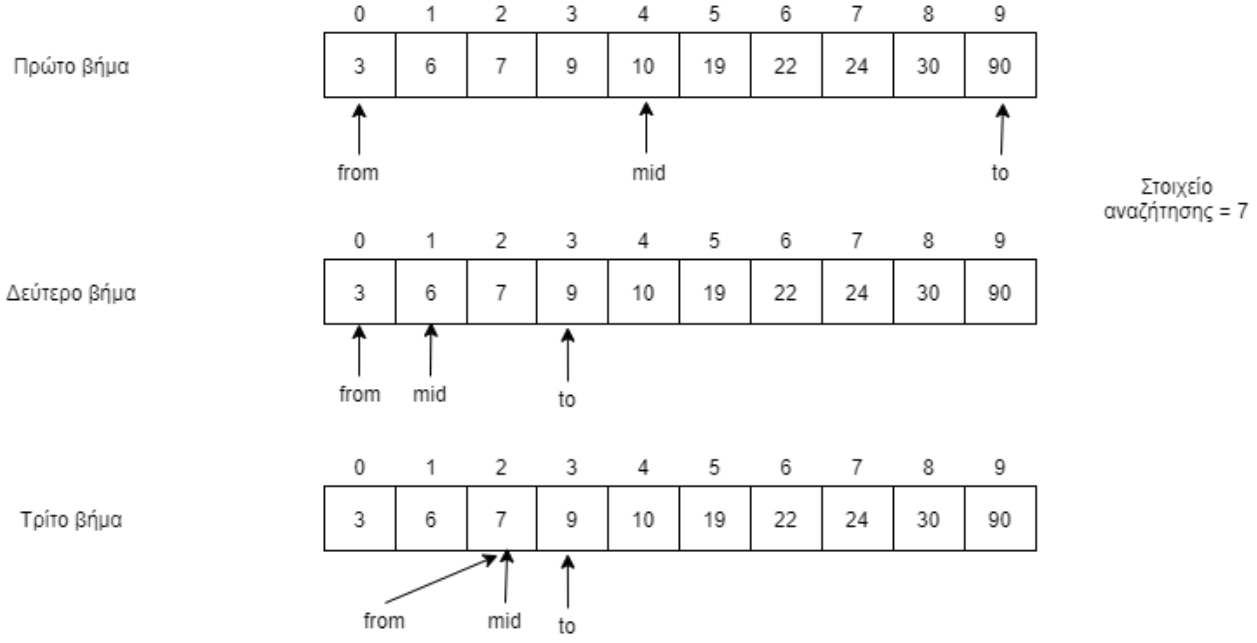
Να αναπτύξετε συνάρτηση Java στην οποία να ορίσετε έναν μονοδιάστατο πίνακα ακεραίων, έναν ακέραιο και στην συνέχεια να πραγματοποιήσετε δυαδική αναζήτηση στον πίνακα να βρείτε τον ακέραιο χωρίς να χρησιμοποιήσετε τις συναρτήσεις δυαδικής αναζήτησης της Arrays.

Λύση

Η βασική ιδέα της δυαδικής αναζήτησης (binary search) [2], είναι να εξετάζουμε κάθε φορά ένα τμήμα του πίνακα για το οποίο ελέγχουμε αν η τιμή που αναζητούμε είναι μεγαλύτερη, μικρότερη ή ίση από την τιμή στο μέσον του τμήματος. Αν είναι ίση, το στοιχείο αναζήτησης βρέθηκε. Αν είναι μικρότερη μας μένει να ψάξουμε μόνο στο πρώτο μισό μέρος του τμήματος του πίνακα. Αν είναι μεγαλύτερη μας μένει να ψάξουμε μόνο στο

δεύτερο μισό μέρος του τμήματος του πίνακα. Με αυτόν τον τρόπο, η δυαδική αναζήτηση υποδιπλασιάζει σε κάθε επαναληπτικό βήμα τα στοιχεία του πίνακα που μένουν προς έλεγχο. Κατά το πρώτο βήμα, το τμήμα του πίνακα που έχουμε να ψάξουμε συμπίπτει με ολόκληρο τον πίνακα.

Για να υλοποιήσουμε την δυαδική αναζήτηση χρειαζόμαστε πέραν του πίνακα και του στοιχείου αναζήτησης και δύο δείκτες, ας τους ονομάσουμε from και to, που καθορίζουν τα όρια του τμήματος που ελέγχουμε σε κάθε επαναληπτικό βήμα. Στο σχήμα 6.3 παρουσιάζεται παράδειγμα δυαδικής αναζήτησης.



Σχήμα 6.3 *Παράδειγμα δυαδικής αναζήτησης*

Στην αρχή της αναζήτησης ο δείκτης from δείχνει στην αρχή του πίνακα και ο δείκτης to στο τέλος. Συμπεριλαμβάνεται έτσι όλος ο πίνακας στην αναζήτηση. Με βάση τις τιμές των from και to υπολογίζεται το μέσο, mid, του πίνακα. Στην συνέχεια, εφόσον το στοιχείο αναζήτησης είναι μεγαλύτερο από την τιμή στο μέσον του πίνακα και ο πίνακας είναι ταξινομημένος κατ' αύξουσα σειρά, εξάγεται το συμπέρασμα πως το στοιχείο αναζήτησης, αν υπάρχει, θα πρέπει να αναζητηθεί στο δεύτερο τμήμα του πίνακα, δηλ. στο τμήμα που αρχίζει από mid+1 και το τέλος του συμπίπτει με το τέλος του πίνακα. Αντίθετα, αν όπως στο παράδειγμά μας, το στοιχείο αναζήτησης είναι μικρότερο από την τιμή στο μέσον του πίνακα, τότε η αναζήτηση θα πρέπει να συνεχίσει με το πρώτο μισό τμήμα του πίνακα, δηλ. να εφαρμοστεί στο τμήμα from έως mid-1.

Έτσι στο δεύτερο βήμα, ο δείκτης to γίνεται ίσος με mid-1 και άρα το νέο μέσο τοποθετείται στην θέση 1 του πίνακα. δείχνει μια θέση μετά το μέσον του πίνακα. Στην συνέχεια, το στοιχείο αναζήτησης είναι μεγαλύτερο από την τιμή στην θέση 1 οπότε ο δείκτης from γίνεται mid+1. Το νέο μέσο βρίσκεται στην θέση 2 όπου βρίσκεται και το στοιχείο αναζήτησης. Τέλος, ελέγχεται το μέσο και βρίσκεται ίσο με το στοιχείο αναζήτησης οπότε η αναζήτηση διακόπτεται.

Ο κώδικας 6.19, παρέχει την υλοποίηση σε Java του αλγόριθμου δυαδικής αναζήτησης.

```
public static void binarySearch() { //Κώδικας 6.19
    int[] tbl = new int[]{1, 3, 4, 5, 7, 9, 11, 23, 34, 50};
    int idx = -1;
    final int schElement = 24;
    int from = 0, to = tbl.length - 1, mid = (to - from) / 2;
    while (to >= from) {
        if (tbl[mid] == schElement) {
            idx = mid;
            break;
        } else if (schElement < tbl[mid]) {
            to = mid - 1;
        } else {
            from = mid + 1;
        }
```

```
            mid = from + (to - from) / 2;
        }
        if (idx == -1) {
            System.out.println("Element " + schElement + " was not found");
        } else {
            System.out.println("Element " + schElement + " found at position " +
idx);
        }
    }
```

Κώδικας 6.19 *Υλοποίηση δυαδικής αναζήτησης*

Στον κώδικα 6.19 καθορίζουμε καταρχάς τον πίνακα στον οποίο θα εφαρμόσουμε την δυαδική αναζήτηση. Στην συνέχεια ορίζουμε τις απαραιτητες βοηθητικές μεταβλητές idx, from, to και mid. Η μεταβλητή from αρχικοποιείται στο 0 και η to στο tbl.length-1. Έτσι, η αναζήτηση αρχικά αφορά όλο τον πίνακα. Αμέσως μετά, καθορίζουμε το στοιχείο αναζήτησης, schElement. Στην συνέχεια, η ροή εισέρχεται σε ένα βρόχο while που ελέγχεται από την συνθήκη to>=from. Όσο αυτή η συνθήκη είναι αληθής υπάρχει τμήμα του πίνακα που δεν έχει ελεγχθεί. Στην συνέχεια, αν η τιμή στο μέσο του πίνακα είναι ίση με την τιμή που αναζητούμε, το στοιχείο βρέθηκε οπότε ενημερώνεται η idx και διακόπτεται η while αλλιώς αν το στοιχείο αναζήτησης είναι μικρότερο από την τιμή στο μέσο του πίνακα ενημερώνεται η μεταβλητή to ώστε η αναζήτηση να περιοριστεί στο πρώτο μισό του πίνακα. Αν όμως το στοιχείο αναζήτησης είναι μεγαλύτερο από την τιμή στο μέσο του πίνακα ενημερώνεται η from ώστε η αναζήτηση να συνεχιστεί στο δεύτερο μισό του πίνακα . Στο επόμενο βήμα υπολογίζεται το νέο μέσο. Τέλος έξω από τον βρόχο while ενημερώνουμε με τα αποτελέσματα της αναζήτησης με την εμφάνιση κατάλληλων μηνυμάτων.

6.5.3 Ισότητα Πινάκων

Να αναπτύξετε συνάρτηση στην οποία να ορίσετε τους ακόλουθους πίνακες.

```
String[] tbl1 = new String[]{new String("John"), "Maria",
"Kely"};
String[] tbl2 = {"John", "Kely", "Maria"};
```

Στην συνέχεια να ελέγξετε αν είναι ίσοι μεταξύ τους με βάση τον ακόλουθο ορισμό ισότητας: Δύο πίνακες από Strings είναι ίσοι μεταξύ τους εφόσον έχουν το ίδιο μέγεθος και κάθε στοιχείο του ενός είναι και στοιχείο του άλλου ανεξάρτητα από την θέση που έχουν τα στοιχεία στους πίνακες.
Προσέξτε πως αυτός ο ορισμός ισότητας πινάκων διαφέρει από τον ορισμό στον οποίο βασίζονται οι συναρτήσεις equals της Array.

Λύση

```
public static void stringEquality() { //Κώδικας 6.20
    String[] tbl1 = new String[]{new String("John"), "Maria", "Kely"};
    String[] tbl2 = {"John", "Kely", "Maria"};
    boolean equal = tbl1.length == tbl2.length;
    for (int i = 0; i < tbl1.length && equal; i++) {
        boolean found = false;
        for (String cName : tbl2) {
            if (tbl1[i].equals(cName)) {
                found = true;
                break;
            }
        }
        equal = found;
    }
    System.out.println(equal);
}
```

Κώδικας 6.20 *Έλεγχος ισότητας πινάκων*

Στον κώδικα 6.20, ορίζουμε αρχικά τους πίνακες όπως δίνονται στην εκφώνηση. Στην συνέχεια, αρχικοποιούμε την μεταβλητή equal στην τιμή της έκφρασης tbl1.length == tbl2.length. Αν οι πίνακες δεν έχουν ίδιο μήκος η equal γίνεται false. Στην συνέχεια η ροή δεν μπαίνει στον βρόχο for και απλά εμφανίζει την τιμή false. Αν όμως τα μήκη των πινάκων είναι ίσα μεταξύ τους, τότε η ροή εισέρχεται στον εξωτερικό βρόχο for. Εκεί ελέγχεται κάθε στοιχείο του tbl1 αν υπάρχει και στον tbl2. Όσο ένα στοιχείο του tbl1 βρίσκεται και στον tbl2, η βοηθητική μεταβλητή found γίνεται true και αντίστοιχα ενημερώνεται και η equal. Αν ένα στοιχείο του tbl1 δεν βρεθεί στον tbl2, η found γίνεται false και διακόπτεται η εσωτερική for. Ακολούθως γίνεται false η equal η οποία θα διακόψει και την εξωτερική for.

6.5.4 Αντιστροφή πίνακα

Να γράψετε κώδικα ο οποίος με δεδομένο τον πίνακα int[] tbl = {1, 2, 3, 14, 5, 6, 7, 8, 9} παράγει έναν πίνακα που περιέχει τα ίδια στοιχεία με τον tbl σε αντεστραμμένη διάταξη, δηλ. στην θέση 0 έχει το 9 και στην θέση 8 το 1.

Λύση

```
public static void reverse() { //Κώδικας 6.21
    int[] tbl = {1, 2, 3, 14, 5, 6, 7, 8, 9};
    int[] reversed = new int[tbl.length];
    for (int i = 0; i < tbl.length; i++) {
        reversed[tbl.length - 1 - i] = tbl[i];
    }
    System.out.print(Arrays.toString(reversed));
}
```

Κώδικας 6.21 *Παραγωγή πίνακα με αντεστραμμένη διάταξη στοιχείων*

Καταρχάς δημιουργούμε τον πίνακα reversed με ίδιο μήκος με τον tbl. Στην συνέχεια τοποθετούμε στην θέση tbl.length - 1 – i του reversed το στοιχείο που βρίσκεται στην θέση i του tbl.

6.5.5 Συνένωση πινάκων

Να γράψετε κώδικα που με δεδομένους τους πίνακες int[] t1={1,2,3,4,5}, t2={6,7,8,9} παράγει ένα πίνακα που συνενώνει τους t1 και t2.

Λύση

```
public static void merge() { //Κώδικας 6.22
    int[] t1 = {1, 2, 3, 4, 5}, t2 = {6, 7, 8, 9};
    int[] rVal = new int[t1.length + t2.length];
    System.arraycopy(t1, 0, rVal, 0, t1.length);
    System.arraycopy(t2, 0, rVal, t1.length, t2.length);
    System.out.println(Arrays.toString(rVal));
}
```

Κώδικας 6.22 *Συνένωση πινάκων*

Αφού δηλώσαμε τους πίνακες t1 και t2, δημιουργούμε τον πίνακα rVal με μήκος ικανό να φιλοξενήσει τα δεδομένα των t1 και t2. Στην συνέχεια, με την arraycopy μεταφέρουμε τα δεδομένα από τους t1 και t2. Τέλος τυπώνουμε τον rVal ώστε να δείξουμε πως η συνένωση έγινε σωστά.

6.5.6 Εφαρμογή της arraycopy

Να γράψετε κώδικα που καταρχάς ορίζει τον πίνακα

```
char[][] copyFrom = {
     {'d', 'e'},
```

```
            {'c', 'a', 'f', 'f', 'e', 'i', 'n'},
            {'a', 't', 'e', 'd'}
        };
```

Στην συνέχεια, ορίστε έναν πίνακα, copyTo, ικανών διαστάσεων ώστε να μπορεί να φιλοξενήσει αντίγραφο του copyFrom. Στην θέση 2 του copyTo τοποθετήστε τους χαρακτήρες 'l', 'i', 'k', 'e'. Αντιγράψτε από τον copyFrom στον copyTo τους πίνακες στις θέσεις 0 και 1. Εμφανίστε τον copyTwo.

Λύση

```
    public static void copy() { ////Κώδικας 6.23
        char[][] copyFrom = {
            {'d', 'e'},
            {'c', 'a', 'f', 'f', 'e', 'i', 'n'},
            {'a', 't', 'e', 'd'}
        };
        char[][] copyTo = new char[3][];
        copyTo[2] = new char[]{'l', 'i', 'k', 'e'};
        System.arraycopy(copyFrom, 1, copyTo, 0, 2);
        for (char[] aElement : copyTo) {
            System.out.println(Arrays.toString(aElement));
        }
    }
```

Κώδικας 6.23 *Λύση της άσκησης 6*

6.5.7 Μεταβολή μεγέθους

Να υλοποιήσετε συνάρτηση στην οποία να δηλώνετε έναν μονοδιάστατο πίνακα ακεραίων 6 θέσεων, έστω t. Στην συνέχεια καταχωρείστε τους ακεραίους 1 έως 6 στον t. Ορίστε επίσης μια ακέραιη μεταβλητή, έστω newSize. Μεταβάλλεται το μήκος του t σε newSize. Φροντίστε ώστε ο κώδικάς σας να λειτουργεί σωστά είτε το newSize είναι μεγαλύτερο από το 6 είτε μικρότερο. Τυπώστε τον t ώστε να διαπιστώσετε πως έχει το αναμενόμενο μήκος.

Λύση

```
1   static void resize() { //Κώδικας 6.24
2       int[] t={1,2,3,4,5,6};
3       int newSize=4;
4       int[] tmp=new int[newSize];
5       System.arraycopy(t, 0, tmp, 0, t.length<newSize?t.length:newSize);
6       t=tmp;
7       System.out.println(Arrays.toString(t));
8   }
```

Κώδικας 6.24 *Μεταβολή μεγέθους πίνακα*

Ορίζουμε τον t και την newSize σύμφωνα με την εκφώνηση. Στην συνέχεια δημιουργούμε έναν άλλο πίνακα με μήκος newSize. Αμέσως μετά αντιγράφουμε δεδομένα από τον t στον tmp. Προσοχή σε αυτό το σημείο. Αν το newSize είναι μεγαλύτερο ή ίσο από το t.length, τότε μπορούν να αντιγραφούν όλα τα δεδομένα του t. Αν όμως είναι μικρότερο, τότε θα πρέπει να αντιγραφούν μόνο όσα χωρούν. Στον κώδικα 6.24, ο έλεγχος αυτός γίνεται με την βοήθεια του τριαδικού τελεστή στην γραμμή 5. Στην γραμμή 6, εκχωρούμε την αναφορά tmp στην t. Στην συνέχεια, η t δείχνει σε πίνακα μεγέθους newSize.

6.6 Ασκήσεις προς λύση

1. Να γράψετε κώδικα στον οποίο να ορίζετε τους δυσδιάστατους πίνακες ακεραίων 3x4, t1 και t2. Στην συνέχεια να δημιουργήσετε ένα πίνακα sum τέτοιον ώστε sum[i][j]=t1[i][j]+t2[i][j]. Εμφανίστε τον sum και ελέγξτε την ορθότητα του κώδικα.

2. Να γράψετε κώδικα που υπολογίζει ποια γραμμή δυσδιάστατου πίνακα πραγματικών έχει το μεγαλύτερο άθροισμα στοιχείων.

3. Να γράψετε κώδικα που υπολογίζει ποια στήλη δυσδιάστατου πίνακα πραγματικών έχει το μεγαλύτερο άθροισμα στοιχείων.

4. Να γράψετε κώδικα που αρχικοποιεί πίνακα ακεραίων 10x5 με τυχαίες τιμές από 0 έως 8. Τα δεδομένα σε κάθε γραμμή του πίνακα αναπαριστούν τις ώρες που εργάστηκε ένας εργαζόμενος κάθε μια από τις 5 εργάσιμες ημέρες μιας εβδομάδας. Υπολογίστε το σύνολο των ωρών που εργάστηκε κάθε εργαζόμενος κατά την διάρκεια της εβδομάδας. Εμφανίστε τα σύνολα ταξινομημένα κατά φθίνουσα σειρά.

5. Να γράψετε κώδικα που αρχικοποιεί ένα δυσδιάστατο πίνακα ακεραίων σε τυχαίες τιμές 0 και 1. Να υπολογίσετε πόσα 0 και πόσα 1 υπάρχουν σε κάθε γραμμή του πίνακα και πόσα σε όλον τον πίνακα.

6. Σε έναν δυσδιάστατο πίνακα από String καταχωρείστε σε κάθε γραμμή του ένα κράτος της Ευρώπης με την πρωτεύουσά του. Στην συνέχεια υποβάλλετε 10 ερωτήσεις στον χρήστη και αξιολογήστε τις απαντήσεις του. Κάθε ερώτηση μπορεί να δίνει την ονομασία του κράτους και να ζητά την πρωτεύουσα ή να δίνει την πρωτεύουσα και να ζητά το κράτος.

7. Να αναπτυχθεί συνάρτηση που λαμβάνει ως παράμετρο ένα πίνακα ακεραίων και επιστρέφει 1 αν ο πίνακας είναι ταξινομημένος κατ’ αύξουσα σειρά, -1 εν είναι ταξινομημένος κατά φθίνουσα σειρά και 0 αν δεν είναι ταξινομημένος.

8. Να δηλώσετε έναν πίνακα, a, από συμβολοσειρές. Να εκχωρήσετε μια συμβολοσειρά σε κάθε θέση του πίνακα. Στην συνέχεια, να δημιουργήσετε ένα αντίγραφο του a τέτοιο ώστε κάθε αλλαγή σε μια συμβολοσειρά του t να μην επηρεάζει την αντίστοιχη συμβολοσειρά στο αντίγραφο.

Βιβλιογραφία

[1] C. Horstmann, Big Java: Early Objects, 7th Edition | Wiley, 7th ed. Laurie Rosatone, 2013. Accessed: Sep. 14, 2021. [Online]. Available: https://www.wiley.com/en-us/Big+Java%3A+Early+Objects%2C+7th+Edition-p-9781119499091

[2] D. Liang, Εισαγωγή στον Προγραμματισμό Java, 10η. Θεσσαλονίκη: Τζιόλας.

Κεφάλαιο 7

Σύνοψη
Σε αυτήν την ενότητα γίνεται εισαγωγή στις στατικές συναρτήσεις. Παρουσιάζονται οι τυπικές και πραγματικές παράμετροι καθώς και η τιμή επιστροφής. Εξηγείται η διαφορά μεταξύ παραμέτρων τιμής και αναφοράς. Αναλύεται η λίστα παραμέτρων μεταβλητού μήκους και οι παράμετροι της main. Επίσης, συζητούνται τρόποι αξιοποίησης στατικών μεταβλητών ενώ γίνονται οι απαραίτητες επισημάνσεις για την πρόκληση τυχόν απροσδόκητων λαθών κατά την κλίση συναρτήσεων. Επιπλέον, παρουσιάζονται μερικές χρήσιμες προκαθορισμένες συναρτήσεις, συζητούνται οι προσδιοριστές προσπέλασης και αναλύεται η χρήση των πακέτων.

Προαπαιτούμενη γνώση
Οι θεμελιώδεις τύποι και βασική γνώση του τύπου String. Βασική είσοδος και έξοδος. Η λειτουργία των τελεστών. Οι δομές επιλογής και οι επαναληπτικές δομές. Η διαχείριση των πινάκων.

Λέξεις κλειδιά
Στατικές συναρτήσεις, στατικές μεταβλητές, Υπερφόρτωση, τυπική παράμετρος, πραγματική παράμετρος, παράμετρος τιμής, παράμετρος αναφοράς, λίστα παραμέτρων μεταβλητού μήκους, τιμή επιστροφής.

7. Στατικές Συναρτήσεις και Μεταβλητές

Είναι συχνό φαινόμενο, το ίδιο τμήμα κώδικα να είναι αναγκαίο σε διάφορα σημεία ενός προγράμματος. Ένας απλοϊκός τρόπος για να αντιμετωπισθεί η αναγκαιότητα αυτή είναι να επαναλάβουμε το ίδιο τμήμα όπου και όσες φορές αυτό χρειάζεται. Όμως η προσέγγιση αυτή παρουσιάζει σοβαρά μειονεκτήματα. Καταρχάς, καθιστά την συντήρηση του κώδικα πολλή δύσκολη. Αν χρειαστεί να κάνω μια βελτίωση ή οποιαδήποτε άλλη μεταβολή στο τμήμα του κώδικα που επαναλαμβάνεται, τότε θα πρέπει να βρω όλα τα σημεία στα οποία ενσωματώνεται και να τα μεταβάλλω ένα προς ένα. Φανταστείτε τώρα πόσο δυσχεραίνεται η συντήρηση του κώδικα αν στο πρόγραμμά σας έχετε πολλά τμήματα κώδικα που το καθένα χρειάζεται σε πολλά σημεία, κάτι που είναι πολύ συνηθισμένο στα σύγχρονα προγράμματα.

Ένα άλλο θέμα μάλλον σημαντικότερο και από την δυσκολία στην συντήρηση είναι η έλλειψη αφαιρετικότητας (abstraction). Ας υποθέσουμε πως έχουμε ένα τμήμα κώδικα που υπολογίζει την δύναμη ενός αριθμού υψωμένου σε μια δύναμη. Έστω κατά την ανάπτυξη της εφαρμογής μας, χρειαζόμαστε αυτόν τον υπολογισμό. Που είναι το τμήμα του κώδικα που τον κάνει; Θα πρέπει να ψάξουμε. Έστω ότι το βρήκαμε. Ναι, όμως εκείνο το τμήμα υπολογίζει την τιμή της μεταβλητής x υψωμένης στην τιμή της μεταβλητής k. Σε ένα άλλο σημείο χρειαζόμαστε τον ίδιο υπολογισμό αλλά για διαφορετικές μεταβλητές. Πρέπει λοιπόν να συντηρούμε σε πολλά σημεία όχι απλώς τον ίδιο κώδικα αλλά περίπου τον ίδιο. Οι μεταβολές όμως από σημείο σε σημείο ενδέχεται να εισάγουν λάθη.

Σίγουρα θα ήταν βολικό να κατασκευάζαμε ένα υποπρόγραμμα που θα είχε κατάλληλη ονομασία, π.χ. power ή calcPower έτσι ώστε όποτε χρειαζόμασταν να υπολογίσουμε την δύναμη ενός αριθμού, να το καλούσαμε να έκανε τους υπολογισμούς και να μας ενημέρωνε με το αποτέλεσμα. Ένα τέτοιο υποπρόγραμμα βέβαια δεν θα έπρεπε να υπολογίζει κάθε φορά την δύναμη του ίδιου αριθμού υψωμένου στον ίδιο εκθέτη. Επομένως θα έπρεπε να υπάρχει τρόπος κάθε φορά που το καλούμε να το ενημερώνουμε σχετικά με την βάση και τον εκθέτη του απαιτούμενου υπολογισμού.

Για καλή μας τύχη, τέτοιου είδους υποπρογράμματα υποστηρίζονται στην Java και ονομάζονται συναρτήσεις. Οι παράμετροι των συναρτήσεων είναι ο μηχανισμός μέσω του οποίου δίνουμε στην συνάρτηση τα δεδομένα στα οποία θα επενεργήσει ενώ ο κύριος τρόπος που η συνάρτηση μας επιστρέφει την απάντησή της είναι η τιμή επιστροφής της συνάρτησης.

Με αυτόν τον τρόπο επιτυγχάνουμε σημαντική αφαιρετικότητα στα προγράμματά μας. Δεν είναι αναγκαίο να γνωρίζουμε τις λεπτομέρειες του κώδικα μιας συνάρτησης για να την χρησιμοποιήσουμε. Αρκεί να γνωρίζουμε τι κάνει (και όχι πως το κάνει), το όνομά της, τις παραμέτρους που δέχεται και τι αποτελέσματα επιστρέφει. Έτσι είναι εύκολο να χρησιμοποιηθούν συναρτήσεις και μάλιστα από ένα ευρύτερο κοινό και όχι μόνο από αυτούς που τις έχουν υλοποιήσει.

Πέραν της τιμής επιστροφής της, μια συνάρτηση είναι δυνατό να παράγει αποτελέσματα και με άλλους τρόπους. Για παράδειγμα, μία συνάρτηση εισάγει σε ένα πίνακα μια σειρά από τιμές και επιστρέφει τον αριθμό

των τιμών που εισήχθησαν. Τα αποτελέσματα που παράγει η συνάρτηση πέραν της τιμής επιστροφής της είναι γνωστά ως παράπλευρα αποτελέσματα (side effects). Η Java όπως θα δούμε αμέσως παρακάτω σε αυτήν την ενότητα υποστηρίζει συναρτήσεις που παράγουν μόνο παράπλευρα αποτελέσματα χωρίς να επιστρέφουν κάποια τιμή. Άλλες γλώσσες διαφοροποιούν τα υποπρογράμματα που παράγουν μόνο παράπλευρα αποτελέσματα από αυτά που επιστρέφουν κάποια τιμή. Για παράδειγμα, η Pascal ονομάζει τα υποπρογράμματα που δεν επιστρέφουν κάποια τιμή διαδικασίες (procedures) ενώ διατηρεί την ονομασία συναρτήσεις (functions) για τα υποπρογράμματα που επιστρέφουν τιμή.

Επάνω στην δυνατότητα να αναπτύσσουμε συναρτήσεις βασίζεται η δυνατότητα να οργανώνουμε τους πηγαίους κώδικες με δομημένο τρόπο. Δημιουργείται έτσι το μοντέλο του δομημένου προγραμματισμού (structured programming). Πρόκειται για ένα σημαντικό μοντέλο προγραμματισμού που διευκολύνει την ανάπτυξη και συντήρηση των προγραμμάτων ενώ αποτελεί συστατικό στοιχείο του Αντικειμενοστρεφούς μοντέλου.

Η Java υποστηρίζει δύο ειδών συναρτήσεις, τις στατικές (static) και τις μη στατικές (non static). Το κύριο θέμα αυτού του κεφαλαίου είναι οι στατικές συναρτήσεις. Οι μη στατικές συναρτήσεις και μεταβλητές παρουσιάζονται αργότερα σε αυτό το εγχειρίδιο κατά την εισαγωγή στο Αντικειμενοστρεφές μοντέλο, στην ενότητα 11. Εδώ μαζί με την παρουσίαση των στατικών συναρτήσεων, παρουσιάζουμε και τις στατικές μεταβλητές της κλάσης.

7.1 Στατικές Συναρτήσεις

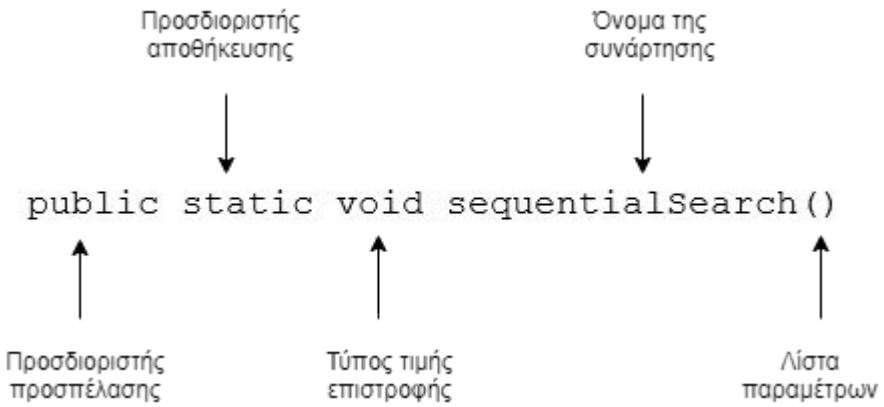
Έχουμε ήδη χρησιμοποιήσει στατικές συναρτήσεις στην πολλή απλή τους μορφή. Ας εξετάσουμε καταρχάς την συνάρτηση sequentialSearch του κώδικα 6.12. Το τμήμα

```
public static void sequentialSearch()
```

ονομάζεται επικεφαλίδα της συνάρτησης (function header). Το μπλοκ που ακολουθεί την επικεφαλίδα ονομάζεται σώμα της συνάρτησης (function body). Η επικεφαλίδα της συνάρτησης ονομάζεται και διεπαφή της συνάρτησης (function interface) γιατί είναι το τμήμα τις λεπτομέρειες του οποίου είναι αναγκαίο να γνωρίζει ο χρήστης της συνάρτησης για να είναι σε θέση να την καλεί και να λαμβάνει την επιστροφή της συνάρτησης. Η επικεφαλίδα μαζί με το σώμα της συνάρτησης αποτελούν τον ορισμό της συνάρτησης (function definition). Ο ορισμός της συνάρτησης από μόνος του δεν παράγει κάποιο αποτέλεσμα σε ένα πρόγραμμα. Το αποτέλεσμα το παράγει η κλήση της συνάρτησης. Για παράδειγμα η sequentialSearch χρειάζεται μια ακέραια μεταβλητή που ονομάζει schElement. Μνήμη για αυτήν την μεταβλητή δεσμεύεται μόνον κατά την κλήση της συνάρτησης.

Στην αρχή της επικεφαλίδας έχουμε έναν προσδιοριστή προσπέλασης, στην συγκεκριμένη περίπτωση τον προσδιοριστή public. Για τους προσδιοριστές προσπέλασης έχουμε ήδη μιλήσει στην ενότητα 2.3. Στην συνέχεια ακολουθεί ο προσδιοριστής αποθήκευσης (class storage specifier), static που καθορίζει πως πρόκειται για στατική συνάρτηση. Μία στατική συνάρτηση μπορεί να κληθεί μέσω της κλάσης στην οποία ορίζεται όπως φαίνεται στον κώδικα 2.6.

Στην συνέχεια ακολουθεί η δεσμευμένη λέξη void που ορίζει πως η συνάρτηση αυτή δεν έχει τιμή επιστροφής. Μετά ακολουθεί το όνομα της συνάρτησης που είναι sequentialSearch και τέλος πριν το σώμα της συνάρτησης παρεμβάλλεται η λίστα παραμέτρων που στο παράδειγμά μας είναι κενή. Στο σχήμα 7.1 παρουσιάζονται τα συστατικά μέρη της διεπαφής της συνάρτησης.



Σχήμα 7.1 *Η διεπαφή της συνάρτησης*

Συνεχίζοντας την εξέταση της sequentialSearch παρατηρούμε πως αυτή ψάχνει να εντοπίσει μια συγκεκριμένη τιμή σε έναν συγκεκριμένο πίνακα. Μάλιστα τιμή αναζήτησης και πίνακας ορίζονται μέσα στην ίδια την συνάρτηση. Αυτό σημαίνει πως η συνάρτηση αυτή δεν μπορεί να χρησιμοποιηθεί για κανέναν άλλο πίνακα αλλά ούτε και για να ψάξει άλλη τιμή έστω και στον ίδιο πίνακα. Ο αλγόριθμος όμως της σειριακής αναζήτησης είναι ίδιος για οποιονδήποτε μονοδιάστατο πίνακα και για οποιαδήποτε τιμή αναζήτησης. Επομένως, θα ήταν χρήσιμο να απομονώσουμε τον αλγόριθμο της σειριακής αναζήτησης κατά τρόπο ώστε να μπορούμε να τον εφαρμόζουμε σε οποιονδήποτε πίνακα και για οποιαδήποτε τιμή. Λύση σε αυτό το ζήτημα παρέχει η λίστα παραμέτρων της συνάρτησης. Η λίστα παραμέτρων δίνει την δυνατότητα σε αυτόν που καλεί την συνάρτηση να καθορίζει τόσο τον πίνακα όσο και την τιμή στα οποία επενεργεί η συνάρτηση. Όμως για να επιτευχθεί αυτό θα πρέπει κατά τον ορισμό της συνάρτησης να καθορίσουμε τους τύπους των παραμέτρων, όπως δείχνει παρακάτω η νέα διεπαφή της συνάρτησης.

```
public static void sequentialSearch(int[] array, int
schElement)
```

Η νέα διεπαφή επιβάλλει ώστε κατά την κλήση της συνάρτησης να δίνεται ένας πίνακας ακεραίων και μια ακέραιη τιμή. Είναι προφανές πως κάποιες μεταβολές θα πρέπει να γίνουν στο σώμα της sequentialSearch. Ας δούμε όμως πρώτα πως καλείται μια συνάρτηση με την νέα διεπαφή της sequentialSearch.

```
int[] t1 = {3, 2, 5, 3, 6, 9};
int[] t2 = {6, 4, 5, 2, 1, 9, 9};
int sItem = 5;
sequentialSearch(t1, 9);
sequentialSearch(t2, sItem);
```

Κώδικας 7.1 *Κλήση της sequentialSearch με παραμέτρους, χωρίς τιμή επιστροφής*

Στον κώδικα 7.1 βλέπουμε δύο κλήσεις της sequentialSearch. Κατά την πρώτη κλήση η συνάρτηση αναζητά στον πίνακα t1 την τιμή 9 και στην δεύτερη κλήση αναζητά στον πίνακα t2 την τιμή της μεταβλητής sItem, δηλ. την τιμή 5.

Ας σημειωθεί πως η παράμετροι στον ορισμό της συνάρτησης ονομάζονται τυπικές παράμετροι (formal parameters) ενώ οι παράμετροι που δίνει ο χρήστης της συνάρτησης κατά την κλήση της ονομάζονται πραγματικές παράμετροι (actual parameters).

Το πρόβλημα στον σχεδιασμό της συνάρτησης όπως τον έχουμε τώρα είναι πως η συνάρτηση δεν επιστρέφει στον κώδικα που την καλεί το αποτέλεσμα της αναζήτησης. Αυτό όμως είναι αναγκαίο καθώς στο αποτέλεσμα της αναζήτησης, ο κώδικας που την καλεί θα βασίσει την περαιτέρω ροή του. Για παράδειγμα, ενδέχεται αν η τιμή βρεθεί, ο χρήστης της συνάρτησης να επιθυμεί να αυξήσει την τιμή στον πίνακα κατά 1. Αν πάλι η τιμή δεν βρεθεί, μπορεί να μην χρειάζεται καμία περαιτέρω ενέργεια για τον συγκεκριμένο πίνακα.

Το πρόβλημα αυτό λύνεται με την τιμή επιστροφής της συνάρτησης. Στο συγκεκριμένο παράδειγμα μια καλή τιμή επιστροφής είναι τύπου int. Στην περίπτωση που θα βρεθεί η τιμή αναζήτησης, ο ακέραιος που θα επιστραφεί θα αναπαριστά την θέση της τιμής αναζήτησης στον πίνακα. Αν πάλι η τιμή δεν βρεθεί μπορεί να επιστραφεί η τιμή -1. Καθώς δεν υπάρχει θέση του πίνακα με αρνητική τιμή, η τιμή -1 μπορεί να αναπαραστήσει

με ασφάλεια την αποτυχία της αναζήτησης. Προσέξτε εδώ πως αν διαλέξουμε να αναπαραστήσουμε την αποτυχία αναζήτησης με μη αρνητικό αριθμό, π.χ. με 0, τότε έχουμε κακό σχεδιασμό της συνάρτησης. Αυτό γιατί η συνάρτηση θα επιστρέφει 0 τόσο στην περίπτωση που το στοιχείο αναζήτησης δεν βρεθεί όσο και στην περίπτωση που βρεθεί στην θέση 0 του πίνακα.

Επομένως η διεπαφή της συνάρτησης τροποποιείται όπως δείχνει ο ακόλουθος κώδικας.

```
public static int sequentialSearch(int[] array, int schElement)
```

Η τιμή επιστροφής έγινε int από void που ήταν στην προηγούμενη έκδοση. Τώρα η συνάρτηση μπορεί να κληθεί όπως δείχνει ο κώδικας 7.2.

```
static void callSeqSearch() { //Κώδικας 7.2
    int[] t1 = {3, 2, 5, 3, 6, 9};
    int sItem = 5;
    int idx = sequentialSearch(t1, sItem);
    if (idx > -1) {
        t1[idx]++;
    }
}
```

Κώδικας 7.2 *Κλήση της sequentialSearch με παραμέτρους και τιμή επιστροφής*

Στον κώδικα 7.2, η κλήση της συνάρτησης επιστρέφει την θέση του πίνακα στην οποία βρέθηκε το sItem ή -1. Αν επιστρέψει -1 καμία ενέργεια επάνω στον πίνακα δεν γίνεται. Αν όμως επιστρέψει μη αρνητική τιμή, τότε προσαυξάνεται η τιμή στην θέση που επιστράφηκε κατά 1.

Ακόμη και αν κάνουμε τις αλλαγές που συζητήσαμε μέχρι τώρα, παραμένει ένα σοβαρό μειονέκτημα στην συνάρτηση. Αυτό συνίσταται στο γεγονός ότι η συνάρτηση εμφανίζει τα αποτελέσματα της αναζήτησης. Προσέξτε, το μειονέκτημα δεν είναι ότι επιστρέφει στον κώδικα που την καλεί τα αποτελέσματα της αναζήτησης αλλά ότι τα εμφανίζει χρησιμοποιώντας μάλιστα την println. Αυτό το μειονέκτημα πολλές φορές γίνεται δύσκολα κατανοητό στους αρχάριους προγραμματιστές. Πράγματι αν κάνω ένα κώδικα που ψάχνει σε συγκεκριμένο πίνακα για να βρει μια συγκεκριμένη τιμή και το μόνο που θέλω είναι να μάθω αν η τιμή βρέθηκε και που, τότε δεν είναι πρόβλημα αν ο κώδικας εμφανίζει το αποτέλεσμα της αναζήτησης. Μια συνάρτηση όμως μπορεί να κληθεί σε ένα σημείο που θέλω να εμφανίσω το αποτέλεσμά της αλλά μπορεί να κληθεί και σε πολλά σημεία που δεν θέλω να το εμφανίζω αλλά απλώς το χρειάζομαι μέσα στον κώδικά μου. Για παράδειγμα, μπορεί την sequentialSearch να την καλέσω σε μια παραθυρική εφαρμογή. Σε αυτήν την περίπτωση η έξοδος με την println που είναι κατάλληλη μόνο για περιβάλλοντα γραμμής εντολών (command line) προφανώς θα μου καταστρέψει την διεπαφή χρήστη (user interface).

Ο κώδικας 7.3 δίνει τον ορισμό της sequentialSearch που ενσωματώνει όλες τις παρατηρήσεις μέχρι εδώ.

```
public static int sequentialSearch(int[] array, int sItem) {
    for (int i = 0; i < array.length; i++) {
        if (array[i] == sItem) {
            return i;
        }
    }
    return -1;
}
```

Κώδικας 7.3 *Συνάρτηση σειριακής αναζήτησης σε μονοδιάστατο πίνακα ακεραίων*

Η sequentialSearch σε αυτή την μορφή μπορεί να κληθεί από οποιονδήποτε κώδικα για να αναζητήσει σε οποιονδήποτε μονοδιάστατο πίνακα ακεραίων οποιαδήποτε ακέραιη τιμή. Προσέξτε πως η επιστροφή τιμής από την συνάρτηση γίνεται με χρήση της δεσμευμένης λέξης return. Η λέξη return συντάσσεται με έκφραση που ο τύπος της είναι υποχρεωτικά ίδιος ή συμβατός με τον τύπο της συνάρτησης. Η return προκαλεί άμεση έξοδο από την συνάρτηση. Σε συναρτήσεις τύπου void, η return συντάσσεται μόνη, ακολουθούμενη απλώς από τον χαρακτήρα ελληνικό ερωτηματικό (semicolon). Σε void συναρτήσεις, η return δεν είναι υποχρεωτική. Ωστόσο σε κάποιες περιπτώσεις είναι χρήσιμη.

Οι παράμετροι των συναρτήσεων στην Java είναι παράμετροι θέσης (positional parameters). Αυτό σημαίνει πως κατά την κλήση συνάρτησης, οι πραγματικές παράμετροι αντιστοιχίζονται με τις τυπικές παραμέτρους ανάλογα με την θέση που διατηρεί κάθε μία στην λίστα παραμέτρων.

Τέλος να επισημάνουμε πως οι συναρτήσεις στα πλαίσια του αντικειμενοστρεφούς προγραμματισμού ονομάζονται και μέθοδοι (methods).

7.2 Υπερφόρτωση Συναρτήσεων

Πάντως, είναι πιθανό να χρειάζεται να κάνουμε αναζήτηση τιμών και σε πίνακες που δεν είναι πίνακες ακεραίων. Ας υποθέσουμε πως χρειάζεται να κάνουμε αναζήτηση σε ένα πίνακα πραγματικών. Σε αυτήν την περίπτωση μπορούμε να υλοποιήσουμε μια άλλη συνάρτηση με παραμέτρους τύπου double[] και double. Το θέμα είναι πως παρότι αλλάζουν οι τύποι των δεδομένων, η λειτουργία παραμένει ίδια. Δεν θέλουμε για την ίδια λειτουργία να έχουμε συναρτήσεις με διαφορετικό όνομα. Παρόλα αυτά, για κάποιες γλώσσες προγραμματισμού, η μόνη λύση είναι να κάνουμε μια συνάρτηση που έχει διαφορετικό όνομα από την συνάρτηση που ψάχνει σε πίνακες ακεραίων. Αυτό όμως δεν είναι βολικό γιατί δεν θέλουμε να αναγκαζόμαστε να εφευρίσκουμε και να θυμόμαστε διαφορετικά ονόματα για την ίδια λειτουργία. Ευτυχώς, η Java όπως και οι περισσότερες σύγχρονες γλώσσες προγραμματισμού, επιλύουν αυτό το θέμα με την υπερφόρτωση συναρτήσεων (function overloading).

