# 3. Τύποι δεδομένων - εντολές - τελεστές

## 3.1 Τύποι δεδομένων στην JAVA

Οι τύποι δεδομένων στην JAVA είναι σαν αυτούς της C με μόνη διαφορά ότι το μέγεθός τους (σε bytes) είναι γνωστό και ίδιο σε όλες τις υλοποιήσεις της JAVA.

| Τύπος | Όνομα | Μέγεθος |
|---|---|---|
| Byte | Byte | 8-bit signed, -128..127 |
| Short | Short | 16-bit signed |
| Int | Ακέραιος | 32-bit signed |
| Long | εκτεταμένος ακέραιος | 64-bit signed |
| Float | πραγματικός (κινητής. υποδιαστολής) | 32-bit signed |
| Double | διπλής ακρίβειας | 64-bit signed |
| Char | unicode character | 16-bit |
| Boolean | Boolean | true or false |

## 3.2 Αναγνωριστικά, literals, σχόλια, διαχωριστές

Τα αναγνωριστικά στην JAVA είναι οποιαδήποτε ακολουθία των χαρακτήρων A..Z, a..,z, 0..9, _, $, η οποία ξεκινά με χαρακτήρα γράμμα (κεφαλαίο ή μικρό) ή _ ή $, αλλά όχι με ψηφίο (0..9). Επίσης δεν θεωρούνται αναγνωριστικά οι δεσμευμένες λέξεις της JAVA.

Τα literals στην JAVA είναι πολλών τύπων, πχ αριθμητικά, λογικά, χαρακτήρων κλπ. Μερικά παραδείγματα φαίνονται παρακάτω :

**Integer literals (ακαίρεοι αριθμοί)**

```
324              // decimal
0xABCD           // hexadecimal
032              // octal
2L               // long decimal integer
```

**Float point literals (πραγματικοί αριθμοί)**

```
3.1415  // float
3.1E12
.1e12
2E12
2.0d or 2.0D          //double
2.0f or 2.0F or 2.0   //float
```

**Boolean literals (λογικά δεδομένα)**

```
true, false           // boolean
```

**Character literals (χαρακτήρες)**

| | | | |
|---|---|---|---|
| continuation | <newline> | \ | // character literals |
| new-line | NL (LF) | \n | |
| horrizontal tab | HT | \t | |
| back space | BS | \b | |
| carriage return | CR | \r | |
| form feed | FF | \f | |
| backslash | \ | \\ | |
| single quote | ' | \' | |
| double quote | " | \" | |
| octal bit pattern | 0ddd | \ddd | |
| hex bit pattern | 0xdd | \xdd | |
| unicode char | 0xdddd | \udddd | |

πχ

'\n' , '\23', 'a'

**String literals (Συμβολοσειρές)**

```
""                  \\ the empty string
"\""
"This is a string"
"This is a \
two-line string"
```

Τα σχόλια στην JAVA υποδηλώνονται με τρεις τρόπους.

1. Με ζεύγος από /* και */ όπως στην C.
2. Με // όπου το κείμενο από το // έως το τέλος της τρέχουσας γραμμής θεωρείται ως σχόλιο
3. Με ζεύγος /** και */. Αυτό είναι πανομοιότυπο με το /* και */ αλλά χρησιμοποιείται από το javadoc για δημιουργία documentation.

Διαχωριστικά (separators) στη JAVA είναι οι χαρακτήρες :

+ - ! % ^ & * | ~ / > <

( ) { } [ ] ; ? : , . =

και επίσης το κενό (SPACE), ο οριζόντιος στηλογνώμονας HT ή \t, η αλλαγή γραμμής LF ή \n καθώς και τα σχόλια.

