Περιεχόμενα

Μεταβλητές και Θεμελιώδεις τύποι

Περίληψη: *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 3 αυτού του εγχειριδίου που περιλαμβάνει τις έννοιες: πακέτο (package), η main ως κύρια είσοδος στο πρόγραμμα, το υποπρόγραμμα ως void συνάρτηση χωρίς παραμέτρους, βασική έξοδο 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).*

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 στην ενότητα 8 ενώ οι μεταβλητές στιγμιότυπου αφορούν το αντικειμενοστραφές μοντέλο και δεν εξετάζονται σε αυτό το εγχειρίδιο.

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

---

[1] Επιπλέον, κάθε μεταβλητή εκ των τριών πρώτων αναφερόμενων ειδών, μπορεί να είναι **μεταβλητή αναφοράς** (*reference variable*) ή **μεταβλητή τιμής** (*value variable*). Οι μεταβλητές στιγμιότυπου συχνά αναφέρονται και ως **μη στατικά πεδία** (*non-Static Fields*).

[2] Πριν τον τύπο μπορεί να τοποθετηθεί ένας τροποποιητής (modifier). 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.

```
int k;
```

δηλώνει μια ακέραιη μεταβλητή με όνομα 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 μια τοπική μεταβλητή.

```
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**[3] (*unitialized*). 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;
   k=0;
   System.out.println(k);
}
```

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

---

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

[4] Μπορούμε ωστόσο να προσπελάσουμε μια μεταβλητή που ενδεχομένως είναι αναχρικοποίητη εφόσον την περικλείσουμε σε try {} catch μπλοκ. Περισσότερα επ'αυτού κατά την μελέτη του αντικειμενοστραφούς μοντέλου.

Εμβέλεια (*Scope*) : Εμβέλεια μιας μεταβλητής ονομάζεται η περιοχή εκείνη του κώδικα που η μεταβλητή είναι ορατή 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 παράγαγει error 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 (*variable life time*). Ο χρόνος ζωής 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 8 που παρουσιάζεται η έννοια της στατικής μεταβλητής, θα έχουμε την ευκαιρία να διαφοροποιήσουμε πληρέστερα μεταξύ των δύο αυτών εννοιών, οπότε θα αποσαφηνιστεί η διαφορά μεταξύ εμβέλειας και χρόνου 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.