Με την υπερφόρτωση συναρτήσεων μπορούμε να ορίσουμε συναρτήσεις με το ίδιο όνομα αρκεί να διαφέρει η ταυτότητά τους (function signature). Η ταυτότητα μιας συνάρτησης αποτελείται από το όνομα της συνάρτησης και την λίστα παραμέτρων. Προσέξτε πως η τιμή επιστροφής δεν συμπεριλαμβάνεται στην ταυτότητα. Επομένως, μπορούμε να κάνουμε συναρτήσεις με το ίδιο όνομα αρκεί να διαφέρει η λίστα παραμέτρων. Ποια από δύο συναρτήσεις που έχουν το ίδιο όνομα θα κληθεί είναι μια απόφαση που λαμβάνει ο μεταγλωττιστής ανάλογα με τις πραγματικές παραμέτρους. Στον κώδικα 7.4 δίνουμε παράδειγμα ορισμού και χρήσης υπερφορτωμένων συναρτήσεων.

```
static void f(int i) { //Κώδικας 7.4
      System.out.println("f(int)");
  }

  static void f(double d) { //Κώδικας 7.4
      System.out.println("f(double)");
  }

  public static void main(String[] args) { //Κώδικας 7.4
      f(1);
      f(1d);
  }
```

Κώδικας 7.4 *Ορισμός και χρήση υπερφορτωμένων συναρτήσεων*

Στον κώδικα 7.4 και οι δύο συναρτήσεις που ορίζουμε ονομάζονται f. Η μία όμως δέχεται μια ακέραιη παράμετρο και η άλλη μια πραγματική παράμετρο. Στην main καλούμε την f την πρώτη φορά με ακέραιη παράμετρο και την δεύτερη με πραγματική. Η έξοδος του κώδικα είναι

```
f(int)
f(double)
```

Η έξοδος αυτή μας δείχνει πως την πρώτη φορά κλήθηκε η f που λαμβάνει την ακέραιη παράμετρο και την δεύτερη φορά κλήθηκε η f που λαμβάνει την πραγματική παράμετρο.
Έτσι αν θέλουμε να έχουμε και μια sequentialSearch ώστε να ψάχνει και σε πραγματικούς πίνακες δεν έχουμε παρά να την υπερφορτώσουμε όπως δείχνει ο κώδικας 7.5.

```
public static int sequentialSearch(double[] array, double sItem) {
    for (int i = 0; i < array.length; i++) {
        if (approximateEquals(array[i], sItem, 0.0001)) {
            return i;
```

```
            }
        }
        return -1;
    }
```

Κώδικας 7.5 *Σειριακή αναζήτηση σε πίνακες πραγματικών*

7.3 Παράμετροι

Έχουμε ήδη συζητήσει σχετικά με τις μεταβλητές αναφοράς και τις μεταβλητές τιμής. Η συμπεριφορά αυτών των δύο τύπων μεταβλητών είναι διαφορετική όταν περνάνε σαν παράμετροι σε συναρτήσεις. Σε αυτήν την ενότητα εξετάζουμε πρώτα τις μεταβλητές τιμής ως παραμέτρους και στην συνέχεια τις μεταβλητές αναφοράς.

7.3.1 Οι μεταβλητές τιμής ως παράμετροι

Ο κώδικας 7.6 παρουσιάζει δύο συναρτήσεις, την increment και την main. Μελετήστε τον κώδικα και επιχειρήστε να υποθέσετε την έξοδό του.

```
static void increment(int i) {
    i++;
    System.out.println(i);
}

public static void main(String[] args) {
    int i = 0;
    increment(i);
    System.out.println(i);
}
```

Κώδικας 7.6 *Παράδειγμα παραμέτρων τιμής*

Η συνάρτηση increment λαμβάνει ως παράμετρο έναν ακέραιο. Το αναγνωριστικό της παραμέτρου είναι i. Στην πρώτη γραμμή αυξάνει την τιμή της παραμέτρου κατά 1 και στην συνέχεια την τυπώνει.

Η main ορίζει μια ακέραιη μεταβλητή, επίσης με το αναγνωριστικό i και την αρχικοποιεί στο 0. Στην συνέχεια καλεί την increment με πραγματική παράμετρο το i. Μετά το πέρας της increment, η main τυπώνει την i.

Πρώτη σημαντική παρατήρηση είναι ότι η i της increment και η i της main δεν ταυτίζονται. Μπορεί να είναι και οι δύο τύπου int και να έχουν το ίδιο αναγνωριστικό, ωστόσο πρόκειται για διαφορετικές μεταβλητές. Εξάλλου θα μπορούσαν να έχουν διαφορετικά αναγνωριστικά. Αν μεταβάλλετε το αναγνωριστικό στην main, π.χ. το κάνετε j, θα διαπιστώσετε πως καμία διαφορά δεν θα έχετε στο αποτέλεσμα του κώδικα.

Αν τρέξουμε τον κώδικα θα διαπιστώσουμε πως η μεν increment τυπώνει 1, η δε main τυπώνει 0. Αυτή η συμπεριφορά εξηγείται ως εξής: Η main δεσμεύει χώρο στην μνήμη κατάλληλο για την μεταβλητή i, δηλ. χώρο κατάλληλο για έναν ακέραιο. Σε αυτήν την θέση μνήμης εκχωρείται η τιμή 0. Όταν καλείται η increment, ένα αντίγραφο της πραγματικής παραμέτρου περνάει στην τυπική παράμετρο. Στην συνέχεια η i της increment λειτουργεί σαν τοπική μεταβλητή. Η increment δηλ. δεσμεύει επίσης χώρο για έναν ακέραιο στον οποίο αντιγράφεται η τιμή της πραγματικής παραμέτρου. Από κει και μετά, η increment επενεργεί επάνω στην δική της μεταβλητή i που είναι εντελώς ανεξάρτητη από την i της main. Με το τέλος εκτέλεσης της increment, ο έλεγχος επανέρχεται στην main. Η μεταβλητή i της main έχει διατηρήσει την τιμή της και έτσι εμφανίζεται το 0.

Οι μεταβλητές όλων των θεμελιωδών τύπων στην Java είναι μεταβλητές τιμής και περνούν στις συναρτήσεις σαν παράμετροι τιμής.

7.3.2 Οι μεταβλητές αναφοράς ως παράμετροι

Είπαμε παραπάνω πως οι θεμελιώδεις τύποι περνάνε στις συναρτήσεις σαν παράμετροι τιμής. Το ίδιο ισχύει και για τις μεταβλητές αναφοράς. Ωστόσο η συμπεριφορά είναι διαφορετική. Μελετήστε τον κώδικα 7.7. Ποια είναι κατά την γνώμη σας η έξοδός του;

```
static void increment(int[] store) { //Κώδικας 7.7
    for (int i = 0; i < store.length; i++) {
        store[i]++;
    }
    System.out.println(Arrays.toString(store));
}

public static void main(String[] args) {
    int[] s = {1, 2, 3};
    increment(s);
    System.out.println(Arrays.toString(s));
}
```

Κώδικας 7.7 *Η μεταβλητή αναφοράς ως παράμετρος*

Η έξοδος του κώδικα 7.7 είναι

```
[2, 3, 4]
[2, 3, 4]
```

Αυτό σημαίνει πως ότι αλλαγές έγιναν στον πίνακα μέσα στην increment ισχύουν και έξω από αυτήν. Η εξήγηση έχει ως εξής: Η μεταβλητή s είναι μία αναφορά που περιέχει την διεύθυνση του πίνακα. Η αναφορά αυτή αντιγράφεται στην παράμετρο store της increment. Εφόσον η s είναι αντίγραφο της store, και οι δύο δείχνουν στην ίδια θέση μνήμης. Άρα η increment επενεργεί στον ίδιο πίνακα στον οποίο αναφέρεται η s. Ότι μεταβολή γίνει στα στοιχεία του πίνακα μέσω μιας αναφοράς σε αυτόν, ισχύει όταν τον προσπελάσουμε μέσω οποιασδήποτε άλλης αναφοράς.
Προσέξτε όμως μια άλλη επίπτωση. Μελετήστε τον κώδικα 7.8. Ποια είναι η έξοδός του κατά την άποψή σας;

```
static void assign(int[] store) { //Κώδικας 7.8
    int[] s = {4, 5, 6};
    store = s;
    System.out.println(Arrays.toString(store));
}

public static void main(String[] args) { //Κώδικας 7.8
    int[] table = {1, 2, 3};
    assign(table);
    System.out.println(Arrays.toString(table));
}
```

Κώδικας 7.8 *Η μεταβλητή αναφοράς ως παράμετρος τιμής*

Η έξοδος του κώδικα 7.8 έχει ως εξής:

```
[4, 5, 6]
[1, 2, 3]
```

Η έξοδος αυτή μας δείχνει πως μέσα στην assign, η store μεταβλήθηκε ώστε να δείχνει στον ίδιο πίνακα με την τοπική μεταβλητή s. Ωστόσο, η table συνεχίζει να αναφέρεται στον πίνακα που δημιουργήθηκε στην main. Στην πράξη, ένα αντίγραφο της table πέρασε στην store. Το αντίγραφο, δηλ. η store, ενημερώθηκε ώστε να δείχνει σε άλλο πίνακα, όμως η table έμεινε ανέπαφη. Το ίδιο ακριβώς αποτέλεσμα θα είχαμε βέβαια αν ονομάζαμε store την table.

Συνοψίζοντας, σε σχέση με τις μεταβλητές αναφοράς ως παραμέτρους, ισχύει το εξής στην Java: Όταν μια μεταβλητή αναφοράς περνάει σαν παράμετρος σε συνάρτηση, η συνάρτηση μπορεί να μεταβάλλει το περιεχόμενο της μνήμης στην οποία αναφέρεται η εν λόγω μεταβλητή, δεν μπορεί όμως να μεταβάλλει το περιεχόμενο της ίδιας της μεταβλητής.

7.3.3 Λίστα παραμέτρων μεταβλητού μήκους

Ας υποθέσουμε τώρα πως αναπτύσσουμε μια εφαρμογή στην οποία απαιτείται συχνά να υπολογίζουμε τον μέσο όρο μιας σειράς πραγματικών τιμών. Το θέμα όμως είναι πως σε ένα σημείο του κώδικα χρειάζεται να υπολογίσουμε τον μέσο όρο τεσσάρων πραγματικών τιμών, σε κάποιο άλλο σημείο χρειάζεται ο μέσος όρος επτά πραγματικών τιμών. Μία λύση θα ήταν να υλοποιήσουμε δύο συναρτήσεις. Η μία με 4 τυπικές παραμέτρους και η άλλη με επτά. Αν αργότερα χρειαστεί να υπολογίσουμε τον μέσο όρο πέντε πραγματικών τιμών τότε θα πρέπει να προσθέσουμε τον ορισμό κατάλληλης συνάρτησης με πέντε τυπικές παραμέτρους. Η αντιμετώπιση αυτή όμως είναι προβληματική. Σε όλες τις περιπτώσεις, ο αλγόριθμος είναι ο ίδιος και το μόνο που αλλάζει είναι το πλήθος των δεδομένων.

Ευτυχώς για μας, η Java προσφέρει μια αποτελεσματική δυνατότητα, την λίστα παραμέτρων μεταβλητού μήκους (variable length parameter list).

Ας δούμε στην πράξη πως υλοποιείται και πως αξιοποιείται μια συνάρτηση που υπολογίζει τον μέσο όρο μιας σειράς πραγματικών αριθμών.

```
    static double average(double... sequence) {
        double sum = 0;
        for (int i = 0; i < sequence.length; i++) {
            sum += sequence[i];
        }
        return sum / sequence.length;
    }

    public static void main(String[] args) {
        double d1 = 12, d2 = 13.5, d3 = 16.1;
        double av1 = average(d1, 14.5);
        double av2 = average(d1, d2, d3);
        double totalAv = average(av1, av2);
        DecimalFormat f = new DecimalFormat("###.##");
        System.out.println(f.format(av1) + " " + f.format(av2) + " " +
f.format(totalAv));
    }
```

Κώδικας 7.9 *Παράμετρος μεταβλητού μήκους*

Η συνάρτηση average στο κώδικα 7.9, περιέχει την παράμετρο μεταβλητού μήκους sequence. Οι τρεις τελείες που ακολουθούν τον τύπο double δηλώνουν πως πρόκειται για παράμετρο μεταβλητού μήκους. Μόνο μια παράμετρος μεταβλητού μήκους επιτρέπεται σε μία συνάρτηση. Αν μάλιστα η συνάρτηση έχει και άλλες παραμέτρους, τότε η παράμετρος μεταβλητού μήκους πρέπει να βρίσκεται τελευταία στην λίστα παραμέτρων.

Στην main, η average καλείται τρεις φορές. Την πρώτη με δύο πραγματικές παραμέτρους, μία μεταβλητή και μία σταθερά, την δεύτερη με τρεις πραγματικές παραμέτρους και την τρίτη φορά καλείται πάλι με δύο πραγματικές παραμέτρους.

Μελετώντας την υλοποίηση της average μπορούμε να διαπιστώσουμε πως οι παράμετροι περνάνε στην πράξη σε έναν πίνακα τύπου double. Εκείνο δηλαδή που συμβαίνει είναι πως ο μεταγλωττιστής τοποθετεί τις τιμές των πραγματικών παραμέτρων σε κατάλληλο πίνακα και περνάει μια αναφορά σε αυτόν τον πίνακα στην average. Έτσι μέσα στην συνάρτηση έχουμε πρόσβαση στις τιμές των πραγματικών παραμέτρων μέσω της average την οποία διαχειριζόμαστε κανονικά ως τυπική αναφορά σε μονοδιάστατο πίνακα πραγματικών. Εξαιτίας αυτού του μηχανισμού, μπορούμε να καλέσουμε την average με πραγματική παράμετρο ένα πίνακα πραγματικών όπως δείχνει ο κώδικας που ακολουθεί

```
double[] d={12,13.5,16.1};
double totalAv=average(d);
```

7.3.4 Οι παράμετροι της main

Μια εφαρμογή Java μπορεί να κληθεί από διάφορα περιβάλλοντα. Στην τυπική περίπτωση, οι εφαρμογές Java σε αυτό το βιβλίο καλούνται μέσα από το NetBeans. Ωστόσο, μια εφαρμογή Java μπορεί να κληθεί μέσα από

πολλά και διαφορετικά περιβάλλοντα. Για παράδειγμα μπορεί να κληθεί από το powershell των Windows ή μέσα από μια άλλη εφαρμογή Java ή μέσα από οποιοδήποτε λειτουργικό στο οποίο τρέχει η JVM.

Επιπλέον μια εφαρμογή θα πρέπει να έχει τρόπο ακριβώς όπως οι συναρτήσεις να λαμβάνει μια σειρά από παραμέτρους ώστε να διαφοροποιεί την συμπεριφορά της ανάλογα με τις τιμές των παραμέτρων. Ωστόσο τα περιβάλλοντα από τα οποία μπορεί να κληθεί μια εφαρμογή Java δεν υποστηρίζουν ακριβώς τους ίδιους τύπους δεδομένων που υποστηρίζει η Java. Προκύπτει επομένως η ανάγκη να υπάρχει μια διεπαφή ικανή να εκκινήσει μια εφαρμογή Java που να είναι ταυτοχρόνως συμβατή με τα διάφορα περιβάλλοντα κλήσης.

Την ανάγκη αυτή καλύπτει η συνάρτηση main. Η main αποτελεί την κύρια είσοδο σε κάθε εφαρμογή Java. Αυτό σημαίνει πως από οποιοδήποτε περιβάλλον και αν καλέσουμε μια εφαρμογή Java, η εικονική μηχανή θα ψάξει να βρει και να καλέσει την main. Στην συνέχεια η main μπορεί να καλέσει ανάλογα με τον σχεδιασμό της εφαρμογής οποιαδήποτε συνάρτηση είναι στην διάθεση της εφαρμογής. Επομένως υπάρχει η ανάγκη να παρέχεται ένας τρόπος με τον οποίο τα διάφορα περιβάλλοντα κλήσης να μεταβιβάζουν παραμέτρους στην main. Ένας τύπος δεδομένων διαθέσιμος σε όλα τα περιβάλλοντα είναι η αλφαριθμητική σειρά. Με την μορφή αλφαριθμητικής σειράς μπορεί να αναπαρασταθούν δεδομένα οποιουδήποτε τύπου. Για παράδειγμα, μια πραγματική τιμή μπορεί να αναπαρασταθεί σαν αλφαριθμητική σειρά. Η τιμή 2.53 ας πούμε, μπορεί να αναπαρασταθεί σαν “2.53”. Αυτός είναι ο λόγος που η main λαμβάνει ως παράμετρο ένα πίνακα από Strings. Μέσα από την παράμετρο αυτή, τα περιβάλλοντα κλήσης μπορούν να περάσουν τις παραμέτρους που απαιτούνται. Από την στιγμή που η main θα έχει τις τιμές των παραμέτρων με την μορφή αλφαριθμητικών τιμών, έχει την δυνατότητα να τις μετατρέψει στους κατάλληλους τύπους και να εργαστεί με αυτές χωρίς πρόβλημα.

Ας πάρουμε όμως τα πράγματα από την αρχή. Ας θυμηθούμε την διεπαφή της main.

```
public static void main(String[] args)
```

Καταρχάς, στην τυπική περίπτωση, η main δηλώνεται ως public. Αυτό είναι λογικό διαφορετικά θα ήταν αδύνατο να κληθεί από τρίτους. Βέβαια ο μεταγλωττιστής δεν θα διαμαρτυρηθεί αν η main δηλωθεί με πρόσβαση πακέτου. Ενδεχομένως μάλιστα σε κάποιες ειδικές περιπτώσεις, όπου η main καλείται μέσα από Java, μια τέτοια δήλωση να είναι χρήσιμη. Ούτε και για τους υπόλοιπους προσδιοριστές προσπέλασης διαμαρτύρεται ο μεταγλωττιστής. Ωστόσο σε όλες τις περιπτώσεις που η main θα τρέξει απευθείας από την εικονική μηχανή και όχι μέσα από κάποιον κώδικα Java, θα πρέπει να χρησιμοποιηθεί ο προσδιοριστής προσπέλασης public. Αν παρόλα αυτά επιχειρήσουμε να τρέξουμε μια εφαρμογή με μη δημόσια main, η εικονική μηχανή θα αποκριθεί με κατάλληλο μήνυμα στο οποίο θα αναφέρει ότι δεν βρίσκει main.

Επιπλέον, η main είναι δηλωμένη ως static. Πρόκειται για απαραίτητη δήλωση. Στην περίπτωση που θα δηλωθεί ως μη στατική συνάρτηση, η εικονική μηχανή θα αποκριθεί και πάλι πως δεν βρίσκει main. Η main ως κύρια είσοδος στην εφαρμογή πρέπει απαραίτητα να δηλωθεί ως στατική συνάρτηση καθώς κλήση των μη στατικών συναρτήσεων μπορεί να πραγματοποιηθεί μόνο μέσα από εφαρμογή Java και άρα έπεται της εκκίνησης της εφαρμογής.

Επίσης, η main πρέπει να είναι τύπου void. Για αυτήν την απαίτηση ίσως να μην υπάρχει αναγκαιότητα. Θα μπορούσε να επιστρέφει int αντί για void όπως η main της C. Στην Java όμως έγινε αυτή η σχεδιαστική επιλογή. Η εικονική μηχανή ψάχνει μια main τύπου void.

Ας δούμε όμως μια πρότυπη main με παραμέτρους.

```
public class MainParameters {

    private static void syntax() {
        System.out.println("MainParameters int String");
        System.exit(1);
    }

    public static void main(String[] prms) {
        if (prms.length != 2) {
            syntax();
        }
        int firstArgument = Integer.parseInt(prms[0]);
        System.out.println("first Argument is " + firstArgument
                + "\nSecond Argument is " + prms[1]);
```

```
        }
}
```

Κώδικας 7.10 *Παράδειγμα main με παραμέτρους*

Στον κώδικα 7.10 παρουσιάζουμε την κλάση MainParameters η οποία περιέχει main με παραμέτρους. Βέβαια όλες οι main που είναι κύριες είσοδοι στο πρόγραμμα διαθέτουν τυπική παράμετρο τύπου String[]. Όσες είδαμε όμως μέχρι εδώ δεν χρησιμοποιούν την παράμετρο. Αντίθετα, η MainParameters την χρησιμοποιεί.

Πιο συγκεκριμένα, η main εδώ χρειάζεται δύο παραμέτρους. Η πρώτη πρέπει να είναι τύπου int και η δεύτερη String. Οι τιμές των δύο παραμέτρων περνάνε στην main ως αλφαριθμητικές σειρές διαμέσου του πίνακα prms. Ας σημειωθεί εδώ πως το αναγνωριστικό prms δεν είναι παρά το αναγνωριστικό μιας παραμέτρου. Μπορεί ο προγραμματιστής που υλοποιεί την main να επιλέξει όποιο αναγνωριστικό κρίνει κατάλληλο. Συνήθως για την παράμετρο της main χρησιμοποιείται το αναγνωριστικό args.

Μέσα στον πίνακα prms λοιπόν αναμένει η main του κώδικα 7.10, δύο παραμέτρους. Η πρώτη που θα τοποθετηθεί στην θέση 0 του πίνακα θα αναπαριστά μια ακέραιη τιμή και η δεύτερη που θα τοποθετηθεί στην θέση 1 θα είναι μια αλφαριθμητική σειρά.

Το πρώτο πράγμα που θα πρέπει να ελέγξουμε στην main είναι αν όντως κλήθηκε με σωστές παραμέτρους. Πράγματι, στην πρώτη γραμμή της main ελέγχουμε αν η εφαρμογή κλήθηκε με 2 παραμέτρους. Αν όχι, καλούμε την συνάρτηση syntax.

Η syntax βγάζει ένα μήνυμα που ενημερώνει τον χρήστη πως η εφαρμογή μας καλείται με δύο παραμέτρους, μία τύπου int και μια τύπου String. Στην συνέχεια καλεί την System.exit με παράμετρο 1. Η System.exit προκαλεί άμεσο τερματισμό της εικονικής μηχανής και άρα και της εφαρμογής που την καλεί. Επομένως αν κληθεί η Syntax βγαίνει ένα μήνυμα στον χρήστη που υποδεικνύει τον ορθό τρόπο κλήσης και η εφαρμογή τερματίζεται καθώς χωρίς τις κατάλληλες παραμέτρους αδυνατεί να προχωρήσει.

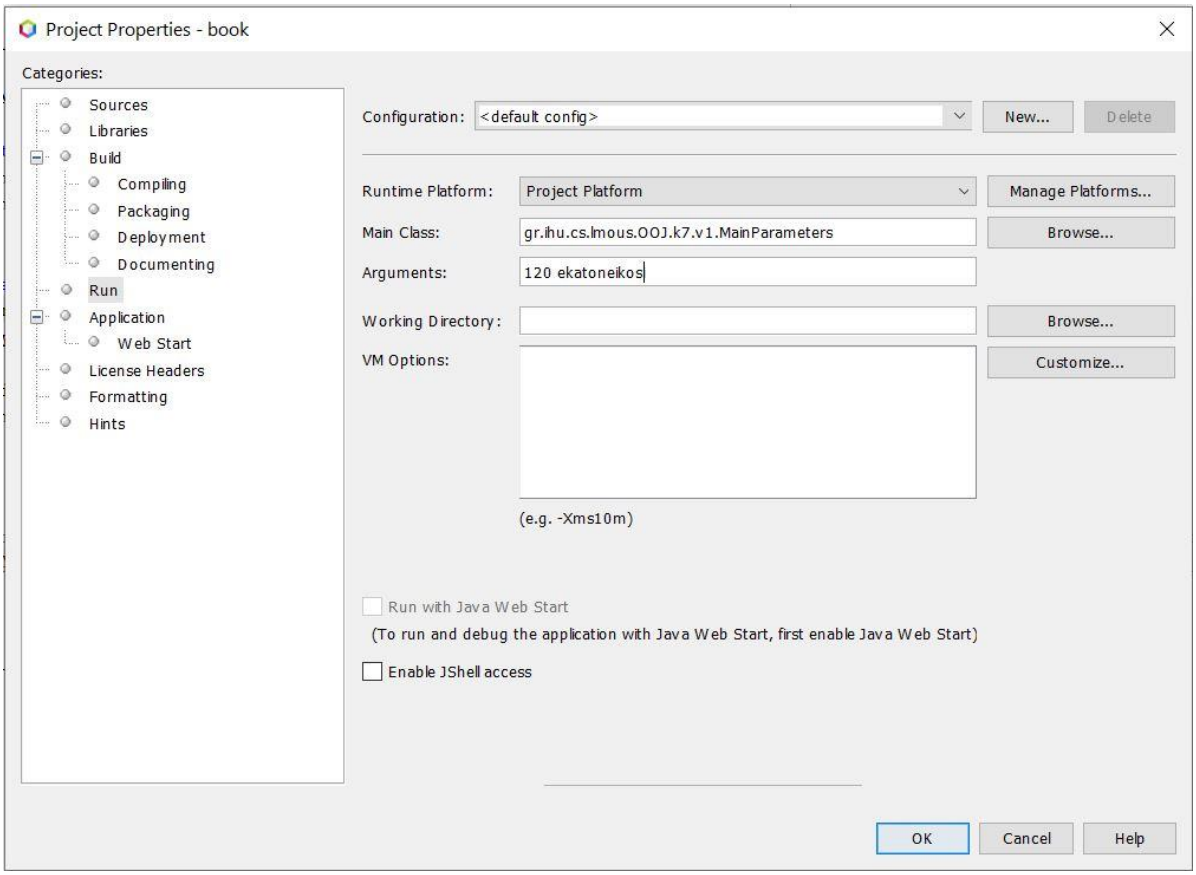
Αν δεν κληθεί η syntax, τότε η εφαρμογή επιχειρεί να μετατρέψει την πρώτη παράμετρο, δηλ. αυτή που βρίσκεται στην θέση μηδέν του prms σε int. Έτσι όπως είναι ο κώδικας 7.10, αν η πρώτη παράμετρος δεν είναι μετατρέψιμη σε int, το πρόγραμμα πέφτει παράγοντας κατάλληλο μήνυμα. Αν η μετατροπή των παραμέτρων γίνει με επιτυχία, τότε η main έχει τις τιμές των παραμέτρων σε τοπικές μεταβλητές και μπορεί να προχωρήσει απρόσκοπτα στην ενδεδειγμένη επεξεργασία. Στον κώδικα 7.10, η πρώτη παράμετρος μετατρέπεται σε int με την βοήθεια της Integer.parseInt και η τιμή της αποθηκεύεται στην τοπική μεταβλητή firstArgument. Η δεύτερη παράμετρος είναι τύπου String και δεν απαιτείται καμιά μετατροπή. Μπορεί να χρησιμοποιηθεί η prms[1].

Να επανέλθουμε λίγο στην System.exit και στην τιμή 1 που της περάσαμε ως πραγματική παράμετρο. Όπως έχουμε εξηγήσει οι συναρτήσεις διαθέτουν μια τιμή επιστροφής που σκοπό έχει να ενημερώνει τον κώδικα που κάλεσε την συνάρτηση με τα αποτελέσματα της κλήσης. Η main όμως είναι εξορισμού void. Η παράμετρος της System.exit αποτελεί έναν τρόπο για να ενημερώνουμε αυτόν που καλεί την εφαρμογή για το αποτέλεσμά της. Σε περίπτωση που η εφαρμογή έτρεξε ομαλά, η τιμή της παραμέτρου πρέπει να είναι μηδέν. Οποιαδήποτε άλλη τιμή μπορεί να χρησιμοποιηθεί για να επικοινωνήσει μη ομαλό τερματισμό της εφαρμογής. Από κει και μετά, οι πιθανές τιμές είναι θέμα σύμβασης μεταξύ της εφαρμογής και του περιβάλλοντος που την καλεί. Στα Windows για παράδειγμα, η τιμή της παραμέτρου της System.exit ενημερώνει την μεταβλητή συστήματος errorlevel. Όλα τα scripts των Windows που καλούν κάποια εφαρμογή Java, μπορούν να ελέγχουν την errorlevel και να λαμβάνουν πληροφορίες σχετικά με τον τερματισμό της εφαρμογής.

Ας δούμε όμως δύο παραδείγματα κλήσης της MainParameters. Πιο συγκεκριμένα, θα δούμε πως καλείται η MainParameters από το NetBeans και από το powershell των Windows.

7.3.4.1 Κλήση από το NetBeans

Στο τμήμα Projects του NetBeans, μεταβείτε στο project στο οποίο έχετε ορίσει την MainParameters και πατήστε δεξί κλικ. Θα ανοίξει ένα πτυσσόμενο μενού. Επιλέξτε Properties. Θα ανοίξει η φόρμα Project Properties.



Εικόνα 7.1 *Καθορισμός παραμέτρων της main στο NetBeans*

Στην αριστερή πλευρά της φόρμας Project Properties είναι τοποθετημένη μία λίστα κάτω από την επικεφαλίδα Categories. Στην λίστα Categories, επιλέξτε Run όπως φαίνεται στην εικόνα 7.1. Στην δεξιά πλευρά της φόρμας βρείτε τα πεδία Main Class και Arguments και ενημερώστε τα κατάλληλα. Στο πεδίο Main Class εισάγετε την κλάση MainParameters με τον προσδιοριστή πακέτου όπως φαίνεται στην εικόνα 7.1. Ο προσδιοριστής πακέτου έχει το όνομα του πακέτου στο οποίο ανήκει η κλάση. Μπορείτε να χρησιμοποιήσετε το button Browse για διευκόλυνσή σας. Στο πεδίο Arguments, τοποθετήστε τις παραμέτρους που απαιτεί η εφαρμογή διαχωρισμένες με ένα διάστημα. Πατήστε το button OK ώστε να αποθηκευτούν οι νέες ιδιότητες.

Μεταβείτε στο project που έχετε ορίσει την κλάση, πατήστε δεξί κλικ και από το πτυσσόμενο μενού, επιλέξτε Run. Στο παράθυρο Output του NetBeans θα δείτε να εμφανίζεται το αποτέλεσμα κλήσης της MainParameters.

7.3.4.2 Κλήση από το powershell

Μεταβείτε στο Project Properties. Στο παράθυρο Categories, επιλέξτε Sources. Επιλέξτε την διαδρομή στην οποία είναι αποθηκευμένο το project σας από το πεδίο Project Folder. Χρησιμοποιήστε Ctrl+c για να αντιγράψετε την διαδρομή του project στο πρόχειρο των Windows (clipboard).

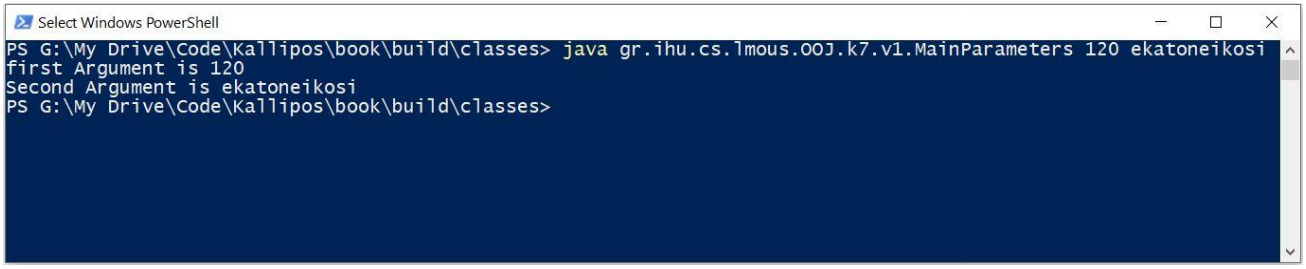
Ανοίξτε τον file explorer των Windows. Στην γραμμή εντολών δώστε Ctrl+V και Enter. Θα μεταβείτε έτσι στον φάκελο του project. Εκεί θα δείτε μια σειρά από φακέλους. Ο φάκελος build περιέχει φάκελο classes. Στον φάκελο classes είναι αποθηκευμένος ο bytecode της εφαρμογής σας οργανωμένος σε υποφακέλους ανάλογα με το πακέτο στο οποίο ανήκει η κάθε κλάση.

Μεταβείτε στον φάκελο classes. Ενώ βρίσκεστε στον φάκελο classes, πηγαίνετε στο μενού file του file explorer και κάντε κλικ στο Open Windows PowerShell. Θα ανοίξει το PowerShell με φάκελο εργασίας τον φάκελο classes.

Εκεί πληκτρολογήστε

```
Java packagePath.MainParameters first-argument second-argument
```

Στην θέση packagePath βάλτε το όνομα του πακέτου στο οποίο ανήκει η MainParameters. Η εικόνα 7.2 παρουσιάζει μια κλήση της MainParameter όταν αυτή έχει δηλωθεί να ανήκει στο πακέτο gr.ihu.cs.lmous.OOJ.k7.v1.



Εικόνα 7.2 *Κλήση εφαρμογής Java από το PowerShell των Windows.*

Όπως φαίνεται στην εικόνα 7.2, σε αυτό το παράδειγμα, η MainParameters καλείται με πρώτη παράμετρο το 112 και δεύτερη παράμετρο την σειρά ekatondwdeka. Η λέξη Java στην αρχή της κλήσης ενεργοποιεί την JVM η οποία στην ουσία τρέχει την κλάση MainParameters.

7.4 Στατικές μεταβλητές

Η Java μας δίνει την δυνατότητα να ορίσουμε στατικές μεταβλητές. Οι στατικές μεταβλητές ορίζονται στο επίπεδο της κλάσης και προσπελαύνονται μέσα από οποιαδήποτε συνάρτηση της κλάσης. Δεν υπάρχει δυνατότητα ορισμού στατικών μεταβλητών που είναι τοπικές μεταβλητές συναρτήσεων. Η πλήρης έννοια των στατικών μεταβλητών μπορεί να αναπτυχθεί μόνο στο πλαίσιο του Αντικειμενοστρεφούς μοντέλου όπου θα συζητηθούν και οι μη στατικές μεταβλητές και θα γίνει διαφοροποίηση μεταξύ στατικών και μη στατικών μεταβλητών.

Για την ώρα θα δούμε την χρησιμότητα των στατικών μεταβλητών μέσα από ένα παράδειγμα αξιοποίησης των. Ας υποθέσουμε πως στην εφαρμογή που αναπτύσσουμε χρειαζόμαστε σε μορφή αλφαριθμητικής σειράς το άθροισμα ή και το γινόμενο πραγματικών αριθμών αποθηκευμένων σε κατάλληλο πίνακα. Σε αυτήν την περίπτωση είναι λογικό να υλοποιήσουμε δύο συναρτήσεις, μία που υπολογίζει και επιστρέφει το άθροισμα και μια για το γινόμενο. Θέλουμε όμως οι δύο συναρτήσεις να διαμορφώνουν τα αποτελέσματά τους με κοινό μοτίβο. Σκοπός μας είναι να εμφανίσουμε άθροισμα και γινόμενο και δεν επιθυμούμε να εμφανίζονται με διαφορετικό πλήθος δεκαδικών. Επομένως χρειαζόμαστε ένα τρόπο διαμόρφωσης του αποτελέσματος κοινό για τις δύο συναρτήσεις. Εδώ μας διευκολύνουν οι στατικές μεταβλητές της κλάσης.

```
import java.text.DecimalFormat;

public class StaticVars {

    public static String format; // "###.##"

    public static String sum(double[] in) {
        double sum = 0;
        for (double current : in) {
            sum += current;
        }
        DecimalFormat f = new DecimalFormat(format);
        return f.format(sum);
    }

    public static String product(double[] in) {
        double product = 1;
        for (double current : in) {
            product *= current;
        }
```

```
        DecimalFormat f = new DecimalFormat(format);
        return f.format(product);
    }

    public static void main(String[] args) {
        format = "###.##";
        double[] d = {5.6, 7.95, 3.23, 45, 96};
        System.out.println(sum(d));
        System.out.println(product(d));
    }
}
```

Κώδικας 7.11 *Χρήση στατικής μεταβλητής*

Στον κώδικα 7.11 δίνεται ο ορισμός της κλάσης StaticVars. Η κλάση περιλαμβάνει ορισμό των συναρτήσεων sum και product. Και οι δύο λαμβάνουν ως παράμετρο ένα πίνακα πραγματικών. Η sum υπολογίζει το άθροισμα των τιμών της παραμέτρου της και η product το γινόμενο. Και οι δύο υπολογίζουν το αποτέλεσμά τους και το επιστρέφουν ως String. Στην κλάση ορίζεται και η στατική μεταβλητή format. Σκοπός της είναι να ελέγχει την διαμόρφωση κατά την εμφάνιση των πραγματικών μεταβλητών. Ως στατική μεταβλητή της κλάσης, η format μπορεί να προσπελαστεί από όλες τις συναρτήσεις της κλάσης. Στην main πριν καλέσουμε τις μεθόδους, φροντίζουμε να ενημερώσουμε κατάλληλα την format. Στην συνέχεια, οι δύο μέθοδοι διαβάζουν την format και επιστρέφουν το αποτέλεσμά τους διαμορφωμένο σύμφωνα με το μοτίβο που αποθηκεύσαμε στην format.

Κατά μία έννοια, μια στατική μεταβλητή αποτελεί μία κοινή μνήμη για όλη την κλάση.

Ας υποθέσουμε τώρα πως χρειαζόμαστε την ίδια λειτουργικότητα, δηλ. να εμφανίζουμε το άθροισμα και το γινόμενο των στοιχείων πινάκων πραγματικών και σε μία άλλη εφαρμογή. Έστω όμως πως στην άλλη εφαρμογή, χρειαζόμαστε διαφορετική διαμόρφωση, π.χ. θέλουμε τρία δεκαδικά ψηφία αντί για δύο. Με δεδομένο ότι οι συναρτήσεις sum και product αλλά και η σειρά format είναι δημόσια μέλη της κλάσης, η λύση είναι απλή και παρατίθεται στον κώδικα 7.12.

```
public class AnotherApp {

    public static void main(String[] args) {
        StaticVars.format = "###.###";
        double[] d = {5.675, 7.915, 3.213, 45, 0.96};
        System.out.println(StaticVars.sum(d));
        System.out.println(StaticVars.product(d));
    }
}
```

Κώδικας 7.12 *Χρήση στατικής μεταβλητής από άλλη κλάση*

Στον κώδικα 7.12, η main της κλάσης AnotherApp, δίνει τιμή στην στατική μεταβλητή format της StaticVars. Στην συνέχεια καλεί τις sum και product της StaticVars προκειμένου να εμφανίσει το άθροισμα και το γινόμενο των στοιχείων του πίνακα d.

Όπως θα αντιληφθήκατε ήδη προκειμένου να προσπελάσουμε μια στατική συνάρτηση ή μια στατική μεταβλητή από κλάση διαφορετική από αυτήν που ορίζονται πρέπει να προηγηθεί του ονόματός τους, το όνομα της κλάσης στην οποία ορίζονται. Σε αυτήν την χρήση, το όνομα της κλάσης θεωρείται ένα προσδιοριστικό ονόματος (name specifier).

Αν έχουμε σύγκρουση μεταξύ αναγνωριστικού στατικής μεταβλητής μιας κλάσης με τοπική μεταβλητή μέσα σε συνάρτηση της ίδιας κλάσης, τότε θα πρέπει να χρησιμοποιήσουμε τον προσδιοριστή ονόματος κατά την αναφορά στην στατική μεταβλητή.

7.5 Απροσδόκητα λάθη

Ας υποθέσουμε πως θέλουμε να αναπτύξουμε μια συνάρτηση που υπολογίζει το παραγοντικό μιας ακέραιης τιμής. Το πρώτο βήμα βέβαια θα είναι να βεβαιωθούμε πως γνωρίζουμε τον σωστό ορισμό του παραγοντικού.

Για την διευκόλυνση της συζήτησής μας παραθέτουμε τον ορισμό.

Το παραγοντικό ενός μη αρνητικού ακέραιου n, συμβολίζεται με n! και είναι ίσο με 1 αν n=0 ή n=1 ενώ αν n>1, είναι το γινόμενο όλων των θετικών ακεραίων που είναι μικρότεροι ή ίσοι με το n.

Με δεδομένο τον ορισμό, η ανάπτυξη της συνάρτησης είναι σχετικά απλή. Ο κώδικας 7.13 μας δίνει την συνάρτηση του παραγοντικού.

```
public class UnexpectedErrors {

    static long factorial(long n) { //Κώδικας 7.13
        if (n == 0 || n == 1) {
            return 1;
        }
        long rVal = 1;
        for (long i = 2; i <= n; i++) {
            rVal *= i;
        }
        return rVal;
    }
}
```

Κώδικας 7.13 *Υπολογισμός παραγοντικού*

Η συνάρτηση factorial είναι τύπου long καθώς το παραγοντικό μεγαλώνει γρήγορα σε σχέση με το όρισμά του. Για παράδειγμα, 3! = 6, 6! = 720 και 9! = 362880. Στην αρχή της συνάρτηση factorial, ελέγχεται το n καθώς αν είναι 0 ή 1 δεν απαιτούνται περαιτέρω υπολογισμοί μια και για αυτές τις τιμές του n, το n! είναι ίσο με 1. Στην συνέχεια αρχικοποιείται η rVal στην τιμή 1 και πολλαπλασιάζεται διαδοχικά με όλους τους ακέραιους που είναι μεγαλύτεροι ή ίσοι του 2 και μικρότεροι ή ίσοι του n. Με την έξοδο από τον βρόχο for, η rVal αναπαριστά την απαιτούμενη τιμή την οποία η συνάρτηση επιστρέφει.

Αν τρέξετε την factorial θα διαπιστώσετε ότι επιστρέφει την σωστή τιμή για όλες τις μη αρνητικές τιμές της παραμέτρου n.

Τι θα συμβεί όμως αν η factorial κληθεί με πραγματική παράμετρο έναν αρνητικό ακέραιο; Καταρχάς, δεν θα επιστρέψει από την αρχική if. Στην συνέχεια θα αρχικοποιήσει την rVal στο 1. Στην συνέχεια δεν θα μπει στον βρόχο for καθώς το θετικό i δεν είναι μικρότερο ή ίσο από οποιονδήποτε αρνητικό αριθμό και τέλος θα επιστρέψει το rVal του οποίου η τιμή παραμένει 1. Με λίγα λόγια, η factorial του κώδικα 7.13 επιστρέφει την τιμή 1 ως αποτέλεσμα υπολογισμού του παραγοντικού οποιουδήποτε αρνητικού ακέραιου.

Αυτό όμως είναι λάθος μια και δεν ορίζεται παραγοντικό για αρνητικούς ακέραιους. Πρόκειται μάλιστα για σοβαρό λάθος με απρόβλεπτες συνέπειες κατά την εκτέλεση της εφαρμογής που το περιέχει.

Μπορεί να αναρωτιέστε γιατί να κληθεί η factorial με αρνητικό ακέραιο. Ενδεχομένως, η factorial να καλείται με όρισμα κάποια μεταβλητή η οποία να προκύπτει από κάποιους υπολογισμούς. Ενδεχομένως, το λάθος να ξεκινά από εκεί. Γενικά, στις ρεαλιστικές εφαρμογές όπου εμπλέκονται πάρα πολλές μεταβλητές και πάρα πολλοί και σύνθετοι υπολογισμοί, τέτοιου είδους λάθη είναι σχεδόν αδύνατο να αποκλειστούν. Εκείνο όμως που μπορεί να γίνει είναι να εντοπισθούν εγκαίρως και να διορθωθούν. Σε αυτό μας βοηθάνε οι εξαιρέσεις (Exceptions) της Java.

Η διαχείριση των απροσδόκητων λαθών είναι μεγάλο κομμάτι κάθε γλώσσας προγραμματισμού και της Java. Αντίστοιχα πολλές είναι και οι κλάσεις που αφορούν την διαχείριση των λαθών. Υπάρχουν διαχειρίσιμα και μη διαχειρίσιμα λάθη. Λεπτομέρειες για την διαχείριση των λαθών συζητάμε στην ενότητα 16. Για την ώρα μας αρκεί μια προσέγγιση που αν και απλή μας βοηθάει να προστατευτούμε από τα απροσδόκητα λάθη στον κώδικά μας. Η προσέγγιση λοιπόν που προτείνουμε εδώ βασίζεται στην παραγωγή μιας εξαίρεσης τύπου RuntimeException. Με άλλα λόγια, στην αρχή της συνάρτησης factorial θα ελέγξουμε το όρισμα και αν βρεθεί αρνητικό θα παράγουμε μια εξαίρεση τύπου RuntimeException. Η παραγωγή της εξαίρεσης, θα προκαλέσει τερματισμό του προγράμματος με εμφάνιση κατάλληλου μηνύματος. Έτσι θα βοηθηθούμε να εντοπίσουμε το πρόβλημα και να το διορθώσουμε.

Καλό είναι αυτός που καλεί την συνάρτησή μας να γνωρίζει πως η συνάρτηση ενδέχεται να παράγει εξαίρεση. Έτσι, η Java υποστηρίζει την δήλωση των εξαιρέσεων που ενδέχεται να παραγάγει μια συνάρτηση στην επικεφαλίδα της. Ο κώδικας 7.14 παρουσιάζει την factorial με ενσωματωμένες τις παρατηρήσεις που κάναμε σε αυτήν την ενότητα.

```
public class UnexpectedErrors {
```

```
    static long factorial(long n) throws RuntimeException { //Κώδικας 7.14
        if (n < 0) {
            throw new RuntimeException();
        }
        if (n == 0 || n == 1) {
            return 1;
        }
        long rVal = 1;
        for (long i = 2; i <= n; i++) {
            rVal *= i;
        }
        return rVal;
    }
}
```

Κώδικας 7.14 *Η factorial με έλεγχο της παραμέτρου*

Όπως φαίνεται στον κώδικα 7.14, η παραγωγή της εξαίρεσης γίνεται με την λέξη-κλειδί throw. Με την βοήθεια της λέξης-κλειδί throws δηλώνεται προαιρετικά στην επικεφαλίδα της συνάρτησης το γεγονός ότι η συνάρτηση ενδέχεται να παράγει εξαίρεση τύπου RuntimeException.

Τρέξτε την νέα έκδοση της factorial με αρνητικό όρισμα και μελετήστε το αποτέλεσμα.