## 3.3 Τελεστές

Οι αριθμητικοί τελεστές είναι οι ακόλουθοι:

| Τελεστής | Περιγραφή |
|---|---|
| + | Πρόσθεση |
| - | Αφαίρεση |
| * | Πολλαπλασιασμός |
| / | Διαίρεση |
| % | Υπόλοιπο |
| ^ | Ύψωση σε δύναμη |

Οι συσχετιστικοί τελεστές είναι οι ακόλουθοι:

| Τελεστής | Περιγραφή |
|---|---|
| <= | Μικρότερο ή ίσο |
| < | Μικρότερο |
| >= | Μεγαλύτερο ή ίσο |
| > | Μεγαλύτερο |

Οι τελεστές ισότητας είναι οι ακόλουθοι:

| Τελεστής | Περιγραφή |
|---|---|
| != | Άνισο με |
| == | Ίσο με == |

Οι λογικοί τελεστές είναι οι ακόλουθοι:

| Τελεστής | Περιγραφή |
|---|---|
| && | ΚΑΙ (AND) |
| \|\| | Η (OR) |
| ! | ΟΧΙ (NOT) |

Τους συσχετιστικούς τελεστές, τους τελεστές ισότητας και τους λογικούς τελεστές τους συναντάμε κυρίως στις δομές ελέγχου (βλέπε παρακάτω). Οι παραπάνω τελεστές

χρησιμοποιούνται για συγκρίσεις μεταξύ αριθμών. Εάν η σύγκριση είναι αληθής τότε το αποτέλεσμα είναι η τιμή **true** διαφορετικά εάν είναι ψευδής τότε το αποτέλεσμα είναι η τιμή **false**.

Ο τελεστής αντιστοίχησης είναι ο:

| Τελεστής | Περιγραφή |
|---|---|
| = | Τελεστής αντιστοίχησης |

Ο τελεστής αύξησης και ο τελεστής μείωσης είναι οι ακόλουθοι:

| Τελεστής | Περιγραφή |
|---|---|
| ++ | αύξηση κατά 1 |
| -- | μείωση κατά 1 |

Οι τελεστές ++ και -- χρησιμοποιούνται όταν θέλουμε να προσθέσουμε ή να αφαιρέσουμε το 1 από μία μεταβλητή. Έτσι

το ++a; ισοδυναμεί με το a=a+1;
ενώ το --a; ισοδυναμεί με το a=a-1;

Οι τελεστές ++ και -- μπορούν να χρησιμοποιηθούν είτε ως προθεματικοί τελεστές (δηλ. πριν την μεταβλητή, πχ ++a ή--a) είτε ως μεταθεματικοί (δηλ. μετά την μεταβλητή, πχ a++ ή a--).

Στην παράσταση ++a η τιμή του a αυξάνει πριν χρησιμοποιηθεί η τιμή της.
Στην παράσταση a++ η τιμή του a αυξάνει αφού χρησιμοποιηθεί η τιμή της.

Παράδειγμα:
Έτσι έστω ότι το a ισούται με 5 δηλαδή υπάρχει η δήλωση:

```
a = 5;
```

τότε η δήλωση

```
b = a++;
```

δίνει στο b την τιμή 5

ενώ η δήλωση

```
b=++a;
```

δίνει στο b την τιμή την τιμή 6. Το a και στις δύο περιπτώσεις γίνεται 6.

Οι τελεστές αντικατάστασης είναι οι ακόλουθοι

| Τελεστής | Περιγραφή |
|---|---|
| += | Τελεστής πρόσθεσης και αντιστοίχησης |
| -= | Τελεστής αφαίρεσης και αντιστοίχησης |
| *= | Τελεστής πολ/μου και αντιστοίχησης |
| /= | Τελεστής διαίρεσης και αντιστοίχησης |
| %= | Τελεστής υπολοίπου και αντιστοίχησης |

Το a += b; ισοδυναμεί με το a = a+b;
Το a -= b; ισοδυναμεί με το a = a-b;
Το a *= b; ισοδυναμεί με το a = a*b;
Το a /= b; ισοδυναμεί με το a = a/b;
Το a %= b; ισοδυναμεί με το a = a %b;