Τοπικές Σταθερές

Σε πολλές περιπτώσεις χρειαζόμαστε μεταβλητές που η τιμή τους να μην αλλάζει μετά την αρχική εκχώρηση (*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;
```

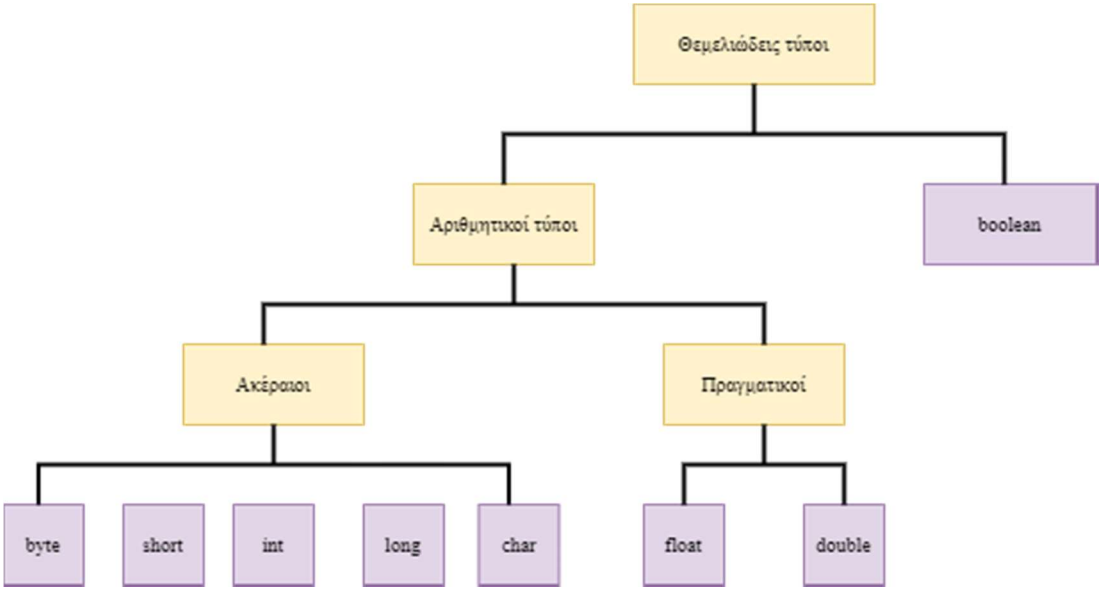
παράγει λάθος μεταγλώττισης.

Θεμελιώδεις τύποι δεδομένων (Primitive Data Types)

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 αδύνατου τύπου** (*Weak typed languages*).

Οι μεταβλητές επομένως στην Java έχουν προκαθορισμένο τύπο. Οι δυνατοί 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 **λέξεις κλειδιά** (keywords). Πλήρη λίστα με τις λέξεις κλειδιά μπορείτε να βρείτε στο παράρτημα 2. Όλοι οι υπόλοιποι τύποι προκύπτουν από συνδυασμό ή επανορισμό των θεμελιωδών τύπων. Στο σχήμα 1, παρουσιάζεται η κατηγοριοποίηση των θεμελιωδών τύπων.



Σχήμα 1: Κατηγοριοποίηση των θεμελιωδών τύπων

`byte` Ο τύπος `byte` είναι προσημασμένος ακέραιος μήκους οκτώ **δυαδικών ψηφίων** (*bits*). Ο τύπος `byte` αναπαρίσταται στην μνήμη με την μορφή συμπληρώματος ως προς 2 (two's complement). Το πρώτο bit χρησιμοποιείται για την αναπαράσταση του πρόσημου. Αν είναι 1, η τιμή είναι αρνητική. Αν είναι 0, η τιμή είναι θετική. Περισσότερες λεπτομέρειες για το συμπλήρωμα ως προς 2 στο παράρτημα 1. Επομένως, για την αναπαράσταση της τιμής απομένουν 7 bits. Συνεπώς, αναπαρίστανται συνολικά 2^7 αρνητικές τιμές και 2^7 μη αρνητικές τιμές. Πιο συγκεκριμένα, αναπαρίστανται οι αρνητικές τιμές -1..-2^7 και οι μη αρνητικές τιμές 0..2^7-1. Προσέξτε πως ο η μικρότερη αρνητική τιμή, δηλ. το -2^7, είναι κατ' απόλυτη τιμή μεγαλύτερη κατά μια μονάδα από την μεγαλύτερη μη αρνητική, δηλ. το 2^7-1. Αυτό συμβαίνει γιατί στις μη αρνητικές τιμές συμπεριλαμβάνεται και το 0. Ανάλυση της αναπαράστασης ακεραίων που περιλαμβάνει εξήγηση του συμπληρώματος ως προς 2, μπορείτε να βρείτε στο παράρτημα 1.

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 αντικειμενοστραφούς 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.

Κώδικας 1: Παράδειγμα χρήσης μεταβλητής τύπου byte

```
1. byte bMin = Byte.MIN_VALUE, bMax = Byte.MAX_VALUE, bLiteral=123;
2. System.out.println("bMin equals to " + bMin);
3. System.out.println("bMax equals to " + bMax);
4. System.out.println("bLiteral equals to " + bLiteral);
5. System.out.println("Size of byte is " + Byte.SIZE + " bits");
```

Στον κώδικα 1, στην γραμμή 1 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, στις γραμμές 2 έως και 4, στέλνουμε στην συσκευή εξόδου, δηλ. στην τυπική περίπτωση στην οθόνη, το περιεχόμενο των bMin, bMax και bLiteral. Τέλος, στην γραμμή 5, τυπώνουμε το μέγεθος του τύπου (Byte.*SIZE*) μετρημένο σε δυαδικά ψηφία (bits). Παρατίθεται η έξοδος του κώδικα 1.

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

`short` Ο τύπος `short` είναι προσημασμένος ακέραιος σε μορφή συμπληρώματος ως προς 2, μήκους 16 δυαδικών ψηφίων. Επομένως, οι τιμές που δέχονται οι μεταβλητές τύπου `short` είναι $0..2^{16}-1$ και $-1..-2^{16}$.

Στον κώδικα 2, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου `short`.

Κώδικας 2: Παράδειγμα χρήσης μεταβλητών τύπου short