7.6 Χρήσιμες συναρτήσεις

Σε αυτήν την ενότητα παρουσιάζουμε συναρτήσεις ενσωματωμένες στο βασικό πλαίσιο της Java που συχνά μας είναι χρήσιμες. Όπως θα έχετε ήδη αντιληφθεί, κάθε συνάρτηση στην Java ανήκει και σε μία κλάση. Επιπρόσθετα λοιπόν κάποιων συναρτήσεων της βιβλιοθήκης της Java που έχουμε ήδη δει, στην ενότητα αυτή θα δούμε κάποιες συναρτήσεις που ανήκουν στις κλάσεις Math και String.

7.6.1 Η κλάση Math

Η κλάση Math περιλαμβάνει μια σειρά από στατικές συναρτήσεις και στατικές σταθερές μεταβλητές που μας επιτρέπουν να κάνουμε βασικούς μαθηματικούς υπολογισμούς. Παρουσιάζουμε εδώ μέρος των συναρτήσεων και σταθερών μεταβλητών της Math. Αυτές που θεωρούμε πως είναι πιο αναγκαίες στα πλαίσια αυτού του εγχειριδίου. Για κάθε μια συνάρτηση παρουσιάζουμε πρώτα την διεπαφή της ακολουθούμενη από μια μικρή περιγραφή και στην συνέχεια επιδεικνύουμε τον τρόπο κλήσης της.
Η σταθερά E αναπαριστά το e, την βάση των φυσικών λογάριθμων. Η διεπαφή της έχει ως εξής:

```
public static final double E
```

Η σταθερά PI αναπαριστά το π, την αναλογία της περιφέρειας ενός κύκλου προς την διάμετρο του κύκλου. Η διεπαφή της είναι:

```
public static final double PI
```

Η συνάρτηση toRadians μετατρέπει τις μοίρες σε ακτίνια.

```
public static double toRadians(double angdeg)
```

Η παράμετρος angdeg αναπαριστά τις μοίρες μιας γωνίας. Η συνάρτηση επιστρέφει τα ακτίνια της ίδιας γωνίας.

Η συνάρτηση toDegrees μετατρέπει τα ακτίνια σε μοίρες.

```
public static double toDegrees(double angrad)
```

Η παράμετρος angrad αναπαριστά τα ακτίνια μιας γωνίας. Η συνάρτηση επιστρέφει τις μοίρες της ίδιας γωνίας.

Η συνάρτηση sin επιστρέφει το ημίτονο μιας γωνίας.

```
public static double sin(double a)
```

Η παράμετρος a αναπαριστά τα ακτίνια μιας γωνίας. Αν επομένως θέλετε να υπολογίσετε το ημίτονο μιας γωνίας 30 μοιρών δεν έχετε παρά να καλέσετε

```
Math.sin(Math.toRadians(30))
```

Η συνάρτηση cos επιστρέφει το συνημίτονο μιας γωνίας.

```
public static double cos(double a)
```

Η παράμετρος a αναπαριστά τα ακτίνια μιας γωνίας. Αν επομένως θέλετε να υπολογίσετε το συνημίτονο των 30 μοιρών καλέστε

```
Math.cos(Math.toRadians(30))
```

Η συνάρτηση tan επιστρέφει την εφαπτομένη γωνίας.

```
public static double tan(double a)
```

Η παράμετρος a αναπαριστά τα ακτίνια μιας γωνίας. Αν επομένως θέλετε να υπολογίσετε την εφαπτομένη των 30 μοιρών καλέστε

```
Math.tan(Math.toRadians(30))
```

Η συνάρτηση log επιστρέφει τον φυσικό λογάριθμο της παραμέτρου της.

```
public static double log(double a)
```

Για να υπολογίσετε τον φυσικό λογάριθμο του 90, καλέστε

```
Math.log(90)
```

Η συνάρτηση log10 επιστρέφει τον λογάριθμο με βάση το 10.

```
public static double log10(double a)
```

Για να υπολογίσετε τον λογάριθμο με βάση 10 του 90, καλέστε

```
Math.log10(90)
```

Η συνάρτηση pow επιστρέφει την δύναμη ενός αριθμού υψωμένου σε έναν εκθέτη.

```
public static double pow(double a,  double b)
```

Η παράμετρος a αναπαριστά την βάση και η b τον εκθέτη. Για να υπολογίσετε την παράσταση 5.24 καλέστε

Math.pow(5.2, 4)

Η συνάρτηση sqrt επιστρέφει την τετραγωνική ρίζα του ορίσματός της.

```
public static double sqrt(double a)
```

Για να υπολογίσετε την τετραγωνική ρίζα του 81, καλέστε

```
Math.sqrt(81)
```

Η συνάρτηση ceil επιστρέφει την μικρότερη ακέραιη τιμή που είναι μεγαλύτερη ή ίση με την παράμετρό της.

```
public static double ceil(double a)
```

Η κλήση ceil(9.1) επιστρέφει 10d ενώ η κλίση ceil(9.0) επιστρέφει 9d.

Η συνάρτηση floor επιστρέφει την μεγαλύτερη ακέραιη τιμή που είναι μικρότερη ή ίση με την παράμετρό της.

```
public static double floor(double a)
```

Η κλήση floor(9.1) επιστρέφει 9d όπως και η κλίση floor(9.0).

Η συνάρτηση rint επιστρέφει την ακέραιη τιμή που είναι πλησιέστερη στην τιμή της παραμέτρου της.

```
public static double rint(double a)
```

Η κλήση rint(9.4) επιστρέφει 9d και η κλίση rint(9.6) επιστρέφει 10d. Αν η τιμή της παραμέτρου βρίσκεται ακριβώς στην μέση μεταξύ δύο ακέραιων τιμών, τότε η rint επιστρέφει τον ακέραιο που είναι άρτιος. Επομένως, η κλίση rint(9.5) θα επιστρέψει 10d όπως ακριβώς και η κλήση rint(10.5).

Η συνάρτηση round επιστρέφει την ακέραιη τιμή που είναι πλησιέστερη στην τιμή της παραμέτρου της. Η συνάρτηση είναι υπερφορτωμένη. Αν κληθεί με όρισμα double επιστρέφει long και αν κληθεί με όρισμα float επιστρέφει int. Αν η τιμή της παραμέτρου βρίσκεται ακριβώς στην μέση μεταξύ δύο ακέραιων τιμών, τότε η round επιστρέφει τον μεγαλύτερο από τους πλησιέστερους ακέραιους.

```
public static long round(double a)
public static int round(float a)
```

Η κλήση Math.round(9.4) επιστρέφει 9 και η κλήση Math.round(9.5) επιστρέφει 10.

Η συνάρτηση max επιστρέφει το μεγαλύτερο από τα ορίσματά της. Η συνάρτηση είναι υπερφορτωμένη και υποστηρίζει όλους τους αριθμητικούς τύπους. Οι διαθέσιμες διεπαφές έχουν ως εξής:

```
public static double max(double a, double b)
public static float max(float a, float b)
public static int max(int a, int b)
public static long max(long a, long b)
```

Αν καλέσετε την max με ορίσματα διαφορετικού τύπου, τότε θα κληθεί η έκδοση που είναι συμβατή και με τους δύο τύπους. Για παράδειγμα, αν την καλέσετε με double και long, τότε θα κληθεί η max(double, double). Έτσι η έκφραση

```
double m=Math.max(1d, 2L);
```

```
είναι σωστή. Ενώ η έκφραση
```

```
long m=Math.max(1d, 2L);
```

είναι λάθος.

Ακριβώς ανάλογη με την max είναι η min.

Η συνάρτηση abs επιστρέφει την απόλυτη τιμή της παραμέτρου της. Η abs είναι επίσης υπερφορτωμένη για όλους τους αριθμητικούς τύπους. Οι διαθέσιμες διεπαφές έχουν ως εξής:

```
public static double abs(double a)
public static float abs(float a)
public static long abs(long a)
public static int abs(int a)
```

7.6.2 Οι κλάσεις των θεμελιωδών τύπων

Έχουμε ήδη εξηγήσει πως για κάθε θεμελιώδη τύπο, η Java παρέχει μια αντίστοιχη κλάση. Κάθε μια τέτοια κλάση παρέχει σταθερές και συναρτήσεις που είναι χρήσιμες κατά την εκτέλεση εργασιών με θεμελιώδεις τύπους. Σε αυτήν την ενότητα συγκεντρώνουμε τις σημαντικότερες στατικές συναρτήσεις αυτών των κλάσεων.

7.6.2.1 Η κλάση Character

Η συνάρτηση isDigit επιστρέφει true αν η παράμετρος της αναπαριστά ψηφίο σύμφωνα με την κωδικοποίηση Unicode. Επομένως, εκτός από τα ψηφία που χρησιμοποιούνται στην Δυτική γραφή, υποστηρίζονται και τα ψηφία από διάφορους πολιτισμούς. Για παράδειγμα, οι χαρακτήρες ‘٢’, ‘٣’ και ‘٤’ αποτελούν ψηφία στο αραβικό σύστημα αρίθμησης. Η διεπαφή της συνάρτησης είναι

```
public static boolean isDigit(char ch)
```

Η συνάρτηση isLetter επιστρέφει true αν η παράμετρος της αναπαριστά κάποιο γράμμα σύμφωνα με την κωδικοποίηση Unicode. Προφανώς, η συνάρτηση δεν είναι αληθής μόνο για τους Λατινικούς χαρακτήρες. Για παράδειγμα, οι χαρακτήρες ‘暘’, ‘睚’ και ‘睛’ αναπαριστούν γράμματα στην Μανδαρίνικη γραφή. Η διεπαφή της συνάρτησης είναι

public static boolean isLetter(char ch)

Η συνάρτηση isLowerCase επιστρέφει true αν η παράμετρος της αναπαριστά πεζό γράμμα.

public static boolean isLowerCase(char ch)

Η συνάρτηση isSpaceChar επιστρέφει true αν η παράμετρος της αναπαριστά χαρακτήρα που είναι διαχωριστικό διαστήματος, γραμμής ή παραγράφου. Υπάρχουν συνολικά 19 τέτοιοι χαρακτήρες στο Unicode με κωδικούς 32, 160, 5760, 8192, 8193, 8194, 8195, 8196, 8197, 8198, 8199, 8200, 8201, 8202, 8232, 8233, 8239, 8287 και 12288. Η διεπαφή της συνάρτησης είναι

public static boolean isSpaceChar(char ch)

Η συνάρτηση isUpperCase επιστρέφει true αν η παράμετρος της αναπαριστά κεφαλαίο γράμμα. Η διεπαφή της συνάρτησης είναι

public static boolean isUpperCase(char ch)

Η συνάρτηση toString επιστρέφει την παράμετρό της ως String μήκους 1. Η διεπαφή της συνάρτησης είναι

public static String toString(char c)

7.6.2.2 Οι κλάσεις Integer, Double και Boolean

Καταρχάς να επισημάνουμε πως ότι αναφέρουμε σε αυτήν την ενότητα για την κλάση Integer ισχύει κατ' αναλογία και για τις κλάσεις Byte, Short και Long. Επίσης ότι αναφέρεται για την κλάση Double ισχύει κατ' αναλογία και για την κλάση Float.

Οι συναρτήσεις

```
public static int parseInt(String s) throws
NumberFormatException
public static double parseDouble(String s) throws
NumberFormatException
public static boolean parseBoolean(String s)
```

ανήκουν στις κλάσεις Integer, Double και Boolean, αντίστοιχα και μετατρέπουν το όρισμά τους σε int, double και boolean, αντίστοιχα.

Προσέξτε πως οι parseInt και parseDouble ενδέχεται να παραγάγουν λάθος τύπου NumberFormatException. Πρόκειται για λάθος στενά συγγενικό με το RuntimeException που είδαμε στην ενότητα 7.5. Αντίθετα, η parseBoolean δεν παράγει κανενός τύπου λάθος. Η διαφορά αυτή εξηγείται ως εξής: Η parseInt για παράδειγμα, για να λειτουργήσει σωστά θα πρέπει το όρισμά της να είναι μετατρέψιμο σε ακέραιο. Για παράδειγμα η σειρά "*12.2" δεν μπορεί να μετατραπεί σε ακέραιο. Αν επομένως κληθεί η parseInt με αυτήν την σειρά θα παραγάγει λάθος.

Αντίθετα η parseBoolean επιστρέφει true μόνο αν το όρισμά της είναι η σειρά "true" ανεξαρτήτως πεζών ή κεφαλαίων και false σε όλες τις άλλες περιπτώσεις.

Η συνάρτηση

```
public static int compare(double d1, double d2)
```

της κλάσης Double, συγκρίνει δύο πραγματικές μεταβλητές ή σταθερές και επιστρέφει 0 αν το d1 είναι αριθμητικά ίσο με το d2, τιμή μικρότερη του μηδενός αν το d1 είναι αριθμητικά μικρότερο από το d2, και τιμή μεγαλύτερη από το 0 αν το d1 είναι αριθμητικά μεγαλύτερο από το d2. Η συνάρτηση θεωρεί πως η τιμή Double.NaN είναι μεγαλύτερη από όλες τις άλλες τιμές τύπου Double και πως το θετικό 0d είναι μεγαλύτερο από το αρνητικό -0d. Διαφέρει η compare σε αυτό το σημείο σε σχέση με τους τελεστές ισότητας και ανισότητας <, <=, ==, >=, >. Για παράδειγμα ο κώδικας

```
System.out.println(Double.NaN>Double.MAX_VALUE);
System.out.println(Double.compare(Double.NaN,Double.MAX_VALUE
));
System.out.println(+0d>-0d);
System.out.println(Double.compare(+0d, -0d));
```

εμφανίζει

```
false
1
false
1
```

Επομένως ο τελεστής μεγαλύτερο από δεν βγάζει την Double.NaN μεγαλύτερη από την Double.MAX_VALUE ενώ η compare την βγάζει. Παρομοίως, ο τελεστής > δεν διαφοροποιεί μεταξύ των 2 μηδέν ενώ η compare διαφοροποιεί.

Οι συναρτήσεις

```
public static boolean isNaN(double v)
public static boolean isInfinite(double v)
```

επιστρέφουν true αν το όρισμά τους είναι Double.NaN ή Double.POSITIVE_INFINITY και Double.NEGATIVE_INFINITY, αντίστοιχα.

Τέλος, όλες οι κλάσεις των θεμελιωδών τύπων διαθέτουν την στατική συνάρτηση toString παρόμοια με την toString της Character, ενότητα 7.6.2.1, που επιστρέφει την τιμή της παραμέτρου της με την μορφή αλφαριθμητικής σειράς.

7.6.3 Η κλάση String

Η συνάρτηση

```
public static String copyValueOf(char[] data)
```

επιστρέφει ένα String που συντίθεται από τους χαρακτήρες του πίνακα data.
Η συνάρτηση

```
public static String copyValueOf(char[] data, int offset, int
count)
```

επιστρέφει ένα String που συντίθεται από count στον αριθμό χαρακτήρες του πίνακα data αρχίζοντας από την θέση offset.

Οι συναρτήσεις valueOf επιστρέφουν την String αναπαράσταση του ορίσματός τους. Η valueOf είναι υπερφορτωμένη για όλους τους θεμελιώδεις τύπους. Ενδεικτικά παραθέτουμε την διεπαφή της συνάρτησης για τον τύπο double

```
public static String valueOf(double d)
```

7.7 Προσδιοριστές προσπέλασης

Έχουμε συζητήσει αρκετά για τα στατικά μέλη της κλάσης. Εκείνο όμως που δεν έχουμε ξεκαθαρίσει είναι με ποια κριτήρια θα καθορίζουμε τον τύπο της πρόσβασης σε αυτά.

Δυστυχώς για αυτήν την απόφαση δεν υπάρχουν κατηγορηματικοί κανόνες και τα πάντα εξαρτώνται από τον στόχο που εξυπηρετεί η κάθε συνάρτηση. Ωστόσο μια συζήτηση επάνω σε αυτό το θέμα μπορεί να μας βοηθήσει να λαμβάνουμε σωστότερες σχετικές αποφάσεις και να προστατεύουμε τους χρήστες των συναρτήσεών μας.

Η συζήτηση εδώ περιορίζεται στον έλεγχο της πρόσβασης σε στατικές συναρτήσεις και μεταβλητές.

Η Java υποστηρίζει 4 προσδιοριστές προσπέλασης, την ιδιωτική (private), την δημόσια (public), την προσπέλαση πακέτου (package access) ή εξορισμού (default access) και την προστατευμένη (protected).

Για την προστατευμένη πρόσβαση δεν μπορούμε να πούμε και πολλά. Πρόκειται για τύπο πρόσβασης που σχετίζεται πολύ στενά με την κληρονομικότητα και παρουσιάζεται αναλυτικά στην ενότητα 13.1.2. Εκείνο όμως που μας ενδιαφέρει σχετικά με τα στατικά μέλη είναι πως παρότι ο μεταγλωττιστής δεν διαμαρτύρεται, σε γενικές γραμμές, η προστατευμένη πρόσβαση δεν ταιριάζει στα στατικά μέλη καθώς αυτά δεν υπακούν στους κανόνες της κληρονομικότητας.

Πρόσβαση πακέτου δηλώνουμε όταν ένα μέλος της κλάσης θέλουμε να μπορεί να χρησιμοποιηθεί από όλες τις κλάσεις του ίδιου πακέτου αλλά όχι από κλάσεις που ανήκουν σε άλλα πακέτα. Θα μπορούσαμε να πούμε πως ότι είναι η ιδιωτική πρόσβαση σε επίπεδο κλάσης είναι και η πρόσβαση πακέτου σε επίπεδο πακέτου. Συχνά μάλιστα αναφέρεται ως ιδιωτική πρόσβαση πακέτου (package-private).

Ας δούμε τώρα ένα παράδειγμα με δύο συναρτήσεις όπου η μια είναι καλύτερο να δηλωθεί ως ιδιωτική και η άλλη ως δημόσια. Έστω λοιπόν πως θέλουμε να υλοποιήσουμε μια βιβλιοθήκη μαθηματικών συναρτήσεων μέσα στην οποία περιλαμβάνεται μια συνάρτηση που υπολογίζει ένα πραγματικό αριθμό υψωμένο σε ακέραιη δύναμη. Για αυτό το παράδειγμα, εκπαιδευτικού χαρακτήρα, υποθέτουμε επίσης πως η συνάρτηση pow της Math δεν είναι διαθέσιμη,

Σε γενικές γραμμές είναι εύκολο να υλοποιήσουμε μια τέτοια συνάρτηση. Ας δούμε μια πρώτη προσέγγιση.

```
static double powerPos(double b, int e) { //Κώδικας 7.15
    if (e == 0) {
```

```
            return 1;
        }
        double rVal = 1;
        for (int i = 0; i < e; i++) {
            rVal *= b;
        }
        return rVal;
    }
```

Κώδικας 7.15 *Συνάρτηση που υπολογίζει την τιμή πραγματικού υψωμένου σε μη αρνητικό ακέραιο εκθέτη*

Στον κώδικα 7.15, η συνάρτηση powerPos επιστρέφει την τιμή της πρώτης παραμέτρου της υψωμένη στην τιμή της δεύτερης παραμέτρου της. Ωστόσο η συνάρτηση επιστρέφει σωστά αποτελέσματα μόνο για μη αρνητικούς εκθέτες. Αν την καλέσετε για οποιονδήποτε αρνητικό εκθέτη θα σας επιστρέψει λανθασμένα την τιμή 1. Αυτό μπορείτε να αντιληφθείτε εύκολα διαβάζοντας τον κώδικα 7.15. Εφόσον η if δεν θα οδηγήσει σε επιστροφή της συνάρτησης, η συνάρτηση θα προχωρήσει αρχικοποιώντας την μεταβλητή rVal στο 1. Στην συνέχεια, η ροή δεν μπαίνει καθόλου στον βρόχο for μια και για την αρχική τιμή 0 του i δεν αληθεύει η i<e. Επομένως, η συνάρτηση καταλήγει να επιστρέφει 1.

Το γεγονός ότι η συνάρτηση μπορεί να παραγάγει λάθος για αρνητικούς εκθέτες την καθιστά εντελώς ακατάλληλη για δημόσια χρήση. Ωστόσο, αυτός που υλοποιεί την κλάση των μαθηματικών συναρτήσεων και γνωρίζει το συγκεκριμένο πρόβλημα θα μπορούσε να αξιοποιήσει την συνάρτηση ώστε να διευκολυνθεί στην ανάπτυξη μιας πιο ολοκληρωμένης συνάρτησης υπολογισμού της δύναμης. Να τονίσουμε πάντως πως μια τέτοια συνάρτηση όπως η powerPos θα πρέπει να χρησιμοποιείται μόνο με πολλή προσοχή και πολύ περιορισμένα. Ας δούμε όμως πως μπορεί να βοηθήσει η powerPos στην ανάπτυξη ορθής συνάρτησης υπολογισμού δύναμης.

```
    static double power(double b, int e) { // Κώδικας 7.16
        if (e >= 0) {
            return powerPos(b, e);
        }
        return 1 / powerPos(b, -e);
    }
```

Κώδικας 7.16 *Συνάρτηση που υπολογίζει την τιμή πραγματικού υψωμένου σε ακέραιο εκθέτη*

Όπως φαίνεται στον κώδικα 7.16, η συνάρτηση power ελέγχει την παράμετρο e που αντιπροσωπεύει τον εκθέτη. Αν η τιμή της είναι μη αρνητική καλεί την powerPos και επιστρέφει σωστά την δύναμη μη αρνητικού εκθέτη. Αν η τιμή της είναι αρνητική, δεν μπαίνει μέσα στην if, οπότε προχωρά και καλεί την powerPos με παράμετρο -e και επιστρέφει και πάλι σωστό αποτέλεσμα για αρνητικό εκθέτη.

Αυτό είναι ένα χαρακτηριστικό παράδειγμα όπου η μεν powerPos επιβάλλεται να δηλωθεί ως ιδιωτική ώστε να προστατεύεται ο χρήστης της βιβλιοθήκης μας από πιθανά λανθασμένη κλήση ενώ η power μπορεί να δηλωθεί ως δημόσια ώστε να μπορεί ο χρήστης να υπολογίσει τις δυνάμεις που επιθυμεί.

7.8 Πακέτα

Ένα πακέτο συντίθεται ως ένα σύνολο κλάσεων ή και άλλων τύπων[6]. Επομένως, ένα πακέτο είναι ταυτόχρονα ένα σύνολο αρχείων πηγαίου κώδικα. Ο σκοπός του είναι να παρέχει έλεγχο πρόσβασης στα μέλη του και έναν μοναδικό ονοματοχώρο (namespace). Τα μέλη ενός πακέτου καθορίζονται από την δήλωση που γίνεται υποχρεωτικά στην πρώτη γραμμή του αντίστοιχου αρχείου. Για τον έλεγχο πρόσβασης έχουμε ήδη μιλήσει στην ενότητα 7.7. Ας δούμε μερικές άλλες χρήσιμες λεπτομέρειες.

Η ανάπτυξη κώδικα στις μέρες μας είναι μια κατεξοχήν συνεργατική εργασία. Έτσι, οι προγραμματιστές που εργάζονται σε μία εταιρεία, συχνά χρησιμοποιούν κώδικες που έχουν αναπτύξει συνάδελφοί τους αλλά και κώδικες που έχουν αναπτυχθεί εκτός της εταιρείας. Προκύπτει εδώ ανά πρόβλημα. Τι θα συμβεί αν στον κώδικά μου έχω ορίσει μια συνάρτηση με ίδια ταυτότητα με μια συνάρτηση που έχει ορισθεί σε κώδικα τρίτου που όμως τον χρησιμοποιώ και εγώ; Για παράδειγμα, στον κώδικά μου χρησιμοποιώ την συνάρτηση public static

[6] Διεπαφών (interface), Απαριθμήσιμων τύπων (enumerated types), και τύπων σχολίων (annotation types).

int f(int). Παράλληλα σε έναν άλλο κώδικα που χρειάζομαι είναι ορισμένη συνάρτηση με την ίδια ταυτότητα σε ομώνυμη κλάση. Όταν καλώ π.χ. f(5), ποια από τις δύο f θα τρέξει; Την λύση εδώ δίνει το πακέτο. Αν η δική μου f έχει ορισθεί στο πακέτο a και η άλλη f στο πακέτο b, τότε μπορώ να διαφοροποιήσω καλώντας a.ClassName.f(5) ή b.ClassName.f(5). Πως όμως θα αποφύγουμε να έχουν και τα δύο πακέτα την ίδια ονομασία; Αυτό επιτυγχάνεται αποκλειστικά μέσω συμβάσεων για την ονοματολογία των πακέτων. Πιο συγκεκριμένα, ισχύουν οι ακόλουθες συμβάσεις:

1. Στην ονομασία ενός πακέτου θα πρέπει να χρησιμοποιούνται μόνο πεζοί χαρακτήρες ώστε να αποφεύγεται σύγκρουση με ονόματα κλάσεων ή άλλων τύπων.
2. Στην ονομασία ενός πακέτου πρέπει να συμμετέχει ένας τομέας διαδικτύου (internet domain). Οι επαγγελματίες συνηθίζουν να χρησιμοποιούν την διεύθυνση email αντεστραμμένη όπως κάνουμε στους κώδικες αυτού του βιβλίου, π.χ. gr.ihu.cs.lmous.OOJ.k2.v2. Η οργάνωση των πακέτων και κλάσεων μέσα σε αυτά ώστε να μην προκύπτει σύγκρουση σε αναγνωριστικά είναι υπόθεση που εύκολα ελέγχεται σε συγκεκριμένο τομέα. Το θέμα είναι πως ένας τομέας δικτύου είναι υποχρεωτικά μοναδικός οπότε έτσι εξασφαλίζεται και η μοναδικότητα στην ονομασία του πακέτου.
3. Σε πολλές περιπτώσεις, η επαναληπτική χρήση του ονόματος ενός πακέτου μέσα στον κώδικα είναι κουραστική. Έτσι η Java μας δίνει την δυνατότητα να εισάγουμε διάφορα μέλη ενός πακέτου στον κώδικά μας και να τα χρησιμοποιούμε χωρίς την αναφορά πακέτου. Η εισαγωγή επιτυγχάνεται με την χρήση της δεσμευμένης λέξης import.

Έτσι για παράδειγμα η πρόταση

```
import a.ClassName;
```

μας επιτρέπει να χρησιμοποιήσουμε την συνάρτηση f της ClassName χωρίς την αναφορά πακέτου, δηλ. ως ClassName.f(5).

Αν πάλι θέλουμε να εισάγουμε όλους τους τύπους από το πακέτο a, τότε θα πρέπει να χρησιμοποιήσουμε την import ως ακολούθως:

```
import a.*;
```

Προσοχή όμως, ο αστερίσκος δεν έχει το σύνηθες νόημα. Αν θέλουμε να εισάγουμε όλους τους τύπους του πακέτου a που το όνομά τους αρχίζει από B, η πρόταση

```
import a.B*;
```

δεν ισχύει και θα προκαλέσει λάθος μεταγλώττισης.

Επίσης, η πρόταση

```
import a.*;
```

εισάγει όλους τους τύπους του πακέτου a αλλά δεν εισάγει τίποτε από τα πακέτα a.b, a.c. Αν θέλουμε να εισάγουμε όλους τους τύπους του πακέτου a και όλους τους τύπους του πακέτου a.b, τότε πρέπει να χρησιμοποιήσουμε δύο import όπως φαίνεται παρακάτω.

```
import a.*;
import a.b.*;
```

7.9 Λυμένες ασκήσεις

Σε αυτήν την ενότητα παρουσιάζουμε μια σειρά από παραδείγματα στατικών συναρτήσεων ώστε να βοηθήσουμε τον αναγνώστη να εξοικειωθεί τόσο με την υλοποίηση όσο και με την χρήση τους.

7.9.1 Μέγιστη τιμή πίνακα

Να υλοποιηθεί συνάρτηση που λαμβάνει ως παράμετρο έναν μονοδιάστατο πίνακα ακεραίων και επιστρέφει την θέση της μέγιστης τιμής του.

Λύση
Καταρχάς θα πρέπει να αποσαφηνίσουμε την διεπαφή της συνάρτησης. Σύμφωνα με την εκφώνηση η συνάρτηση λαμβάνει ένα μονοδιάστατο πίνακα ακεραίων. Επομένως, η λίστα παραμέτρων είναι σαφής. Επίσης, σύμφωνα με την εκφώνηση, η συνάρτηση επιστρέφει το μέγιστο στοιχείο του πίνακα. Επομένως, επιστρέφει έναν ακέραιο. Ο κώδικας 7.17 παρουσιάζει την συνάρτηση.

```
public static int max(int[] tbl) { //Κώδικας 7.17
    int max = tbl[0];
    int idx = 0;
    for (int i = 1; i < tbl.length; i++) {
        if (tbl[i] > max) {
            idx = i;
            max = tbl[i];
        }
    }
    return idx;
}
```

Κώδικας 7.17 *Συνάρτηση εύρεσης της μέγιστης τιμής σε πίνακα ακεραίων*

Για την υλοποίηση της συνάρτησης χρησιμοποιούμε δύο βοηθητικές μεταβλητές, την max και την idx. Η max χρησιμεύει για την αποθήκευση της τιμής του μεγαλύτερου στοιχείου και η idx για την αποθήκευση της θέσης του μεγαλύτερου στοιχείου. Η idx αρχικοποιείται ώστε να αναφέρεται στην αρχή του πίνακα και η max στην τιμή του στοιχείου στην θέση 0.

Στην συνέχεια, σε μια επαναληπτική διαδικασία ελέγχονται ένα προς ένα τα υπόλοιπα στοιχεία του πίνακα. Κάθε φορά που το υπό έλεγχο στοιχείο είναι μεγαλύτερο από την τιμή της max ενημερώνεται τόσο η max όσο και η idx. Στο τέλος του βρόχου for, η max είναι ενημερωμένη με την τιμή του μεγαλύτερου στοιχείου του πίνακα και η idx με την θέση του.

Προσέξτε πως η for ξεκινάει από 1 και όχι από 0 όπως συνήθως. Αν ξεκινούσε από 0, στο πρώτο επαναληπτικό βήμα θα εξέταζε αν το στοιχείο στην θέση 0 είναι μεγαλύτερο από την τιμή της max. Η max όμως έχει αρχικοποιηθεί στην τιμή του στοιχείου στην θέση 0. Επομένως, πρόκειται για περιττή σύγκριση.

Επίσης, ενδιαφέρον έχει πως η συνάρτηση επιστρέφει την θέση της μεγαλύτερης τιμής και όχι την ίδια την τιμή. Πράγματι, είναι πιο χρήσιμη η θέση παρά η τιμή καθώς αν έχω την θέση, εύκολα μπορώ να προσπελάσω την τιμή.

Τέλος, ας σημειωθεί πως αν στον πίνακα υπάρχουν περισσότερα από ένα μέγιστα στοιχεία, η υλοποίηση του κώδικα 7.17 επιστρέφει την θέση του πρώτου από τα μέγιστα.

7.9.2 Εξίσωση β' βαθμού

Να υλοποιηθεί συνάρτηση που επιστρέφει την λύση εξίσωσης β' βαθμού.

Λύση
Καταρχάς θα πρέπει να δούμε πως θα μοντελοποιήσουμε το πρόβλημα, δηλ. στην προκειμένη περίπτωση να αποφασίσουμε ποια θα είναι η διεπαφή της συνάρτησης. Γνωρίζουμε από τα μαθηματικά πως η εξίσωση β' βαθμού έχει την μορφή

$$ax^2 + bx + c = 0, με\ a, b, c\ \in R\ και\ a \neq 0$$

Επομένως, η εξίσωση ορίζεται από τους συντελεστές της. Αυτούς πρέπει να περάσουμε παραμετρικά στην συνάρτηση. Τι όμως θα επιστρέφει η συνάρτηση; Όπως γνωρίζουμε, οι λύσεις εξαρτώνται από την τιμή της διακρίνουσας. Αν η διακρίνουσα, Δ, είναι μεγαλύτερη από το 0, τότε έχουμε δύο διακριτές πραγματικές λύσεις,

αν η διακρίνουσα είναι ίση με το 0, τότε έχουμε μια κοινή λύση και αν η διακρίνουσα είναι μικρότερη από το 0, η εξίσωση δεν έχει λύση.

Μία αντιμετώπιση είναι να επιστρέφουμε ένα πίνακα πραγματικών μεγέθους 2. Στην περίπτωση που $\Delta > 0$, τότε στην πρώτη θέση του πίνακα θα έχουμε την μία ρίζα και στην δεύτερη την άλλη, αν $\Delta = 0$ και στις δύο θέσεις θα έχουμε την κοινή ρίζα, τέλος αν $\Delta < 0$, τότε και στις δύο θέσεις θα τοποθετήσουμε την τιμή Double.NaN.

Βέβαια όλα αυτά για να τα κάνουμε θα πρέπει να υπολογίσουμε την διακρίνουσα. Παρότι, ο χρήστης της συνάρτησης, θα πληροφορηθεί για το πρόσημο της διακρίνουσας δεν θα γνωρίζει την ακριβή τιμή της. Μία εναλλακτική προσέγγιση είναι η συνάρτησή μας να επιστρέφει πίνακα μήκους 3, όπου στην θέση 0 να τοποθετεί την τιμή της διακρίνουσας. Στον κώδικα 7.18 δίνεται αυτή η τελευταία εκδοχή της συνάρτησης.

```
static double[] triwnymo(double a, double b, double c) { // Κώδικας 7.18
    if (a == 0) {
        throw new RuntimeException();
    }
    double[] rVal = new double[3];
    double d = b * b - 4 * a * c;
    rVal[0] = d;
    if (d < 0) {
        rVal[1] = rVal[2] = Double.NaN;
        return rVal;
    }
    rVal[1] = (-b + Math.sqrt(d)) / (2 * a);
    rVal[2] = (-b - Math.sqrt(d)) / (2 * a);
    return rVal;
}
```

Κώδικας 7.18 *Συνάρτηση λύσης εξίσωσης β' βαθμού*

Σε αυτήν την περίπτωση η εξίσωση μπορεί να κληθεί όπως δείχνει ο κώδικας 7.19.

```
static void klisiTriwnymo() { // Κώδικας 7.19
    double[] solution = triwnymo(2, 5, -1);
    if (solution[0] < 0) {
        System.out.println("Η Εξίσωση σεν έχει λύση στο R");
    } else {
        System.out.println("χ1=" + solution[1] + " x2=" + solution[2]);
    }
}
```

Κώδικας 7.19 *Κλήση της συνάρτησης triwnymo*

Μία εναλλακτική κλήση δίνεται στον κώδικας 7.20

```
static void klisiTriwnymo2() { // Κώδικας 7.20
    double[] solution = triwnymo(2, 5, -1);
    if (Double.isNaN(solution[0])) {
        System.out.println("Η Εξίσωση σεν έχει λύση στο R");
    } else {
        System.out.println("χ1=" + solution[1] + " x2=" + solution[2]);
    }
}
```

Κώδικας 7.20 *Εναλλακτική κλήση της συνάρτησης triwnymo*

Προσοχή όμως δεν είναι σωστή η κλήση

```
if (solution[0]==Double.NaN)….
```

καθώς εξορισμού η έκφραση Double.NaN==Double.NaN είναι ψευδής.

Σε μια εναλλακτική προσέγγιση μπορούμε να αναπαραστήσουμε το τριώνυμο ως ένα πίνακα πραγματικών μήκους 3. Η συνάρτηση μπορεί να υπερφορτωθεί εύκολα ώστε να υποστηρίζεται και αυτή η αναπαράσταση του τριωνύμου, όπως δείχνει ο κώδικας 7.21.

```
static double[] triwnymo(double[] syn) { // Κώδικας 7.21
    return triwnymo(syn[0], syn[1], syn[2]);
}
```

Κώδικας 7.21 *Υπερφόρτωση της συνάρτησης triwnymo*

Σε αυτήν την περίπτωση, γίνεται η αναπαράσταση εξισώσεων β' βαθμού ευκολότερη, όπως δείχνει ο κώδικας 7.22.

```
double[] ex1 = {2, 5, -1}, ex2 = {4, 4, -9};
double[] solution1 = triwnymo(ex1);
double[] solution2 = triwnymo(ex2);
```

Κώδικας 7.22 *Αναπαράσταση εξισώσεων β βαθμού ως double[]*

7.9.3 Fibonacci

Να αναπτυχθεί συνάρτηση που επιστρέφει τον ν-οστό όρο της ακολουθίας Fibonacci.

Λύση
Καταρχάς θα πρέπει να δούμε ποιος ακριβώς είναι ο ορισμός της ακολουθίας Fibonacci. Στο διαδίκτυο μπορούμε εύκολα να τον βρούμε. Αν συμβολίσουμε τον ν-οστό με Fn, τότε ο ορισμός της ακολουθίας έχει ως εξής:

$$n = 0 \rightarrow F_n = 0$$
$$n = 1 \rightarrow F_n = 1$$
$$n > 1 \rightarrow F_{n-1} + F_{n-2}$$

Επομένως, η συνάρτησή μας θα πρέπει να λαμβάνει μία ακέραιη παράμετρο, μπορεί όμως να επιστρέφει ένα long, δεδομένου πως η τιμή του ν-οστού όρου είναι σημαντικά μεγαλύτερη από το n. Ο κώδικας 7.23 δίνει μια υλοποίηση της συνάρτησης.

```
static long fibonacci(int fibIdx) { //Κώδικας 7.23
    if (fibIdx < 0) {
        throw new RuntimeException();
    } else if (fibIdx <= 1) {
        return fibIdx;
    }
    int n = 2;
    long current = 1, previous = 1, beforePrevious;
    while (++n <= fibIdx) {
        beforePrevious = previous;
        previous = current;
        current = beforePrevious + previous;
    }
    return current;
}
```

Κώδικας 7.23 *Υλοποίηση της συνάρτησης Fibonacci*

Καθώς δεν υπάρχει αρνητικός όρος της ακολουθίας, αν η συνάρτηση κληθεί με παράμετρο μικρότερη του μηδενός, σημαίνει πως κάποιο λάθος έχει συμβεί. Διαφορετικά αν η παράμετρος fibIdx είναι μικρότερη ή ίση με το 1, δηλ. αν είναι 0 ή 1, βάσει ορισμού, επιστρέφεται η ίδια η τιμή της παραμέτρου. Αν παρόλα αυτά δεν επιστρέψει η συνάρτηση σημαίνει ότι κλήθηκε με παράμετρο μεγαλύτερη του ένα, οπότε η τιμή του ν-οστού όρου πρέπει να υπολογιστεί ως το άθροισμα των δύο προηγούμενων όρων. Η μεταβλητή n αναπαριστά την

τάξη του όρου και ξεκινά από το 2, οι δε μεταβλητές current, previous και beforePrevious αναπαριστούν τις τιμές του ν-οστού όρου, του προηγούμενου του και του όρου πριν τον προηγούμενο του ν-οστού. Ο βρόχος while επισκέπτεται έναν-έναν τους όρους της ακολουθίας έως ότου φτάσει στον όρο που εκφράζει η παράμετρος fibIdx. Σε κάθε βήμα αυτής της επαναληπτικής διαδικασίας ενημερώνονται κατάλληλα οι προηγούμενοι όροι. Με την έξοδο από την while, η current είναι ενημερωμένη με την τιμή του όρου στην θέση fibIdx.

7.9.4 Λεκτικό αριθμού

Να υλοποιηθεί συνάρτηση static String lektiko(int num) που δέχεται μια ακέραιη παράμετρο από 0 έως 999 και επιστρέφει την διατύπωσή της στα Ελληνικά.

Λύση

Σύμφωνα με την εκφώνηση και με την διεπαφή που δίνεται μια κλήση όπως η lektiko(125) θα πρέπει να επιστρέψει την αλφαριθμητική σειρά “εκατόν είκοσι πέντε”. Εδώ έχουμε δύο θέματα να μας απασχολήσουν. Πρώτον, πόσα και ποια λεκτικά μας είναι απαραίτητα για να σχηματίσουμε την απαιτούμενη διατύπωση όλων των αριθμών από 0 έως 999 και δεύτερον πως θα εξάγουμε πόσες εκατοντάδες έχουμε, πόσες δεκάδες και πόσες μονάδες. Προσέξτε πως αν διαιρέσετε έναν ακέραιο με το 100 (ακέραιη διαίρεση), τότε λαμβάνεται τον αριθμό εκατοντάδων που περιέχει. Αν στην συνέχεια βγάλετε τις εκατοντάδες μένει το υπόλοιπο του ακεραίου. Για παράδειγμα. Έστω num=475, h ο αριθμός των εκατοντάδων και r το υπόλοιπο, τότε h=num/100=4, r=num-h*100=75. Στην συνέχεια διαιρούμε το υπόλοιπο με το 10 για να λάβουμε τον αριθμό των δεκάδων. Τέλος, οι μονάδες είναι το υπόλοιπο μετά την αφαίρεση των δεκάδων. Ο κώδικας 7.24 επιστρέφει το λεκτικό ενός ακέραιου από το 0 έως το 999.

```
static String lektiko(int num) { //Κώδικας 7.24
    if (num < 0 || num > 999) {
        throw new RuntimeException();
    }
    if (num == 0) {
        return "μηδέν";
    }
    int ekatontades = num / 100;
    num -= ekatontades * 100;
    int dekades = num / 10;
    num -= dekades * 10;
    String hundreds = "", tens = "", units = "";
    switch (ekatontades) {
        case 0:
            break;
        case 1:
            hundreds = "εκατόν";
            break;
        case 2:
            hundreds = "διακόσια";
            break;
        case 3:
            hundreds = "τριακόσια";
            break;
        case 4:
            hundreds = "τετρακόσια";
            break;
        case 5:
            hundreds = "πεντακόσια";
            break;
        case 6:
            hundreds = "εξακόσια";
            break;
        case 7:
```

```
            hundreds = "επτακόσια";
            break;
        case 8:
            hundreds = "οκτακοσια";
            break;
        case 9:
            hundreds = "εννιακόσια";
            break;
    }
    switch (dekades) {
        case 0:
            break;
        case 1:
            tens = "δέκα";
            break;
        case 2:
            tens = "είκοσι";
            break;
        case 3:
            tens = "τριάντα";
            break;
        case 4:
            tens = "σαράντα";
            break;
        case 5:
            tens = "πενήντα";
            break;
        case 6:
            tens = "εξήντα";
            break;
        case 7:
            tens = "εβδομήντα";
            break;
        case 8:
            tens = "ογδόντα";
            break;
        case 9:
            tens = "ενενήντα";
            break;
    }
    switch (num) {
        case 0:
            break;
        case 1:
            if (tens.equals("δέκα")) {
                tens = "έντεκα";
            } else {
                units = "ένα";
            }
            break;
        case 2:
            if (tens.equals("δέκα")) {
                tens = "δώδεκα";
            } else {
                units = "δύο";
            }
            break;
        case 3:
            units = "τρία";
            break;
        case 4:
```

```
                units = "τέσσερα";
                break;
            case 5:
                units = "πέντε";
                break;
            case 6:
                units = "έξη";
                break;
            case 7:
                units = "επτά";
                break;
            case 8:
                units = "οκτώ";
                break;
            case 9:
                units = "εννέα";
                break;
        }
        String rVal = hundreds + (tens != "" ? " " : "") + tens + (units != "" ?
" " : "") + units;
        //return rVal;
        return rVal.trim();
    }
```

Κώδικας 7.24 *Μετατροπή ακεραίου σε λεκτικό*

Προσέξτε την έκφραση tens!=””?” ”:””. Σημαίνει πως αν η μεταβλητή tens έχει πάρει τιμή διάφορη από ““, τότε επιστρέφουμε ένα διάστημα που θα διαχωρίσει τις εκατοντάδες από τις δεκάδες. Αν όμως η tens παραμένει ““, τότε σημαίνει πως δεν έχουμε δεκάδες, επομένως δεν υπάρχει λόγος να προσθέσουμε το αντίστοιχο διάστημα. Ίδια λογική ισχύει και για την units!=””?” ”:””.