Οι τελεστές πράξεων με bits είναι οι ακόλουθοι :

| Τελεστής | Περιγραφή |
|---|---|
| & | AND για bit |

| | |
|---|---|
| \| | OR για bit |
| ^ | XOR για bit |
| ! | NOT για bit |
| << | Ολίσθηση αριστερά |
| >> | Ολίσθηση δεξιά |

Οι παραπάνω τελεστές αφορούν πράξεις σε επίπεδο bits.
Οι τελεστές &, |, ^ και ~ αντιστοιχούν στις απλές πράξεις της άλγεβρας Boole.
Οι τελεστές >> και << προκαλούν ολίσθηση στα δεξιά και στα αριστερά αντίστοιχα.

Έτσι για παράδειγμα εάν η μεταβλητή a είναι ο δυαδικός αριθμός 01101000 τότε
Η δήλωση

```
b = a >> 2;
```

δίνει στη μεταβλητή b την τιμή 00011010.

Προτεραιότητα των τελεστών

| **Προτεραιότητα τελεστών** |
|---|
| **. [ ] ( )** |
| **++ -- ! ~ instanceof** |
| *** / %** |
| **+ -** |
| **<< >> >>>** |
| **< > <= >=** |
| **== !=** |
| **&** |
| **^** |
| **\|** |
| **&&** |
| **\|\|** |
| **? :** |
| **= op=** |
| **,** |

Οι προτεραιότητα των τελεστών μικραίνει καθώς κατεβαίνουμε στον πίνακα, πχ το ***** έχει μεγαλύτερη προτεραιότητα από το **!=**. Σε περίπτωση ύπαρξης σε μια παράσταση τελεστών με την ίδια προτεραιότητα οι πράξεις γίνονται από αριστερά προς δεξιά.

Τέλος το **op=** είναι εξής τελεστές : **+= -= *= /= %= &= |= ^= <<= >>= >>>=**

## 3.4 Συμβολοσειρές (Strings)

Χαρακτήρες
char c='A';

Οι συμβολοσειρές (Strings) είναι ακολουθίες χαρακτήρων. Οι συμβολοσειρές στην Java είναι ένα αντικείμενα της κλάσης String.

Παράδειγα String:

```
String ntua="National Technical University of Athens";
```

Ή εναλακτικά όπως είπαμε στο δεύτερο κεφάλαιο

```
String ntua = new String("National Technical University of Athens");
```

Η εμφάνιση συμβολοσειρών γίνεται χρησιμοποιώντας την μέθοδο println() η οποία πραγματοποιεί αλλαγή γραμμής μετα την εκτύπωση

Παράδειγμα εκτύπωσης:

```
System.out.println(ntua); //Αλλαγή γραμμής στην εκτύπωση
```

Ή εναλακτικά

```
System.out.println("National Technical University of Athens"); //Αλλαγή γραμμής στην
                                                                  // εκτύπωση
```

Για την εμφάνιση συμβολοσειρών χωρίς αλλάγή γραμμής μετά την εκτύπωση χρησιμοποιείται η μέθοδος print()

Παράδειγμα εκτύπωσης:

```
System.out.print("National");
System.out.print(" Technical ");
System.out.println(" University ");
System.out.println(" of ");
System.out.println(" Athens ");
```

Παράδειγμα επικόλλησης συμβολοσειρών

```
String ntuaHMMY = ntua + " HMMY Department";
```

Η συμβολοσειρά ntuaHMMY περιέχει το:
"National Technical University of Athens HMMY Department".