```
1. short s = Short.MIN_VALUE, s1 = Short.MAX_VALUE, s2=399;
2. System.out.println("short s equals to " + s);
3. System.out.println("short s1 equals to " + s1);
4. System.out.println("short s2 equals to " + s2);
5. System.out.println("Size of short is " + Short.SIZE + " bits");
```

Ακολουθεί η έξοδος του κώδικα 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 σε μορφή συμπληρώματος ως προς 2, μήκους 32 δυαδικών ψηφίων. Επομένως, οι τιμές που δέχονται οι μεταβλητές τύπου `int` είναι $0..2^{32}-1$ και $-1..-2^{32}$.

Στον κώδικα 3, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου `int`.

Κώδικας 3: Δήλωση και αρχικοποίηση μεταβλητών τύπου int

```
1.  int iMin = Integer.MIN_VALUE,
2.  iMax = Integer.MAX_VALUE, iLiteral=43127;
3.  System.out.println("int iMin equals to " + iMin);
4.  System.out.println("int iMax equals to " + iMax);
5.  System.out.println("int iLiteral equals to " + iLiteral);
6.  System.out.println("Size of int is " + Integer.SIZE + " bits");
```

Ακολουθεί η έξοδος του κώδικα 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 σε μορφή συμπληρώματος ως προς 2, μήκους 64 δυαδικών ψηφίων. Επομένως, οι τιμές που δέχονται οι μεταβλητές τύπου `long` είναι $0..2^{64}-1$ και $-1..-2^{64}$.

Στον κώδικα 4, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου `long`.

Κώδικας 4: Δήλωση και αρχικοποίηση μεταβλητών τύπου long

```
1.  long lMin = Long.MIN_VALUE,
2.  lMax = Long.MAX_VALUE, lLiteral=2_147_483_699L;
3.  System.out.println("long lMin equals to " + lMin);
4.  System.out.println("long lMax equals to " + lMax);
5.  System.out.println("long lLiteral equals to " + lLiteral);
6.  System.out.println("Size of long is " + Long.SIZE + " bits");
```

Ακολουθεί η έξοδος του κώδικα 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 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 απο 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*), mxE^n, όπου m ένας δεκαδικός αριθμός, n ακέραιος και E=10. Οι float αριθμοί χαρακτηρίζονται ως **αριθμοί κινητής υποδιαστολής απλής ακρίβειας** (*single-precision floating point numbers*).

Στον κώδικα 5, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου `float`.

Κώδικας 5: Δήλωση και αρχικοποίηση μεταβλητών τύπου float

```
1. float fMin = Float.MIN_VALUE, fMax = Float.MAX_VALUE, fLit=3.15f;
2. System.out.println("float fMin equals to " + fMin);
3. System.out.println("float fMax equals to " + fMax);
4. System.out.println("float fLit equals to " + fLit);
5. System.out.println("Size of float is " + Float.SIZE + " bits");
```

Ακολουθεί η έξοδος του κώδικα 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 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. 1.4×10^{-45} και η μέγιστη είναι 3.4028235E38, δηλ. 3.4028235×10^{38}, 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 συμετέχουν 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 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 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 !=) 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 συγκρίνεται 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 διαφορά των δύο float είναι μικρότερη από αυτήν την οριακή τιμή, οι αριθμοί θεωρούνται ίσοι.

Γενικότερα, ο τύπος float θα πρέπει να αποφεύγεται για αποθήκευση 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 6, παρατίθεται παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου double.

Κώδικας 6: Δήλωση και αρχικοποίηση μεταβλητών τύπου double

```
1. double dMin = Double.MIN_VALUE, dMax = Double.MAX_VALUE,
        dLit=3.4028235E38;
2. System.out.println("double dMin equals to " + dMin);
3. System.out.println("double dMax equals to " + dMax);
4. System.out.println("double dLit equals to " + dLit);
5. System.out.println("Size of double is " + Double.SIZE + "bits");
```

Ακολουθεί η έξοδος του κώδικα 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 $1.7976931348623157x10^{308}$ 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 που μπορεί να αναπαρασταθεί με αυτούς τους τύπους 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 [2].

Στον κώδικα 7, παρουσιάζουμε παράδειγμα δήλωσης και αρχικοποίησης μεταβλητών τύπου boolean.

Κώδικας 7: Δήλωση και αρχικοποίηση boolean μεταβλητών