7.9.5 Κλάση MyArrays

Να υλοποιηθεί η κλάση MyArrays που περιλαμβάνει τις ακόλουθες στατικές συναρτήσεις χωρίς να χρησιμοποιηθεί η κλάση Arrays.

```
public static int seqSearch(int[] t, int sE)
```

Πραγματοποιεί σειριακή αναζήτηση στον t για την τιμή sE. Επιστρέφει την θέση στον t της sE εφόσον αυτή βρεθεί, διαφορετικά -1.

```
public static int seqSearch(String[] t, String sE)
```

Πραγματοποιεί σειριακή αναζήτηση στον t για την τιμή sE. Επιστρέφει την θέση στον t της sE εφόσον αυτή βρεθεί, διαφορετικά -1.

```
public static int cntOccurances(int[] t, int sE)
```

Μετράει και επιστρέφει το πλήθος τιμών sE στον t.

```
public static String toString(int[] t)
```

Επιστρέφει μια String αναπαράσταση του t.

```
public static String toString(int[][] t)
```

Επιστρέφει μια String αναπαράσταση του t.

```
public static int[] reverse(int[] t)
```

Επιστρέφει ένα πίνακα ακεραίων που περιέχει τα στοιχεία του t με αντεστραμμένη διάταξη.

```
public static int sum(int[] t)
```

Επιστρέφει το άθροισμα των στοιχείων του t.

```
public static int max(int[] t)
```

Επιστρέφει την θέση του μέγιστου στοιχείου του t.

```
public static boolean equals(int[] tA, int[] tB)
```

Επιστρέφει true αν κάθε στοιχείο του tA είναι και στοιχείο του tB και κάθε στοιχείο του tB είναι και στοιχείο του tA.

```
public static boolean equals(String[] tA, String[] tB)
```

Επιστρέφει true αν κάθε στοιχείο του tA είναι και στοιχείο του tB και κάθε στοιχείο του tB είναι και στοιχείο του tA.

```
public static int binarySearch(int[] t, int sE)
```

Πραγματοποιεί δυαδική αναζήτηση στον t για την τιμή sE. Αν η τιμή βρεθεί επιστρέφει την θέση της στον t, διαφορετικά επιστρέφει -1.

```
public static void swap(int[] t, int idx1, int idx2)
```

Αντιμεταθέτει το στοιχείο στην θέση idx1 του t με το στοιχείο στην θέση idx2.

```
public static int idxOfMin(int[] array, int startIdx)
```

Επιστρέφει την θέση του ελάχιστου στοιχείου του t από την θέση startIdx και μετά.

```
public static void sort(int[] t)
```

Ταξινομεί κατ'αύξουσα αριθμητική σειρά των πίνακα t.

Λύση
Ο κώδικας 7.25 παραθέτει μια λύση της άσκησης και ακολουθούν σχόλια.

```
public class MyArrays {

    public static int seqSearch(int[] t, int sE) {
        for (int i = 0; i < t.length; i++) {
            if (t[i] == sE) {
                return i;
            }
        }
        return -1;
    }

    public static int seqSearch(String[] t, String sE) {
        for (int i = 0; i < t.length; i++) {
            if (t[i].equals(sE)) {
```

```
                return i;
            }
        }
        return -1;
    }

    public static int cntOccurances(int[] t, int sE) {
        int cnt = 0;
        for (int e : t) {
            if (e == sE) {
                cnt++;
            }
        }
        return cnt;
    }

    public static String toString(int[] t) {
        if (t.length == 0) {
            return "[]";
        }
        String s = "[";
        for (int i = 0; i < t.length; i++) {
            s += t[i];
            if (i < t.length - 1) {
                s += ", ";
            }
        }
        s += "]";
        return s;
    }

    public static String toString(int[][] t) {
        if (t.length == 0) {
            return "[]";
        }
        String s = "[";
        for (int i = 0; i < t.length; i++) {
            s += toString(t[i]);
            if (i < t.length - 1) {
                s += ",\n";
            }
        }
        s += "]";
        return s;
    }

    public static int[] reverse(int[] input) {
        int[] rVal = new int[input.length];
        for (int i = 0; i < input.length; i++) {
            rVal[rVal.length - 1 - i] = input[i];
        }
        return rVal;
    }

    public static int sum(int[] t) {
        int sum = 0;
        for (int i : t) {
            sum += i;
        }
        return sum;
    }
```

```
public static int max(int[] t) {
    int max = t[0];
    int idx = 0;
    for (int i = 1; i < t.length; i++) {
        if (t[i] > max) {
            idx = i;
            max = t[i];
        }
    }
    return idx;
}

public static boolean equals(int[] tA, int[] tB) {
    if (tA.length != tB.length) {
        return false;
    }
    for (int cE : tA) {
        if (seqSearch(tB, cE) == -1) {
            return false;
        }
    }
    return true;
}

public static boolean equals(String[] tA, String[] tB) {
    if (tA.length != tB.length) {
        return false;
    }
    for (String cE : tA) {
        if (seqSearch(tB, cE) == -1) {
            return false;
        }
    }
    return true;
}

public static int binarySearch(int[] tbl, int schElement) {
    int from = 0, to = tbl.length - 1, mid = (to - from) / 2;
    while (to >= from) {
        if (tbl[mid] == schElement) {
            return mid;
        } else if (schElement < tbl[mid]) {
            to = mid - 1;
        } else {
            from = mid + 1;
        }
        mid = from + (to - from) / 2;
    }
    return -1;
}

public static void swap(int[] t, int idx1, int idx2) {
    int i = t[idx1];
    t[idx1] = t[idx2];
    t[idx2] = i;
}

public static int idxOfMin(int[] t, int startIdx) {
    int min = t[startIdx];
    int rVal = startIdx;
```

```
        for (int i = startIdx + 1; i < t.length; i++) {
            if (t[i] < min) {
                min = t[i];
                rVal = i;
            }
        }
        return rVal;
    }

    public static void sort(int[] t) {
        for (int i = 0; i < t.length; i++) {
            int mIdx = idxOfMin(t, i);
            if (mIdx != i) {
                swap(t, mIdx, i);
            }
        }
    }

}
```

Κώδικας 7.25 *Η κλάση MyArrays*

Οι δύο πρώτες συναρτήσεις είναι απλές. Η πρώτη κάνει σειριακή αναζήτηση σε πίνακα ακεραίων και η δεύτερη σε πίνακα αλφαριθμητικών σειρών. Η μόνη λεπτομέρεια που πρέπει να προσέξουμε είναι πως για μεν τους ακέραιους, η σύγκριση γίνεται με τον τελεστή ==, για δε τις αλφαριθμητικές σειρές, η σύγκριση γίνεται με την συνάρτηση equals.

Η cntOccurances επίσης βασίζεται σε σειριακή προσπέλαση όλων των στοιχείων του πίνακα. Κάθε φορά που βρίσκει τιμή ίση με την δεύτερη παράμετρό της, αυξάνει έναν μετρητή τον οποίον και επιστρέφει.

Η toString για μονοδιάστατο πίνακα ακεραίων, καταρχάς επιστρέφει [] αν η παράμετρός της έχει μήκος μηδέν. Στην συνέχεια, αρχικοποιεί την τοπική μεταβλητή s στην σειρά [. Μετά προσπελαύνει ένα προς ένα τα στοιχεία τα στοιχεία του t και προσθέτει στο s την τιμή του στοιχείου ακολουθούμενη από κόμμα και διάστημα. Εξαιρεί όμως την πρόσθεση κόμματος και διαστήματος στο τελευταίο στοιχείο. Τέλος, κλείνει το s προσθέτοντας το].

Η toString για δυσδιάστατο πίνακα έχει ανάλογη λογική με την toString για μονοδιάστατο. Μόνο που εδώ τα στοιχεία είναι πίνακες, οπότε χρησιμοποιεί την toString για να πάρει την αναπαράσταση των μονοδιάστατων πινάκων στοιχείων του t.

Η reverse ορίζει τον πίνακα rVal με μήκος ίσο με το μήκος του πίνακα input. Στην συνέχεια αντιγράφει ένα προς ένα τα στοιχεία του input στον rVal σε αντιδιαμετρικά αντίθετη θέση. Προσέξτε πως όταν ο δείκτης i είναι ίσος με μηδέν, δηλ. αναφερόμαστε στην πρώτη θέση του input, η έκφραση rVal.length-1-i αναφέρεται στην τελευταία θέση του rVal, όταν ο δείκτης i είναι ίσος με 1, δηλ. αναφερόμαστε στην δεύτερη θέση του input, η έκφραση rVal.length-1-i αναφέρεται στην προτελευταία θέση του rVal, κοκ.

Η sum εφαρμόζει σειριακή προσπέλαση και προσθέτει κάθε τιμή του t στην τοπική μεταβλητή sum την οποία τελικά επιστρέφει.

Παρόμοια η max εφαρμόζει σειριακή προσπέλαση. Αρχικοποιεί την τοπική μεταβλητή max στην τιμή στην θέση μηδέν του t. Στην συνέχεια, επισκέπτεται τα στοιχεία του t από την θέση 1 και μετά και κάθε φορά που βρίσκει μεγαλύτερη τιμή ενημερώνει την max και την idx που κρατάει την θέση της max. Επιστρέφει την idx.

Η equals για πίνακες ακεραίων συγκρίνει καταρχάς τα μήκη των πινάκων. Αν δεν είναι ίσα επιστρέφει false. Αν δεν επιστρέψει από τον έλεγχο των μηκών, ψάχνει στην συνέχεια αν κάθε στοιχείο του tA είναι και στοιχείο του tB. Προσέξτε πως ο αντίστροφος έλεγχος είναι περιττός από την στιγμή που τα μήκη είναι ίσα.

Ίδια είναι και η λογική της equals για πίνακες από Strings.

Η binarySearch εφαρμόζει την λογική που αναπτύξαμε στην ενότητα 6.5.2.

Η swap κρατάει την πρώτη από τις τιμές που αντιμεταθέτει σε προσωρινή μεταβλητή, εκχωρεί την δεύτερη τιμή στην πρώτη και στην θέση της δεύτερης τιμής εκχωρεί την τιμή της προσωρινής μεταβλητής.

Η sort υλοποιεί τον αλγόριθμο που παρατίθεται στον κώδικα 6.18. Η διαφορά της είναι πως την εύρεση του μικρότερου στοιχείου την αναθέτει στην idxMin.

7.9.6 Ταξινόμηση δυσδιάστατου πίνακα με βάση το άθροισμα γραμμών

Να υλοποιηθεί συνάρτηση με διεπαφή public static void sortSubArraysBySum(int[][] t) που ταξινομεί τους πίνακες του t κατά αύξουσα αριθμητική σειρά του αθροίσματος των στοιχείων τους. Για παράδειγμα, ο πίνακας

```
{
  {100, 200, 300, 400},
  {1, 2, 3, 4, 5},
  {10, 20, 30, 40, 50, 60}
}
```

θα πρέπει να ταξινομηθεί ως

```
{
   {1, 2, 3, 4, 5},
   {10, 20, 30, 40, 50, 60},
   {100, 200, 300, 400}
}
```

καθώς 1+2=3+4+5 < 10+20+30+40+50+60 < 100+200+300+400΄

Λύση
Θα εφαρμόσουμε παρόμοια λογική με αυτήν της ταξινόμησης μονοδιάστατου πίνακα. Έτσι, θα χρειαστούμε μια συνάρτηση που εντοπίζει τον πίνακα-στοιχείο του t που έχει το μικρότερο άθροισμα.

```
static int idxOfMinSum(int[][] t, int startIdx) {
    int min = MyArrays.sum(t[startIdx]);
    int rVal = startIdx;
    for (int i = startIdx + 1; i < t.length; i++) {
        if (MyArrays.sum(t[i]) < min) {
            min = MyArrays.sum(t[i]);
            rVal = i;
        }
    }
    return rVal;
}
```

Κώδικας 7.26 *Εύρεση πίνακα στοιχείου με το μικρότερο άθροισμα*

Ο κώδικας 7.26 παρουσιάζει την idxOfMinSum που αρχίζοντας μετά από την θέση startIdx εντοπίζει ποιος πίνακας-στοιχείο του t έχει το μικρότερο άθροισμα στοιχείων.
Έχοντας στην διάθεσή μας την idxOfMinSum, μπορούμε να υλοποιήσουμε την απαιτούμενη ταξινόμηση εφαρμόζοντας αντίστοιχη λογική με αυτήν της ταξινόμησης μονοδιάστατου πίνακα. Ο κώδικας 7.27 παρουσιάζει την σχετική υλοποίηση.

```
public static void sortSubArraysBySum(int[][] t) { //Κώδικας 7.27
    for (int i = 0; i < t.length; i++) {
        int mIdx = idxOfMinSum(t, i);
        if (mIdx != i) {
            int[] local = t[i];
            t[i] = t[mIdx];
            t[mIdx] = local;
        }
    }
}
```

Κώδικας 7.27 *Ταξινόμηση σύμφωνα με το άθροισμα των γραμμών δυσδιάστατου πίνακα*

7.9.7 Διαχείριση ψηφιοσειρών

Να υλοποιηθεί η κλάση BitMaps που περιλαμβάνει τις λειτουργίες

```
public static int setBit(int i, int p)
```

Θέτει στο bit που βρίσκεται στην θέση p αντιγράφου του ακεραίου i την τιμή 1. Επιστρέφει τον παραγόμενο ακέραιο.

```
public static int resetBit(int i, int p)
```

Θέτει στο bit που βρίσκεται στην θέση p αντιγράφου του ακεραίου i την τιμή 0. Επιστρέφει τον παραγόμενο ακέραιο.

```
public static int getBit(int i, int p)
```

Επιστρέφει την τιμή του bit που βρίσκεται στην θέση p του ακεραίου i

```
public static int reverseBit(int i, int p)
```

Αντιστρέφει σε αντίγραφο του i το bit στην θέση p. Επιστρέφει τον παραγόμενο ακέραιο.

Λύση

Παρουσιάζουμε τον κώδικα της κλάσης BitMap ακολουθούμενο από τις αναγκαίες επεξηγήσεις.

```
public class BitMap { //Κώδικας 7.28

    public static int setBit(int i, int p) {
        int local = 1;
        local <<= p;
        return i | local;
    }

    public static int resetBit(int i, int p) {
        int local = setBit(0, p);
        local = ~local;
        return i & local;
    }

    public static int getBit(int i, int p) {
        int local = setBit(0, p);
        if ((local & i) == local) {
            return 1;
        }
        return 0;
    }

    public static int reverseBit(int i, int p) {
        if (getBit(i, p) == 1) {
            return resetBit(i, p);
        } else {
            return setBit(i, p);
        }
    }

    public static void main(String[] args) {
        System.out.println(setBit(1, 1));
        System.out.println(resetBit(3, 0));
        System.out.println(getBit(3, 1));
        System.out.println(getBit(3, 2));
        System.out.println(reverseBit(3,0));
    }
}
```

Κώδικας 7.28 *Διαχείριση ψηφιοσειρών*

Η συνάρτηση setBit, στην πρώτη γραμμή αρχικοποιεί την μεταβλητή local στην τιμή 1. Επομένως, ο ψηφιοχάρτης της local γίνεται 00000000000000000000000000000001, δηλ. 31 μηδενικά ακολουθούμενα από ένα άσσο. Παρότι ο int έχει μήκος 32 bits, για απλοποίηση, στο εξής θα θεωρούμε ότι είναι μήκους 8 bits. Με αυτήν την υπόθεση, η local έχει το ακόλουθο περιεχόμενο 00000001. Στην συνέχεια εκτελείται διολίσθηση στα αριστερά για όσες θέσεις καθορίζει η παράμετρος p. Για παράδειγμα, θεωρήστε την κλήση setBit(1,1) στην πρώτη γραμμή της main. Η p εδώ έχει την τιμή 1. Επομένως, η local θα αποκτήσει περιεχόμενο 00000010. Στην συνέχεια γίνεται bitwise OR μεταξύ της παραμέτρου i και της local. Η παράμετρος i έχει τιμή 1, δηλ. περιεχόμενο 00000001. Συνεπώς 00000010 | 00000001=00000011, δηλ. 3 στο δεκαδικό σύστημα που είναι και η τιμή που εμφανίζεται από την println στην πρώτη γραμμή της main.

Με άλλα λόγια διαμορφώσαμε ένα ψηφιοχάρτη στην μεταβλητή local ο οποίος έχει όλα τα bits ίσα με το 0, εκτός από το bit που θέλουμε να κάνουμε 1. Στην συνέχεια, το bitwise OR μεταξύ της local και οποιουδήποτε ακέραιου, x, θα μετατρέψει σε 1 το bit του x που βρίσκεται στην θέση που έχει η local τιμή 1.

Παρόμοια εργαζόμαστε και για την resetBit. Ορίζουμε την μεταβλητή local που έχει όλα τα bits 0 εκτός από το bit στην θέση p. Με bitwise not αντιστρέφουμε όλα τα bits της local οπότε σε όλες τις θέσεις έχει πλέον 1 εκτός από την θέση p που έχει 0. Στην συνέχεια το bitwise and με την παράμετρο i, έχει ως αποτέλεσμα όλα τα bits να παραμείνουν ως έχουν εκτός από το bit στην θέση p που αποκτά την τιμή 0.

Η getBit δημιουργεί τον ακέραιο local που αρχικοποιείται ώστε ο ψηφιοχάρτης του να αποτελείται από 0 εκτός από την θέση p. Επομένως, αν το bitwise and μεταξύ της local και της i επιστρέψει ακέραιο ίσο με την local αυτό σημαίνει πως το bit στην θέση p της i είναι 1 αλλιώς 0.

Τέλος η reverseBit είναι απλή καθώς βασίζεται στις getBit, resetBit και setBit και δεν κάνει απευθείας πράξεις σε επίπεδο ψηφίου.

7.10 Ασκήσεις προς λύση

1. Να αναπτύξετε συνάρτηση double abs(double n) που επιστρέφει την απόλυτη τιμή του n

2. Να αναπτύξετε συνάρτηση που υπολογίζει και τυπώνει την τιμή της f(x)=x4+5x3+2x2+8x+1 για τις τιμές του x=1..100, όπου x ακέραιος.

3. Να αναπτύξετε συνάρτηση που τυπώνει τους ν πρώτους όρους της ακολουθίας Fibonacci

4. Να αναπτυχθεί συνάρτηση που επιστρέφει true αν η ακέραιη παράμετρός της είναι πρώτος αριθμός.

5. Να αναπτυχθεί συνάρτηση που τυπώνει τους πρώτους αριθμούς από το 1 έως το 100.

6. Να αναπτυχθεί συνάρτηση που τυπώνει τους αριθμούς από το 1 έως το 100 που δεν είναι πρώτοι.

7. Να αναπτυχθεί συνάρτηση που λαμβάνει μια boolean παράμετρο και επιστρέφει έναν πίνακα με τους χαρακτήρες της Αγγλικής, κεφαλαίους ή πεζούς, ανάλογα με την τιμή της παραμέτρου.

8. Να υλοποιηθεί Java application που λαμβάνει 3 παραμέτρους που αντιπροσωπεύουν μια εξίσωση 2ου βαθμού ($ax^2+bx+c=0$) και τυπώνει την λύση της εφόσον υπάρχει ή διαφορετικά το μήνυμα δεν υπάρχει λύση, διακρίνουσα <0.

9. Να αναπτυχθεί συνάρτηση που επιστρέφει το άθροισμα των ψηφίων ενός ακεραίου.

10. Να αναπτυχθεί συνάρτηση με διεπαφή static int max(int…in) που λαμβάνει μια σειρά από ακεραίους και επιστρέφει τον μεγαλύτερο.

11. Να αναπτυχθεί συνάρτηση που λαμβάνει ως παράμετρο έναν πραγματικό αριθμό που αναπαριστά θερμοκρασία σε Celsius και επιστρέφει την θερμοκρασία σε Fahrenheit.

12. Υπερφορτώστε την equals της άσκησης 7.8.5 ώστε να εξετάζει και πίνακες πραγματικών.

13. Αναπτύξτε συνάρτηση με διεπαφή static int[] resize(int[] t, int newSize) που επιστρέφει ένα αντίγραφο του t με μήκος newSize. Αν η τιμή της newSize είναι μικρότερη από το αρχικό μήκος του t, τότε ενδέχεται να απωλεσθούν δεδομένα από τον πίνακα.

14. Οποιοσδήποτε βαθμός μικρότερος του 5 αντιστοιχεί σε αποτυχία. Η βαθμολογία από 5 έως 6,4 αντιστοιχεί στο χαρακτηρισμό «καλώς», από 6,5 έως 8,4 στο «λίαν καλώς» και από 8,5 έως 10 στο «άριστα». Να

αναπτύξετε συνάρτηση που λαμβάνει ως είσοδο ένα βαθμό από 0 έως 10 και επιστρέφει τον αντίστοιχο χαρακτηρισμό.

15. Να αναπτυχθεί εφαρμογή Java που παράγει 100 τυχαίους αριθμούς από το 1 έως το 40 και στην συνέχεια μετράει

16. Πόσοι από τους παραγόμενους αριθμούς είναι μοναδικοί

17. Πόσοι είναι άρτιοι και πόσοι περιττοί

18. Ποιοι αριθμοί από το 1 έως το 40 δεν υπάρχουν καθόλου στην παραγόμενη ακολουθία

19. Ποιος ή ποιοι αριθμοί εμφανίζονται περισσότερες φορές

20. Να αναπτυχθεί συνάρτηση με διεπαφή static void reverse(char[] t) που αντιστρέφει τις θέσεις των στοιχείων του t, δηλ. τοποθετεί το στοιχείο από την πρώτη στην τελευταία θέση του πίνακα, από την δεύτερη στην προτελευταία, κοκ.

21. Να αναπτυχθεί συνάρτηση που επιστρέφει τον μέγιστο κοινό διαιρέτη ενός πλήθους ακεραίων.

22. Να αναπτυχθεί συνάρτηση που επιστρέφει το ελάχιστο κοινό πολλαπλάσιο ενός πλήθους ακεραίων.

23. Να αναπτύξετε συνάρτηση που λαμβάνει ως παράμετρο ένα πίνακα ακεραίων και επιστρέφει επίσης ένα πίνακα ακεραίων που αναπαριστά το σύνολο των ακεραίων του πίνακα εισόδου. Προσέξτε πως σε ένα σύνολο, ένα στοιχείο μπορεί να υπάρχει μόνο μία φορά.

24. Να αναπτύξετε συνάρτηση που υλοποιεί την ταξινόμηση φυσαλίδας σε πίνακα πραγματικών.

25. Να αναπτύξετε συνάρτηση που εξετάζει αν ένας πίνακας είναι ταξινομημένος. Η συνάρτηση να επιστρέφει 0 για αταξινόμητους πίνακες, -1 για ταξινομημένους κατά φθίνουσα σειρά και 1 για ταξινομημένους κατά αύξουσα σειρά.

26. Να αναπτύξετε συνάρτηση που συγχωνεύει δύο ταξινομημένους πίνακες σε έναν νέο πίνακα επίσης ταξινομημένο.

27. Να αναπτύξετε συνάρτηση που λαμβάνει ως παράμετρο δύο μονοδιάστατους πίνακες ακεραίων και επιστρέφει έναν πίνακα που περιέχει μόνο τα κοινά στοιχεία των παραμέτρων της έτσι ώστε κάθε κοινό στοιχείο να υπάρχει μόνο μία φορά στον πίνακα επιστροφής.

28. Να αναπτύξετε συνάρτηση που λαμβάνει ως παράμετρο δύο τετραγωνικούς πίνακες πραγματικών και επιστρέφει το άθροισμά τους. Σημειώστε πως τετραγωνικοί λέγονται οι πίνακες που έχουν ίδιο αριθμό γραμμών και στηλών.

29. Να αναπτύξετε συνάρτηση που ανακατανέμει τα στοιχεία ενός μονοδιάστατου πίνακα.

30. Να αναπτύξετε συνάρτηση που ανακατανέμει τα στοιχεία ενός δυσδιάστατου πίνακα έτσι ώστε ένα τελικό στοιχείο να είναι πιθανό να βρεθεί σε οποιαδήποτε γραμμή και στήλη του πίνακα.

31. Ο τόκος ενός κεφαλαίου υπολογίζεται με βάση το επιτόκιο. Όταν ο τόκος μίας περιόδου προστίθεται στο κεφάλαιο, τότε ο τόκος της επόμενης περιόδου υπολογίζεται επι του αρχικού κεφαλαίου αυξημένου κατά τον τόκο της προηγούμενης περιόδου. Το φαινόμενο αυτό ονομάζεται ανατοκισμός. Να αναπτύξετε συνάρτηση που λαμβάνει ως παραμέτρους το κεφάλαιο, το επιτόκιο και τον αριθμό των περιόδων και υπολογίζει το σύνολο του τόκου.

32. Να αναπτυχθεί συνάρτηση που επιστρέφει την κύρια διαγώνιο τετραγωνικού πίνακα.

33. Να αναπτυχθεί συνάρτηση που επιστρέφει το Ελάχιστο Κοινό Πολλαπλάσιο μιας σειράς ακεραίων.

34. Να αναπτυχθεί συνάρτηση που επιστρέφει τον Μέγιστο Κοινό Διαιρέτη μιας σειράς ακεραίων.

35. Έστω nb ένας αριθμός εκφρασμένος σε θεσιακό αριθμητικό σύστημα με βάση b. Η μετατροπή του στο δεκαδικό σύστημα αρίθμησης γίνεται από τον τύπο

$$n_b = s_k \times b^k + s_{k-1} \times b^{k-1} + \cdots + s_1 \times b^1 + s_0 \times b^0 + s_{-1} \times b^{-1} + s_{-2} \times b^{-2} + \cdots + s_m \times b^m$$

Θεωρήστε ως θεσιακά αριθμητικά συστήματα μόνο αυτά που μας ενδιαφέρουν στην Πληροφορική, δηλ. το δυαδικό, το οκταδικό και το δεκαεξαδικό.

36. Να υλοποιηθεί συνάρτηση που λαμβάνει δύο παραμέτρους τύπου String. Η πρώτη αναπαριστά έναν αριθμό που μπορεί να περιέχει ή όχι , υποδιαιρέσεις της μονάδας, π.χ. 101101.98, 25634.01, AB32.A ή 101101, 25634, AB32. Η δεύτερη αναπαριστά την βάση του αριθμητικού συστήματος στο οποίο εκφράζεται ο αριθμός και μπορεί να λάβει τις τιμές 2, 8 ή 16. Η συνάρτηση επιστρέφει την τιμή του αριθμού στο δεκαδικό σύστημα ως double.

Καλέστε την συνάρτηση και μετατρέψτε του ακόλουθους αριθμούς στο δεκαδικό σύστημα.

```
110111102
65128
A2E3416
10110.0112
763.2348
A2E34.A16
BB135.1B16
```

Βρείτε κατάλληλο υπολογιστή μετατροπών στο διαδίκτυο και ελέγξτε τα αποτελέσματα.

Κεφάλαιο 8

Σύνοψη
Σε αυτήν την ενότητα παρουσιάζονται τα βασικά στάδια της ανάπτυξης εφαρμογών προσαρμοσμένα στις γνώσεις που έχει αποκομίσει ο αναγνώστης μέχρι αυτό το σημείο του βιβλίου. Ο στόχος είναι να διευκολυνθεί ο αρχάριος προγραμματιστής να υιοθετήσει ένα δομημένο τρόπο ανάλυσης, σχεδίασης και υλοποίησης τόσο των ασκήσεων του βιβλίου όσο και γενικότερων εφαρμογών. Προς αυτόν τον στόχο, παρουσιάζονται οι έννοιες του δομημένου προγραμματισμού, τα στάδια της ανάπτυξης, η συλλογή απαιτήσεων, η ανάλυση και ο σχεδιασμός, η τεκμηρίωση και απολαθοποίηση του κώδικα καθώς και οι διαδικασίες και τα εργαλεία ελέγχου της ορθότητάς του.

Προαπαιτούμενη γνώση
Οι θεμελιώδεις τύποι και βασική γνώση του τύπου String. Βασική είσοδος και έξοδος. Η λειτουργία των τελεστών. Οι δομές επιλογής και οι επαναληπτικές δομές. Η διαχείριση των πινάκων. Οι συναρτήσεις, η έννοια και βασική διαχείριση των απροσδόκητων λαθών, οι προσδιοριστές προσπέλασης και η χρήση των πακέτων.

Λέξεις κλειδιά
Δομημένος προγραμματισμός, απαιτήσεις, ανάλυση, σχεδιασμός, τεκμηρίωση, απολαθοποίηση.

8 Ανάπτυξη εφαρμογών

Η ανάπτυξη των εφαρμογών δεν αρχίζει ούτε τελειώνει με την συγγραφή του κώδικα σε κάποια γλώσσα προγραμματισμού. Αντίθετα, μια σειρά από διαδικασίες που λαμβάνουν χώρα πριν, μετά και κατά την διάρκεια της συγγραφής του κώδικα διαδραματίζουν κρίσιμο ρόλο στην ανάπτυξη μιας ολοκληρωμένης εφαρμογής που απαντά αποτελεσματικά στις ανάγκες των χρηστών της. Στις διαδικασίες αυτές συμπεριλαμβάνονται κατ' ελάχιστο, η συλλογή και ανάλυση των απαιτήσεων, ο σχεδιασμός της εφαρμογής, η κωδικοποίηση και απολαθοποίηση, η τεκμηρίωση του κώδικα και ο έλεγχος της ορθότητάς του.

Αυτές τις διαδικασίες παρουσιάζει αυτή η ενότητα προσαρμοσμένες στις ανάγκες μιας εισαγωγής στον προγραμματισμό. Ο σκοπός της ενότητας δεν είναι να αντικαταστήσει την μελέτη των διάφορων τεχνικών ανάλυσης και σχεδίασης λογισμικού. Εξάλλου η μελέτη αυτών των τεχνικών είναι έξω από την εμβέλεια αυτού του βιβλίου. Ωστόσο κάποιες κατευθυντήριες γραμμές είναι χρήσιμες και μπορεί να βοηθήσουν στην αποτελεσματική ανάπτυξη ορθότερων και δομημένων εφαρμογών ήδη από το στάδιο της εισαγωγής στον Προγραμματισμό.

Με βάση αυτό το σκεπτικό, σε αυτήν την ενότητα παρουσιάζεται μια απλοποιημένη εκδοχή των διαδικασιών συλλογής και ανάλυσης απαιτήσεων και σχεδιασμού λογισμικού. Οι υπόλοιπες διαδικασίες, δηλ. η υλοποίηση, η απολαθοποίηση, η τεκμηρίωση και ο έλεγχος ορθότητας του προγράμματος, παρουσιάζονται στο πλαίσιο της Java, αξιοποιούν εργαλεία που σχετίζονται άμεσα με την Java και εκθέτονται με μεγαλύτερη λεπτομέρεια. Δεν θα πρέπει όμως να θεωρηθεί ότι τα θέματα που πραγματεύεται το κεφάλαιο αυτό αναπτύσσονται σε πλήρη έκταση. Ο αναγνώστης που επιθυμεί να εμβαθύνει θα πρέπει να ανατρέξει σε σχετική βιβλιογραφία. Ας εξηγήσουμε όμως καταρχάς την έννοια του δομημένου προγράμματος

8.1 Δομημένος Προγραμματισμός

Οι στόχοι του Δομημένου Προγραμματισμού (Structured Programming) είναι να καταστήσει τα προγράμματα ευανάγνωστα, να περιορίσει τα λάθη σε αυτά και να επιταχύνει τον χρόνο ανάπτυξης. Το κύριο συστατικό του είναι η ρουτίνα, δηλ. το γνωστό μας υποπρόγραμμα ή η συνάρτηση στο πλαίσιο της Java. Η ρουτίνα θα πρέπει να είναι αυτόνομη και να δίνει απάντηση σε ένα σαφώς ορισμένο πρόβλημα ενώ ο συνδυασμός των ρουτινών θα πρέπει να γίνεται με τρόπο που δομημένα ανακλά το πρόβλημα που αποτελεί το κίνητρο της ανάπτυξης.

Ο Δομημένος Προγραμματισμός προέκυψε την δεκαετία του 1950 με την ανάπτυξη των γλωσσών Algol [1] και σύντομα επικράτησε τόσο στην ακαδημαϊκή κοινότητα όσο και στον χώρο των επαγγελματιών προγραμματιστών. Η τεχνολογία του προγραμματισμού εκείνη την εποχή περιλάμβανε περίπου τις έννοιες που έχουν παρουσιαστεί μέχρι εδώ σε αυτό το βιβλίο. Ωστόσο, ο δομημένος προγραμματισμός κάθε άλλο παρά ξεπερασμένο μοντέλο θεωρείται. Αντίθετα αποτελεί θεμελιώδες μοντέλο προγραμματισμού καθώς ενυπάρχει

στα περισσότερα πιο σύγχρονα μοντέλα όπως ο Αντικειμενοστρεφής Προγραμματισμός ή ο Προγραμματισμός Πρακτόρων (Agent Oriented Programming).

Αρχικά το μοντέλο βασίσθηκε στο θεώρημα του δομημένου προγράμματος (structured program theorem) [2] σύμφωνα με το οποίο κάθε πρόγραμμα μπορεί να αναπτυχθεί συνδυάζοντας ρουτίνες με τρεις μόνο βασικές δομές.

1. Σειριακή εκτέλεση, δηλ. η μία ρουτίνα εκτελείται σειριακά μετά από μία άλλη
2. Επιλογή, δηλ. ο έλεγχος μιας λογικής συνθήκης αποφασίζει ποια ρουτίνα θα εκτελεσθεί
3. Επανάληψη, δηλ. μια ή περισσότερες ρουτίνες που συνδυάζονται σειριακά ή με επιλογή μπορούν να επαναλαμβάνονται κάτω από τον έλεγχο μιας λογικής συνθήκης.

Κάποιοι συγγραφείς προσθέτουν και την αναδρομή, δηλ. την δομή κατά την οποία μια ρουτίνα καλεί τον εαυτό της κάτω από μία συνθήκη ελέγχου. Ωστόσο, η αναδρομή δύσκολα μπορεί να θεωρηθεί βασική δομή καθώς τα προβλήματα στα οποία αξιοποιείται μπορούν να επιλυθούν και με χρήση των τριών προαναφερόμενων δομών.

Σύντομα, η αποδοχή του μοντέλου έλαβε καθολικές διαστάσεις. Χαρακτηριστικό δείγμα η εξαφάνιση εντολών goto από τα προγράμματα παγκοσμίως. Οι εντολές τύπου goto αποτελούν ανεξέλεγκτες μεταπηδήσεις (unconditional jumps) από ένα σημείο του προγράμματος σε ένα άλλο οπότε παραβιάζουν το θεώρημα του δομημένου προγράμματος ενώ στην πράξη αποδεικνυόταν πως συχνά αποτελούσαν πηγή λαθών.

Στην πορεία στο μοντέλο έγιναν κάποιες παραλλαγές με κυριότερες την προσθήκη διαχείρισης εξαιρέσεων (Exceptions) και την πρόωρη έξοδο (early exit) από επαναληπτικές δομές και ρουτίνες.

Ο μηχανισμός των εξαιρέσεων αναπτύχθηκε την δεκαετία του 1960 [3] σαν μηχανισμός διαχείρισης απροσδόκητων γεγονότων που είναι πιθανό να συμβούν κατά την εκτέλεση του προγράμματος, π.χ. ένα πρόγραμμα επιχειρεί να ανοίξει ένα αρχείο για να επεξεργαστεί τα δεδομένα του αλλά το αρχείο δεν βρίσκεται στην καθορισμένη θέση στον δίσκο. Οι εξαιρέσεις συνδυάζουν δεδομένα από δύο διαφορετικές περιοχές του κώδικα, από το εσωτερικό της συνάρτησης κατά την εκτέλεση της οποίας συμβαίνει το απροσδόκητο γεγονός και από την περιοχή του κώδικα που καλεί αυτήν την συνάρτηση. Με αυτήν την έννοια, οι εξαιρέσεις δεν υπακούν στις τρεις αρχές του θεωρήματος του δομημένου προγράμματος. Ωστόσο είναι αναγκαία η χρησιμοποίησή τους και όλες οι σύγχρονες γλώσσες ενσωματώνουν κάποιο μηχανισμό εξαιρέσεων.

Μια συνέπεια του θεωρήματος του δομημένου προγράμματος είναι η λεγόμενη μοναδική έξοδος (single exit). Σύμφωνα με την αρχή της μοναδικής εξόδου, οι επαναληπτικές δομές και οι συναρτήσεις οφείλουν να τερματίζουν από ένα μόνο συγκεκριμένο σημείο. Σήμερα υπάρχουν διαφορετικές προσεγγίσεις επάνω σε αυτό το θέμα με κυριότερη την πρόωρη έξοδο (early exit). Με βάση την προσέγγιση αυτή μια συνάρτηση μπορεί να ελέγξει κάποιες συνθήκες και να προχωρήσει σε έξοδο από οποιοδήποτε σημείο. Κάτι τέτοιο είναι ασφαλέστερο να γίνει σε ένα αρχικό στάδιο πριν η συνάρτηση προχωρήσει σε διαχείριση πόρων. Σε πολλές περιπτώσεις, με την πρόωρη έξοδο ο κώδικας γίνεται απλούστερος, ασφαλέστερος και πιο κατανοητός. Η πρόωρη έξοδος εφαρμόζεται σε πολλούς κώδικες όπου θεωρείται χρήσιμη σε αυτό το βιβλίο. Η πρόωρη έξοδος υποστηρίζεται από όλες σχεδόν τις σύγχρονες γλώσσες και για τις επαναληπτικές δομές με την χρήση των λέξεων-κλειδιών continue και break.

Οι έννοιες αναγνωσιμότητα προγράμματος (code readability), επαναχρησιμοποίηση κώδικα (code reusability) και αφαιρετικότητα (abstraction) σχετίζονται στενά μεταξύ τους αλλά και με το υπόδειγμα του δομημένου προγραμματισμού. Ωστόσο, οι έννοιες αυτές συναντώνται και σε άλλα μοντέλα προγραμματισμού, όπως ο αντικειμενοστρεφής προγραμματισμός. Η συζήτηση εδώ όμως αναφέρεται στις συγκεκριμένες έννοιες και το ειδικό περιεχόμενο που αυτές προσλαμβάνουν στο επίπεδο του δομημένου προγραμματισμού.

Ας δούμε όμως τι σημαίνει αφαιρετικότητα και επαναχρησιμοποίηση κώδικα στην πράξη και στο πλαίσιο της Java. Ας υποθέσουμε πως έχουμε ένα πρόβλημα στα πλαίσια του οποίου χρειάζεται να ψάξουμε σε ένα πίνακα μήκους 10 να διαπιστώσουμε αν υπάρχει ένα στοιχείο ή όχι. Μία λύση είναι να παρεμβάλλουμε εκεί όπου χρειάζεται ένα κώδικα που ψάχνει στον συγκεκριμένο πίνακα. Μια καλύτερη λύση όμως είναι να κάνουμε μια συνάρτηση που λαμβάνει παραμετρικά τον πίνακα και το υπό αναζήτηση στοιχείο και επιστρέφει κατά πόσο το στοιχείο περιλαμβάνεται στον πίνακα. Η συνάρτηση αυτή μπορεί να χρησιμοποιηθεί και αλλού όπου είναι χρήσιμη. Η συνάρτηση θα πρέπει να σχεδιαστεί με όσο το δυνατόν μεγαλύτερη αφαιρετικότητα, δηλ. με όσο το δυνατόν μικρότερη σχέση με το συγκεκριμένο πρόβλημα που αποτέλεσε κίνητρο για την ανάπτυξή της. Με άλλα λόγια η συνάρτηση δεν θα πρέπει να αξιοποιεί το γεγονός ότι ο πίνακας στον οποίο θέλουμε να ψάξουμε έχει μήκος 10. Αντίθετα, θα πρέπει να γραφεί κατά τρόπο που να μπορούμε να την καλέσουμε για οποιοδήποτε πίνακα ανεξάρτητα από το μήκος του. Είναι προφανές πως η αφαιρετικότητα συμβάλει στην δυνατότητα επαναχρησιμοποίησης του κώδικα.

Στο συγκεκριμένο πρόβλημα αναζήτησης μας ενδιαφέρει αν μια τιμή υπάρχει σε έναν πίνακα, επομένως, η συνάρτηση θα μπορούσε να είναι τύπου boolean. Παρόλα αυτά μπορούμε να αυξήσουμε την δυνατότητα επαναχρησιμοποίησης της συνάρτησης αν αυτή επιστρέφει ακέραιο ο οποίος στην περίπτωση που βρεθεί το στοιχείο μεταφέρει την θέση του στον πίνακα αλλιώς μια τιμή που δεν μπορεί να αποτελεί θέση σε πίνακα, π.χ. μια αρνητική τιμή. Έτσι το πρόγραμμα-πελάτης, δηλ. η εφαρμογή που θα καλεί την συνάρτησή μας, μπορεί να την καλέσει απλώς για να ελέγξει αν υπάρχει ή όχι το στοιχείο αλλά μπορεί εφόσον το χρειάζεται να αξιοποιήσει την θέση του στοιχείου στον πίνακα.

Ας σημειώσουμε εδώ πως η έννοια της δόμησης δεν αφορά στενά την κωδικοποίηση σε μια γλώσσα προγραμματισμού αλλά σχετίζεται και με την ανάλυση (structured analysis) και με τον σχεδιασμό (structured design).

8.2 Τα Στάδια ανάπτυξης

Στην υλοποίηση των εφαρμογών θα διευκολυνθούμε αν ακολουθούμε τα βήματα:

Συλλογή απαιτήσεων (Requirements Gathering). Θα πρέπει να καταγράψουμε με όσο μεγαλύτερη ακρίβεια τις απαιτήσεις των δυνητικών χρηστών της εφαρμογής, δηλ. τι πρέπει να κάνει η εφαρμογή. Η ανάπτυξη σεναρίων χρήσης (use cases) μπορεί να φανεί πολλή χρήσιμη. Τα σενάρια χρήσης περιγράφουν καταστάσεις κατά τις οποίες το υπό ανάπτυξη λογισμικό μπορεί να είναι χρήσιμο ή υπό μία διαφορετική έννοια, περιγράφουν την ανταπόκριση του λογισμικού σε πιθανές εισόδους του χρήστη. Αποφάσεις σχετικά με το πώς θα επιτευχθούν οι απαιτήσεις ή ποια εργαλεία λογισμικού θα χρησιμοποιηθούν για τον σκοπό αυτό δεν περιλαμβάνονται σε αυτό το στάδιο.

Ανάλυση Απαιτήσεων (Requirements Analysis). Η ανάλυση των απαιτήσεων έχει σαν σκοπό να αντιστοιχίσει τις απαιτήσεις των χρηστών σε μια σειρά σαφώς προσδιορισμένων εργασιών του υπό ανάπτυξη συστήματος.

Σχεδιασμός του Συστήματος (System Design). Ο στόχος εδώ είναι ο σχεδιασμός των αυτόνομων ρουτινών και της σύνθεσής τους με σκοπό την υλοποίηση των εργασιών του συστήματος όπως αυτές καθορίστηκαν στο στάδιο της ανάλυσης.

Υλοποίηση (Implementation). Εδώ περνάμε στην υλοποίηση των ρουτινών που εντοπίστηκαν στον σχεδιασμό, δηλ. στην κωδικοποίησή τους σε ένα πλαίσιο προγραμματισμού.

Έλεγχος (test). Τα λάθη στους κώδικες είναι συχνό και ως ένα βαθμό αναπόφευκτο πρόβλημα. Σε αυτό το στάδιο, θα πρέπει να ελέγξουμε συστηματικά την εκτέλεση και τα αποτελέσματα της εφαρμογής μας με σκοπό να εντοπίσουμε πιθανά προβλήματα.

Επειδή καμία εφαρμογή δεν παράγεται πλήρως ολοκληρωμένη, συχνά τα παραπάνω βήματα επαναλαμβάνονται είτε γιατί αποκαλύπτονται αδύνατα σημεία είτε γιατί προστίθενται επιπλέον απαιτήσεις. Έτσι μιλάμε για κυκλική ανάπτυξη (cyclical development) ή για κύκλο ζωής ανάπτυξης εφαρμογών (Application development life cycle). Συχνά μάλιστα, στο περιβάλλον του επαγγελματικού προγραμματισμού συμπεριλαμβάνονται στα στάδια ανάπτυξης, η διανομή του λογισμικού (deployment) και η παρακολούθηση του στον πελάτη (monitoring).

Στην συνέχεια, βασιζόμενοι σε ένα απλό παράδειγμα προσπαθούμε να δείξουμε πως μπορούμε να εφαρμόζουμε τα στάδια της ανάπτυξης στο επίπεδο των απλών εφαρμογών αυτού του βιβλίου ώστε από την αρχή να μάθουμε να δομούμε τις εφαρμογές όσο το δυνατόν αποτελεσματικότερα.

8.3 Συλλογή απαιτήσεων

Η συλλογή των απαιτήσεων (requirements gathering) είναι από μόνη της απαιτητική δουλειά. Σε γενικές γραμμές, οι απαιτήσεις χωρίζονται σε δύο κατηγορίες, στις απαιτήσεις του χρήστη (user requirements) και στις λειτουργικές απαιτήσεις (functional requirements).

Στις απαιτήσεις του χρήστη συμπεριλαμβάνονται απαιτήσεις τόσο των άμεσων χρηστών της εφαρμογής όσο και απαιτήσεις άλλων παραγόντων όπως η αγορά, η επιχείρηση για την οποία σχεδιάζεται το λογισμικό, κλπ. Για παράδειγμα, ο χρήστης θέλει να μπορεί να πλοηγείται σε λίστα προϊόντων και να βρίσκει εύκολα το προϊόν που επιθυμεί. Αυτό είναι μια απαίτηση του χρήστη. Όταν ο χρήστης πλοηγείται σε λίστα προϊόντων θα πρέπει τα προϊόντα στην λίστα να ταξινομούνται ανάλογα με το τι έχει ήδη αγοράσει ο συγκεκριμένος χρήστης.

Αυτό είναι μια λειτουργική απαίτηση. Με άλλα λόγια, οι απαιτήσεις του χρήστη περιγράφουν τι θέλει ο χρήστης από την εφαρμογή ενώ οι λειτουργικές απαιτήσεις περιγράφουν τι κάνει η εφαρμογή.