Απλές μέθοδοι στις συμβολοσειρές:

Έστω οι συμβολοσειρές s1 και s2.

| Μέθοδος | Ενέργεια |
|---|---|
| s1.length(); | Προσδιορισμός μήκους (αριθμός χαρακτήρων) συμβολοσειράς s1 |
| s2=s1.toUpperCase(); | Μετατροπή συμβολοσειράς s1 σε κεφαλαία |
| s2=s1.toLowerCase(); | Μετατροπή συμβολοσειράς s1 σε πεζά |
| s2.equals(s1) | Σύγκριση συμβολοσειρών s1 και s2 |
| int a=s2.indexOf(s1); | Το a περιέχει την θέση της s2 στην s1 |
| >> | Ολίσθηση δεξιά |

## 3.5 Εντολές και Δομές ελέγχου

Οι εντολές είναι σχεδόν πανομοιότυπες με αυτές της C, με μόνη εξαίρεση ότι στη JAVA μπορούμε να κάνουμε δηλώσεις μεταβλητών όχι μόνο στην αρχή ενός block αλλά και σε οποιοδήποτε σημείο του, αρκεί η δήλωση να γίνει πριν από τη χρήση της εν λόγω μεταβλητής. Να σημειώσουμε επίσης ότι ισχύουν όλοι οι κανόνες εμβέλειας, τοπικών μεταβλητών και περάσματος παραμέτρων (σε μεθόδους) της C.

Όσον, τώρα, αφορά τις δομές ροής ελέγχου (if - else, for, while, do - while και switch), οι μόνες αλλαγές που έχουν γίνει σε σχέση με τη C είναι το ότι στο if-else, while, do-while οι εκφράσεις που αποτιμώνται προκειμένου να γίνει άλμα ή επανάληψη, θα πρέπει να είναι boolean και όχι αριθμητικές. Δηλαδή :

*while (1) {...}  // ΕΙΝΑΙ ΛΑΘΟΣ*
*while (true) {....} // ΕΙΝΑΙ ΣΩΣΤΟ*

Στη συνέχεια θα παρουσιάσουμε συνοπτικά τις δομές ροής ελέγχου προγράμματος της Java. Υπάρχουν γενικά δύο ήδη δομών ελέγχου ροής (control flow):

- Οι δομές επιλογής και
- Οι δομές επανάληψης

Ο ακόλουθος πίνακας συνοψίζει αυτές τις δομές για τη Java

| Είδος δομής ελέγχου ροής | Δομή ελέγχου ροής |
|---|---|
| Δομές επιλογής | if-else |
| | switch-case |
| Δομές επανάληψης | for |
| | while |
| | do-while |

Πέρα από τις εντολές του πιο πάνω πίνακα ο έλεγχος ροής σε ένα πρόγραμμα Java μπορεί να μεταφερθεί και σε κάποιο άλλο σημείο εξαιτίας της πρόκλησης μιας εξαίρεσης. Αλλά για τις εξαιρέσεις (exceptions) και τους χειριστές τους (exception handlers) θα μιλήσουμε σε επόμενα μαθήματα.

Επίσης κάποιες άλλες εντολές πέρα από τις ίδιες τις δομές ελέγχου ροής που είδαμε προηγουμένως είναι οι: break, continue και return που μεταφέρουν το έλεγχο ροής σε άλλα σημεία του προγράμματος. Την χρήση αυτών των εντολών θα τη δούμε στη συνέχεια.

Η Java επιτρέπει και τη χρήση ετικετών αλλά δεν υποστηρίζει την εντολή goto. Αντί αυτής μπορούν και με τις ετικέτες να χρησιμοποιηθούν οι εντολές break και continue.