```
1. boolean b1=true, b2=false;
2. System.out.println("b1 equals to "+b1);
3. System.out.println("b2 equals to "+b2);
```

Ακολουθεί η έξοδος του κώδικα 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 8, δίνουμε παράδειγμα χρήσης μεταβλητών του τύπου char.

Κώδικας 8: Παράδειγμα χρήσης του τύπου char

```
1.  char cMin = Character.MIN_VALUE, cMax = Character.MAX_VALUE,
        cLit = 'A', c='\u0043';
2.  System.out.println(cMin + " " + (int) cMin);
3.  System.out.println(cMax + " " + (int) cMax);
4.  System.out.println(c + " "+ (int) c);
5.  System.out.println(cLit + " " + (int) cLit);
6.  System.out.println((char) (cLit + 1));
```

Ο κώδικας 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 ο τύπος κωδικοποιείται στην μνήμη σαν 16-bit ακέραιος χωρίς πρόσημο. 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.

Ακολουθίες ελέγχου (Control Sequences)

Ο χαρακτήρας \ (backslash) ακολουθούμενος από έναν ειδικό χαρακτήρα ονομάζεται **ακολουθία ελέγχου** (*control sequence*) ή **χαρακτήρας διαφυγής** (*escape character*).

Στον πίνακα 1 παρουσιάζονται οι ακολουθίες ελέγχου της Java.

Πίνακας 1: Ακολουθίες Ελέγχου		
\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

Ακολουθεί κώδικας χρήσης των ακολουθιών ελέγχου

Κώδικας 9: Παραδείγματα χρήσης ακολουθιών ελέγχου

```
1. System.out.println("slash_t"+'\t'+"second ");
2. System.out.println("slash_b"+'\b'+"second ");
3. System.out.println("slash_n"+'\n'+"second ");
4. System.out.println("slash_r"+'\r'+"second ");
5. System.out.println("slash_singlequote"+'\''+"second ");
6. System.out.println("slash_doublequote"+'\"'+"second ");
7. System.out.println("slash_backslash"+'\\'+"second ");
8. System.out.println("slash_unicode"+'\u0041'+"second ");
```

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 1 τυπώνει “slash_t”, μετά αφήνει ένα tab και τέλος τυπώνει την σειρά “second”.

Στην γραμμή 2, ο χαρακτήρας ‘\b’ ΄φέρνει την σειρά “second” έναν χαρακτήρα πιο πίσω με αποτέλεσμα την εξαφάνιση του b από την “slash_b”.

Στην γραμμή 3, ο χαρακτήρας newline έχει ως αποτέλεσμα να εκτυπωθεί η “slash_n” και η “second” να εκτυπωθεί στην επόμενη γραμμή.

Στην γραμμή 4, το carriage return φέρνει την έξοδο στην αρχή της γραμμής με αποτέλεσμα να εξαφανίζεται η σειρά “slash_r”.

Η γραμμή 5 τυπώνει ένα quote ανάμεσα στην “slash_singlequote” και στην “second”.

Η γραμμή 6 τυπώνει ένα double quote ανάμεσα στην “slash_doublequote” και στην “second”.

Η γραμμή 7 τυπώνει ένα backslash ανάμεσα στην “slash_backslash” και στην “second”.

Η γραμμή 8 τυπώνει τον χαρακτήρα ‘A’ ανάμεσα στην “slash_unicode” και στην “second”.

Αναγνωριστικά

Είναι προφανές από την μέχρι τώρα παρουσίαση πως κάθε τοπική μεταβλητή λαμβάνει ένα όνομα που αποτελεί το αναγνωριστικό 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 2, δίνονται παραδείγματα από έγκυρα και μη έγκυρα αναγνωριστικά

Πίνακας 2: Παραδείγματα έγκυρων και άκυρων αναγνωριστικών

Έγκυρα Αναγνωριστικά	Μη έγκυρα
rVal	!color
costPerUnit	1start
_event23	*profit
net_weight	start-up
$score	last/digit

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 **αυτό-τεκμηριωμένος** (*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 τοπικών μεταβλητών ισχύουν και για τις παραμέτρους. Αντίθετα, όλα τα γράμματα στα αναγνωριστικά των σταθερών μεταβλητών πρέπει να είναι κεφαλαία.

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 10, παρουσιάζουμε δύο παραδείγματα υπερχείλισης

Κώδικας 10: Υπερχείλιση ακεραίων