Δεν πρόκειται να επεκταθούμε άλλο στο μεγάλο και σημαντικό θέμα της συλλογής απαιτήσεων. Ο αναγνώστης που ενδιαφέρεται για περισσότερες λεπτομέρειες μπορεί να διαβάσει το [4]. Στην συνέχεια, θα υποθέσουμε πως το κείμενο που ακολουθεί περιγράφει τις απαιτήσεις για μια εφαρμογή. Πρόκειται στην ουσία για ένα παράδειγμα αναφορικά με το οποίο θα παρουσιαστούν και οι υπόλοιπες διεργασίες του κύκλου της ανάπτυξης.

Παράδειγμα απαιτήσεων Χρήστη

Τρεις φίλοι, ο John, ο Jim και ο Jack θα παίξουν ένα παιχνίδι ζαριών. Σύμφωνα με τους κανόνες του παιχνιδιού, κατά την διάρκεια μιας παρτίδας, οι παίκτες ρίχνουν εναλλάξ το ζάρι 10 φορές και μετράνε πόσες φορές έφερε ο καθένας τους άρτια ζαριά. Νικητής είναι εκείνος που έφερε τις περισσότερες άρτιες ζαριές. Αν όμως δύο ή περισσότεροι παίκτες έχουν ισοβαθμήσει, η παρτίδα επαναλαμβάνεται για όλους. Η επανάληψη συνεχίζεται μέχρις ότου να επικρατήσει ένας μοναδικός νικητής.
Οι συγκεκριμένοι παίκτες όμως έχουν τις ιδιομορφίες τους. Ο John είναι ένας τίμιος παίκτης με την έννοια πως κάθε φορά που πετάει το ζάρι έχει την ίδια πιθανότητα να φέρει οποιοδήποτε αποτέλεσμα από 1 έως 6. Από την άλλη μεριά, ο Jim είναι τυχερός καθώς έχει αποδειχτεί πως όταν πετάει το ζάρι έχει 15% πιθανότητα για κάθε ένα από τα αποτελέσματα 1, 3 και 5, 18% για κάθε ένα από τα αποτελέσματα 2 ή 4 και 19% για να φέρει 6. Επομένως, ο Jim φέρνει άρτια ζαριά με πιθανότητα 55% και περιττή ζαριά με πιθανότητα 45%.
Αντίθετα, ο Jack είναι άτυχος καθώς φέρνει 2, 4, 6 με πιθανότητα 15% για κάθε ένα από αυτά τα αποτελέσματα, ενώ 1 και 3 φέρνει με πιθανότητα 18% και 5 φέρνει με πιθανότητα 19%. Επομένως, ο Jack φέρνει άρτια ζαριά με πιθανότητα 45% και περιττή ζαριά με πιθανότητα 55%.
Να υλοποιηθεί εφαρμογή που προσομοιώνει μια παρτίδα ζαριών μεταξύ των τριών παικτών και αναφέρει τον νικητή.

8.4 Σχεδιασμός

Γενικά υπάρχουν δύο βασικές προσεγγίσεις στον σχεδιασμό ενός προγράμματος. Η από πάνω προς τα κάτω (top-down) και η από κάτω προς τα επάνω (bottom-up).

Σύμφωνα με την top-down θα πρέπει αρχικά να σχεδιάσουμε την εφαρμογή κατ' αρχάς στο υψηλότερο επίπεδο. Για παράδειγμα, μιλώντας με όρους Java, θα πρέπει πρώτα να σχεδιάσουμε την main. Σε αυτό το επίπεδο πρέπει να γνωρίζουμε τι θα κάνουν οι συναρτήσεις που καλούνται από την main χωρίς όμως να έχουμε καθορίσει τις λεπτομέρειες υλοποίησης των. Στην συνέχεια, με τον ίδιο τρόπο σχεδιάζουμε κάθε συνάρτηση που καλείται από την main, δηλ. καταρχάς χρησιμοποιούμε κλήσεις συναρτήσεων που δεν έχουμε ακόμη υλοποιήσει. Έτσι, σύμφωνα με την προσέγγιση top-down, σε κάθε βήμα αποσυνθέτουμε ένα πρόβλημα σε άλλα απλούστερα προβλήματα. Εξ' αυτού του λόγου, η μέθοδος αυτή συχνά ονομάζεται και σταδιακή υλοποίηση (stepwise refinement).

Αντίθετα, η προσέγγιση bottom-up ξεκινά τον σχεδιασμό στο χαμηλότερο επίπεδο και σε κάθε βήμα συνθέτει επι μέρους στοιχεία που οδηγούν σε υψηλότερο επίπεδο, δηλ. σε σύστημα σταδιακά πιο πολύπλοκο.

Κάθε μια από τις μεθοδολογίες αυτές έχει πλεονεκτήματα και μειονεκτήματα. Η top-down προσέγγιση αργεί σχετικά να παράγει ολοκληρωμένες συναρτήσεις με αποτέλεσμα να καθυστερεί η διαδικασία ελέγχου και άρα η ολοκλήρωση του έργου. Η bottom-up είναι ένα στοίχημα καθώς είναι δύσκολο να καθοριστούν με ακρίβεια οι λεπτομερείς ανάγκες πριν καθοριστεί το πλαίσιο στο οποίο αυτές θα αξιοποιηθούν. Συχνά, στην πράξη χρησιμοποιείται ένας συνδυασμός των δύο προσεγγίσεων. Για το παράδειγμα του παιχνιδιού ζαριών χρησιμοποιούμε την top-down προσέγγιση.

Αναλύοντας λοιπόν το κείμενο των απαιτήσεων του χρήστη, βλέπουμε πως εκείνο που ζητά είναι η προσομοίωση ενός παιχνιδιού ζαριών και η ανακήρυξη του νικητή. Ένα παιχνίδι όμως ζαριών συνίσταται στην επανάληψη μιας παρτίδας έως ότου ανακηρυχθεί μοναδικός νικητής.

Με βάση αυτήν την πληροφορία, εύκολα μπορούμε να αρχίσουμε τον σχεδιασμό χρησιμοποιώντας διαγράμματα ροής ή ψευδοκώδικα.

```
game:
     do {
          scores=oneLot;
```

```
} while withdraw(scores);
return winner(scores);
```

Κώδικας 8.1 *Ψευδοκώδικας για το παιχνίδι ζαριών*

Ο κώδικας 8.1 παρουσιάζει σε μορφή ψευδοκώδικα έναν σχεδιασμό για την υλοποίηση της κύριας απαίτησης του χρήστη. Πράγματι, εδώ ορίζεται μια ρουτίνα που ονομάζεται game και συνίσταται στην επανάληψη μιας παρτίδας ζαριών (oneLot) έως ότου προκύψει μοναδικός νικητής. Σε αυτό το στάδιο δεν μας ενδιαφέρει πως ακριβώς θα επιτυγχάνει τους στόχους της η oneLot αλλά μόνο πως αυτή είναι μια διαδικασία που προσομοιώνει μια παρτίδα και επιστρέφει τα σκορ των παικτών. Το ίδιο ισχύει και για την withdraw. Δεν μας αφορούν οι λεπτομέρειες υλοποίησής της αλλά μόνο ότι ανιχνεύει ενδεχόμενη ισοπαλία με βάση μια λίστα από σκορ. Τέλος, η winner λαμβάνει μια λίστα από σκορ τέτοια ώστε σε αυτήν υπάρχει ένα μόνο μέγιστο σκορ και επιστρέφει τον νικητή. Υποθέτουμε πως η winner λαμβάνει λίστα από σκορ στην οποία υπάρχει μία μόνο μέγιστη τιμή καθώς η λίστα έρχεται από το παιχνίδι το οποίο φτάνει σε αυτό το σημείο μόνο εφόσον δεν ανιχνευθούν ισοπαλίες.

Θα πρέπει τώρα να συνεχίσουμε την ανάλυση και τον σχεδιασμό στο αμέσως επόμενο επίπεδο, δηλ. να παράγουμε ψευδοκώδικα για τις oneLot, withdraw και winner.

Ας σχεδιάσουμε λοιπόν την παρτίδα, oneLot. Αυτή συνίσταται σε 10 ζαριές που ρίχνουν εναλλάξ, οι τρεις παίκτες. Όμως η ρίψη κάθε παίκτη έχει διαφορετικά χαρακτηριστικά. Επομένως, η ρίψη της ζαριάς θα πρέπει να παραμετροποιηθεί ως προς την κατανομή πιθανοτήτων στην βάση των οποίων υπολογίζεται η ρίψη κάθε παίκτη. Επιπλέον, θα ήταν καλό να παραμετροποιήσουμε την παρτίδα ως προς τον αριθμό ρίψεων. Με αυτόν τον τρόπο θα ανταποκριθούμε πιο εύκολα σε περίπτωση που θα γίνει κάποια σχετική αλλαγή στον αριθμό ρίψεων της παρτίδας. Στον κώδικα 8.2 χρησιμοποιούμε την noOfRolls αντί για την σταθερά 10.

```
oneLot:
     for (noOfRolls times) {

          rslt = roll(Fair);
          if (rslt is even)
               update score for JOHN;

          rslt = roll(lucky);
          if (rslt is even)
               update score for JIM;

          rslt = roll(unlucky);
          if (rslt is even)
               update score for JACK;

     }
     return scores;
```

Κώδικας 8.2 *Ψευδοκώδικας για μια παρτίδα ζαριών*

Ο ψευδοκώδικας στον κώδικα 8.2, δίνει επαναληπτικά την δυνατότητα σε κάθε παίκτη να πραγματοποιήσει μία ρίψη και αν το αποτέλεσμα είναι άρτιος ενημερώνει το αντίστοιχο σκορ. Σε αυτό το βήμα προκύπτει η ανάγκη να δούμε σε δεύτερο επίπεδο τον σχεδιασμό της roll και της ενημέρωσης του σκορ κάθε παίκτη.

Συνεχίζοντας όμως σε αυτό το επίπεδο μας μένει να σχεδιάσουμε τον έλεγχο της ισοπαλίας και τον εντοπισμό του νικητή. Πρόκειται για δύο πολύ απλές διαδικασίες. Ο εντοπισμός του νικητή πρέπει απλώς να ψάξει σε μια λίστα με σκορ και να επιστρέψει το μεγαλύτερο, οπότε δεν χρειάζεται να επεκταθούμε περισσότερο σε αυτό. Ο έλεγχος της ισοπαλίας πρέπει να ψάξει σε μια λίστα με σκορ να διαπιστώσει εάν υπάρχουν περισσότερες από μία μέγιστες τιμές. Επομένως

```
withdraw:
     return countOccurances(score, maxScore)>1
```

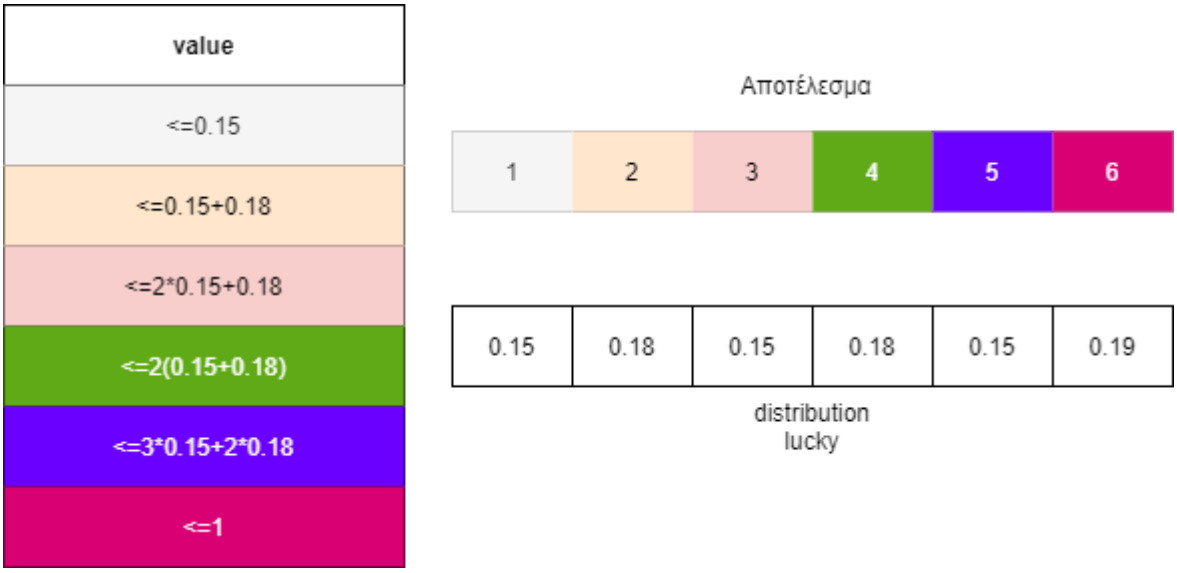
Κώδικας 8.3 *Ανίχνευση ισοπαλίας*

Στο επόμενο επίπεδο θα πρέπει να σχεδιάσουμε καταρχάς την διαδικασία ρίψης, roll. Η ενημέρωση του σκορ του χρήστη είναι προφανώς πολύ απλή διαδικασία και δεν χρειάζεται περαιτέρω ανάλυση σε αυτό το στάδιο. Καθώς έχουμε τρεις roll οι οποίες επιστρέφουν έναν αριθμό από το 1 έως το 6 με διαφορετική κατανομή πιθανοτήτων για κάθε δυνατό αποτέλεσμα, μπορούμε να σχεδιάσουμε μια διαδικασία παραμετροποιημένη ως προς την κατανομή πιθανοτήτων. Επομένως, θεωρούμε πως η roll λαμβάνει μια λίστα που περιέχει έξη πιθανότητες, μία για κάθε δυνατό αποτέλεσμα μιας ρίψης. Στον ψευδοκώδικα 8.4, η λίστα αυτή ονομάζεται distribution.

```
roll:
     value = Random value from 0 to 1;
     sum = 0;
     rslt=1;
     for (each percentage in distribution) {
          sum += percentage;
          if (value <= sum)
               return rslt;
          rslt++;
     }
}
```

Κώδικας 8.4 *Η διαδικασία προσομοίωσης της ζαριάς*

Το σκεπτικό του κώδικα 8.4 έχει ως εξής: Παράγουμε έναν τυχαίο αριθμό από 0 έως 1, την value. Αν η value είναι μικρότερη ή ίση από την πιθανότητα για αποτέλεσμα 1, επιστρέφουμε 1. Διαφορετικά, αν η value είναι μεγαλύτερη από την πιθανότητα για 1 αλλά μικρότερη από το άθροισμα πιθανοτήτων για 1 και 2, τότε επιστρέφουμε 2. Την ίδια λογική ακολουθούμε για όλα τα δυνατά αποτελέσματα. Με αυτόν τον τρόπο πετυχαίνουμε κάθε αποτέλεσμα με βάση την πιθανότητα που δίνεται στο distribution.



Σχήμα 8.1 *Υπολογισμός της ρίψης με βάση την κατανομή πιθανοτήτων για κάθε δυνατό αποτέλεσμα.*

Στο σχήμα 8.1, οι τιμές της value αντιστοιχίζονται χρωματικά με τα δυνατά αποτελέσματα για την τυχερή κατανομή. Αν η τιμή της value είναι μικρότερη ή ίση του 0.15, η roll επιστρέφει 1. Επομένως, το 1 επιστρέφεται με πιθανότητα 15%. Αν η τιμή της value είναι μικρότερη ή ίση του 0.15+0.18 αλλά δεν είναι μικρότερη ή ίση από 0.15, περίπτωση κατά την οποία η roll θα είχε ήδη επιστρέψει 1, η τιμή επιστροφής είναι 2. Επομένως, η πιθανότητα να επιστρέψει η roll 2 είναι ίση με την πιθανότητα η value να έχει τιμή μικρότερη ή ίση από 0.15+0.18 μείον την πιθανότητα να έχει η value τιμή 0.15. Άρα, η πιθανότητα να επιστρέψει 2 είναι 0.18, δηλ. σύμφωνη με την κατανομή. Με τον ίδιο τρόπο ερμηνεύεται και ο υπολογισμός των υπόλοιπων τιμών.

8.5 Υλοποίηση

Έχοντας ολοκληρώσει τον σχεδιασμό, μπορούμε να προχωρήσουμε στην υλοποίηση. Στο στάδιο αυτό πρέπει να φροντίσουμε να αξιοποιήσουμε όλες τις δυνατότητες που μας προσφέρει η γλώσσα υλοποίησης, δηλ. στην προκειμένη περίπτωση, η Java, ώστε ο κώδικάς μας να είναι ευανάγνωστος, απαλλαγμένος κατά το δυνατό από λάθη και εύκολα συντηρήσιμος, Στην συνέχεια παρουσιάζουμε πρώτα την υλοποίηση και μετά την σχολιάζουμε.

```
import java.util.Random;

public class ADiceGame {

    static final double EPSILON = 0.00001;
    private static final Random gen = new Random();
    static final double[] fair = {1/6d, 1/6d, 1/6d, 1/6d, 1/6d, 1/6d};
    static final double[] lucky = {15d/100, 18d/100, 15d/100, 18d/100, 15d/100,
19d/100};
    static final double[] unlucky = {18d/100, 15d/100, 18d/100, 15d/100, 19d/100,
15d/100};
    private static final int JOHN = 0;
    private static final int JIM = 1;
    private static final int JACK = 2;

    public static boolean approximateEquals(double d1, double d2) {
        return Math.abs(d1 - d2) < EPSILON;
    }

    public static double sum(double[] array) {
        double sum = 0;
        for (double d : array) {
            sum += d;
        }
        return sum;
    }

    public static int roll(double[] distribution) {
        if (distribution.length != 6 || !approximateEquals(sum(distribution),
1d)) {
            throw new RuntimeException();
        }
        double value = gen.nextDouble();
        double sum = 0;
        int rslt = 1;
        for (double percentage : distribution) {
            sum += percentage;
            if (value <= sum) {
                return rslt;
            }
            rslt++;
        }
        return 0;
    }

    public static int max(int[] array) {
        int idx = 0;
        int max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
                idx = i;
```

```
            }
        }
        return idx;
    }

    public static int cntOccurances(int[] array, int value) {
        int cnt = 0;
        for (int i = 0; i < array.length; i++) {
            if (value == array[i]) {
                cnt++;
            }
        }
        return cnt;
    }

    public static boolean withdraw(int[] score) {
        return cntOccurances(score, score[max(score)]) > 1;
    }

    private static int[] aLot(int noOfRolls) {
        int[] score = new int[3];
        for (int i = 0; i < noOfRolls; i++) {
            int rslt = roll(fair);
            if (rslt % 2 == 0) {
                score[JOHN]++;
            }

            rslt = roll(lucky);
            if (rslt % 2 == 0) {
                score[JIM]++;
            }

            rslt = roll(unlucky);
            if (rslt % 2 == 0) {
                score[JACK]++;
            }
        }
        return score;
    }

    public static int game() {
        final int NOOFROLLS = 10;
        int[] score;

        do {
            score = aLot(NOOFROLLS);
        } while (withdraw(score));
        return max(score);
    }

    private static String winnerName(int winnerId) {

        switch (winnerId) {
            default:
                throw new RuntimeException();
            case JOHN:
                return "John";
            case JIM:
                return "Jim";
            case JACK:
                return "Jack";
```

```
        }
    }

    public static void main(String[] args) {
        int winnerId = game();
        System.out.println("winner is " + winnerName(winnerId));
    }
}
```

Κώδικας 8.5 *Εφαρμογή – Ένα παιχνίδι ζαριών*

Ο κώδικας 8.5 παρουσιάζει την υλοποίηση του παιχνιδιού ζαριών. Στην main καλείται η game που επιστρέφει τον νικητή. Στην συνέχεια, τυπώνεται το όνομά του με την βοήθεια της winnerName.
Η game υλοποιεί ένα παιχνίδι με συγκεκριμένα χαρακτηριστικά παρτίδας. Έτσι δηλώνει ως τοπική σταθερά τον αριθμό ρίψεων κάθε παρτίδας. Στην συνέχεια, σύμφωνα με τον σχεδιασμό καλεί την aLot μέσα σε επαναληπτική διαδικασία που ελέγχεται από την withdraw. Σε κάθε κλήση, η aLot επιστρέφει έναν πίνακα με τα σκορ των παικτών. Τα σκορ ελέγχει η withdraw. Αν η withdraw δεν ανιχνεύσει ισοπαλία τερματίζεται ο βρόχος do και επιστρέφεται η θέση του πίνακα των σκορ με την μεγαλύτερη τιμή. Εκεί βρίσκεται ο νικητής.

Η aLot λαμβάνει ως παράμετρο τον αριθμό των ρίψεων ανά παρτίδα και παίκτη. Δηλώνει τον πίνακα score όπου αποθηκεύονται οι ζαριές με άρτιο αποτέλεσμα για κάθε παίκτη κατά την διάρκεια της παρτίδας. Προκειμένου να αντιστοιχίζεται κάθε παίκτης ευκολότερα, με περισσότερη ασφάλεια και να είναι το πρόγραμμα ευανάγνωστο, έχουν δηλωθεί οι σταθερές JOHN, JIM και JACK ως 0, 1 και 2. Έτσι γίνεται μια αντιστοιχία μεταξύ των θέσεων στον πίνακα των σκορ με τα ονόματα των παικτών. Μέσα στον βρόγχο for της aLot κάθε παίκτης ρίχνει την ζαριά του με την βοήθεια της roll και αν το αποτέλεσμα είναι άρτιο ενημερώνεται το σκορ του.

Η withdraw, σύμφωνα με τον σχεδιασμό μετράει πόσες φορές στον πίνακα των σκορ υπάρχει το μέγιστο σκορ. Επιστρέφει true μόνο αν βρει περισσότερες από μια μέγιστες τιμές. Την απαιτούμενη μέτρηση υλοποιεί η cntOccurances με την βοήθεια της max που εντοπίζει την μέγιστη τιμή.

Η roll λαμβάνει ως παράμετρο έναν πίνακα από πραγματικούς που αναπαριστά την κατανομή των πιθανοτήτων για κάθε δυνατό αποτέλεσμα. Επειδή στο συγκεκριμένο παιχνίδι έχουμε τρεις πιθανές κατανομές, έχουν ορισθεί τρεις πίνακες, ο fair, ο lucky και ο unlucky που αναπαριστούν τις τρεις κατανομές. Κάθε πίνακας έχει 6 στοιχεία καθώς δίνει μια πιθανότητα για κάθε ένα από τα έξη πιθανά αποτελέσματα. Επειδή ρίχνοντας το ζάρι η πιθανότητα να έρθει ένα από τα έξη πιθανά αποτελέσματα είναι 100%, το άθροισμα των στοιχείων κάθε ενός πίνακα πρέπει να είναι 1. Τους ελέγχους αυτούς κάνει η roll. Αν δεν επαληθευτούν παράγει εξαίρεση. Στην συνέχεια, εφαρμόζει τον αλγόριθμο σύμφωνα με τον σχεδιασμό και επιστρέφει το αποτέλεσμα.

Η sum και η approximateEquals είναι προφανείς και έχουν ήδη συζητηθεί.

Προσέξτε πως κάποια μέλη της ADiceGame είναι δημόσια, άλλα ιδιωτικά και άλλα έχουν πρόσβαση πακέτου. Πιο συγκεκριμένα, η EPSILON και οι κατανομές έχουν πρόσβαση πακέτου. Όπως θα δούμε, οι μεταβλητές αυτές θα μας είναι χρήσιμες κατά την ανάπτυξη του ελέγχου του κώδικα και για αυτόν τον λόγο τους δώσαμε πρόσβαση πακέτου διαφορετικά θα μπορούσαμε να σκεφτούμε να τις ορίσουμε ως ιδιωτικές καθώς σχετίζονται πολύ στενά με την συγκεκριμένη εφαρμογή. Η γεννήτρια τυχαίων αριθμών gen είναι ένα αντικείμενο που χρησιμοποιείται στην συγκεκριμένη κλάση για την παραγωγή τυχαίων αριθμών και δεν υπάρχει λόγος να δώσουμε πρόσβαση σε αυτήν έξω από την κλάση. Σε περίπτωση που άλλες κλάσεις επιθυμούν να παράγουν τυχαίους αριθμούς μπορούν να δημιουργήσουν δικές τους γεννήτριες. Τα ονόματα των παικτών αφορούν αποκλειστικά το συγκεκριμένο παιχνίδι και δεν χρειάζεται να έχει πρόσβαση σε αυτά κανείς άλλος.

Επίσης ως ιδιωτικές έχουν δηλωθεί οι aLOt, game και winnerName. Αυτή η επιλογή οφείλεται στο γεγονός ότι οι τρεις αυτές συναρτήσεις είναι πολύ εξειδικευμένες και αφορούν αποκλειστικά την συγκεκριμένη εφαρμογή.

Τα υπόλοιπα μέλη της κλάσης έχουν ορισθεί ως δημόσια κυρίως με την έννοια πως παρουσιάζουν τέτοια χρησιμότητα ώστε είναι πιθανό να αξιοποιηθούν και από τρίτους.

8.6 Τεκμηρίωση

Η τεκμηρίωση των προγραμμάτων της Java γίνεται με το ειδικό εργαλείο παραγωγής τεκμηρίωσης JavaDoc [3, 4]. Όλες οι προκαθορισμένες κλάσεις της Java είναι τεκμηριωμένες με το JavaDoc. Αν πάτε σε ένα κώδικα στο NetBeans σε ένα σημείο που αναφέρεται το όνομα μιας προκαθορισμένης κλάσης, π.χ. String και πατήσετε

Alt+F1 θα διαπιστώσετε πως θα ανοίξει η τεκμηρίωση της κλάσης String[7]. Εκεί μπορείτε να βρείτε τεκμηρίωση για κάθε μέλος της κλάσης.

Η συγγραφή της τεκμηρίωσης κάθε κλάσης βασίζεται σε ειδικά σχόλια, τα επονομαζόμενα σχόλια τεκμηρίωσης (doc comments). Σχόλιο τεκμηρίωσης θεωρείται οποιοδήποτε κείμενο βρίσκεται μεταξύ των χαρακτήρων /** που σηματοδοτούν την αρχή του σχολίου και τους χαρακτήρες */ που σηματοδοτούν το τέλος του. Επομένως, υπάρχει διαφορά από τα απλά σχόλια που περιλαμβάνονται μεταξύ των χαρακτήρων /* και */. Τα μεν σχόλια τεκμηρίωσης απαιτούν δύο αστερίσκους μετά την πλάγια κάθετο ενώ τα απλά ένα. Τα απλά σχόλια δεν λαμβάνονται υπόψη από το JavaDoc.

Τα σχόλια τεκμηρίωσης τοποθετούνται πριν την δήλωση οποιασδήποτε κλάσης ή οποιουδήποτε μέλους κλάσης. Ωστόσο, υπάρχει η δυνατότητα να παράγουμε τεκμηρίωση που αφορά ένα πακέτο, δηλ. ένα σύνολο κλάσεων ή και για ένα σύνολο πακέτων [5].

Η τεκμηρίωση JavaDoc αφορά κυρίως τα δημόσια μέλη των κλάσεων. Παρότι το εργαλείο προσφέρει την δυνατότητα τεκμηρίωσης και των μη δημόσιων μελών, η χρησιμότητά της αμφισβητείται από την μεγάλη πλειοψηφία των επαγγελματιών του χώρου. Αντί της τεκμηρίωσης JavaDoc για τα μη δημόσια μέλη είναι προτιμητέα τα απλά σχόλια με βάση το σκεπτικό πως τα δημόσια μέλη θα χρησιμοποιηθούν από τρίτους επομένως μόνο η τεκμηρίωση των δημόσιων μελών έχει ενδιαφέρον προς τρίτους.

Κατά την συγγραφή τεκμηρίωσης μπορούμε να χρησιμοποιήσουμε μια σειρά από tags για να προσδιορίσουμε τον τύπο της πληροφορίας που περιγράφουμε. Πριν την χρήση των tags, μπορεί να προστεθεί μια περιγραφή του υπό τεκμηρίωση αντικειμένου. Παρουσιάζουμε εδώ τα πολύ βασικά tags. Πλήρης λίστα των διαθέσιμων tags μπορεί να βρεθεί στην [6].

Βασικά tags
author
see
param
return
throws

Πίνακας 8.1 *Βασικά tags JavaDoc*

Στο tag author αναφέρεται προφανώς το όνομα του συγγραφέα ή των συγγραφέων του κώδικα. Το tag εισάγεται αυτόματα από τα περισσότερα IDE και από το NetBeans. Ωστόσο, δεν συμπεριλαμβάνεται στο παραγόμενο JavaDoc.

Το tag see χρησιμοποιείται όταν η τεκμηρίωση αναφέρεται σε κάποια άλλη κλάση ή μέλος κλάσης από αυτήν στην οποία ανήκει το tag. Το tag δημιουργεί ένα σύνδεσμο στο αναφερόμενο αντικείμενο.

Το tag param χρησιμοποιείται για την περιγραφή παραμέτρου συνάρτησης.

Το tag return για την περιγραφή της επιστρεφόμενης τιμής από συνάρτηση.

Τέλος, το tag throws για την περιγραφή τυχόν εξαιρέσεων που μπορεί να παράγονται από συνάρτηση.

Στην συνέχεια στον κώδικα 8.6 παρουσιάζουμε τον κώδικα του παιχνιδιού ζαριών με απλά σχόλια και με σχόλια τεκμηρίωσης ανάλογα με τον χαρακτηρισμό προσπέλασης συναρτήσεων και μεταβλητών.

```
import java.util.Random;

  /**
   *Η κλάση προσομοιώνει το ακόλουθο παιχνίδι ζαριών. Τρεις
   *φίλοι, ο John, ο Jim και ο Jack παίζουν ένα παιχνίδι
   *ζαριών. Σύμφωνα με τους κανόνες του παιχνιδιού, κατά την
   *διάρκεια μιας παρτίδας, οι παίκτες ρίχνουν εναλλάξ το ζάρι
   *10 φορές και μετράνε πόσες φορές έφερε ο καθένας τους άρτια
```

[7] Αν με Alt+F1 δεν ανοίξει αυτόματα η τεκμηρίωση της κλάσης String, μεταβείτε στο NetBeans στο μενού Tools/Java Platforms. Στο παράθυρο που θα ανοίξει, επιλέξτε το (Default) JDK. Σύμφωνα με τις οδηγίες αυτού του βιβλίου πρέπει να είναι το JDK 17. Από τις καρτέλες δεξιά, επιλέξτε την καρτέλα JavaDoc και μεταβάλλετε το URL https://docs.oracle.com/en/Java/Javase/16/docs/api σε https://docs.oracle.com/en/Java/Javase/16/docs/api/Java.base.

```
    *ζαριά.
    * Νικητής είναι εκείνος που έφερε τις περισσότερες άρτιες
    *ζαριές. Αν όμως 2 ή περισσότεροι παίκτες έχουν
    *ισοβαθμήσει, η παρτίδα επαναλαμβάνεται για όλους.
    * Η επανάληψη συνεχίζεται μέχρις ότου να επικρατήσει ένας
    *μοναδικός νικητής. Οι συγκεκριμένοι παίκτες όμως έχουν τις
    *ιδιομορφίες τους. Ο John είναι ένας τίμιος παίκτης με την
    *έννοια πως κάθε φορά που πετάει το ζάρι έχει την ίδια
    *πιθανότητα να φέρει οποιοδήποτε αποτέλεσμα από 1 έως 6. Από
    *την άλλη μεριά, ο Jim είναι τυχερός καθώς έχει αποδειχτεί
    *πως όταν πετάει το ζάρι έχει 15% πιθανότητα για κάθε ένα
    *από τα αποτελέσματα 1, 3 και 5, 18% για κάθε ένα από τα
    *αποτελέσματα 2 ή 4 και 19% για να φέρει 6. Επομένως, ο Jim
    *φέρνει άρτια ζαριά με πιθανότητα 55% και περιττή ζαριά με
    *πιθανότητα 45%. Αντίθετα, ο Jack είναι άτυχος καθώς φέρνει
    *2, 4, 6 με πιθανότητα 15% για κάθε ένα από αυτά τα
    *αποτελέσματα, ενώ 1 και 3 φέρνει με πιθανότητα 18% και 5
    *φέρνει με πιθανότητα 19%. Επομένως, ο Jack φέρνει άρτια
    *ζαριά με πιθανότητα 45% και περιττή ζαριά * με πιθανότητα
    *55%.
    *
    * @author Lefteris Moussiades
    */
public class ADiceGame {

    /**
     * Χρησιμοποιείται απο τον έλεγχο ισότητας πραγματικών
     *
     * @see approximateEquals
     */
    static final double EPSILON = 0.00001;
    private static final Random gen = new Random();
    static final double[] fair = {1/6d, 1/6d, 1/6d, 1/6d, 1/6d, 1/6d};
    static final double[] lucky = {15d/100, 18d/100, 15d/100, 18d/100, 15d/100,
19d/100};
    static final double[] unlucky = {18d/100, 15d/100, 18d/100, 15d/100, 19d/100,
15d/100};
    private static final int JOHN = 0;
    private static final int JIM = 1;
    private static final int JACK = 2;

    /**
     * Συγκκρίνει προσεγγιστικά δύο πραγματικούς τύπου double για έλεγχο
     * ισότητας
     *
     * @param d1
     * @param d2
     * @return Επιστρέφει true αν η απόλυτη τιμή της διαφορας d1-d2 είναι
     * μικρότερη απο την τιμή της EPSILON
     * @see EPSILON
     */
    public static boolean approximateEquals(double d1, double d2) {
        return Math.abs(d1 - d2) < EPSILON;
    }

    /**
     * Υπολογίζει το άθροισμα των στοιχείων της παραμέτρου array
     *
     * @param array
     * @return Το άθροισμα των στοιχείων της array
     */
```

```
    public static double sum(double[] array) {
        double sum = 0;
        for (double d : array) {
            sum += d;
        }
        return sum;
    }

    /**
     * Προσομοιώνει την ρίψη ζαριού
     *
     * @param distribution Πίνακας πραγματικών. Κατά την κλήση της μεθόδου, η
     * distribution πρέπει να περιλαμβάνει 6 στοιχεία με άθροισμα ίσο με 1.
     * @return Έναν αριθμό απο 1 έως 6 ανάλογα με την πιθανότητα που δίνεται
     * για κάθε τιμή μέσω της παραμέτρου distribution
     * @throws RuntimeException
     */
    public static int roll(double[] distribution) {
        if (distribution.length != 6 || !approximateEquals(sum(distribution),
1d)) {
            throw new RuntimeException();
        }
        double value = gen.nextDouble();
        double sum = 0;
        int rslt = 1;
        for (double percentage : distribution) {
            sum += percentage;
            if (value <= sum) {
                return rslt;
            }
            rslt++;
        }
        return 0;
    }

    /**
     * Εντοπίζει την μέγιστη τιμή του πίνακα array
     *
     * @param array Ο πίνακας του οποίου το μέγιστο στοιχείο θέλουμε να
     * βρούμε
     * @return Επιστρέφει την θέση της πρώτης απο τις μέγιστες τιμές του
     * πίνακα
     * array
     */
    public static int max(int[] array) {
        int idx = 0;
        int max = array[0];
        for (int i = 1; i < array.length; i++) {
            if (array[i] > max) {
                max = array[i];
                idx = i;
            }
        }
        return idx;
    }

    /**
     * Υπολογίζει πόσες φορές υπάρχει μια τιμή σε ένα πίνακα
     *
     * @param array Ο πίνακας στον οποίο θα μετρήσουμε πόσες φορές υπάρχει η
     * value
```

```
     * @param value Η τιμή που θα μετρήσουμε πόσες φορές υπάρχει στον array
     * @return Επιστρέφει το πλήθος των στοιχείων με τιμή ίση με value στον
     * πίνακα array
     */
    public static int cntOccurances(int[] array, int value) {
        int cnt = 0;
        for (int i = 0; i < array.length; i++) {
            if (value == array[i]) {
                cnt++;
            }
        }
        return cnt;
    }

    private static boolean withdraw(int[] score) {
        return cntOccurances(score, score[max(score)]) > 1;
    }

    /*
    Η aLot  λαμβάνει ως παράμετρο τον αριθμό των ρίψεων ανα παρτίδα και
    παίκτη.
    Δηλώνει τον πίνακα score όπου αποθηκεύονται οι ζαριές με άρτιο αποτέλεσμα
    για κάθε παίκτη κατά την διάρκεια της παρτίδας.
    Προκειμένου να αντιστοιχίζεται κάθε παίκτης ευκολότερα, με περισσότερη
    ασφάλεια και να είναι το πρόγραμμα ευανάγνωστο,
    έχουν δηλωθεί οι σταθερές JOHN, JIM και JACK ως 0, 1 και 2. Έτσι γίνεται
    μια αντιστοιχία μεταξύ των θέσεων στον πίνακα των σκορ
    με τα ονόματα των παικτών.
    Μέσα στον βρόγχο for της aLot κάθε παίκτης ρίχνει την ζαριά του με την
    βοήθεια της roll και αν το αποτέλεσμα είναι άρτιο ενημερώνεται
    το σκορ του.
    */
    private static int[] aLot(int noOfRolls) {
        int[] score = new int[3];
        for (int i = 0; i < noOfRolls; i++) {
            int rslt = roll(fair);
            if (rslt % 2 == 0) {
                score[JOHN]++;
            }

            rslt = roll(lucky);
            if (rslt % 2 == 0) {
                score[JIM]++;
            }

            rslt = roll(unlucky);
            if (rslt % 2 == 0) {
                score[JACK]++;
            }
        }
        return score;
    }

    /*
    Η game υλοποιεί ένα παιχνίδι με συγκεκριμένα χαρακτηριστικά παρτίδας.
    Έτσι δηλώνει ως τοπική σταθερά τον αριθμό ρίψεων κάθε παρτίδας.
    Στην συνέχεια, σύμφωνα με τον σχεδιασμό καλεί την aLot μέσα σε
    επαναληπτική διαδικασία που ελέγχεται από την withdraw.
    Σε κάθε κλήση, η aLot επιστρέφει έναν πίνακα με τα σκορ των παικτών.
    Τα σκορ ελέγχει η withdraw. Αν η withdraw δεν ανιχνεύσει ισοπαλία
    τερματίζεται ο βρόχος do και επιστρέφεται η θέση του πίνακα
```

```
    των σκορ με την μεγαλύτερη τιμή. Εκεί βρίσκεται ο νικητής.
     */
    private static int game() {
        final int NOOFROLLS = 10;
        int[] score;

        do {
            score = aLot(NOOFROLLS);
        } while (withdraw(score));
        return max(score);
    }

    private static String winnerName(int winnerId) {

        switch (winnerId) {
            default:
                throw new RuntimeException();
            case JOHN:
                return "John";
            case JIM:
                return "Jim";
            case JACK:
                return "Jack";
        }
    }

    public static void main(String[] args) {
        int winnerId = game();
        System.out.println("winner is " + winnerName(winnerId));

    }
}
```

Κώδικας 8.6 *Ο κώδικας του παιχνιδιού ζαριών σχολιασμένος*

Μετά την ολοκλήρωση σχολιασμού του κώδικα, μεταβείτε στην περιοχή Projects, κάντε δεξί κλικ στο όνομα του project στο οποίο έχετε σχολιάσει τις κλάσεις και επιλέξτε Generate Javadoc. Η τεκμηρίωση του κώδικα σε μορφή HTML θα ανοίξει στον φυλλομετρητή του συστήματος. Επιπλέον, με δεξί κλικ επάνω σε κάθε κλάση και επιλογή Tools/Analyze Javadoc μπορείτε να λάβετε μια αναφορά για τυχόν παραλείψεις στην τεκμηρίωση της κλάσης.

8.7 Απολαθοποίηση

Η απολαθοποίηση (debugging) είναι μια σημαντική λειτουργία χρήσιμη τόσο κατά το στάδιο της υλοποίησης όσο και κατά το στάδιο του ελέγχου όπου ενδέχεται να προκύψουν προβλήματα και να κληθούν οι προγραμματιστές να εντοπίσουν την εστία τους μέσα στον κώδικα και να τα θεραπεύσουν. Η απολαθοποίηση παρέχει την δυνατότητα να εκτελέσουμε τον κώδικα ελεγχόμενα, βήμα προς βήμα και σε κάθε βήμα να ελέγχουμε τις τιμές των μεταβλητών και των εκφράσεων.
Θα δούμε την λειτουργία του απολαθοποιητή στο NetBeans με ένα παράδειγμα.

```
import gr.ihu.cs.lmous.OOJ.k7.v1.StaticFunctions;
import java.util.Arrays;

public class Debugging {

    static int[] resize(int[] in, int newSize) {
        int[] rVal=new int[newSize];
        for (int i=0; i<in.length && i<rVal.length; i++)
            rVal[i]=in[i];
        return rVal;
```

```
    }

    static int[] createSet(int... elements) {
        int[] set=new int[elements.length];
        int setIdx=0;
        for (int element:elements) {
            int idx=StaticFunctions.sequentialSearch(set, element);
            if (idx<0)
                set[setIdx++]=element;
        }
        resize(set, setIdx);
        return set;
    }

    static int[] union(int[] set1, int[] set2) {
        int[] union=new int[set1.length+set2.length];
        System.arraycopy(set1, 0, union, 0, set1.length);
        System.arraycopy(set2, 0, union, set1.length, set2.length);
        return createSet(union);
    }

    public static void main(String[] args) {
        int[] set1=createSet(2,4,6,3), set2=createSet(3,5,7,2);
        int[] union=union(set1, set2);
        System.out.println(Arrays.toString(union));
    }
}
```

Κώδικας 8.7 *Κώδικας προς απολαθοποίηση*

Ο κώδικας 8.7 είναι μια εφαρμογή διαχείρισης συνόλων ακεραίων. Κάθε στοιχείο ενός συνόλου είναι μοναδικό, δηλ. σε ένα σύνολο δεν μπορεί να υπάρχει δύο η περισσότερες φορές το ίδιο στοιχείο. Η συνάρτηση createSet λαμβάνει ως παράμετρο μια σειρά ακεραίων και επιστρέφει ένα σύνολο ακεραίων με την μορφή πίνακα. Για να το πετύχει αυτό, εισάγει στον πίνακα που επιστρέφει μια τιμή από τις πραγματικές παραμέτρους μόνο εφόσον αυτή δεν υπάρχει ήδη. Μάλιστα πριν επιστρέψει καλεί την resize ώστε να ρυθμίσει το μέγεθος του πίνακα που επιστρέφει ανάλογα με το πλήθος των στοιχείων του συνόλου. Η συνάρτηση union λαμβάνει ως παραμέτρους δύο πίνακες που αντιπροσωπεύουν σύνολα και επιστρέφει την ένωσή τους. Στην main δημιουργούμε δύο σύνολα, υπολογίσουμε την ένωσή τους και την τυπώνουμε.
Αν τρέξετε τον κώδικα θα δείτε πως η έξοδός του είναι

```
[2, 4, 6, 3, 5, 7, 0, 0]
```

Το σύνολο-ένωση όμως που αναμένουμε είναι το [2, 4, 6, 3, 5, 7]. Επομένως κάποιο λάθος υπάρχει στον κώδικα. Μπορείτε να το βρείτε; Αξίζει να προσπαθήσετε καταρχάς απλώς διαβάζοντας τον κώδικα. Αν ωστόσο δεν τα καταφέρετε, ο απολαθοποιητής είναι εδώ για να σας βοηθήσει.
Μεταβείτε στο NetBeans και δημιουργήστε την κλάση Debugging. Στην πρώτη γραμμή της main, κάντε κλικ στην γκρίζα κάθετη αριθμημένη λωρίδα στα αριστερά του κώδικα. Η γραμμή θα αποκτήσει ένα ροζ φόντο όπως φαίνεται στην εικόνα 8.1.

```
39      public static void main(String[] args) {
            int[] set1=createSet(2,4,6,3), set2=createSet(3,5,7,2);
41          int[] union=union(set1, set2);
42          System.out.println(Arrays.toString(union));
43      }
```

Εικόνα 8.1 *Breakpoint στην πρώτη γραμμή της main*

Έχετε τοποθετήσει ένα Breakpoint (σημείο διακοπής). Μπορείτε τώρα να εκτελέσετε τον κώδικα κατά τρόπο ώστε να διακοπεί η ροή του όταν συναντήσει το Breakpoint. Ανάλογα με τις ανάγκες σας μπορείτε να τοποθετήσετε και άλλα Breakpoint.
Μεταβείτε στην κλάση στο πλαίσιο Projects. Με δεξί κλικ πάνω στην κλάση ανοίγει ένα μενού, επιλέξτε Debug File. Θα μεταβείτε στην κατάσταση που δείχνει η εικόνα 8.2.

```
39      public static void main(String[] args) {
            int[] set1=createSet(2,4,6,3), set2=createSet(3,5,7,2);
41          int[] union=union(set1, set2);
42          System.out.println(Arrays.toString(union));
43      }
```

Εικόνα 8.2 *Διακοπή της ροής στο Breakpoint*

Η εφαρμογή εκτελείται αλλά έχει σταματήσει στην πρώτη γραμμή της main, εκεί δηλ. που συνάντησε το πρώτο Breakpoint. Στην συνέχεια έχουμε έλεγχο στην ροή του προγράμματος με την βοήθεια των εντολών προς τον απολαθοποιητή. Στην γραμμή εργαλείων, κάτω από το κεντρικό μενού βλέπετε την εργαλειοθήκη με τις διαθέσιμες εντολές προς τον απολαθοποιητή.