**Η δομή επιλογής if-else**

Η εντολή if μας βοηθάει στον έλεγχο μίας λογικής έκφρασης (της συνθήκης) και αν είναι αληθής εκτελούνται μία ή περισσότερες εντολές ενώ αν είναι ψευδής δεν εκτελούνται. Αν οι εντολές είναι περισσότερες από μία τότε θα πρέπει οι εντολές να περικλειστούν ανάμεσα από άγκιστρα (τα οποία ομαδοποιούν εντολές).
Το συντακτικό της εντολής if είναι το ακόλουθο:

```
if  (συνθήκη)
   εντολές
```

π.χ.

```
if (x!=0) {
   System.out.println("Το x δεν είναι 0");
   y = 1/x;
}
```

Η εντολή if έχει επίσης και τη φράση else με την οποία επιτρέπεται η εκτέλεση μίας ομάδας εντολών εναλλακτικά αν δεν ισχύει η συνθήκη. Φυσικά μπορούμε να έχουμε και εμφωλευμένα if δηλαδή if μέσα σε άλλα if όσες φορές θέλουμε.
Το συντακτικό της if επαυξημένο με τη φράση else είναι το ακόλουθο:

```
if (συνθήκη)
```

```
 ομάδα-εντολών-1
else
 ομάδα-εντολών-2
```

π.χ.

```
if (x!=0) {
    System.out.println("Το x δεν είναι 0");
    y=1/x;
}
else {
   System.out.println("Το x είναι 0");
   y=0;
}
```

**Η δομή επιλογής switch-case**

Όταν οι εναλλακτικές περιπτώσεις που πρέπει να ελέγξουμε με την if είναι πάρα πολλές και αφορούν τον έλεγχο για ισότητα της τιμής μιας μεταβλητής ή μιας έκφρασης με κάποιες τιμές προτιμάται η switch-case η οποία έχει την ακόλουθη γενική μορφή:

```
switch (έκφραση) {
    case τιμή-1:
        εντολές-1; [break;]
           …
    case τιμή-ν:
        εντολές-ν; [break;]
    [default:
        εντολές; [break;]]
}
```

Η switch-case αποτιμά την τιμή της έκφρασης που ελέγχεται και στη συνέχεια διατρέχει με τη σειρά όλες τις περιπτώσεις που δίνονται. Αν κάποια περίπτωση βρεθεί αληθής τότε εκτελούνται οι εντολές που δίνονται μετά την άνω-κάτω τελεία γι' αυτή τη περίπτωση. Αν καμία περίπτωση δεν βρεθεί αληθής τότε εκτελείται η default περίπτωση αν υπάρχει.
Προσοχή χρειάζεται στη χρήση του break που είναι μεν προαιρετική αλλά απαιτείται τις περισσότερες φορές, μια και αν δεν υπάρχει τότε θα εκτελεστούν και οι εντολές τις επόμενης περίπτωσης (χωρίς να ελεγχθεί η τιμή της) και πιθανώς και άλλων, μέχρι να βρεθεί το επόμενο break ή να τελειώσει η switch-case.
Παρόλα αυτά ενδέχεται να υπάρχουν κάποιες περιπτώσεις που αυτό θα ήταν βολικό, όπως δείχνει το ακόλουθο τμήμα κώδικα που υπολογίζει τις μέρες ενός μήνα ανάλογα με το ποιος μήνας είναι (η τιμή της μεταβλητής month) και το αν το έτος (year) είναι δίσεκτο:

```
...
switch (month) {
   case 1:
   case 3:
   case 5:
   case 7:
   case 8:
   case 10:
   case 12:
      numDays = 31;
      break;
   case 4:
   case 6:
   case 9:
   case 11:
      numDays = 30;
      break;
```

```
   case 2:
      if ( ((year % 4 == 0) && !(year % 100 == 0))
            || (year % 400 == 0) )
         numDays = 29;
      else
         numDays = 28;
      break;
}
...
```

**Η δομή επανάληψης for**

Η εντολή for είναι χρήσιμη για την επανάληψη μιας σειράς εντολών όταν είναι γνωστό εκ των προτέρων πόσες φορές θέλουμε να επαναληφθούν. Έτσι συνήθως το for ελέγχεται από ένα μετρητή ο οποίος μεταβάλλεται από μία αρχική τιμή μέχρι να ξεπεράσει μία τελική τιμή. Στη for επίσης καθορίζεται το πόσο θα μεταβάλλεται αυτός ο μετρητής σε κάθε βήμα.
Η γενική μορφή της for είναι η ακόλουθη:

```
for (αρχικοποίηση; τερματισμός; αύξηση)
   εντολές;
```