```
1. int i=Integer.MAX_VALUE, j=Integer.MIN_VALUE;
2. i=i+1;
3. System.out.println(i);
4. System.out.println(Integer.MIN_VALUE-1);
```

Η έξοδος του κώδικα 10 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.

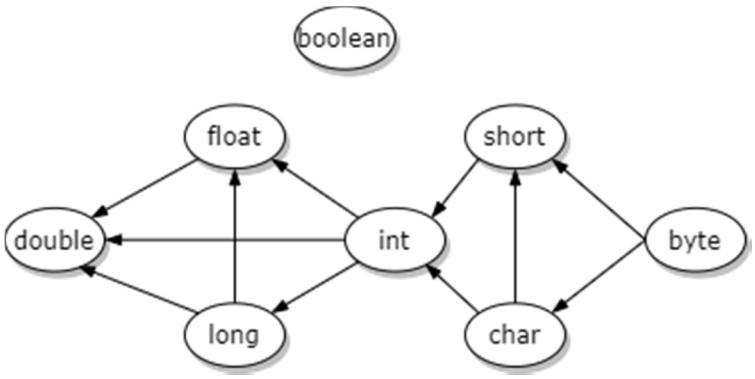
Αν παρόλα αυτά θέλετε να προσθέσετε δύο int και δεν είστε βέβαιοι ότι το άθροισμά τους είναι στα όρια του τύπου, μπορείτε να κάνετε μετατροπή τύπου τουλάχιστον στον ένα από τους δύο και στην συνέχεια να εκχωρήσετε το αποτέλεσμα σε έναν long. Στον κώδικα 11, δίνεται σχετικό παράδειγμα.

Κώδικας 11: Άθροισμα ακεραίων εκτός ορίων του τύπου

```
1. int i=Integer.MAX_VALUE;
2. long k=(long)i+1;
3. System.out.println(k);
```

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 και μεγαλύτερης ή ίσης χωρητικότητας από τον int. Μπορούμε ωστόσο να εκχωρήσουμε μια μεταβλητή ενός τύπου σε μεταβλητή άλλου τύπου με την προυπόθεση 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 2: Αυτόματες μετατροπές μεταξύ θεμελιωδών τύπων

Όπως φαίνεται στο σχήμα 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 σε κανέναν άλλο τύπο 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 2^{970}, το αποτέλεσμα της έκφρασης θα γίνει Infinity. Παρόμοια, 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. Αντίθετα, 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 άπειρο πλήθος πραγματικών τιμών.

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.

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 Strings είναι η σύνδεση των String (*concatenation*). Η σύνδεση των String γίνεται με την βοήθεια του τελεστή +.

Κώδικας 11: Παράδειγμα μεταβλητών τύπου String

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

Στον κώδικα 11, στην γραμμή 1, δηλώνουμε και αρχικοποιούμε την μεταβλητή s1 τύπου String, στην γραμμή 2, δηλώνουμε και αρχικοποιούμε την μεταβλητή s2. Τέλος, στην γραμμή 3, 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. Από σύμβαση τα ονόματα των κλάσεων ξεκινούν με κεφαλαίο. Μια άλλη πολύ ουσιαστική διαφορά 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). Περισσότερες λεπτομέρειες σχετικά με το σημαντικό αυτό ζήτημα σχετίζονται με την μελέτη του αντικειμενοστραφούς μοντέλου.

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; int finalsum; char Start; final char END; double 3oAthroisma; boolean ExitLoop; byte day; _10; _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 Nov. 15, 2020).

[2] L. Doug, *Java All-in-One for Dummies*, 5th ed. John Wiley & Sons, Inc., 2017.

[3] “The Unicode Standard, Version 13.0.” The Unicode Consortium, Accessed: Sep. 11, 2020. [Online]. Available: http://www.unicode.org/charts/PDF/UFFF0.pdf.

[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/jls-5.html#jls-5.1.4 (accessed Nov. 11, 2020).