Εικόνα 8.3 *Η εργαλειοθήκη του απολαθοποιητή.*

Με αναφορά στην αρίθμηση της εικόνας 8.3, οι διαθέσιμες εντολές προς τον απολαθοποιητή έχουν ως εξής:

Finish Debugger Session: Τερματίζεται η λειτουργία του απολαθοποιητή.

Pause: Προσωρινή παύση της εκτέλεσης του υπό απολαθοποίηση προγράμματος.

Continue: Συνέχιση της εκτέλεσης του υπό απολαθοποίηση προγράμματος.

Step Over: Εκτέλεση της τρέχουσας συνάρτησης.

Step Over Expression: Εκτέλεση της τρέχουσας έκφρασης.

Step Into: Είσοδος στην τρέχουσα συνάρτηση.

Step Out: Έξοδος από την τρέχουσα συνάρτηση.

Run to cursor: Εκτέλεση του προγράμματος μέχρι του σημείου που έχουμε τοποθετήσει τον cursor στον κώδικα.

Καθώς το πρόγραμμα έχει μπει σε κατάσταση απολαθοποίησης, στο πλαίσιο κάτω από τον κώδικα, εκεί όπου βρίσκεται το παράθυρο Output, έχουν ανοίξει δύο ακόμη παράθυρα, το Variables και το Breakpoints. Στο παράθυρο Breakpoints μπορούμε να βλέπουμε και να διαχειριζόμαστε τα Breakpoints που τοποθετούμε στον κώδικα και στο παράθυρο Variables βλέπουμε τις μεταβλητές που είναι εντός εμβέλειας ανάλογα με το σημείο του προγράμματος στο οποίο βρισκόμαστε. Για παράδειγμα, στην τρέχουσα κατάσταση, μπορούμε να δούμε πως η παράμετρος args έχει μήκος 0, κλήθηκε επομένως η εφαρμογή χωρίς παραμέτρους της main.

Πατήστε το εικονίδιο Step Over. Η πρώτη γραμμή έχει τρέξει. Ο απολαθοποιητής περιμένει στην επόμενη γραμμή ενώ παράλληλα στο παράθυρο Variables έχουν εμφανιστεί οι μεταβλητές set1 και set2.

Μπορούμε να ανοίξουμε τις μεταβλητές για να δούμε τα περιεχόμενά τους. Είναι αυτά που αναμένουμε; Πράγματι, μια πρώτη επιθεώρηση μας δίνει τις αναμενόμενες τιμές τόσο για την set1 όσο και για την set2. Δεδομένου ότι έχουμε ένα πολύ μικρό πρόγραμμα και είδαμε πως η πρώτη γραμμή εκτελείται με το σωστό αποτέλεσμα, λογικό είναι να υποθέσουμε πως κάτι δεν πάει καλά στην δεύτερη γραμμή όπου εκτελείται η union. Για αυτόν τον λόγο θα επιθεωρήσουμε πιο στενά την union.

Πατήστε Step Into, η ροή του κώδικα θα σας μεταφέρει στην πρώτη γραμμή της union. Αν προχωρήσουμε βήμα προς βήμα με Step Over μέχρι την τελευταία γραμμή της union δεν θα συναντήσουμε κάποιο πρόβλημα. Επομένως θα υποπτευθούμε πως κάτι μπορεί να συμβαίνει στην τελευταία γραμμή της union

όπου καλείται η createSet. Μα την createSet την έχουμε ήδη δοκιμάσει και φαίνεται να πήγε καλά. Ας μπούμε καλύτερα μέσα με Step Into να δούμε με μεγαλύτερη λεπτομέρεια.

Με Step Over στις γραμμές της create και έλεγχο των διαθέσιμων μεταβλητών διαπιστώνουμε πως το πρόβλημα είναι πως το μέγεθος του πίνακα set δεν μεταβάλλεται μετά την εκτέλεση της resize. Ενώ ο πίνακας set δημιουργήθηκε αρχικά με μέγεθος το άθροισμα των μεγεθών των δύο συνόλων που ενώνονται, μετά την resize το μέγεθός του αναμένεται μειωμένο κατά δύο καθώς τα στοιχεία 2 και 3 υπάρχουν και στα δύο σύνολα.

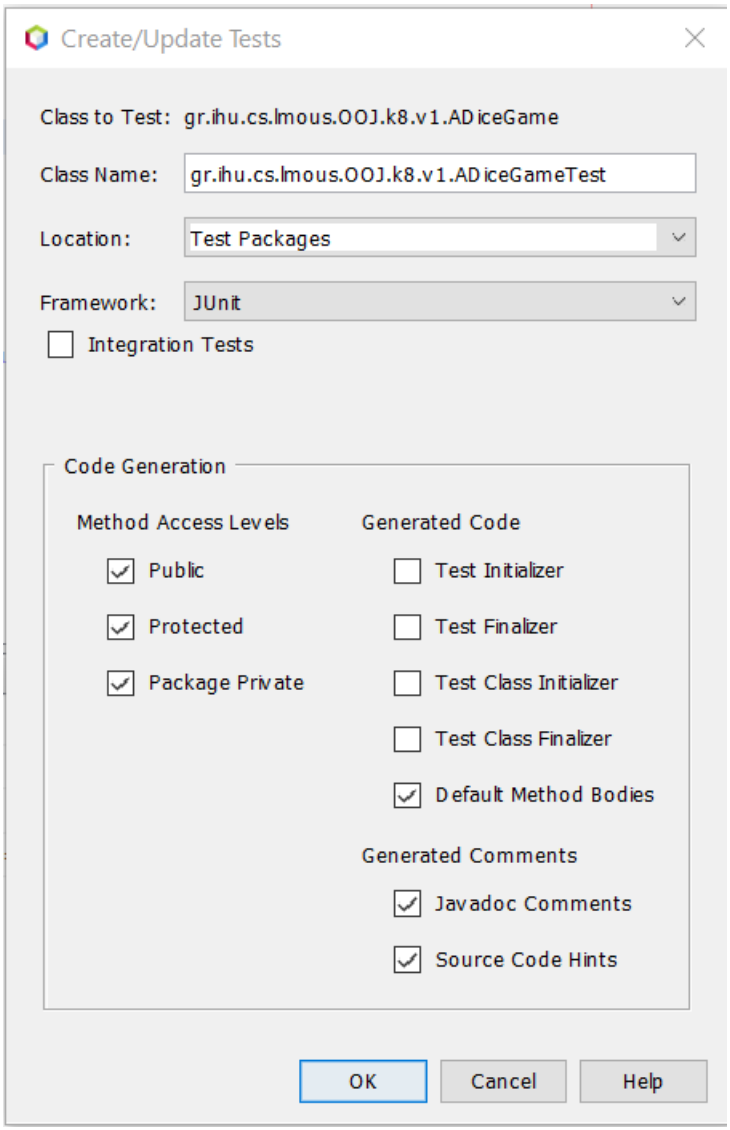
Μετά την διαπίστωση αυτή, παρατηρήστε πως η resize δεν μεταβάλλει το μήκος του πίνακα που περνάει ως παράμετρος σε αυτήν. Αντίθετα, δημιουργεί και επιστρέφει ένα αντίγραφο της παραμέτρου με το σωστό μήκος. Εφόσον επιθυμούμε να αλλάξουμε το μέγεθος του πίνακα set, θα πρέπει να εκχωρήσουμε την τιμή που επιστρέφει η resize στην μεταβλητή set. Επομένως αλλάξτε την γραμμή resize(set, setIdx) σε set=resize(set, setIdx), τερματίστε τον απολαθοποιητή και ξανατρέξτε την κλάση. Θα διαπιστώσετε πως η ένωση των συνόλων εμφανίζεται σωστά αυτήν την φορά.

8.8 Έλεγχος

Ο έλεγχος (test) αποτελεί ένα απαραίτητο στοιχείο της ανάπτυξης. Είναι σημαντικό να κατανοήσουμε πως ο έλεγχος δεν γίνεται αποκλειστικά μετά την ολοκλήρωση της υλοποίησης. Αντίθετα, ξεκινά παράλληλα με την υλοποίηση και διεξάγεται καθόλη την διάρκειά της.

Ο έλεγχος επιτυγχάνεται με την συγγραφή και εκτέλεση κατάλληλων συναρτήσεων. Θα πρέπει δηλ. για κάθε μια μη ιδιωτική συνάρτηση της εφαρμογής να αναπτύξουμε μια αντίστοιχη συνάρτηση ελέγχου. Θα δούμε περισσότερες λεπτομέρειες στην πράξη αναπτύσσοντας κώδικα ελέγχου για την κλάση ADiceGame.

Στο παράθυρο Projects, μεταβείτε στην 1η έκδοση της κλάσης και πατήστε δεξί κλικ. Στο παράθυρο που θα ανοίξει επιλέξτε Tools/Create Tests. Θα ανοίξει το παράθυρο που φαίνεται στην εικόνα 8.4.



Εικόνα 8.4 *Παράμετροι δημιουργίας test*

Στην ενότητα Code Generation, θέστε τις επιλογές όπως φαίνονται στην εικόνα 8.4 και πατήστε OK. Θα διαπιστώσετε πως δημιουργήθηκαν δύο νέοι φάκελοι κάτω από το project. Οι φάκελοι Test Packages και Test Libraries. Κάντε δεξί κλικ στον φάκελο Test Libraries και στο μενού που θα ανοίξει επιλέξτε Add Library. Θα ανοίξει τότε ένα παράθυρο με τις διαθέσιμες βιβλιοθήκες. Εντοπίστε το Junit 4.12, επιλέξτε το και πατήστε Add Library. Ανοίξτε τον φάκελο Test Libraries και διαγράψτε (remove) τις υπόλοιπες βιβλιοθήκες πλην της Junit 4.12 και της Hamcrest 1.3. Αν δεν βλέπετε την Hamcrest προσθέστε την όπως προσθέσαμε την Junit 4.12.

Μεταβείτε στον φάκελο Test Packages, εντοπίστε το ADiceGameTest και διαγράψτε το. Τέλος μεταβείτε και πάλι στην 1η έκδοση της κλάσης, πατήστε δεξί κλικ και δημιουργήστε εκ' νέου το ADiceGameTest. Η κλάση θα ανοίξει αυτόματα στην περιοχή του κώδικα. Περιηγηθείτε στον κώδικα της κλάσης και παρατηρήστε πως για κάθε μη ιδιωτική συνάρτηση της κλάσης AdiceGame, η κλάση AdiceGameTest περιλαμβάνει μια συνάρτηση ελέγχου. Για παράδειγμα, για την συνάρτηση approximateEquals περιλαμβάνει την TestApproximateEquals, για την sum την TestSum, κοκ. Θα πρέπει να διορθώσουμε κατάλληλα τις συναρτήσεις ελέγχου ώστε να επιτελούν το έργο τους.

Αντικαταστήστε την ADiceGameTest που παρήχθη αυτόματα με την ADiceGameTest του κώδικα 8.9 όπου έχουν γίνει οι απαιτούμενες αλλαγές.

```
import java.util.Random;
import org.junit.Test;
import static org.junit.Assert.*;
```

```java
/**
 *
 * @author Lefteris Moussiades <lmous@cs.ihu.gr>
 */
public class ADiceGameTest {

    /**
     * Test of approximateEquals method, of class ADiceGame.
     */
    @Test
    public void testApproximateEquals() {
        System.out.println("approximateEquals");
        double d1 = 0.0;
        double d2 = ADiceGame.EPSILON;
        boolean result = ADiceGame.approximateEquals(d1, d2);
        assertEquals(false, result);
        d2 = ADiceGame.EPSILON / 2;
        assertEquals(true, ADiceGame.approximateEquals(d1, d2));
    }

    /**
     * Test of sum method, of class ADiceGame.
     */
    @Test
    public void testSum() {
        System.out.println("sum");
        double[] array = ADiceGame.fair;
        double result = ADiceGame.sum(array);
        assertEquals(1d, result, ADiceGame.EPSILON);
        array = ADiceGame.lucky;
        result = ADiceGame.sum(array);
        assertEquals(1d, result, ADiceGame.EPSILON);
        array = ADiceGame.unlucky;
        result = ADiceGame.sum(array);
        assertEquals(1d, result, ADiceGame.EPSILON);
    }

    /**
     * Test of roll method, of class ADiceGame.
     */
    @Test
    public void testRoll() {
        System.out.println("roll");
        double[] distribution;
        int sumFair = 0, sumLucky = 0, sumUnlucky = 0, rslt;
        for (int i = 0; i < 10000; i++) {
            rslt = ADiceGame.roll(ADiceGame.fair);
            if (rslt < 1 || rslt > 6) {
                fail("roll result " + rslt);
            }
            sumFair += rslt % 2 == 0 ? 1 : 0;
            rslt = ADiceGame.roll(ADiceGame.lucky);
            if (rslt < 1 || rslt > 6) {
                fail("roll result " + rslt);
            }
            sumLucky += rslt % 2 == 0 ? 1 : 0;
            rslt = ADiceGame.roll(ADiceGame.unlucky);
            if (rslt < 1 || rslt > 6) {
                fail("roll result " + rslt);
            }
            sumUnlucky += rslt % 2 == 0 ? 1 : 0;
```

```
        }
        System.out.println(sumLucky/10000d+" "+sumFair/10000d+"
"+sumUnlucky/10000d);
        assertTrue(sumLucky > sumFair && sumFair > sumUnlucky);
    }

    /**
     * Test of max method, of class ADiceGame.
     */
    public static void shuffle(int[] t) {
        Random r = new Random();

        for (int i = 0; i < t.length; i++) {
            int rIdx = r.nextInt(t.length);
            int tmp = t[i];
            t[i] = t[rIdx];
            t[rIdx] = tmp;
        }
    }

    private int[] constructArray(int initialValue) {
        int[] rVal = new int[10];
        for (int i = 0; i < 10; i++) {
            rVal[i] = initialValue + i;
        }
        shuffle(rVal);
        return rVal;
    }

    @Test
    public void testMax() {
        System.out.println("max");
        int arrayNo = 0, initialValue = 1;
        int[] array;
        while (arrayNo < 1000) {
            array = constructArray(initialValue);
            assertEquals(initialValue + 9, array[ADiceGame.max(array)]);
            initialValue += 10;
            arrayNo++;
        }
    }

    /**
     * Test of cntOccurances method, of class ADiceGame.
     */
    @Test
    public void testCntOccurances() {
        System.out.println("cntOccurances");
        int[] array = {1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5, 5, 5, 5, 6, 6, 6, 6,
6, 6};
        assertEquals(1, ADiceGame.cntOccurances(array, 1));
        assertEquals(2, ADiceGame.cntOccurances(array, 2));
        assertEquals(3, ADiceGame.cntOccurances(array, 3));
        assertEquals(4, ADiceGame.cntOccurances(array, 4));
        assertEquals(5, ADiceGame.cntOccurances(array, 5));
        assertEquals(6, ADiceGame.cntOccurances(array, 6));
    }

    /**
     * Test of withdraw method, of class ADiceGame.
     */
```

```
    @Test
    public void testWithdraw() {
        System.out.println("withdraw");
        int[] score = {1100,5,1100};
        ADiceGame.withdraw(score);
        assertTrue(ADiceGame.withdraw(score));
        score[2] = 6;
        assertFalse(ADiceGame.withdraw(score));
    }

}
```

Κώδικας 8.9 *Test για την κλάση AdiceGame*

Καταρχάς να επισημάνουμε τον ρόλο μερικών νέων συναρτήσεων που περιλαμβάνονται στην ADiceGameTest και συναντούμε για πρώτη φορά.

assertEquals με δύο παραμέτρους ίδιου τύπου. Ελέγχει τις παραμέτρους και αν δεν είναι ίσες σημειώνει λάθος.

assertEquals με τρεις πραγματικές παραμέτρους. Ελέγχει τις δύο πρώτες παραμέτρους για προσεγγιστική ισότητα. Η Τρίτη παράμετρος είναι η ανεκτή διαφορά. Στην δική μας περίπτωση χρησιμοποιούμε το EPSILON της εφαρμογής.

fail σημειώνει λάθος με μήνυμα την πραγματική παράμετρό της.

assertTrue σημειώνει λάθος εφόσον η παράμετρός της είναι ψευδής.

assertFalse σημειώνει λάθος εφόσον η παράμετρός της είναι αληθής.

Ας συζητήσουμε τώρα κάποιες από τις συναρτήσεις της ADiceGameTest ώστε να κατανοήσουμε πως πρέπει να γράφουμε τις συναρτήσεις ελέγχου.

Η συνάρτηση testSum ελέγχει την συνάρτηση sum της ADiceGame. Γνωρίζουμε ότι έχουμε τρεις πίνακες πραγματικών δηλωμένους στην ADiceGame, τον fair, τον lucky και τον unlucky και πως το άθροισμα των στοιχείων κάθε ενός από αυτούς τους πίνακες πρέπει να είναι ίσο με 1. Μπορούμε να αναπτύξουμε την testSum βασιζόμενοι σε αυτό το δεδομένο. Πράγματι, αν εξετάσετε τον κώδικα της testSum θα διαπιστώσετε περιλαμβάνει διαδοχικές κλήσεις στην sum, μία για κάθε έναν από τους fair, lucky και unlucky. Λαμβάνει το αποτέλεσμα της κλήσης στην μεταβλητή result την οποία συγκρίνει προσεγγιστικά μέσω της assertEquals με το 1. Αν η assertEquals εντοπίσει απόκλιση θα σημειώσει λάθος.

Η testRoll καλεί την roll 10.000 φορές για κάθε μία κατανομή, δηλ. για τους πίνακες fair, lucky και unlucky. Αν παραχθεί αριθμός έξω από το όριο 1 έως 6, καλεί την fail και σημειώνει το λάθος. Στο τέλος του βρόγχου καλείται η assertTrue για να ελέγξει την συνθήκη sumLucky > sumFair && sumFair > sumUnlucky. Πράγματι, γνωρίζουμε πως η κατανομή lucky φέρνει περισσότερα άρτια αποτελέσματα από ότι η fair και η fair περισσότερα από ότι η unlucky. Μετά από 10.000 ρίψεις, οι πιθανότητες λογικά επαληθεύονται. Σε περίπτωση όμως που δεν επαληθευτούν η assertTrue θα σημειώσει λάθος.

Στην testMax δημιουργούμε 1000 πίνακες με την βοήθεια της constructArray κατά τρόπο που γνωρίζουμε για κάθε πίνακα την τιμή του μέγιστου στοιχείου του. Προσέξτε πως η constructArray ανακατεύει τα στοιχεία του πίνακα που δημιουργεί έτσι ώστε το στοιχείο με την μέγιστη τιμή να μην βρίσκεται πάντα στην τελευταία θέση του πίνακα. Στην συνέχεια ελέγχουμε αν η τιμή που επιστρέφει η max ανταποκρίνεται στην αναμενόμενη.

Εφόσον έχετε ακολουθήσει τα βήματα έως εδώ με ακρίβεια, το τεστ θα τρέξει επιτυχώς. Καλό είναι όμως να δείτε τι συμβαίνει σε περίπτωση που υπάρχει κάποια αποτυχία στο τεστ.
Μετατρέψτε προσωρινά τις cntOccurances και withdraw της ADiceGame όπως δείχνει ο κώδικας 8.10.

```
    public static int cntOccurances(int[] array, int value) {
        int cnt = 0;
        for (int i = 0; i < array.length; i++) {
            if (array[value] == array[i]) {
                cnt++;
            }
        }
        return cnt;
```

```
    }

    public static boolean withdraw(int[] score) {
        return cntOccurances(score, max(score)) > 1;
    }
```

Κώδικας 8.10 *Προσωρινή έκδοση της cntOccurances και withdraw.*

Μετά την μετατροπή, τρέξτε και πάλι την κλάση ADiceGameTest. Μελετήστε το αποτέλεσμα. Επαναφέρετε τις cntOccurances και withdraw στην προηγούμενη σωστή μορφή τους.

8.9 Ασκήσεις προς λύση

8.9.1 Τεκμηρίωση, Απολαθοποίηση και Έλεγχος

Δίνεται η κλάση CharArrayLib στον κώδικα 8.11. Πρόκειται για μια βιβλιοθήκη διαχείρισης πινάκων χαρακτήρων. Στην main της κλάσης δημιουργούνται δύο πίνακες χαρακτήρων, ο s1={‘E’.’r’,’i’,’k’} και ο s2={‘s’,’o’,’n’}. Στην συνέχεια δημιουργείται ένας τρίτος πίνακας σαν συνένωση των δύο πρώτων. Αμέσως μετά ελέγχεται αν ο πίνακας συνένωση περιλαμβάνει τον s1 και τον s2. Η έξοδος του κώδικα είναι 0 και -1. Αυτό σημαίνει πως ο πίνακας s1 βρέθηκε στον πίνακα συνένωση και αρχίζει στην θέση 0. Ο πίνακας s2 όμως δεν βρέθηκε ενώ περιέχεται στον πίνακα-συνένωση.
Να τεκμηριωθεί και να απολαθοποιηθεί η κλάση και να υλοποιηθεί κατάλληλος κώδικας ελέγχου.

```
public class CharArrayLib {

    static char[] create(String s) {
        return s.toCharArray();
    }

    static int compare(char[] s1, char[] s2) {
        int idx = 0;
        while (idx < s1.length && idx < s2.length) {
            if (s1[idx] > s2[idx]) {
                return 1;
            }
            if (s1[idx] < s2[idx]) {
                return -1;
            }
            idx++;
        }
        return s1.length-s2.length;
    }

    static int compareIgnoreCase(char[] s1, char[] s2) {
        int idx = 0;
        while (idx < s1.length && idx < s2.length) {
            char u1 = Character.toUpperCase(s1[idx]),
                 u2 = Character.toUpperCase(s2[idx]);
            if (u1 > u2) {
                return 1;
            }
            if (u1 < u2) {
                return -1;
            }
            idx++;
        }

        return s1.length - s2.length;
    }
```

```
    static char[] concat(char[] s1, char[] s2) {
        char[] rVal = new char[s1.length + s2.length];
        System.arraycopy(s1, 0, rVal, 0, s1.length);
        System.arraycopy(s2, 0, rVal, s1.length, s2.length);
        return rVal;
    }

    public static boolean equals(char[] tA, char[] tB) {
        if (tA.length != tB.length) {
            return false;
        }
        for (int i = 0; i < tA.length; i++) {
            if (tA[i] != tB[i]) {
                return false;
            }
        }
        return true;
    }

    static char[] subString(char[] s, int from, int to) {
        char[] rVal = new char[to - from];
        int idx = 0;
        for (int i = from; i < to; i++) {
            rVal[idx++] = s[i];
        }
        return rVal;
    }

    static int contains(char[] s, char... cS) {
        if (cS.length > s.length) {
            return -1;
        }

        for (int i = 0; i < s.length - cS.length; i++) {
            char[] sub = subString(s, i, cS.length + i);
            if (equals(sub, cS)) {
                return i;
            }
        }
        return -1;
    }

    public static void main(String[] args) {
        char[] s1 = create("Erik"), s2 = create("son");
        char[] con = concat(s1, s2);
        int idx = contains(con, s1);
        System.out.println(idx);
        idx = contains(con, s2);
        System.out.println(idx);
        char[] s4 =create("EriksoN");
        System.out.println(compareIgnoreCase(con,s4));
    }
}
```

Κώδικας 8.11 *Βιβλιοθήκη διαχείρισης πινάκων χαρακτήρων*

8.9.2 Παιχνίδι με τράπουλα

Τέσσερις φίλοι, ο John, ο Jim, η Mary και η Ann παίζουν χαρτιά. Το παιχνίδι που παίζουν έχει τους ακόλουθους κανόνες. Καταρχάς, ανακατεύεται η τράπουλα, 52 φύλλων (χωρίς μπαλαντέρ). Στον πρώτο γύρο τοποθετούνται

τέσσερις κάρτες ανοικτές σε στοίβα στο τραπέζι και μοιράζονται από τέσσερις κάρτες σε κάθε παίκτη. Στους επόμενους γύρους λαμβάνουν από τέσσερις κάρτες μόνο οι παίκτες. Οι γύροι επαναλαμβάνονται έως ότου εξαντληθεί η τράπουλα.

Σε κάθε γύρο, οι παίκτες ανοίγουν τις κάρτες τους εναλλάξ μια προς μια και τις τοποθετούν στην κορυφή της στοίβας. Αν η κάρτα που άνοιξε ο παίκτης έχει ίδια τιμή με την κάρτα που βρίσκεται στην κορυφή της στοίβας, ο παίκτης κερδίζει τις κάρτες της στοίβας. Επίσης, κερδίζει τις κάρτες της στοίβας εφόσον η κάρτα που άνοιξε είναι βαλές. Όταν εξαντληθεί η τράπουλα, ο παίκτης που έχει τις περισσότερες κάρτες κερδίζει τις τυχόν εναπομείνασες κάρτες της στοίβας. Τέλος, οι παίκτες αθροίζουν τις τιμές των καρτών που έχουν μαζέψει και όποιος συγκεντρώνει το μεγαλύτερο άθροισμα, ανακηρύσσεται νικητής.

Να υλοποιηθεί προσομοίωση μιας παρτίδας παιχνιδιού μεταξύ των τεσσάρων φίλων. Να συνοδευτεί από κατάλληλο κώδικα ελέγχου.

Βιβλιογραφία

[1] L. B. Wilson and R. G. Clark, Comparative Programming Languages. Pearson Education, 2001.

[2] C. Böhm and G. Jacopini, "Flow diagrams, turing machines and languages with only two formation rules," Commun. ACM, vol. 9, no. 5, pp. 366–371, May 1966, doi: 10.1145/355592.365646.

[3] "Exception handling," Wikipedia. Sep. 19, 2021. Accessed: Oct. 15, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Exception_handling&oldid=1045188202

[4] K. Wiegers and J. Beatty, Software Requirements, 3rd edition. Redmond, Washington: Microsoft Press, 2013.

[5] "javadoc-The Java API Documentation Generator." https://docs.oracle.com/javase/7/docs/technotes/tools/windows/javadoc.html (accessed Sep. 14, 2021).

[6] "How to Write Doc Comments for the Javadoc Tool." https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html (accessed Sep. 14, 2021).

Κεφάλαιο 9

Σύνοψη
Σε αυτήν την ενότητα αναλύεται σε βάθος η έννοια της αναδρομής. Καταρχάς, αναλύεται η έννοια σε γενικότερο πλαίσιο, στην συνέχεια εξηγείται η λειτουργία της μνήμης stack (στοίβα) και μετά παρουσιάζονται οι προγραμματιστικές αναδρομικές τεχνικές, δηλ. η άμεση και η αμοιβαία αναδρομή. Τέλος, περιλαμβάνεται συζήτηση για τα πλεονεκτήματα και μειονεκτήματα της αναδρομής σε σχέση με τις μη αναδρομικές προσεγγίσεις.

Προαπαιτούμενη γνώση
Οι θεμελιώδεις τύποι και βασική γνώση του τύπου String. Βασική είσοδος και έξοδος. Η λειτουργία των τελεστών. Οι δομές επιλογής και οι επαναληπτικές δομές. Η διαχείριση των πινάκων. Οι συναρτήσεις, η έννοια και βασική διαχείριση των απροσδόκητων λαθών, οι προσδιοριστές προσπέλασης και η χρήση των πακέτων.

Λέξεις κλειδιά
Μνήμη stack, αναδρομή, αμοιβαία αναδρομή

9 Αναδρομή

Όταν στον ορισμό μιας έννοιας χρησιμοποιείται η ίδια αυτή έννοια, λέμε πως έχουμε αναδρομικό ορισμό (recursive definition). Χαρακτηριστικό παράδειγμα αναδρομικού ορισμού είναι ο ορισμός της ακολουθίας Fibonacci που είδαμε στην άσκηση 7.8.3. Πιο συγκεκριμένα, αν Fn αναπαριστά τον ν-οστό όρο, η τιμή του υπολογίζεται ως εξής:

$$n = 0 \rightarrow F_n = 0$$
$$n = 1 \rightarrow F_n = 1$$
$$n > 1 \rightarrow F_n = F_{n-1} + F_{n-2}$$

Δηλαδή, αν το n είναι ίσο με το 0 ή το 1, ο ν-οστός όρος ισούται με το n. Για τιμές μεγαλύτερες του 1, ο ν-οστός όρος ισούται με το άθροισμα των δύο προηγούμενων όρων. Επομένως, η τιμή ενός όρου της ακολουθίας Fibonacci ορίζεται με βάση την τιμή δύο όρων της ίδιας ακολουθίας.

Πως όμως μπορούμε να υπολογίσουμε το F3; Με βάση αυτόν τον ορισμό, F3=F2+F1. Το F1 έχει τιμή 1. Δεν γνωρίζουμε όμως την τιμή του F2. Ωστόσο, γνωρίζουμε πως F2=F1+F0. Επομένως, F2=1 και άρα F3=2.

Ένα άλλο χαρακτηριστικό παράδειγμα είναι ο αναδρομικός ορισμός της δύναμης ενός αριθμού υψωμένου σε εκθέτη. Ας πάρουμε την απλή περίπτωση που η βάση είναι πραγματικός και ο εκθέτης ακέραιος. Σε αυτήν την περίπτωση ο γνωστός μας μη αναδρομικός ορισμός έχει ως εξής;

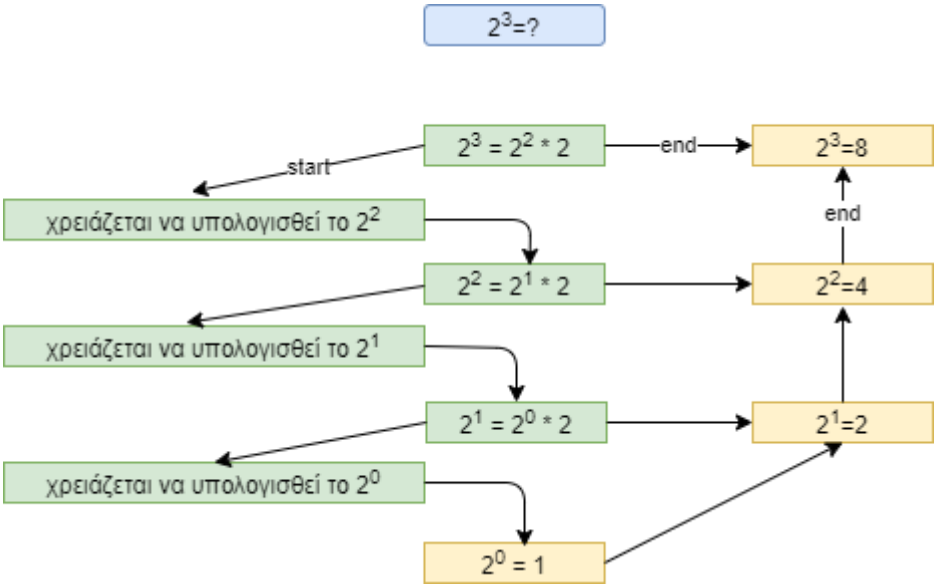
$$a^e = \begin{cases} 1 \; \alpha\nu \; e = 0 \\ \alpha * a * \dots a, e \; \varphi o\rho\acute{\varepsilon}\varsigma \; \alpha\nu \; e > 0 \\ \dfrac{1}{a * a * \dots a, e \; \varphi o\rho\acute{\varepsilon}\varsigma} \; \alpha\nu \; e < 0 \end{cases}$$ [8]

Όμως ισχύει και ο αναδρομικός ορισμός.

$$a^e = \begin{cases} 1, \alpha\nu \; e = 0 \\ a^{e-1} * a, \alpha\nu \; e > 0 \\ \dfrac{1}{a^{-e}}, \alpha\nu \; e < 0 \end{cases}$$

[8] Ας σημειωθεί εδώ πως το 0^0 δεν ορίζεται. Ωστόσο σε πολλές περιπτώσεις θεωρείται πως 0^0=1. Πολλές γλώσσες προγραμματισμού συμπεριλαμβανομένων των Java, Javascript και C, C++, υπολογίζουν το 0^0 ίσο με 1. Για παράδειγμα, αν καλέσετε την συνάρτηση Math.pow(0, 0) θα διαπιστώσετε πως επιστρέφει 1. Για συμβατότητα θεωρούμε σε αυτό το βιβλίο πως το $0^0 = 1$.

Όπως βλέπουμε στον αναδρομικό ορισμό της δύναμης το ae ορίζεται ως το ae-1*a όταν το e είναι θετικός και ως 1/a-e όταν το e είναι αρνητικός. Πως θα υπολογίσουμε αναδρομικά την τιμή του 23; Ας το συζητήσουμε με την βοήθεια του σχήματος 9.1.



Σχήμα 9.1 *Ο αναδρομικός υπολογισμός του 23*

Όπως φαίνεται στο σχήμα 9.1, για να υπολογίσουμε το 23 θα πρέπει πρώτα να υπολογίσουμε το 22. Παρόμοια, για να υπολογίσουμε το 22, θα πρέπει πρώτα να υπολογίσουμε το 21 και για να υπολογίσουμε το 21 θα πρέπει να υπολογίσουμε το 20. Το 20 όμως εξορισμού είναι ίσο με το 1. Επομένως τώρα εύκολα υπολογίζουμε το 21 που είναι ίσο με 2. Γνωρίζοντας το 21, μπορούμε να υπολογίσουμε το 22 που είναι ίσο με 4. Τέλος, γνωρίζοντας το 22, εύκολα υπολογίζουμε το 23 που είναι ίσο με 8.

Γενικότερα, η αναδρομή αποδομεί ένα πρόβλημα τάξης ν σε ένα ή περισσότερα ίδια προβλήματα τάξης μικρότερης του ν. Η διαδικασία της αποδόμησης επαναλαμβάνεται έως ότου φτάσουμε σε ένα ή περισσότερα απλά προβλήματα με προφανή λύση. Τότε τα αποτελέσματα από τα προφανή προβλήματα αξιοποιούνται για την λύση των προβλημάτων του αμέσως ανώτερου επιπέδου των οποίων η λύση επίσης συνεισφέρει στην λύση των προβλημάτων του ανώτερου επιπέδου, κοκ, έως ότου φθάσουμε στην λύση του αρχικού προβλήματος.

Στην συνέχεια, θα δούμε την αναδρομή στην Java και θα συζητήσουμε τα πλεονεκτήματα και μειονεκτήματά της. Πρώτα όμως θα δώσουμε μερικές απαραίτητες πληροφορίες για μια περιοχή της μνήμης ενός προγράμματος που ονομάζεται στοίβα (stack).

9.1 Η λειτουργία της στοίβας

Για ευκολία θα υποθέσουμε πως κάθε εφαρμογή Java διαθέτει μια μνήμη που ονομάζεται στοίβα[9]. Σύμφωνα με τις προδιαγραφές (specification) της Java [1], η στοίβα μπορεί να είναι σταθερού ή δυναμικά μεταβαλλόμενου μεγέθους. Και στις δύο περιπτώσεις, η στοίβα μπορεί να γεμίσει με δεδομένα. Στην πρώτη περίπτωση αφού καλυφθεί το προκαθορισμένο μέγεθός της και στην δεύτερη αφού καταλάβει όλη την διαθέσιμη δυναμική μνήμη. Αν το πρόγραμμά μας απαιτεί επιπλέον μνήμη στοίβας, τότε έχουμε αδυναμία συνέχισης της εκτέλεσης του προγράμματος. Επομένως, η εξάντληση της στοίβας είναι ένα πολύ σοβαρό πρόβλημα που πάντα πρέπει να λαμβάνει ο προγραμματιστής υπόψιν του.

Ας δούμε όμως τι αποθηκεύεται στην στοίβα και ποια μπορεί να είναι η πιθανή αιτία εξάντλησής της. Στον κώδικα 9.2 παρουσιάζουμε μια απλή μελέτη περίπτωσης.

```
public class StackFunctionality {
```

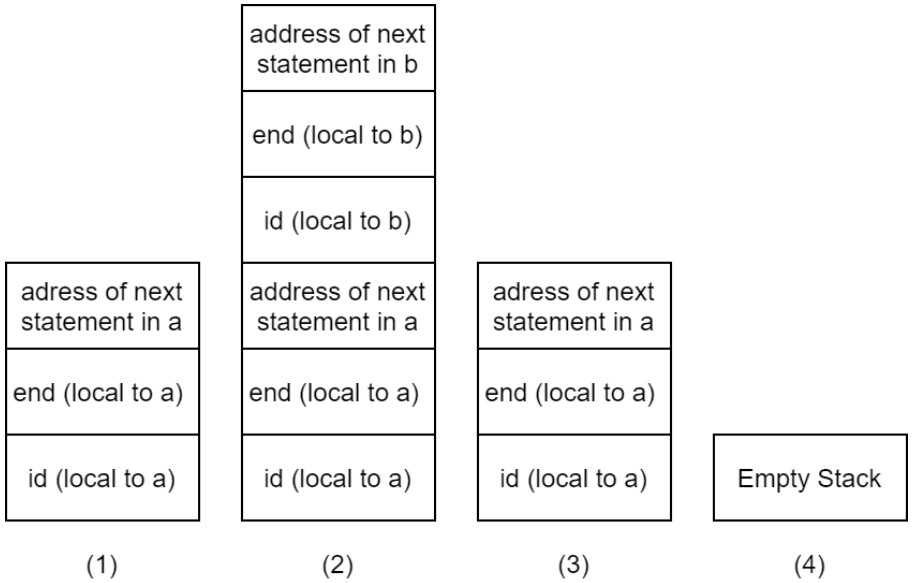
[9] Στην πραγματικότητα, στις πολυνηματικές (multithreading) εφαρμογές, διαμορφώνεται μια στοίβα για κάθε νήμα.

```
    static void a() {
        String id = "static function a";
        System.out.println("start " + id);
        String end = "end";
        b();
        System.out.println(end + " " + id);
    }

    static void b() {
        String id = "static function b";
        System.out.println("start " + id);
        String end = "end";
        c();
        System.out.println(end + " " + id);
    }

    static void c() {
        String id = "static function c";
        System.out.println("start " + id);
        String end = "end";
        System.out.println("c is running");
        System.out.println(end + " " + id);
    }

    public static void main(String[] args) {
        a();
    }
}
```

Κώδικας 9.1 *Περίπτωση μελέτης λειτουργίας της στοίβας*

Όπως βλέπουμε στον κώδικα 9.1, η συνάρτηση a καλεί την b και αυτή με την σειρά της, την c. Όταν όμως η c θα ολοκληρωθεί, θα πρέπει η b να συνεχίσει. Από ποιο σημείο όμως; Μα από την γραμμή της a που έπεται της κλήσης της c. Το ίδιο θα συμβεί όταν ολοκληρωθεί η εκτέλεση της b, δηλ. θα πρέπει να συνεχίσει η a. Προφανώς, από την γραμμή του κώδικα που έπεται της κλήσης της b. Η συνάρτηση a προσπελαύνει τις μεταβλητές end και id. Όμως, η b διαθέτει δικές της τέτοιες μεταβλητές. Η προσπέλαση με τα αναγνωριστικά end και id μέσα στην b αναφέρεται στις τοπικές μεταβλητές της b. Αντίθετα, οι τοπικές μεταβλητές της a είναι απροσπέλαστες κατά την εκτέλεση της b. Επομένως, είναι φανερό πως η κλήση συνάρτησης μέσα από άλλη συνάρτηση απαιτεί μια ειδική διαχείριση. Η διαχείριση αυτή βασίζεται στην στοίβα.

Η στοίβα είναι μια Last In First Out (LIFO) δομή, δηλ. ότι μπαίνει τελευταίο στην στοίβα βγαίνει πρώτο ή αλλιώς ότι εισάγεται στην στοίβα, τοποθετείται στην κορυφή της και ότι εξάγεται, λαμβάνεται από την κορυφή της.



Σχήμα 9.2 *Η στοίβα κατά την εκτέλεση της main του κώδικα 9.1*

Ας δούμε όμως πως λειτουργεί η στοίβα κατά την εκτέλεση του κώδικα 9.1. Για απλοποίηση, στην περιγραφή αυτή παραλείπουμε τις κλήσεις πέραν των κλήσεων της a, της b και της c. Καταρχάς, έχουμε την κλήση της a. Η a κατανέμει τις τοπικές μεταβλητές της και προχωρά στην εκτέλεση της μέχρι του σημείου που καλεί την b. Πριν ακριβώς, ξεκινήσει η b, οι τοπικές μεταβλητές της a αποθηκεύονται στην στοίβα όπως δείχνει το σχήμα 9.2-1. Όπως φαίνεται σε αυτό το σχήμα, εκτός από τις τοπικές μεταβλητές, στην στοίβα αποθηκεύεται και η διεύθυνση του κώδικα από την οποία θα συνεχιστεί η εκτέλεση της a όταν η b θα έχει τερματίσει. Στην συνέχεια εκτελείται η b μέχρι του σημείου που καλείται η c. Πριν την κλήση της c, οι τοπικές μεταβλητές της b και η διεύθυνση της γραμμής που έπεται της κλήσης της c, επίσης αποθηκεύονται στην στοίβα. Έτσι, η στοίβα διαμορφώνεται όπως δείχνει το σχήμα 9.2-2. Με τον τερματισμό της c, η b εξάγει τις απαραίτητες πληροφορίες από την κορυφή της στοίβας και συνεχίζει την εκτέλεσή της, οπότε η στοίβα διαμορφώνεται όπως δείχνει το σχήμα 9.2-3. Παρόμοια, με τον τερματισμό της b, η a εξάγει από την στοίβα τις πληροφορίες που την αφορούν και οι οποίες μετά την εξαγωγή των πληροφοριών της b, βρίσκονται στην κορυφή της στοίβας. Έτσι, η a συνεχίζει απρόσκοπτα την λειτουργία της ενώ η στοίβα έχει αδειάσει (σχήμα 9.2-4).

Επομένως, κάθε φορά που μια συνάρτηση καλείται από μία άλλη, στην στοίβα τοποθετούνται διάφορες πληροφορίες. Άρα είναι δυνατόν αν έχουμε πολύ μεγάλο βάθος κλήσεων, η στοίβα να μην διαθέτει άλλο χώρο. Ωστόσο, κάτι τέτοιο είναι απίθανο να συμβεί μέσα από μη αναδρομικές κλήσεις. Μπορεί όμως να συμβεί από αναδρομικές κλήσεις. Πόσες φορές θα καλέσει μία συνάρτηση τον εαυτό της, εξαρτάται από το μέγεθος του προβλήματος που επιχειρεί να επιλύσει.

9.2 Αναδρομικές συναρτήσεις

Σύμφωνα με τα όσα έχουμε πει, είναι εύκολο να υποθέσουμε πως μια συνάρτηση είναι αναδρομική όταν καλεί τον εαυτό της. Προσέξτε την συνάρτηση endless στον κώδικα 9.2.

```
static void endless(int i) { //Κώδικας 9.2
    System.out.println(i);
    endless(i + 1);
}
```

Κώδικας 9.2 *Αναδρομική συνάρτηση χωρίς συνθήκη ελέγχου*

Καταρχάς, είναι φανερό πως πρόκειται για αναδρομική συνάρτηση. Αν καλέσουμε endless(0), τότε η συνάρτηση θα εμφανίσει 0 και στην συνέχεια θα καλέσει endless(1). Η endless(1) θα εμφανίσει 1 και θα καλέσει endless(2). Παρόμοια, η endless(2) θα εμφανίσει 2 και θα καλέσει endless(3). Η αναδρομική κλήση θα συνεχιστεί στην ουσία μέχρις ότου γεμίσει η στοίβα και το πρόγραμμα πέσει.

Επομένως, κάθε αναδρομική συνάρτηση πρέπει να περιλαμβάνει μια συνθήκη ελέγχου της αναδρομικής κλήσης. Με άλλα λόγια, μια αναδρομική συνάρτηση πρέπει να περιλαμβάνει κάποιο κώδικα που κάτω από κάποιες συνθήκες θα τερματίζει την αναδρομική κλήση.

```
static void withEnd(int i) { //Κώδικας 9.3
    System.out.println(i);
    if (i < 3) {
        withEnd(i + 1);
    }
}
```

Κώδικας 9.3 *Απλή αναδρομή με συνθήκη τερματισμού*

Όπως φαίνεται στον κώδικα 9.3, η withEnd είναι επίσης αναδρομική. Ωστόσο, η αναδρομική κλήση γίνεται μόνο εφόσον το i είναι μικρότερο του 3. Για παράδειγμα, αν καλέσουμε withEnd(0), η συνάρτηση θα εμφανίσει 0, στην συνέχεια ελέγχει το i, το βρίσκει μικρότερο του 3 καλεί την endless(1). Με την ίδια διαδικασία, τυπώνει 1, μετά 2, καλεί και withEnd(3), εμφανίζει και το 3 και μετά βρίσκει ότι η έκφραση i<3 είναι ψευδής και σταματά την αναδρομική κλήση. Σε αυτήν την περίπτωση μπορούμε να πούμε ότι η withEnd δεν πέφτει. Ισχύει όμως αυτό γενικευμένα; Η απάντηση είναι όχι. Όπως αναφέραμε και προηγουμένως εξαρτάται από το μέγεθος του προβλήματος. Για παράδειγμα, αν καλέσουμε withEnd(Integer.MIN_VALUE), θα έχουμε τόσες αναδρομικές κλήσεις ώστε η συνάρτηση θα πέσει πολύ πριν κληθεί με παράμετρο 3.