Για παράδειγμα το ακόλουθο τμήμα κώδικα εμφανίζει στην οθόνη τους αριθμούς από το 1 μέχρι το 10:

```
...
for (int i=1; i<=10; i++)
   System.out.println(i);
...
```

Στη φράση της αρχικοποίησης το i δηλώνεται (η εμβέλειά του είναι το for loop) και αρχικοποιείται στο1.
Στη φράση του τερματισμού το i ελέγχεται για το αν έχει ξεπεράσει το 10. Άρα ο συγκεκριμένος βρόχος θα επαναληφθεί μέχρι το i να ξεπεράσει το 10.
Στη φράση της μεταβολής το i αυξάνεται κατά 1. Αυτό σημαίνει ότι το i θα πάρει διαδοχικά τις τιμές 1, 2, …, 10
Σε κάθε βήμα της επανάληψης ελέγχεται η τιμή του i. Αν το i ξεπεράσει το 10 ο βρόχος τερματίζεται, αν όχι αυξάνεται κατά 1 και εκτελείται ξανά η μία και μοναδική εντολή αυτού του βρόχου.

**Η δομή επανάληψης while**

Η εντολή for που είδαμε προηγουμένως είναι κατάλληλη όταν γνωρίζουμε πόσες επαναλήψεις θα γίνουν. Αν δεν γνωρίζουμε πόσες επαναλήψεις θα γίνουν μία καλύτερη εντολή είναι η while.
Η while έχει την ακόλουθη γενική μορφή:

```
while (συνθήκη)
   εντολές;
```

Η εντολή ή εντολές που ακολουθούν το while θα εκτελεστούν όσο η συνθήκη είναι αληθής. Ο βρόχος δηλαδή θα τερματιστεί όταν η συνθήκη - που είναι μία boolean έκφραση - γίνει ψευδής

Στο ακόλουθο τμήμα κώδικα ελέγχεται αν η τιμή x βρίσκεται μέσα στο πίνακα numbers. Κάνουμε δηλαδή μία σειριακή αναζήτηση στα στοιχεία του πίνακα. Αν το στοιχείο βρεθεί η μεταβλητή found γίνεται true και βγαίνουμε από το βρόχο. Επίσης από το βρόχο θα βγούμε αν ελεγχθούν και τα δέκα στοιχεία του πίνακα και δεν έχει βρεθεί ακόμα το x.

```
…
boolean found = false;
int i=0;
```

```
while (!found && i!=10)
    if (numbers[i++] == x)
        found = true;
…
```

**Η δομή επανάληψης do-while**

Υπάρχουν κάποιες περιπτώσεις στις οποίες θα θέλαμε οι εντολές μέσα στο βρόχο να εκτελεστούν τουλάχιστον μία φορά και στη συνέχεια να ελεγχθεί η συνθήκη εξόδου. Σ' αυτές τις περιπτώσεις προτιμάται η χρήση της do-while αντί της while.
Η do-while έχει την ακόλουθη γενική μορφή:

```
do
    εντολές;
while (συνθήκη);
```

Στην do-while πρώτα εκτελούνται οι εντολές και στη συνέχεια ελέγχεται η συνθήκη. Ο βρόχος τερματίζεται αν η συνθήκη βρεθεί ψευδής.
Η do-while δεν χρησιμοποιείται πολύ συχνά αλλά έχει κι αυτή τις χρήσεις της.