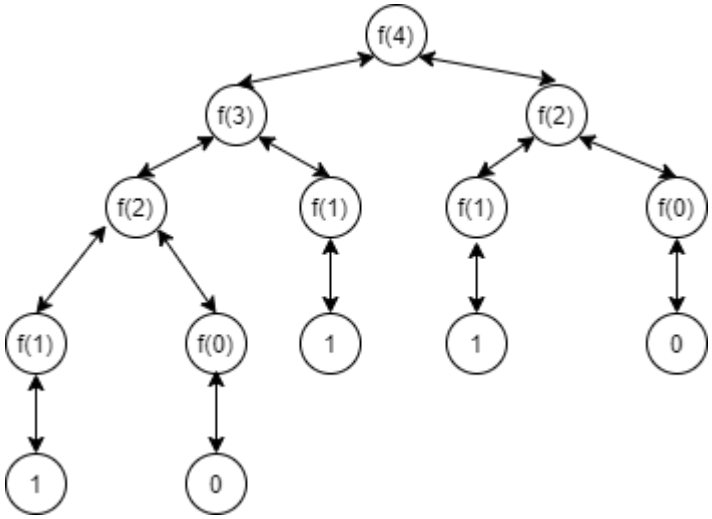
Αυτό είναι και το βασικό μειονέκτημα της αναδρομής. Οποιαδήποτε αναδρομική συνάρτηση μπορεί να αντιμετωπίσει πρόβλημα χωρητικότητας του στοίβας ανάλογα με το μέγεθος του προβλήματος που επιχειρεί να αντιμετωπίσει.

Στον κώδικα 9.4 παραθέτουμε την αναδρομική υλοποίηση υπολογισμού του ν-οστού όρου της ακολουθίας Fibonacci.

```
static int fibonacci(int n) { //Κώδικας 9.4
    if (n < 0) {
        throw new RuntimeException();
    }
    if (n <= 1) {
        return n;
    } else {
        return fibonacci(n - 1) + fibonacci(n - 2);
    }
}
```

Κώδικας 9.4 *Αναδρομική υλοποίηση υπολογισμού το ν-οστού όρου της ακολουθίας Fibonacci*

Στην ακολουθία Fibonacci δεν ορίζονται όροι αρνητικής τάξης. Έτσι αν η συνάρτηση κληθεί με αρνητικό όρισμα παράγει εξαίρεση. Στην συνέχεια, ο κώδικας 9.4 υλοποιεί ακριβώς τον ορισμό της ακολουθίας που δόθηκε στην αρχή της ενότητας 9. Ας δούμε πως λειτουργεί η συνάρτηση όταν κληθεί με όρισμα 4. Στο σχήμα 9.3, για απλοποίηση, η συνάρτηση συμβολίζεται με f αντί για fibonacci.



Σχήμα 9.3 *Κλήσεις και τιμές επιστροφής της fibonacci*

Όπως βλέπουμε στο σχήμα 9.3, καταρχάς έχουμε κλήση της f(4). Σύμφωνα με τον κώδικα 9.4 η κλήση αυτή θα επιστρέψει f(3)+f(2). Επομένως, καλείται η f(3) και η f(2) όπως δείχνει και το σχήμα 9.3. Στην συνέχεια, ο υπολογισμός της f(3) απαιτεί την κλήση της f(2) και f(1) και ο υπολογισμός της f(2) απαιτεί την κλήση f(1) και f(0). Σε αυτό το βάθος τρεις κόμβοι του δένδρου, οι f(1), f(1) και f(0), υπολογίζονται χωρίς περαιτέρω αναδρομική κλήση. Αντίθετα ο κόμβος f(2) απαιτεί την κλήση των f(1) και f(0). Κάθε τερματικός κόμβος επιστρέφει την τιμή του στο αμέσως προηγούμενο επίπεδο. Έτσι το f(1) υπολογίζεται ως 1, το f(0) ως 0, το f(2) ως 1, το f(3) ως 2 και το f(4) ως 3.

Ας σημειωθεί πως η περίπτωση κατά την οποία το n είναι μικρότερο ή ίσο του 1 ονομάζεται βασική περίπτωση (base case). Γενικότερα, στην αναδρομή, βασική περίπτωση ονομάζεται η περίπτωση κατά την οποία υπάρχει προφανής λύση που ο υπολογισμός της δεν απαιτεί αναδρομική κλήση.

Το παράδειγμα που θα δούμε τώρα είναι απλό, ωστόσο ιδιαίτερα κρίσιμο για την κατανόηση του μηχανισμού αναδρομικών κλήσεων.

```
static void prt(int i) { // Κώδικας 9.5 prt(1) 123321
    System.out.print(i);
    if (i < 3) {
        prt(i + 1);
    }
    System.out.print(i);
}
```

Κώδικας 9.5 *Συνέχεια της συνάρτησης μετά την αναδρομική κλήση*

Προσέξτε την συνάρτηση prt του κώδικα 9.5. Τι εμφανίζει κατά την άποψή σας η κλήση prt(3); Συχνά, λαμβάνω λανθασμένη απάντηση σε αυτήν την ερώτηση. Πολλοί φοιτητές στο στάδιο της εκμάθησης της αναδρομής θεωρούν πως η prt(0) θα εμφανίσει 0123. Για να το δούμε βήμα προς βήμα. Όταν αρχίσει η εκτέλεση της prt(0) είναι προφανές πως καταρχάς θα εμφανίσει 0. Στην συνέχεια, ελέγχει την συνθήκη i<3 την οποία βρίσκει αληθή και εκτελεί το μπλοκ της if. Με την ολοκλήρωση του μπλοκ της if, ο έλεγχος επανέρχεται στην κλήση prt(0). Η συνάρτηση εξάγει την τιμή της i από την στοίβα και προχωράει στην εκτέλεση της τελευταίας εντολής της στην οποία εμφανίζει και πάλι το 0. Άρα, μέχρι εδώ έχουμε υπολογίσει μια έξοδο ως εξής: 0[Έξοδος από το μπλοκ της if, με i==0]0. Στο μπλοκ της if καλείται καταρχάς, η prt(1) η οποία λειτουργεί ακριβώς με τον ίδιο τρόπο, δηλ. το αποτέλεσμά της μπορεί να περιγραφεί ως 1[Έξοδος από το μπλοκ της if, με i==1]1 ενώ το συνολικό αποτέλεσμα μέχρι στιγμής είναι 01[Έξοδος από το μπλοκ της if, με i==1]01. Συνεχίζοντας με αυτόν τον τρόπο φτάνουμε στο τελικό αποτέλεσμα που είναι 01233210.

Το κλειδί εδώ είναι να προσέξουμε το εξής: Η prt(0) καλεί κάποια στιγμή την prt(1). Αυτό όμως δεν σημαίνει πως η prt(0) έχει ολοκληρωθεί. Όταν η αναδρομική κλήση θα ολοκληρωθεί, η prt(0) θα συνεχίσει από την επόμενη γραμμή κώδικα από όπου έγινε η κλήση prt(1).

Ας δούμε τώρα την αναδρομική συνάρτηση δύναμης με βάση πραγματικό και εκθέτη ακέραιο.

```
static double power(double b, int e) { //Κώδικας 9.6
    if (e > 0) {
        return power(b, e - 1) * b;
    }
    if (e < 0) {
        return 1 / power(b, -e);
    }
    return 1;
}
```

Κώδικας 9.6 *Αναδρομική υλοποίηση του υπολογισμού δύναμης*

Όταν έχουμε διαθέσιμο έναν αναδρομικό ορισμό είναι εύκολο να υλοποιήσουμε την αντίστοιχη αναδρομική συνάρτηση. Θυμηθείτε τον αναδρομικό ορισμό της δύναμης στην αρχή της ενότητας 9 και συγκρίνετέ τον με τον κώδικα 9.6. Θα διαπιστώσετε σημαντική ομοιότητα. Στην ουσία, πρόκειται για απλή μεταγραφή από την γλώσσα των μαθηματικών στην γλώσσα του προγραμματισμού.

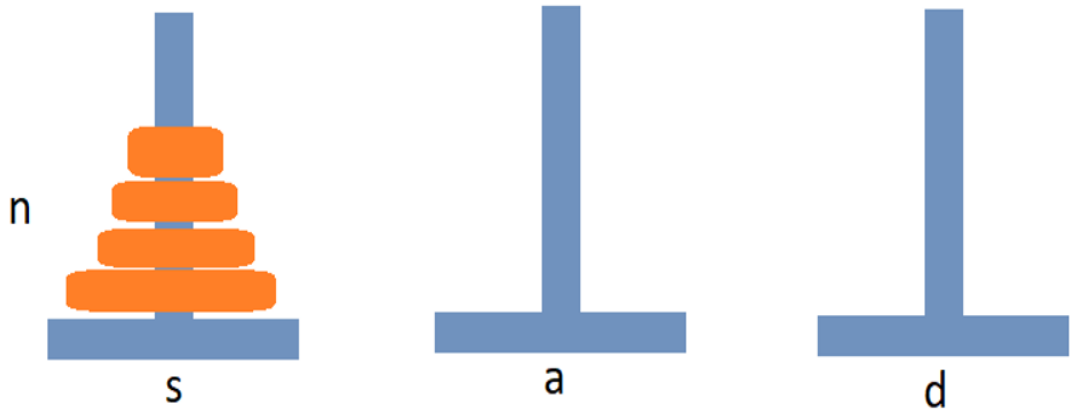
Η συνάρτηση power ελέγχει τον εκθέτη e, αν είναι θετικός επιστρέφει power(b, e-1)*b, δηλ. ακριβώς ότι λέει και ο μαθηματικός ορισμός. Αν πάλι ο εκθέτης είναι αρνητικός επιστρέφει 1/power(b,-e). Τέλος, εφόσον ο εκθέτης είναι 0, δεν θα γίνουν οι αναδρομικές κλήσεις που προβλέπονται για θετικό και αρνητικό εκθέτη οπότε η συνάρτηση θα επιστρέψει 1.

9.3 Οι πύργοι του Ανόι

Οι πύργοι του Ανόι (towers of Hanoi) είναι ένα μαθηματικό πρόβλημα που χρησιμοποιείται ευρύτατα για την εκμάθηση της αναδρομής στον προγραμματισμό. Η λύση του φαίνεται αρχικά αρκετά περίπλοκη σε σχέση με τις λύσεις των προβλημάτων που έχουμε δει ως τώρα. Ωστόσο, αν σκεφτεί κανείς με τον κατάλληλο τρόπο θα διαπιστώσει πως η αναδρομική λύση απαιτεί περιορισμένο κώδικα και τελικά αποδεικνύεται απλούστερη της μη αναδρομικής.

9.3.1 Περιγραφή του προβλήματος

Έχουμε τρεις στύλους, τον s από το source (προέλευση), τον d από το destination (προορισμός) και τον a από το auxiliary (βοηθητικός). Στον στύλο s είναι τοποθετημένοι n δίσκοι όπως φαίνεται στο σχήμα 9.4. Πιο συγκεκριμένα, οι δίσκοι έχουν διαφορετική διάμετρο και είναι τοποθετημένοι έτσι ώστε ένας δίσκος με μεγαλύτερη διάμετρο δεν βρίσκεται ποτέ επάνω από ένα δίσκο με μικρότερη διάμετρο.



Σχήμα 9.4 *Οι πύργοι του Ανόι*

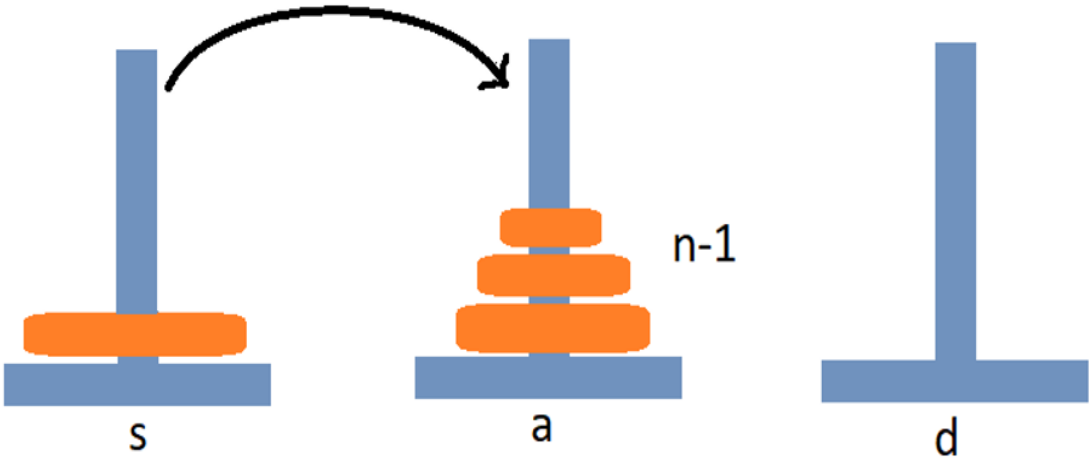
Το ζητούμενο είναι να μεταφέρουμε τους δίσκους από τον στύλο s στον στύλο d χρησιμοποιώντας ως βοηθητικό τον στύλο a. Η μετακίνηση όμως θα πρέπει να γίνει με τους εξής κανόνες:

- Κάθε φορά μπορούμε να μετακινήσουμε ένα μόνο δίσκο από έναν στύλο σε έναν άλλο.

- Δεν πρέπει ποτέ ένας δίσκος με μεγαλύτερη διάμετρο να τοποθετηθεί επάνω από ένα δίσκο με μικρότερη διάμετρο.

9.3.2 Λύση

Η απλούστερη οδός για να λύσουμε το πρόβλημα με αναδρομή είναι να σκεφτούμε με την φιλοσοφία της αναδρομής. Ας υποθέσουμε πως έχουμε κάποιο τρόπο για να μεταφέρουμε τους n-1 δίσκους από το s στο a χρησιμοποιώντας σαν βοηθητικό το d. Σε αυτήν την περίπτωση, στο s θα έχει μείνει ένας δίσκος και στο a θα έχουν τοποθετηθεί οι n-1 δίσκοι όπως δείχνει το σχήμα 9.5. Επομένως, ο δίσκος στο s μπορεί να μετακινηθεί στο d χωρίς πρόβλημα. Αν υποθέσουμε τώρα πως με τον ίδιο τρόπο που μεταφέραμε τους n-1 στύλους στο a, θα τους μεταφέρουμε στο d με βοηθητικό το s, τότε έχουμε λύσει το πρόβλημα. Επομένως, η μεταφορά n δίσκων από το s στο d με βοηθητικό το a έχει αναλυθεί σε δύο προβλήματα μεταφοράς n-1 δίσκων. Αν λύσουμε τα δύο αυτά προβλήματα, έχουμε λύση και στο αρχικό μας πρόβλημα.



Σχήμα 9.5 *Οι n-1 δίσκοι έχουν μετακινηθεί στον στύλο a χρησιμοποιώντας σαν βοηθητικό τον στύλο d.*

Πως θα λύσουμε τώρα αυτά τα δύο προβλήματα; Με τον ίδιο τρόπο. Στο πρώτο πρόβλημα, έχουμε να μεταφέρουμε n-1 δίσκους από το s στο a χρησιμοποιώντας σαν βοηθητικό το d. Αν μεταφέρουμε τους n-2 δίσκους από το s στο d χρησιμοποιώντας σαν βοηθητικό το a, στον s θα έχουν μείνει δύο στύλοι και ο a θα είναι άδειος, οπότε ο στύλος στην κορυφή του s μπορεί να μετακινηθεί χωρίς πρόβλημα στον a. Στην συνέχεια θα πρέπει να μεταφέρουμε τους n-2 από το d στο a. Επομένως έχουμε αναλύσει το ένα πρόβλημα μεταφοράς n-1 δίσκων σε δύο προβλήματα μεταφοράς n-2 δίσκων.

Η ανάλυση μπορεί να συνεχισθεί για n-1 βήματα. Στο πρώτο βήμα αναλύεται το πρόβλημα μεγέθους n σε δύο υποπροβλήματα μεγέθους n-1, στο δεύτερο βήμα αναλύονται τα δύο υποπροβλήματα μεγέθους n-1 σε τέσσερα υποπροβλήματα μεγέθους n-2. Η διαδικασία συνεχίζεται και στο βήμα n-1, έχει αναλυθεί το πρόβλημα σε 2n-1 υποπροβλήματα μεγέθους n-(n-1) το καθένα, δηλ. υποπροβλήματα μεγέθους 1. Κάθε πρόβλημα όμως μεγέθους 1, έχει προφανή λύση όπως έχουμε ήδη εξηγήσει. Επομένως, η λύση σε αυτό το επίπεδο, δίνει την δυνατότητα λύσης στο αμέσως προηγούμενο το οποίο με την σειρά του δίνει λύση στο προηγούμενό του επίπεδο. Μέσα από αυτήν την αναδρομική διαδικασία φτάνουμε και στην λύση του αρχικού προβλήματος.

```
public class TowersOfAnoi {

    private static void moveOne(int diskNo, char from, char to) {
            System.out.println("move disk " + diskNo + " from " + from + " to " +
to);
    }

    public static void moveDisks(int n, char s, char d, char a) {
        if (n == 1) {
            moveOne(n, s, d);
        } else {
```

```
            moveDisks(n - 1, s, a, d);
            moveOne(n, s, d);
            moveDisks(n - 1, a, d, s);
        }
    }

    public static void main(String[] args) {
        moveDisks(4,'S','D','A');
    }
}
```

Κώδικας 9.7 *Αναδρομική λύση του προβλήματος towers of Hanoi*

Όπως δείχνει ο κώδικας 9.7, έχουμε κωδικοποιήσει το πρόβλημα ως την συνάρτηση moveDisks που λαμβάνει τέσσερις παραμέτρους. Η παράμετρος n αντιπροσωπεύει τον αριθμό των δίσκων και οι υπόλοιπες τρεις παράμετροι τους ομώνυμους στύλους s, d και a. Με αυτήν την αναπαράσταση μπορούμε να θεωρήσουμε ότι κάθε δίσκος έχει μία ταυτότητα. Ο δίσκος που βρίσκεται πάνω-πάνω στην στοίβα των δίσκων έχει την ταυτότητα 1, ο αμέσως επόμενος την ταυτότητα 2. Έτσι χαρακτηρίζονται όλοι οι δίσκοι μέχρι τον δίσκο στην βάση της στοίβας που είναι ο δίσκος n.

Παράλληλα, έχουμε και την συνάρτηση moveOne. Πρόκειται για μια απλή συνάρτηση που το μόνο που κάνει είναι να εμφανίζει ένα μήνυμα που μας πληροφορεί ποιος δίσκος μετακινήθηκε και από ποιον στύλο έφυγε και σε ποιον τοποθετήθηκε.

Κατά τα λοιπά, η moveDisks εφαρμόζει ακριβώς την λογική που περιγράψαμε. Πιο συγκεκριμένα, αν κληθεί με n=1, τότε η μετακίνηση του δίσκου είναι μια απλή μετακίνηση οπότε καλεί την moveOne. Διαφορετικά, μεταφέρει τους n-1 δίσκους από το s στο a με βοηθητικό το d. Μετά μεταφέρει τον μοναδικό δίσκο στο s και τον τοποθετεί στο d και τέλος μεταφέρει τους n-1 από το a όπου τους έχει τοποθετήσει προηγουμένως στο d χρησιμοποιώντας σαν βοηθητικό το s.

9.4 Αμοιβαία αναδρομή

Η αναδρομή κατά την οποία μια συνάρτηση καλεί τον εαυτό της ονομάζεται ειδικότερα άμεση αναδρομή (direct recursion). Είναι όμως πιθανό μια συνάρτηση a να καλεί μια συνάρτηση b και η b να καλεί την a. Πρόκειται και πάλι για αναδρομή που όμως ονομάζεται αμοιβαία αναδρομή (mutual) ή έμμεση (indirect) αναδρομή.

Ας δούμε ένα πρόβλημα που μπορεί να επιλυθεί με αμοιβαία αναδρομή. Έστω ότι έχουμε μια διαδικασία εξέτασης και μια διαδικασία αξιολόγησης της εξέτασης. Η εξέταση βαθμολογεί ένα γραπτό και καλεί την αξιολόγηση. Η δε αξιολόγηση ελέγχει τον βαθμό και αν είναι προβιβάσιμος ενημερώνει με κατάλληλο μήνυμα. Αν όχι, πάλι ενημερώνει με κατάλληλο μήνυμα αλλά καλεί επαναληπτική εξέταση. Η διαδικασία συνεχίζεται μέχρις ότου επιτευχθεί προβιβάσιμος βαθμός.

Ο κώδικας 9.8 δίνει λύση σε αυτό το πρόβλημα με αμοιβαία αναδρομή;

```
    static void exam() { //Κώδικας 9.8
        Random generator = new Random();
        int mark = generator.nextInt(10) + 1;
        evaluate(mark);
    }

    static void evaluate(int mark) { //Κώδικας 9.8
        System.out.print("Your mark is: " + mark);
        if (mark >= 7) {
            System.out.println(" Success");
        } else {
            System.out.println(" Fail");
            exam();
        }
    }
```

Κώδικας 9.8 *Εξέταση και αξιολόγηση με αμοιβαία αναδρομή*

Η συνάρτηση exam παράγει έναν τυχαίο ακέραιο από 1 έως 10. Ο αριθμός αυτός υποτίθεται πως αναπαριστά τον βαθμό ενός γραπτού. Στην συνέχεια καλεί την συνάρτηση evaluate με παράμετρο τον βαθμό. Η evaluate με την σειρά της ανακοινώνει τον βαθμό και ανάλογα με την τιμή του ενημερώνει πως έχουμε επιτυχία ή αποτυχία. Στην περίπτωση της αποτυχίας καλεί και πάλι την exam.
Παρόμοια με την συνάρτηση fibonacci που ορίζεται αναδρομικά, οι συναρτήσεις Hofstadter male και female [2] ορίζονται με αμοιβαία αναδρομή. Πρόκειται για δύο σειρές ακεραίων που δίνονται από τον ακόλουθο ορισμό.

$$f(0) = 1$$
$$m(0) = 0$$
$$f(n) = n - m\big(f(n-1)\big), \qquad n > 0$$
$$m(n) = n - f\big(m(n-1)\big), \qquad n > 0$$

Ο κώδικας 9.9 δίνει την υλοποίηση σε Java των συναρτήσεων f και m.

```
static int f(int n) {
    if (n == 0) {
        return 1;
    }
    return n - m(f(n - 1));
}

static int m(int n) {
    if (n == 0) {
        return 0;
    }
    return n - f(m(n - 1));
}
```

Κώδικας 9.9 *Υπολογισμός σειρών Hofstadter male και female*

Παρότι η υλοποίηση των συναρτήσεων με δεδομένο τον αμοιβαία αναδρομικό ορισμό τους είναι σχετικά εύκολη, η παρακολούθηση των κλήσεων και των επιστροφών είναι αρκετά περίπλοκη. Στον πίνακα 9.1, παραθέτουμε τις κλήσεις και επιστροφές που προκαλούνται προκειμένου να υπολογισθούν οι m(2) και f(2).

1	m(2)	f(2)
2	m(1)	f(1)
3	m(0)	f(0)
4	m(0)->0	f(0)->1
5	f(0)	m(1)
6	f(0)->1	m(0)
7	m(1)->0	m(0)->0
8	f(0)	f(0)
9	f(0)->1	f(0)->1
10	m(2)->1	m(1)->0
		f(1)->1
		m(1)
		m(0)
		m(0)->0
		f(0)
		f(0)->1
		m(1)->0
		f(2)->2

Πίνακας 9.2 *Κλήσεις και επιστροφές για τον υπολογισμό των m(2) και f(2)*

Στον πίνακα 9.2, ένα στοιχείο που δεν περιέχει τον χαρακτήρα -> επισημαίνει την είσοδο στην αντίστοιχη συνάρτηση. Αντίθετα τα στοιχεία που περιέχουν -> επισημαίνουν την επιστροφή της συνάρτησης. Ας μελετήσουμε την κλήση m(2).

Προκειμένου να υπολογισθεί η m(2) καλείται η f(m(1)). Κατά την κλήση της f(m(1)) πριν ο κώδικάς της αρχίσει να εκτελείται, θα πρέπει να υπολογισθεί η παράμετρός της, δηλ. η m(1). Για αυτόν τον λόγο στον πίνακα 9.2 στην δεύτερη γραμμή βλέπουμε κλήση της m(1). Η m(1) υπολογίζεται ως f(m(0)). Κατά την κλήση της f(m(0)) πριν ο κώδικάς της αρχίσει να εκτελείται, θα πρέπει να υπολογισθεί η παράμετρός της, δηλ. η m(0). Για αυτόν τον λόγο στον πίνακα 9.2 στην τρίτη γραμμή βλέπουμε κλήση της m(0). Η m(0) όμως είναι βασική περίπτωση και υπολογίζεται χωρίς περαιτέρω αναδρομική κλήση. Έτσι στην γραμμή 4 βλέπουμε πως η επιστροφή της m(0) είναι 0. Επομένως, η κλήση f(m(0)) έχει υπολογίσει την παράμετρό της και μπορεί να προχωρήσει στην εκτέλεση του σώματος της συνάρτησης. Έτσι στην γραμμή 5 βλέπουμε την f(0) η οποία επιστρέφει χωρίς περαιτέρω αναδρομική κλήση όπως φαίνεται στην γραμμή 6. Τώρα είναι εφικτός ο υπολογισμός της m(1) όπως φαίνεται στην γραμμή 7. Θυμηθείτε πως η m(2) που είναι το τελικό ζητούμενο είναι ίση με f(m(1)), εφόσον m(1)=0, καλείται η f(0) όπως φαίνεται στην γραμμή 8 που επιστρέφει 1 (γραμμή 9). Τέλος, η m(2) είναι ίση με 1. Παρόμοια, μπορείτε να αναλύσετε την κλήση f(2).

9.5 Πλεονεκτήματα και μειονεκτήματα

Πολλά προβλήματα μπορούν να λυθούν με αναδρομή ευκολότερα με πολύ μικρότερο και πιο κομψό κώδικα από ότι με μη αναδρομικές προσεγγίσεις. Για παράδειγμα, προσπαθήστε να κωδικοποιήσετε τους πύργους του Ανόι μη αναδρομικά και θα διαπιστώσετε πως πρόκειται για πρόβλημα σημαντικής δυσκολίας.

Επιπλέον, πολλά προβλήματα έχουν αναδρομικό ορισμό και επομένως η αναδρομική υλοποίησή τους είναι προφανής.

Από την άλλη πλευρά, η αναδρομή δεν είναι χωρίς μειονεκτήματα. Το σοβαρότερο από όλα είναι η πιθανή εξάντληση της στοίβας. Επομένως, θα πρέπει να χρησιμοποιείτε αναδρομή μόνο εκεί που γνωρίζετε πως τα μεγέθη των προβλημάτων είναι αδύνατο να εξαντλήσουν το μέγεθος της στοίβας. Οι προδιαγραφές της Java επιτρέπουν στοίβα σταθερού μεγέθους ή προσαρμοζόμενου. Εξαρτάται από την υλοποίηση της Java αν υποστηρίζει στοίβα σταθερού ή μεταβλητού μεγέθους. Στην πρώτη περίπτωση, η εξάντληση της στοίβας προκαλεί παραγωγή λάθους τύπου υπερχείλιση της στοίβας (stack overflow) και στην δεύτερη περίπτωση παραγωγή λάθους τύπου υπερχείλιση της μνήμης (memory overflow). Και στις δύο περιπτώσεις, το αποτέλεσμα για την εφαρμογή είναι το ίδιο, δηλ. ο ανορθόδοξος τερματισμός της.

Επίσης, η αναδρομή σε αρκετές περιπτώσεις μπορεί να δίνει αποτελέσματα σε μεγαλύτερο χρονικό διάστημα από την μη αναδρομική λύση. Αν παρατηρήσετε το δένδρο των κλήσεων για τον υπολογισμό του fibonacci(4) στο σχήμα 9.3 θα διαπιστώσετε ότι κάποιες κλήσεις επαναλαμβάνονται. Για παράδειγμα, η τιμή f(2) υπολογίζεται 2 φορές. Κατά τον υπολογισμό του εικοστού όρου της ακολουθίας, δηλ. του f(20), η f(2) καλείται 4181 φορές. Επομένως σε αυτήν την περίπτωση, 4181 φορές επαναλαμβάνεται ο ίδιος υπολογισμός. Ωστόσο αυτό το πρόβλημα έρχεται να λύσει μια ειδική αναδρομική τεχνική γνωστή ως Απομνημόνευση που συζητάμε στην ενότητα 18.

Σε πολλές περιπτώσεις, οι εταιρείες παραγωγής λογισμικού υλοποιούν την λύση ενός προβλήματος αρχικά αναδρομικά ώστε να βγουν εγκαίρως στην αγορά και στην συνέχεια αντικαθιστούν την λύση με μη αναδρομική έκδοση. Σε κάθε περίπτωση, μια αναδρομική υλοποίηση μπορεί πάντα να μεταγραφεί με χρήση αποκλειστικά μη αναδρομικών τεχνικών.

Συμπερασματικά, χρησιμοποιήστε αναδρομή όπου σας διευκολύνει και εφόσον σας το επιτρέπει το μέγεθος του προβλήματος που έχετε να λύσετε.

9.6 Λυμένες Ασκήσεις

Σε αυτήν την ενότητα παρουσιάζουμε μια σειρά από παραδείγματα αναδρομικών συναρτήσεων ώστε να βοηθήσουμε τον αναγνώστη να εξοικειωθεί τόσο με την υλοποίηση όσο και με την χρήση τους.

9.6.1 Παραγοντικό

Να αναπτυχθεί αναδρομική συνάρτηση που επιστρέφει το παραγοντικό της παραμέτρου της.

Λύση
Το παραγοντικό (factorial) ενός ακεραίου n είναι το γινόμενο όλων των θετικών ακεραίων που είναι μικρότεροι ή ίσοι του n. Το παραγοντικό του n συμβολίζεται σαν n!. Επομένως

$$n! = \begin{Bmatrix} 1 \times 2 \times 3 \ldots \times n, για\ n > 0 \\ 1, για\ n = 0 \end{Bmatrix}$$

Εκτός όμως από αυτόν τον ορισμό του παραγοντικού υπάρχει και ένας άλλος, αναδρομικός, ορισμός.

$$n! = \begin{Bmatrix} 1, για\ n = 0\ και\ n = 1 \\ n * (n-1)!, για\ n > 1 \end{Bmatrix}$$

Έχοντας τον αναδρομικό ορισμό είναι εύκολο να υλοποιήσουμε την αντίστοιχη συνάρτηση.

```
static int factorial(int n) { //Κώδικας 9.10
    if (n == 0 || n == 1) {
        return 1;
    } else {
        return factorial(n - 1) * n;
    }
}
```

Κώδικας 9.10 *Υλοποίηση υπολογισμού του παραγοντικού με αναδρομή*

9.6.2 Ελάχιστο Κοινό Πολλαπλάσιο

Να αναπτυχθεί συνάρτηση αναδρομικού υπολογισμού του Ελάχιστου Κοινού Πολλαπλάσιου μιας σειράς ακεραίων. Στην main να αναπτυχθεί βασικός κώδικας ελέγχου.

Λύση
Το Ελάχιστο Κοινό Πολλαπλάσιο (ΕΚΠ) μιας σειράς ακεραίων, όπως φανερώνει το όνομά του, είναι το μικρότερο από τα κοινά πολλαπλάσια της σειράς. Υπάρχουν διάφοροι τρόποι υπολογισμού του. Ο τρόπος που χρησιμοποιούμε εδώ βασίζεται στο ακόλουθο σκεπτικό. Ο πρώτος υποψήφιος ακέραιος που μπορεί να είναι το ΕΚΠ της σειράς ακεραίων είναι ο μεγαλύτερος ακέραιος της σειράς. Αν ο ακέραιος αυτός δεν είναι το ΕΚΠ, τότε αναγκαστικά, ΕΚΠ θα είναι κάποιο πολλαπλάσιό του.

```
public class ElaxistoKoinoPollaplasio {

    private static int max(int[] nums) {
        int rVal = nums[0];
        for (int n : nums) {
            if (n > rVal) {
                rVal = n;
            }
        }
        return rVal;
    }

    private static boolean divideAll(int[] nums, int max) {
        for (int n : nums) {
            if (max % n != 0) {
                return false;
            }
        }
        return true;
    }
```

```
    public static int eKP(int... nums) { //variable length list of parameters
        return eKPR(1, nums);
    }

    private static int eKPR(int factor, int... nums) {
        int max = max(nums) * factor;
        if (divideAll(nums, max)) {
            return max;
        } else {
            return eKPR(++factor, nums);
        }

    }

    public static void main(String[] args) {
        System.out.println(eKP(24, 90, 60, 4, 89));
    }
}
```

Κώδικας 9.11 *Αναδρομικός υπολογισμός του ΕΚΠ σειράς ακεραίων*

Στον κώδικα 9.11, η αναδρομική συνάρτηση υπολογισμού του ΕΚΠ είναι η eKPR. Αυτή λαμβάνει ως παραμέτρους τον ακέραιο factor και μια σειρά ακεραίων. Επομένως, η δεύτερη παράμετρός της είναι μεταβλητού μήκους. Στην πρώτη γραμμή της συνάρτησης υπολογίζεται ο μεγαλύτερος ακέραιος της σειράς που είναι ο πρώτος υποψήφιος ακέραιος που πιθανώς είναι το ΕΚΠ. Ο μεγαλύτερος ακέραιος πολλαπλασιάζεται με την παράμετρο factor. Η παράμετρος factor διευκολύνει την αναδρομική κλήση. Κατά την αρχική κλήση της συνάρτησης, η factor πρέπει να έχει την τιμή 1, διαφορετικά ο υπολογισμός του ΕΚΠ πιθανότατα θα καταλήξει σε λάθος. Επομένως, κατά την πρώτη κλήση, η μεταβλητή max έχει την τιμή του μεγαλύτερου ακέραιου της σειράς. Προσέξτε πως η συνάρτηση max δεν επιστρέφει την θέση του μεγαλύτερου ακεραίου αλλά την τιμή του. Στην συνέχεια, ελέγχουμε αν ο max διαιρείται ακριβώς με όλους τους ακεραίους της σειράς. Αυτό επιτυγχάνεται με κλήση της divideAll. Αν η divideAll επιστρέψει true, το ΕΚΠ έχει βρεθεί και επιστρέφεται. Αν όμως επιστρέψει false, ο κώδικας προχωράει σε αναδρομική κλήση αυξάνοντας την factor κατά 1 ώστε να ελεγχθεί το πρώτο πολλαπλάσιο του μεγαλύτερου της σειράς. Στην επόμενη αναδρομική κλήση θα ελεγχθεί το επόμενο πολλαπλάσιο έως ότου βρεθεί ακέραιος που διαιρείται ακριβώς με όλους τους αριθμούς της σειράς εισόδου.

Ίσως να προσέξατε πως όλες οι συναρτήσεις στις οποίες αναφερθήκαμε είναι private, όπως και eKPR. Πράγματι, προκειμένου να κάνουμε την eKPR αναδρομική έχουμε προσθέσει την παράμετρο factor. Ο χρήστης της συνάρτησής μας όταν θέλει να υπολογίσει το ΕΚΠ μιας σειράς ακεραίων θα πρέπει να δίνει μόνο την σειρά ακεραίων. Δεν οφείλει να γνωρίζει πως η συνάρτηση είναι αναδρομική και πως κατά την κλήση της πρέπει να της περάσουμε και μια επιπλέον παράμετρο, ούτε τον ενδιαφέρει ο ρόλος αυτής της παραμέτρου. Σε αυτές τις περιπτώσεις κάνουμε την αναδρομική συνάρτηση ιδιωτική και παρέχουμε μια δημόσια συνάρτηση περικάλυμμα (wrapper). Στην περίπτωσή μας η συνάρτηση περιτύλιγμα είναι η eKP η οποία το μόνο που κάνει είναι να καλεί την eKPR με factor ίσο με 1.

9.6.3 Πρώτοι αριθμοί

Να αναπτυχθεί αναδρομική συνάρτηση που εξετάζει αν η παράμετρός της είναι πρώτος αριθμός. Στην main να αναπτυχθεί βασικός κώδικας ελέγχου.

Λύση

Πρώτος είναι ένας ακέραιος που διαιρείται ακριβώς μόνο με τον εαυτό του και το 1. Επιπλέον εξορισμού το 1 και το 2 δεν είναι πρώτοι. Επομένως, για να ελέγξουμε αν ένας ακέραιος, n, είναι πρώτος θα πρέπει να ελέγξουμε αν οποιοσδήποτε ακέραιος από το 2 έως n/2 διαιρεί ακριβώς τον n. Αν βρεθεί τέτοιος αριθμός, ο n δεν είναι πρώτος. Είναι προφανές πως κανένας ακέραιος μεγαλύτερος από n/2 δεν μπορεί να διαιρεί ακριβώς τον n.

```
public class PrimeNumber {
```

```
    private static boolean isPrime(int num, int divider) {
        if (num <= 2) {
            return false;
        }
        if (num % divider == 0) {
            return false;
        } else if (divider > num / 2) {
            return true;
        } else {
            return isPrime(num, divider + 1);
        }
    }

    public static boolean isPrime(int num) {
        return isPrime(num, 2);
    }

    public static void main(String[] args) {
        for (int i = 0; i < 100; i++) {
            System.out.println(i + " " + isPrime(i));
        }
    }
}
```

Κώδικας 9.12 *Αναδρομική αξιολόγηση πρώτου αριθμού*

Καταρχάς, στον κώδικα 9.12, στην συνάρτηση isPrime(int, int), ελέγχουμε αν ο υπό εξέταση αριθμός είναι μικρότερος ή ίσος από το 2 οπότε επιστρέφουμε false. Στην συνέχεια προκύπτουν δύο βασικές περιπτώσεις. Στην πρώτη περίπτωση, ο num διαιρείται ακριβώς με τον divider οπότε ο num δεν είναι πρώτος και επιστρέφουμε false. Στην δεύτερη βασική περίπτωση, ελέγχουμε αν ο divider υπερβαίνει το num/2. Σε αυτήν την περίπτωση έχουν ελεγχθεί όλοι οι ακέραιοι από 2 έως num/2 και κανένας δεν βρέθηκε να διαιρεί ακριβώς τον num, επομένως ο num είναι πρώτος και επιστρέφουμε true. Αν καμία από τις βασικές περιπτώσεις δεν προκαλέσει επιστροφή της isPrime, τότε προχωρούμε στην αναδρομική κλήση αυξάνοντας τον divider κατά 1.

Παρόμοια με την προηγούμενη άσκηση, ο divider ως παράμετρος εξυπηρετεί στην αναδρομική κλήση. Ο χρήστης της συνάρτησής μας δεν χρειάζεται να γνωρίζει καν την ύπαρξή του. Έτσι έχουμε κάνει την isPrime(int, int) private και παρέχουμε κατάλληλο περιτύλιγμα, την isPrime(int) που είναι public.

9.6.4 Selection Sort

Να αναπτυχθεί αναδρομική συνάρτηση ταξινόμησης μονοδιάστατου πίνακα ακεραίων με βάση τον αλγόριθμο selection sort (άσκηση 6.5.1). Να κατασκευαστεί κατάλληλος κώδικας ελέγχου με χρήση της Junit.

Λύση

Την λογική του selection sort έχουμε ήδη παρουσιάσει στην άσκηση 6.5.1. Στον κώδικα 9.13 δίνουμε την αναδρομική και δομημένη υλοποίησή της.

```
import java.util.Arrays;

public class SelectionSort {//Κώδικας 9.13

    private static void swap(int[] array, int idx1, int idx2) {
        int i = array[idx1];
        array[idx1] = array[idx2];
        array[idx2] = i;
    }

    private static int idxOfMin(int[] array, int startIdx) {
        int min = array[startIdx];
        int rVal = startIdx;
```

```
        for (int i = startIdx + 1; i < array.length; i++) {
            if (array[i] < min) {
                min = array[i];
                rVal = i;
            }
        }
        return rVal;
    }

    private static void sort(int[] array, int startIdx) {
        int posOfMin = idxOfMin(array, startIdx);
        if (posOfMin != startIdx) {
            swap(array, posOfMin, startIdx);
        }
        startIdx++;
        if (startIdx < array.length - 1) {
            sort(array, startIdx);
        }
    }

    public static void sort(int[] array) {
        sort(array, 0);
    }

    public static void main(String[] args) {
        int[] tbl = {2, 1, 8, 6, 5, 4};
        sort(tbl);
        System.out.println(Arrays.toString(tbl));
    }
}
```

Κώδικας 9.13 *Αναδρομική υλοποίηση της selection sort*

Στον κώδικα 9.13, η αναδρομική υλοποίηση της selection sort δίνεται από την συνάρτηση sort(int[], int). Η δεύτερη παράμετρος προκύπτει για διευκόλυνση της αναδρομικής κλήσης. Παρόμοια με τις δύο προηγούμενες ασκήσεις παρέχεται μια συνάρτηση-περιτύλιγμα, η sort(int[]). Οι υπόλοιπες συναρτήσεις είναι γνωστές καθώς έχουν ήδη συζητηθεί.

Ας δούμε λοιπόν, την sort(int[], int). Η παράμετρος startIdx καθορίζει την θέση του πίνακα από την οποία και μετά, ο πίνακας είναι αταξινόμητος. Αρχικά, η συνάρτηση εντοπίζει την θέση του μικρότερου ακέραιου στον υπό ταξινόμηση τμήμα του πίνακα, δηλ. από το startIdx και μετά. Αν το μικρότερο στοιχείο βρίσκεται μετά την θέση startIdx, τότε ανταλλάσσει το μικρότερο στοιχείο με το στοιχείο στην θέση startIdx. Στην συνέχεια, αυξάνει το startIdx κατά 1 έτσι ώστε να συνεχίσει η ταξινόμηση από την επόμενη θέση στον πίνακα. Αν το startIdx δείχνει πριν την τελευταία θέση, σημαίνει ότι υπολείπεται τμήμα του πίνακα προς ταξινόμηση οπότε γίνεται η αναδρομική κλήση, διαφορετικά τερματίζεται η αναδρομική κλήση και ο πίνακας είναι ταξινομημένος.

Στην συνέχεια παρουσιάζουμε τον κώδικα ελέγχου.

```
public class SelectionSortTest {

    public SelectionSortTest() {
    }

    /**
     * Test of sort method, of class SelectionSort.
     */
    @Test
    public void testSort() {
        System.out.println("sort");
        Random gen = new Random();
        for (int i = 0; i < 100; i++) {
```

```
            int[] array = new int[gen.nextInt(100) + 10];
            for (int j = 0; j < array.length; j++) {
                array[j] = gen.nextInt();
            }
            SelectionSort.sort(array);
            for (int j = 0; j < array.length - 1; j++) {
                if (array[j + 1] < array[j]) {
                    fail();
                }
            }
        }
    }

}
```

Κώδικας 9.14 *Κώδικας ελέγχου της SelectionSort.sort*

Στον κώδικα 9.14 κατασκευάζουμε 100 πίνακες με μέγεθος μεταξύ 10 και 110. Στην συνέχεια εκχωρούμε σε κάθε θέση ενός πίνακα έναν τυχαίο ακέραιο. Σημειώστε πως η nextInt χωρίς παραμέτρους μπορεί να επιστρέψει οποιαδήποτε ακέραιη τιμή. Αφού γεμίσουμε τον πίνακα με τυχαίες τιμές, προχωρούμε στην ταξινόμησή του. Τέλος ελέγχουμε αν ο πίνακας είναι σωστά ταξινομημένος. Σε περίπτωση που εντοπίσουμε μικρότερο στοιχείο του πίνακα να έπεται μεγαλύτερου, καλούμε την fail και σημειώνουμε το λάθος.

9.6.5 Δυαδική αναζήτηση

Να αναπτυχθεί αναδρομική συνάρτηση δυαδικής αναζήτησης σε μονοδιάστατο πίνακα ακεραίων. Να κατασκευαστεί κατάλληλος κώδικας ελέγχου με χρήση της Junit.

Λύση
Τον αλγόριθμο δυαδικής αναζήτησης τον έχουμε ήδη συζητήσει και έχουμε δώσει και την μη αναδρομική υλοποίησή του (ενότητα 6.5.2). Έτσι, εδώ θα συζητήσουμε αποκλειστικά την αναδρομική υλοποίησή του.

```
public class BinarySearch {

    private static int binarySearch(int[] tbl, int schElement, int from, int to)
{
        if (from <= to) {
            int mid = from + (to - from) / 2;
            if (tbl[mid] == schElement) {
                return mid;
            }
            if (schElement < tbl[mid]) {
                return binarySearch(tbl, schElement, from, mid - 1);
            }
            return binarySearch(tbl, schElement, mid + 1, to);
        }
        return -1;
    }

    public static int binarySearch(int[] tbl, int schElement) {
        return binarySearch(tbl, schElement, 0, tbl.length - 1);
    }

}
```

Κώδικας 9.15 *Αναδρομική δυαδική αναζήτηση*

Στον κώδικα 9.15, η binarySearch(int[], int, int, int) είναι η αναδρομική συνάρτηση. Οι παράμετροι from και to έχουν προστεθεί για να διευκολύνουν την αναδρομή. Η συνάρτηση-περικάλυμμα καλεί την αναδρομική με παραμέτρους 0 και tbl.length-1 ώστε η αναζήτηση να αφορά καταρχάς όλον τον πίνακα. Μέσα στην

αναδρομική συνάρτηση γίνεται έλεγχος εάν το from είναι μικρότερο από το to, δηλ. αν υπάρχει τμήμα του πίνακα που δεν έχει ελεγχθεί. Αν υπάρχει τέτοιο τμήμα, ενημερώνεται κατάλληλα η mid που δείχνει στο μέσον του υπό αναζήτηση τμήματος. Στην συνέχεια ελέγχεται αν το μέσον έχει την τιμή αναζήτησης. Αν ναι, η συνάρτηση επιστρέφει το μέσον ως την θέση του στοιχείου που αναζητούμε. Αν όχι, η συνάρτηση προχωράει σε αναδρομική κλήση. Αν η τιμή του στοιχείου αναζήτησης είναι μικρότερη από την τιμή στο μέσον, η αναδρομική κλήση αφορά το πρώτο μισό του υπό εξέταση τμήματος διαφορετικά αφορά το δεύτερο μισό.
Στην συνέχεια δίνουμε τον κώδικα ελέγχου.

```
import java.util.Random;
import org.junit.Test;
import static org.junit.Assert.*;

/**
 *
 * @author Lefteris Moussiades
 */
public class BinarySearchTest {

     /**
      * Test of binarySearch method, of class BinarySearch.
      */
     @Test
     public void testBinarySearch() {
         System.out.println("binarySearch");
         Random gen=new Random();
         for (int i=0; i<100; i++) {
             int[] array=new int[gen.nextInt(100)+10];
             for (int j=0; j<array.length; j++) {
                 array[j]=gen.nextInt();
             }
             SelectionSort.sort(array);
             int idx=gen.nextInt(array.length);
             idx=idx==array.length?array.length-1:idx;
             int schElement=array[idx];
             int result = BinarySearch.binarySearch(array, schElement);
             assertTrue(array[result]==schElement);
         }
     }
}
```

Κώδικας 9.16 *Έλεγχος της binarySearch*

Στον κώδικα 9.16 κατασκευάζουμε 100 πίνακες ακεραίων με τυχαίο μήκος από 10 έως 110 και τους γεμίζουμε με τυχαίες ακέραιες τιμές. Στην συνέχεια, ταξινομούμε κάθε πίνακα και επιλέγουμε την τιμή ενός τυχαίου στοιχείου του. Τέλος καλούμε την binarySearch ώστε να αναζητήσουμε το στοιχείο που γνωρίζουμε ότι όντως υπάρχει στον πίνακα.

9.7 Ασκήσεις προς λύση

1. Να αναπτυχθεί συνάρτηση που υπολογίζει αναδρομικά τον Μέγιστο Κοινό Διαιρέτη σειράς ακεραίων. Να αναπτυχθεί κατάλληλος κώδικας ελέγχου.

2. Να αναπτυχθεί αναδρομική συνάρτηση static void prtSquareRoots(int i, int n) που τυπώνει την σειρά των τετραγώνων των αριθμών από i έως n.

3. Να αναπτυχθεί αναδρομική συνάρτηση static void prtSquareRoots(int i, int n) που τυπώνει την σειρά των τετραγώνων των αριθμών από i έως n, στην ίδια γραμμή, χωρισμένα με κόμμα και διάστημα. Μετά την εκτύπωση της σειράς, η συνάρτηση να προετοιμάζει ώστε η επόμενη έξοδος να βγαίνει στην επόμενη γραμμή. Για παράδειγμα, τα τετράγωνα των αριθμών από 0 έως 9 να τυπωθούν ως εξής:

```
0, 1, 4, 9, 16, 25, 36, 49, 64, 81
```

4. Να αναπτύξετε αναδρομική συνάρτηση υπολογισμού του αθροίσματος

$$\sum_{i=1}^{n} \frac{1}{i}$$

5. Πόσες αναδρομικές κλήσεις προκαλεί η κλήση fibonacci(9);

   Υπόδειξη: Τοποθετήστε την αναδρομική fibonacci σε μία κλάση. Ορίστε στην κλάση μια ακέραιη στατική μεταβλητή cnt. Αυξήστε την cnt κατάλληλα μέσα στην fibonacci. Εμφανίστε το αποτέλεσμα.

6. Να αναπτυχθεί αναδρομική συνάρτηση που επιστρέφει την μέγιστη τιμή ενός πίνακα ακεραίων.

7. Να αναπτυχθεί αναδρομική συνάρτηση που μετράει τους πεζούς χαρακτήρες σε ένα πίνακα χαρακτήρων.

8. Πόσες αναδρομικές κλήσεις απαιτεί η λύση των πύργων του Ανόι για 10 δίσκους;

9. Να αναπτυχθεί αναδρομική συνάρτηση που εμφανίζει όλες τις αναδιατάξεις (permutations) μιας συμβολοσειράς. Για παράδειγμα, έστω η συμβολοσειρά can, οι δυνατές αναδιατάξεις είναι can, cna, acn, anc, nca, nac.

10. Να αναπτυχθεί αναδρομική συνάρτηση που λαμβάνει ως παραμέτρους δύο συμβολοσειρές και εξετάζει αν η δεύτερη περιέχεται στην πρώτη.

11. Ένας τρόπος για τον υπολογισμό της τετραγωνικής ρίζας ενός πραγματικού αριθμού, x>0, έχει ως εξής: Έστω y μια τυχαία λύση. Αν το y υψωμένο στο τετράγωνο προσεγγίζει ικανοποιητικά το x, τότε η τετραγωνική ρίζα του x είναι το y, διαφορετικά αντικαθιστούμε στην θέση του y, το (y+x/y)/2. Υλοποιήστε αναδρομική συνάρτηση υπολογισμού της τετραγωνικής ρίζας πραγματικού αριθμού.

Βιβλιογραφία

[1] “Chapter 2. The Structure of the Java Virtual Machine.” https://docs.oracle.com/javase/specs/jvms/se8/html/jvms-2.html#jvms-2.5 (accessed Oct. 15, 2021).

[2] E. W. Weisstein, “Hofstadter Male-Female Sequences.” https://mathworld.wolfram.com/HofstadterMale-FemaleSequences.html (accessed Sep. 14, 2021).

Κριτήρια αξιολόγησης

Στατικές και τοπικές μεταβλητές

Έστω ο κώδικας

```
public class KA01 {

     static int i = 2;

     public static void main(String[] args) {
          for (int i = 5; i < 10; i++) {
               i++;
          }
          System.out.println(i--);
     }
}
```

Ποια είναι η έξοδός του;

```
A. 56789
B. 5678910
Γ. 2
Δ. 1
Ε. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Οι απαντήσεις Α και Β θα μπορούσαν να απασχολήσουν τον αναγνώστη, εφόσον μέσα στην επαναληπτική διαδικασία τυπώναμε το i. Εδώ το i απλώς αυξάνεται χωρίς να εμφανίζεται. Επιπλέον, η στατική μεταβλητή i είναι διαφορετική από την τοπική i της οποίας η εμβέλεια περιορίζεται στην for. Μεταβολές στην τοπική i δεν επηρεάζουν την στατική i. Η απάντηση Δ είναι επίσης λανθασμένη. Παρότι η στατική i στέλνεται στην έξοδο συνοδευόμενη από τελεστή προσαύξησης, η προσαύξηση γίνεται αλλά το i-- αξιολογείται στην τιμή πριν την προσαύξηση καθώς χρησιμοποιείται ο μεταθεματικός τελεστής. Ο κώδικας δεν περιέχει λάθος μεταγλώττισης, επομένως και η απάντηση Ε είναι λανθασμένη. Τέλος, η σωστή απάντηση είναι η Γ.

Εμβέλεια τοπικών μεταβλητών

Έστω ο κώδικας

```
public class KA02 {

     public static void main(String[] args) {
          int i = 1;
          do {
               int j = i++;
               System.out.print(j);
          } while (j < 5);
     }
}
```

Ποια είναι η έξοδός του;

```
A. 1234
B. 2345
```

```
Γ. 12345
Δ. Λάθος χρόνου εκτέλεσης
Ε. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Το j είναι τοπική μεταβλητή στο μπλοκ της do. Επομένως η εμβέλειά του περιορίζεται σε αυτό το μπλοκ. Η συνθήκη ελέγχου στην while βρίσκεται εκτός του μπλοκ της do και αναφέρεται στο j. Το j όμως σε εκείνη την περιοχή του κώδικα είναι αθέατο στον μεταγλωττιστή ο οποίος παράγει κατάλληλο μήνυμα λάθους. Επομένως, η σωστή απάντηση είναι η Δ.

Μεταβλητή ελέγχου της for

Έστω ο κώδικας

```
public class KA03 {

     public static void main(String[] args) {
          int i = 5;
          for (i = 10; i > 0; i--) {
          System.out.print(--i);
          }
          System.out.println(i);
     }
}
```

Ποια είναι η έξοδός του;

```
Α. 975310
Β. 987654321
Γ. 97531
Δ. Λάθος χρόνου εκτέλεσης
Ε. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Η μεταβλητή ελέγχου της for ορίζεται μια φορά πριν τον βρόχο οπότε μπορεί να χρησιμοποιηθεί χωρίς πρόβλημα. Έτσι, λάθος χρόνου εκτέλεσης δεν υπάρχει, άρα η απάντηση Δ δεν ισχύει. Επίσης, η απάντηση Ε δεν είναι σωστή καθώς δεν υφίσταται κάποιο λάθος. Η απάντηση Β επίσης αποκλείεται καθώς το i μειώνεται δύο φορές σε κάθε επαναληπτικό βήμα. Η απάντηση Γ είναι επίσης λανθασμένη. Η έξοδος 97531 πραγματοποιείται σαν αποτέλεσμα του βρόχου. Ωστόσο, μετά το πέρας του βρόχου εκτελείται η System.out.println(i) η οποία τυπώνει το 0. Προσέξτε πως μετά την εκτύπωση του 1 από το σώμα της for, ο έλεγχος μεταβαίνει στην πρόταση της for όπου το i μειώνεται και γίνεται 0. Στην συνέχεια, η συνθήκη ελέγχου προκαλεί τερματισμό της for και ο κώδικας εκτελεί την τελική println με τιμή του i ίση με 0. Επομένως σωστή απάντηση είναι η Α.

Αρχικοποίηση τοπικών μεταβλητών

Έστω ο κώδικας

```
public class KA04 {
     public static void main(String[] args) {
          int x, y=5;
          if (y>5 || x>=Integer.MIN_VALUE) {
               x=y;
          }
          else {
```

```
                x=-y;
            }
            System.out.println(x*y);
        }
}
```

Ποια είναι η έξοδός του;

```
A. 25
B. -25
Γ. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Στην συνθήκη ελέγχου της if επιχειρείται προσπέλαση της x που είναι αναρχικοποίητη τοπική μεταβλητή. Αυτό έχει σαν αποτέλεσμα την παραγωγή λάθους χρόνου μεταγλώττισης. Επομένως σωστή είναι η απάντηση Γ.

Πράξεις με ακέραιους

Έστω ο κώδικας

```
public class KA05 {
     public static void main(String[] args) {
          short k=1;
          k=k+1;
          int m=k++ + 1;
          System.out.println(m+k);
     }
}
```

Ποια είναι η έξοδός του;

```
A. 5
B. 6
Γ. Λάθος χρόνου μεταγλώττισης
Δ. 7
Ε. Λάθος χρόνου εκτέλεσης
```

Απάντηση/Λύση

Στην δεύτερη γραμμή της main επιχειρείται η εκχώρηση k=k+1. Παρότι το k είναι short, η έκφραση k+1 είναι τύπου int. Καθώς ο int είναι μεγαλύτερου μεγέθους από τον short, η εκχώρηση αυτή δεν είναι επιτρεπτή οπότε η σωστή απάντηση είναι η Γ.

Switch 1

Έστω ο κώδικας

```
public class KA06 {
     public static void main(String[] args) {
          int x=1;
          int c=++x+1;
          switch (c) {
               case 0 : System.out.println(0);
               case 1 : System.out.println(1);
               case x : System.out.println(x);
```

```
                case 3 : System.out.println(c);
            }
        }
}
```

Ποια είναι η έξοδός του;

```
A. 0123
B. 123
Γ. 23
Δ. 3
Ε. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Η τρίτη case της switch αναφέρεται στην μεταβλητή x. Στην θέση αυτή είναι αποδεκτές είτε κυριολεκτικές σταθερές είτε σταθερές μεταβλητές, δηλ. μεταβλητές final. Η x δεν είναι ούτε το ένα, ούτε το άλλο. Επομένως, ο κώδικας παράγει λάθος χρόνου μεταγλώττισης και η σωστή απάντηση είναι η Ε.

Switch 2

Έστω ο κώδικας

```
public class KA07 {
     public static void main(String[] args) {
     int k=2;
     switch (k) {
          case 1: System.out.print(1);
          case 2: System.out.print(2);
          default: System.out.print(3);
     }
     }
}
```

Ποια είναι η έξοδός του;

```
A. 1
B. 2
Γ. 12
Δ. 3
Ε. 23
```

Απάντηση/Λύση

Η μεταβλητή k έχει την τιμή 2. Επομένως, η ροή του κώδικα κατευθύνεται στην case 2. Στην συνέχεια εκτελούνται οι προτάσεις μέχρι το τέλος της switch ή μέχρι το επόμενο break. Στο παράδειγμα αυτό δεν έχουμε break επομένως εμφανίζεται το 2 και στην συνέχεια το 3. Η σωστή απάντηση είναι η Ε.

Εκχώρηση και ισότητα

Έστω ο κώδικας

```
public class KA08 {
     public static void main(String[] args) {
          boolean b=7<=5;
          if (b=true) {
               System.out.println("7<=5");
```

```
            }
            else {
                System.out.println("7>5");
            }
        }
}
```

Ποια είναι η έξοδός του;

```
A. 7<=5
B. 7>5
Γ. Λάθος χρόνου μεταγλώττισης
Δ. Λάθος χρόνου εκτέλεσης
```

Απάντηση/Λύση

Δεν υπάρχει λάθος χρόνου μεταγλώττισης, ούτε λάθος χρόνου εκτέλεσης. Προσέξτε πως στην if δεν χρησιμοποιείτε ο τελεστής ισότητας αλλά ο τελεστής εκχώρησης. Έτσι το b δεν συγκρίνεται με το true αλλά εκχωρείται το true στο b. Η τιμή επιστροφής της εκχώρησης επομένως είναι true με αποτέλεσμα ο κώδικας να εισέρχεται στο μπλοκ της if και να εμφανίζει 7<=5. Η σωστή απάντηση επομένως είναι η Α.

Διαίρεση

Έστω ο κώδικας

```
public class KA09 {
    public static void main(String[] args) {
        System.out.println(3/2+""+3.0/2.0);
    }
}
```

Ποια είναι η έξοδός του;

```
A. 1.5 1.5
B. 1 1
Γ. 11.5
```

Απάντηση/Λύση

Το 2 και το 3 είναι κυριολεκτικές σταθερές ακέραιου τύπου. Επομένως και η διαίρεση μεταξύ τους είναι η ακέραιη διαίρεση, δηλ. 3/2=1. Αντίστοιχα, 3.0 και 2.0 είναι πραγματικές σταθερές και η μεταξύ τους διαίρεση είναι διαίρεση μεταξύ πραγματικών, δηλ. 3.0/2.0=1.5. Δεδομένου ότι η προτεραιότητα της διαίρεσης είναι μεγαλύτερη από την προτεραιότητα της σύνδεσης-πρόσθεσης, η έξοδος του κώδικα είναι 11.5 και σωστή είναι η απάντηση Γ.

Διαίρεση και Σύνδεση

Έστω ο κώδικας

```
public class KA10 {
    public static void main(String[] args) {
        double i=1, j=0;
        System.out.print("sum equals to "+i+j);
        if (i/j> 1000) {
            System.out.println("-Very Big");
        }
```

```
            else {
                  System.out.println("-Normal");
            }
      }
}
```

Ποια είναι η έξοδός του;

```
A. sum equals to 1-VeryBig
B. sum equals to 1-Normal
Γ. sum equals to 10-VeryBig
Δ. sum equals to 10-Normal
Ε. Λάθος χρόνου εκτέλεσης
ΣΤ. Τίποτε από τα παραπάνω
```

Απάντηση/Λύση

Η πρώτη print εμφανίζει sum equals to 10. Η έκφραση “sum equals to”+i+j αξιολογείται από αριστερά προς τα δεξιά. Η αλφαριθμητική σειρά “sum equals to “ συνδέεται με το i το οποίο μετατρέπεται αυτόματα σε String. Το αποτέλεσμα “sum equals to 1“ συνδέεται στην συνέχεια με το j. Στην συνέχεια, καθώς τα i και j είναι τύπου double, η διαίρεση i/j δεν προκαλεί λάθος χρόνου εκτέλεσης αλλά επιστρέφει Infinite που θεωρείτε μεγαλύτερο από το 1000, οπότε η συνθήκη της if αληθεύει και εμφανίζεται το -VeryBig. Επομένως, η σωστή απάντηση είναι η Γ.

Όρια Πίνακα 1

Έστω ο κώδικας

```
import java.util.Random;

public class KA11 {
      public static void main(String[] args) {
            Random g=new Random();
            int[] t={1,2,3,4,5,6};
            int idx=g.nextInt(6)+1;
            System.out.println(t[idx]);
      }
}
```

Ποια είναι η έξοδός του;

```
A. Τυπώνει ένα αριθμό από το 1 έως το 6
B. Τυπώνει ένα αριθμό από το 2 έως το 6
Γ. Λάθος χρόνου εκτέλεσης
Δ. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Η συνάρτηση nextInt(6) επιστρέφει έναν ακέραιο από το 0 έως το 5. Επομένως, οι πιθανές τιμές για την μεταβλητή idx είναι από 1 έως 6. Παράλληλα το μήκος του πίνακα t είναι 6. Κατά συνέπεια, οι δείκτες του πίνακα λαμβάνουν τιμές στην περιοχή από 0 έως 5. Επομένως, αν η τιμή της idx είναι από 1 έως 5, η συνάρτηση θα τυπώσει έναν αριθμό από το 2 έως το 6. Ωστόσο, ο κώδικας δεν εγγυάται αυτές τις τιμές για το idx. Αντίθετα, όπως εξηγήσαμε παραπάνω, η τιμή 6 είναι πιθανή για την idx. Σε αυτήν την περίπτωση, στην τελευταία πρόταση επιχειρούμε να προσπελάσουμε έξω από τα όρια του πίνακα. Αυτό θα προκαλέσει λάθος χρόνου εκτέλεσης. Η σωστή απάντηση είναι η Γ.

Όρια Πίνακα 2

Έστω ο κώδικας

```
public class KA12 {

     public static void main(String[] args) {
          int[][] t2Dim = {{1, 2, 3, 4},
           {5, 6, 7, 8},
           {9, 10, 11, 12}
          };
          int[] t={1,0,2,3};
          for (int i=0; i<t.length; i+=2) {
               System.out.print(t2Dim[t[i]][t[i]]);
          }
     }
}
```

Ποια είναι η έξοδός του;

```
Α. 15
Β. 17
Γ. 19
Δ. 111
Ε. 611
ΣΤ. Λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Η for στην main ξεκινά από 0 και αυξάνεται με βήμα 2 έως ότου δεν υπερβεί το μήκος του πίνακα. Επομένως, η πρώτη τιμή της μεταβλητής i είναι το 0 και η δεύτερη το 2. Κατά συνέπεια, η t[i] για i=0 είναι ίση με 1 και η t[i] για i=2 είναι ίση με 2. Έτσι, ισχύει t2Dim[t[i]][t[i]]=t2Dim[1][1] για i=0 και t2Dim[t[i]][t[i]]=t2Dim[2][2]. Το μεν t2Dim[1][1] είναι ίσο με 6, το δε t2Dim[2][2] είναι ίσο με 11. Λάθος χρόνου μεταγλώττισης δεν υπάρχει, επομένως, η έξοδος είναι 611 και η σωστή απάντηση είναι η Ε.

Αναδρομή

Έστω ο κώδικας

```
public class KA13 {
     static void prt(int i) {
          System.out.print(i);
          if (i>=10)
               return;
          if (i%2==0) {
               prt(i+1);
          }
          else {
               prt(i+3);
          }
     }

     public static void main(String[] args) {
          prt(0);
     }
}
```

Ποια είναι η έξοδός του;

```
Α. 01458912
Β. 014589
Γ. 0347811
Δ. 03478
Ε. Καμία από τις παραπάνω
```

Απάντηση/Λύση

Κατά την κλήση prt(0) καταρχάς εμφανίζεται το 0. Στην συνέχεια, η συνθήκη ελέγχου i>=10 είναι ψευδής ενώ η συνθήκη ελέγχου i%2==0 είναι αληθής, οπότε καλείται η prt(1). Η prt(1) αφού τυπώσει 1, καλεί την prt(4). Παρόμοια, η prt(4) εμφανίζει το 4 και καλεί prt(5). Εμφανίζεται το 5 και καλείται η prt(8). Εμφανίζεται το 8 και καλείται prt(9). Εμφανίζεται το 9 και καλείται η prt(12). Αυτή εμφανίζει πρώτα το 12 και στην συνέχεια βρίσκει false την έκφραση i>=10 οπότε τερματίζεται η αναδρομική κλήση. Καθώς από την επιστροφή από την αναδρομική κλήση, η prt δεν έχει άλλο κώδικα να εκτελέσει, η prt(0) τερματίζεται οριστικά. Επομένως, η έξοδος της prt(0) είναι 01458912 και σωστή απάντηση είναι η Α.

Αμοιβαία Αναδρομή

Έστω ο κώδικας

```
public class KA14 {

     static int f(int i) {
          if (i > 0) {
               return g(i);
          }
          return 0;
     }

     static int g(int i) {
          System.out.print(i);
          return f(i - 1);
     }

     public static void main(String[] args) {
          System.out.print(g(3));
     }
}
```

Ποιες από τις παρακάτω προτάσεις είναι σωστές;

```
Α. Η έξοδος είναι 3210
Β. Η g καλείται 3 φορές
Γ. Η f καλείται 3 φορές
Δ. Η έξοδος της κλήσης g(3) είναι 321
```

Απάντηση/Λύση

Η κλήση g(3) εμφανίζει 3 και καλεί την f(2). Η f(2) καλεί την g(2). Η g(2) εμφανίζει 2 και καλεί την f(1). Η f(1) καλεί την g(1) που εμφανίζει 1 και καλεί την f(0). Η f(0) επιστρέφει 0 και τερματίζει την αναδρομική κλήση. Επομένως, η g καλείται 3 φορές, το ίδιο και η f. Άρα οι προτάσεις Β και Γ είναι σωστές. Επιπλέον, η έξοδος της g(3) είναι 321, άρα και η πρόταση Δ είναι σωστή. Τέλος, η τιμή επιστροφής της g(3) είναι 0. Η main εμφανίζει την τιμή αυτή, επομένως η έξοδος του κώδικα είναι 3210, δηλ. και η πρόταση Α είναι σωστή.

Κλάση Math

Έστω ο κώδικας

```
public class KA15 {
     public static void main(String[] args) {
          float f;
          for (f=0; f<=9; f++); //γραμμή 4
          f=Math.sqrt(f) + 1;   //γραμμή 5
          f=f/1;
          System.out.println(f);
     }
}
```

Ποια από τις παρακάτω προτάσεις είναι σωστή;

```
Α. Η main έχει λάθος χρόνου μεταγλώττισης στην γραμμή 4
Β. Η main έχει λάθος χρόνου μεταγλώττισης στην γραμμή 5
Γ. Η main τυπώνει 4
Δ. Καμία από τις παραπάνω
```

Απάντηση/Λύση

Στην γραμμή 4 δεν υπάρχει λάθος. Όπως έχουμε εξηγήσει, κανένα από τα τμήματα της for δεν είναι υποχρεωτικό. Στην γραμμή 5, η sqrt επιστρέφει τιμή τύπου double ενώ το f είναι float. Καθώς ο float είναι μικρότερης χωρητικότητας από τον double, η εκχώρηση αυτή δεν επιτρέπεται. Επομένως, η γραμμή 5 έχει λάθος χρόνου μεταγλώττισης. Κατά συνέπεια ο κώδικας δεν μεταγλωττίζεται άρα δεν τυπώνει 4. Η σωστή απάντηση είναι η Β.

Κλάση Boolean

Έστω ο κώδικας

```
public class KA16 {
     public static void main(String[] args) {
          int i=1, j=0;
          boolean b= i++>j && i>j+1; //2
          System.out.println(b                              &&
          Boolean.parseBoolean("TRue"));
     }
}
```

Ποια είναι η έξοδος του;

```
Α. false
Β. true
Γ. Περιέχει λάθος χρόνου εκτέλεσης
```

Απάντηση/Λύση

Η έκφραση στην γραμμή 2 αξιολογείται ως εξής: Πρώτα αξιολογείται η έκφραση στα δεξιά της εκχώρησης, δηλ. η έκφραση i++>j && i>j+1. Η αξιολόγηση εδώ γίνεται από αριστερά προς τα δεξιά. Η i++>j αυξάνει το i σε 2 και είναι αληθής, επομένως θα αξιολογηθεί και η i>j+1 η οποία επίσης είναι αληθής. Επομένως, η μεταβλητή b λαμβάνει την τιμή true. Η parseBoolean επιστρέφει επίσης true όταν το όρισμά της είναι η σειρά true ανεξαρτήτως πεζών ή κεφαλαίων. Το λογικό and μεταξύ δύο εκφράσεων που είναι αληθείς επιστρέφει true. Λάθος χρόνου εκτέλεσης δεν υπάρχει. Συνεπώς, η main εμφανίζει true και η σωστή απάντηση είναι η Β.

Συστήματα Αρίθμησης

Έστω ο κώδικας

```
public class KA17 {
     public static void main(String[] args) {
          int i=010, j=011;
          System.out.println(i+j);
     }
}
```

Ποια είναι η έξοδος του;

```
A. 21
B. 17
Γ. Περιέχει λάθος χρόνου εκτέλεσης
Δ. Περιέχει λάθος χρόνου μεταγλώττισης
```

Απάντηση/Λύση

Όπως έχουμε αναφέρει στην ενότητα 3.6, οι κυριολεκτικές ακέραιες σταθερές που αρχίζουν από μηδέν εκφράζονται στο οκταδικό σύστημα. Ο 010 είναι η βάση του οκταδικού συστήματος, δηλ. το 8. Εύκολα, μπορεί κανείς να καταλάβει πως ο 011 είναι ο επόμενος ακέραιος από τον 010, δηλ. το 9. Επομένως, η έξοδος του κώδικα είναι 17 και η σωστή απάντηση η Δ.

Κεφάλαιο 10

Σύνοψη
Καταρχάς γίνεται μια σύντομη ιστορική αναδρομή της ανάπτυξης του αντικειμενοστρεφούς μοντέλου. Στην συνέχεια παρουσιάζονται τα θεμελιώδη χαρακτηριστικά του αντικειμενοστρεφούς μοντέλου, η Ενσωμάτωση (Encapsulation), η Κληρονομικότητα (Inheritance), ο Πολυμορφισμός (Polymorphism) και η Αφαιρετικότητα (Abstraction). Συζητείται η αναλογία μεταξύ των θεμελιωδών χαρακτηριστικών και των προβλημάτων του φυσικού κόσμου και εξηγούνται οι τρόποι με τους οποίους το αντικειμενοστρεφές μοντέλο διευκολύνει στην μοντελοποίηση των προβλημάτων του φυσικού κόσμου.

Προαπαιτούμενη γνώση
Η εισαγωγική αυτή ενότητα προϋποθέτει γνώση του διαδικαστικού προγραμματισμού με Java.

Λέξεις κλειδιά
Αντικειμενοστρεφής προγραμματισμός (Object Oriented Programming), Διαδικαστικός Προγραμματισμός (Procedural Programming), Ενσωμάτωση (Encapsulation), Κληρονομικότητα (Inheritance), Πολυμορφισμός (Polymorphism), Αφαιρετικότητα (Abstraction).

10 Εισαγωγή στο Αντικειμενοστρεφές μοντέλο

Το αντικειμενοστρεφές μοντέλο αντλεί την καταγωγή του από την εφαρμογή Sketchpad [1] του Ivan Sutherland που δημιουργήθηκε στις αρχές της δεκαετίας του 1960. Η Sketchpad υποστήριζε κληρονομικότητα βασιζόμενη σε πρωτότυπα με πολλά κοινά στοιχεία με την σημερινή Javascript και δυναμική διασύνδεση, βασικό χαρακτηριστικό των περισσοτέρων γλωσσών αντικειμενοστρεφούς προγραμματισμού, σήμερα. Την ίδια περίοδο έκανε την εμφάνισή της η AED-0, μια έκδοση της Algol που υποστήριζε την άμεση διασύνδεση δεδομένων και συναρτήσεων που αποτελεί θεμελιώδες χαρακτηριστικό του αντικειμενοστρεφούς μοντέλου. Το 1965 εμφανίστηκε η Simula [2], η πρώτη γλώσσα προγραμματισμού που στην συνέχεια αναγνωρίστηκε ευρέως ως αντικειμενοστρεφής. Ωστόσο, τον όρο «Αντικειμενοστρεφής Προγραμματισμός» επινόησε ο Alan Kay το 1967.

Την δεκαετία του 1970 αναπτύχθηκε η Smalltalk [3] που εισήγαγε την έννοια των κλάσεων από τους Alan Kay, Dan Ingalls, Adele Goldberg και άλλους στην Xerox PARC.

Στα μέσα του 1980 αναπτύχθηκε η Objective-C που προσέθεσε στην C στοιχεία από την SmallTalk και αποτέλεσε μέχρι το 2014, οπότε αντικαταστάθηκε από την Swift, την κύρια γλώσσα προγραμματισμού για τα λειτουργικά macOS και iOS της Apple.

Ήδη από το 1979, ο Bjarne Stroustrup, άρχισε την ανάπτυξη μιας προέκτασης της C που ονόμασε C with classes ενώ το 1982 άρχισε την ανάπτυξη της C++ [4], η αρχική έκδοση της οποίας βγήκε το 1985.

Την δεκαετία του 1990, το αντικειμενοστρεφές μοντέλο έγινε το κυρίαρχο μοντέλο προγραμματισμού καθώς οι αντικειμενοστρεφείς γλώσσες διαδόθηκαν ευρέως.

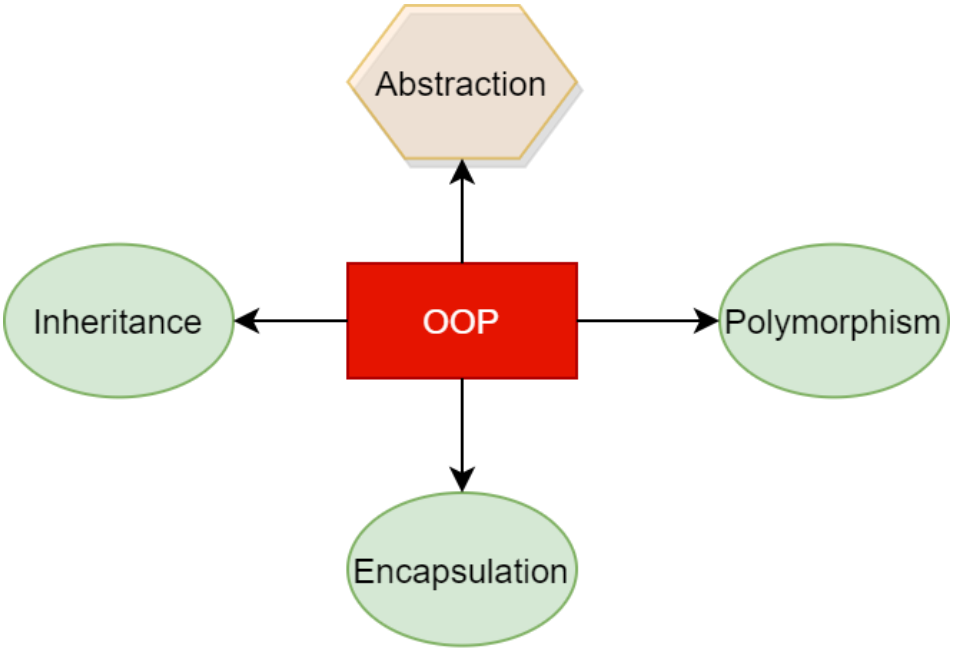
Η πρώτη έκδοση της Python κυκλοφόρησε το 1994 με δημιουργό τον Van Rossum. Με την εμφάνισή της, η Python ενσωματώνει τα βασικά χαρακτηριστικά του αντικειμενοστρεφούς μοντέλου τα οποία συνδυάζει με το λειτουργικό (functional) μοντέλο. Λίγο αργότερα, το 1996, κυκλοφόρησε και η πρώτη έκδοση της Java το 1996 από την Sun Microsystems.

Πέραν των προαναφερόμενων γλωσσών, εμφανίστηκαν και αρκετές άλλες ενώ χαρακτηριστικά του αντικειμενοστρεφούς μοντέλου, προστέθηκαν σε αρκετές προυπάρχουσες.

Πρόσφατα, πολλές αντικειμενοστρεφείς γλώσσες, συμπεριλαμβανομένης της Java, ενσωματώνουν χαρακτηριστικά του λειτουργικού (functional) προγραμματισμού, ενισχύοντας έτσι την αποτελεσματικότητά τους.

10.1 Τα θεμελιώδη χαρακτηριστικά του Αντικειμενοστρεφούς Προγραμματισμού

Το αντικειμενοστρεφές μοντέλο σχεδιάστηκε ώστε να εξυπηρετεί δύο βασικούς στόχους. Ο πρώτος αφορά στην διευκόλυνση της μοντελοποίησης των προβλημάτων του πραγματικού κόσμου και ο δεύτερος στον έλεγχο των λαθών που όχι σπάνια υπεισέρχονται στους κώδικες εφαρμογών.
Τους στόχους του, το αντικειμενοστρεφές μοντέλο επιτυγχάνει βασιζόμενο στα θεμελιώδη χαρακτηριστικά του: την Ενσωμάτωση (Encapsulation), την Κληρονομικότητα (Inheritance) και τον Πολυμορφισμό (Polymorphism). Στο σχήμα 10.1, παρουσιάζονται τα θεμελιώδη χαρακτηριστικά του αντικειμενοστρεφούς μοντέλου. Στο σχήμα συμπεριλαμβάνεται και η Αφαιρετικότητα (Abstraction) που παρουσιάζεται παρακάτω σε αυτήν την ενότητα.

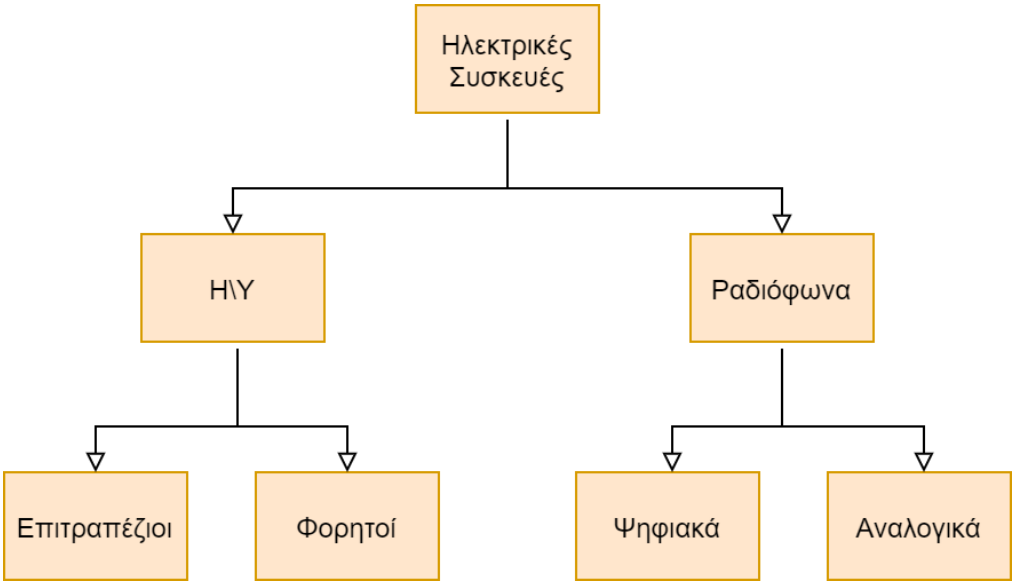


Σχήμα 10.1 *Τα θεμελιώδη χαρακτηριστικά του αντικειμενοστρεφούς μοντέλου*

Ο πραγματικός κόσμος αποτελείται από αντικείμενα, π.χ. ένας Ηλεκτρονικός Υπολογιστής, ένα βιβλίο, μια καρέκλα. Τα αντικείμενα συχνά διαθέτουν ιδιότητες και επιτελούν λειτουργίες. Για παράδειγμα, ένας Ηλεκτρονικός Υπολογιστής διαθέτει ιδιότητες όπως η χωρητικότητα της μνήμης RAM, ο τύπος του επεξεργαστή του, κα. Επίσης, επιτελεί λειτουργίες, όπως εκκίνηση λειτουργίας, τερματισμός, κα. Το αντικειμενοστραφές μοντέλο μας δίνει την δυνατότητα να ορίσουμε οντότητες μέσα στον κώδικά μας που ενσωματώνουν ιδιότητες και λειτουργίες και να αναπαραστήσουμε έτσι τα αντικείμενα του πραγματικού κόσμου με ευκολία. Πρόκειται για την πιο βασική αρχή του αντικειμενοστραφούς προγραμματισμού, την Ενσωμάτωση. Λεπτομέρειες για την Ενσωμάτωση θα βρείτε στην ενότητα 11.

Τα αντικείμενα του πραγματικού κόσμου είναι οργανωμένα σε κατηγορίες. Για παράδειγμα, ο δικός μου υπολογιστής ανήκει στην κατηγορία των επιτραπέζιων υπολογιστών, ο φορητός υπολογιστής της συζύγου μου ανήκει στην κατηγορία των φορητών υπολογιστών, το ραδιόφωνό που έχουμε στο σαλόνι ανήκει στην κατηγορία των αναλογικών ραδιοφώνων και αυτό που έχουμε στην κρεβατοκάμαρα στην κατηγορία των ψηφιακών ραδιοφώνων. Στα πλαίσια του αντικειμενοστρεφούς μοντέλου, οι κατηγορίες αντικειμένων αναφέρονται συνήθως με τον όρο κλάσεις (classes). Έτσι, έχουμε την κλάση των επιτραπέζιων υπολογιστών και την κλάση των φορητών υπολογιστών, την κλάση των αναλογικών ραδιοφώνων και την κλάση των ψηφιακών ραδιοφώνων. Προσέξτε όμως πως οι επιτραπέζιοι και οι φορητοί υπολογιστές συνιστούν υποκατηγορίες ή υποκλάσεις (subclasses) της κλάσης των ηλεκτρονικών υπολογιστών. Αντίστοιχα, η κλάση των ηλεκτρονικών υπολογιστών είναι υπερκλάση (superclass) της κλάσης των επιτραπέζιων και της κλάσης των φορητών υπολογιστών. Παρόμοια, τα αναλογικά και τα ψηφιακά ραδιόφωνα αποτελούν υποκλάσεις της γενικότερης κλάσης των ραδιοφώνων. Επιπλέον, τόσο τα ραδιόφωνα όσο και οι ηλεκτρονικοί υπολογιστές είναι υποκλάσεις της κλάσης των ηλεκτρικών συσκευών.

Γενικότερα, τα αντικείμενα του πραγματικού κόσμου οργανώνονται σε κλάσεις οι οποίες σχετίζονται μεταξύ τους με ιεραρχικές σχέσεις. Στο σχήμα 10.2 απεικονίζονται οι ιεραρχικές σχέσεις των κλάσεων που αναφέραμε ως παραδείγματα.



Σχήμα 10.2 *Ιεραρχικές σχέσεις μεταξύ των κλάσεων*

Οι ηλεκτρικές συσκευές διαθέτουν μια λειτουργία που τους επιτρέπει να συνδέονται ή να αποσυνδέονται στο ηλεκτρικό ρεύμα. Την λειτουργία αυτή όμως διαθέτουν και οι Η\Υ και τα ραδιόφωνα καθώς τόσο οι Η\Υ όσο και τα ραδιόφωνα είναι ηλεκτρικές συσκευές. Το ίδιο ισχύει για τις κλάσεις των επιτραπέζιων και φορητών υπολογιστών, των ψηφιακών και των αναλογικών ραδιοφώνων. Γενικότερα, μια λειτουργία που ορίζεται σε μια υπερκλάση αποτελεί επίσης λειτουργία κάθε υποκλάσης της. Το αντικειμενοστραφές μοντέλο μέσα από την δεύτερη θεμελιώδη αρχή του, την Κληρονομικότητα, μας δίνει την δυνατότητα να ορίζουμε μια ιδιότητα ή λειτουργία σε μια κλάση και στην συνέχεια η λειτουργία/ιδιότητα κληρονομείται αυτόματα σε όλες τις υποκλάσεις της. Το χαρακτηριστικό αυτό διευκολύνει την εφαρμογή της επαναχρησιμοποίησης κώδικα (code reusability) και διαμορφώνει το μοντέλο για την εφαρμογή του τρίτου θεμελιώδους χαρακτηριστικού του, του Πολυμορφισμού. Λεπτομέρειες για την Κληρονομικότητα θα βρείτε στην ενότητα 13.

Όπως αναφέρθηκε ήδη, ο υπολογιστής μου είναι επιτραπέζιος, επομένως ανήκει στην κλάση των επιτραπέζιων υπολογιστών. Ταυτόχρονα όμως είναι ένας (is a) Η\Υ αλλά και μια ηλεκτρική συσκευή. Σε αντιστοιχία με τον πραγματικό κόσμο, το αντικειμενοστρεφές μοντέλο μας δίνει την δυνατότητα να δημιουργήσουμε ένα αντικείμενο τύπου επιτραπέζιου υπολογιστή και ανάλογα με τις ανάγκες του κώδικά μας να το διαχειριζόμαστε σαν Η\Υ ή σαν ηλεκτρική συσκευή, δηλ. σαν να είναι τύπου μιας υπερκλάσης. Αυτή η δυνατότητα συνιστά την βάση του Πολυμορφισμού και είναι πολύ σημαντική όπως θα δούμε στην ενότητα 13 που εξετάζουμε αναλυτικά τον πολυμορφισμό.

Πολλοί συγγραφείς θεωρούν ένα τέταρτο θεμελιώδες χαρακτηριστικό του αντικειμενοστραφούς προγραμματισμού, την Αφαιρετικότητα (Abstraction) [1,42]. Αφαιρετικότητα σημαίνει απόκρυψη των μη αναγκαίων πληροφοριών από τον χρήστη του κώδικα. Ωστόσο, άλλοι συγγραφείς δεν θεωρούν την αφαιρετικότητα ως θεμελιώδες χαρακτηριστικό του αντικειμενοστρεφούς μοντέλου [2,41]. Παρότι υπάρχει καθολική συμφωνία πως η αφαιρετικότητα είναι ένα χαρακτηριστικό του αντικειμενοστρεφούς προγραμματισμού, προκύπτει διχογνωμία για το κατά πόσο αποτελεί θεμελιώδες γνώρισμα του. Οι συγγραφείς που δεν θεωρούν την αφαιρετικότητα ως θεμελιώδες χαρακτηριστικό, επιχειρηματολογούν υποστηρίζοντας πως αυτή συναντάται και στο διαδικαστικό μοντέλο επάνω στο οποίο στηρίζεται το αντικειμενοστρεφές. Επίσης, κάθε πληροφοριακό σύστημα, σωστά οργανωμένο, χαρακτηρίζεται από κάποιο βαθμό αφαιρετικότητας. Επιπλέον, παρότι η αφαιρετικότητα διαθέτει και τους δικούς της ιδιαίτερους μηχανισμούς σε αρκετές γλώσσες και στην Java, είναι ταυτόχρονα αναγκαίο αποτέλεσμα της ενσωμάτωσης, της κληρονομικότητας και του πολυμορφισμού. Με αυτά τα δεδομένα, τείνουν να θεωρούν την αφαιρετικότητα ως παράγωγο παρά ως θεμελιώδες γνώρισμα.

Βιβλιογραφία

[1] “Sketchpad,” Wikipedia. Dec. 28, 2020. Accessed: Feb. 10, 2021. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Sketchpad&oldid=996705015

[2] J. R. Holmevik, “Compiling SIMULA: a historical study of technological genesis,” IEEE Annals of the History of Computing, vol. 16, no. 4, pp. 25–37, Winter 1994, doi: 10.1109/85.329756.

[3] “The Early History of Smalltalk,” Jul. 10, 2008. https://web.archive.org/web/20080710144930/http://gagne.homedns.org/~tgagne/contrib/EarlyHistoryST.html (accessed Feb. 10, 2021).

[4] “Stroustrup: The C++ Programming Language.” https://www.stroustrup.com/1st.html (accessed Feb. 10, 2021).