Για παράδειγμα όταν διαβάζουμε χαρακτήρες από ένα αρχείο μέχρι να διαπιστώσουμε το τέλος του αρχείου θα πρέπει να διαβάσουμε τουλάχιστον ένα χαρακτήρα, όπως δείχνει και το ακόλουθο τμήμα κώδικα:

```
...
int c;
Reader in;
...
do {
   c = in.read();
   ...
} while (c != 1);
...
```

**Εντολές διακλάδωσης**

Η Java έχει τρεις εντολές διακλάδωσης οι οποίες είναι βολικές σε αρκετές περιπτώσεις και ιδιαίτερα με τις επαναληπτικές δομές που έχουμε συζητήσει.
Οι εντολές διακλάδωσης είναι οι: break, continue και return
Την εντολή break ήδη την είδαμε σε μία χρήση της με την switch-case. Μία ακόμα αρκετά συνήθη χρήση της break είναι η χρήση της για άμεση έξοδο από κάποιο βρόχο.

Για παράδειγμα η σειριακή αναζήτηση που είδαμε με το while θα μπορούσε να γίνει όπως στο For.java ως εξής:

```
int i;
boolean found = false;
for (i=0; i<10; i++)
   if (numbers[i]==x) {
      found = true;
      break;
   }
```

Η χρήση της continue μέσα σε ένα βρόχο προκαλεί την άμεση αποτίμηση και πάλι της συνθήκης εξόδου. Για να γίνει αυτό στις εντολές for και while ο έλεγχος μεταφέρεται στη πρώτη γραμμή του βρόχου ενώ στο do-while στη τελευταία. Στη συνέχεια αποτιμάται και πάλι η συνθήκη τερματισμού και η επαναληπτική διαδικασία τερματίζεται ή συνεχίζεται ανάλογα με την τιμή της συνθήκης (true ή false) ως συνήθως.

Οι εντολές break και continue έχουν και μία μορφή για ετικέτες (labeled break και labeled continue) που δεν θα συζητήσουμε εδώ μια και στη πράξη οι ετικέτες δεν χρησιμοποιούνται αφού οι υπόλοιπες δομές που είδαμε αρκούν για όλες τις πιθανές χρήσεις, χωρίς να εμφανίζουν πολλά από τα προβλήματα που πιθανώς να εμφανιστούν με την χρήση των ετικετών.

Η εντολή return διακόπτει αμέσως την εκτέλεση της μεθόδου μέσα στην οποία βρίσκετε και σε περίπτωση που η μέθοδος επιστρέφει κάποια τιμή, επιστρέφει την τιμή που βρίσκεται στα δεξιά της. Ο τύπος της επιστρεφόμενης τιμής πρέπει να είναι ίδιος με τον τύπο-αποτελέσματος που δηλώθηκε στην δήλωση της μεθόδου.

Στην ακόλουθη μέθοδο η εντολή return επιστρέφει ένα String που συγκεκριμένα την τιμή που περιέχεται στην μεταβλητή name. Η εκτέλεση της μεθόδου getName( ) τερματίζει ακόμα και αν υπάρχουν και άλλες εντολές κάτω από την εντολή: **return name;**

```
public String getName()
{
        ...
        return name;
}
```

Στην παρακάτω μέθοδο η εντολή: **return interest;** τερματίζει την μέθοδο και επιστρέφει τον ακέραιο που περιέχει η μεταβλητή interest;

```
public int calculateInterest( )
{
        ...     return interest;
}
```

## 3.6 Πίνακες

Οι πίνακες στην JAVA δηλώνονται ως εξής:

```
        <Τύπος ή κλάση> []      <μεταβλητή>;
ή
        <Τύπος ή κλάση>                 <μεταβλητή>[];
```

Παραδείγματα

```
        int[]   nums;
        byte    buff[];
        float   matrix[][];     //Διδιάστατος πίνακας
        A       a[];
        B[]     b;
```

Όπως βλέπουμε ο πίνακας δεν δημιουργείται κατά την δήλωσή του (αφού άλλωστε δεν του έχουμε προσδιορίσει το μέγεθος που θα έχει). Για την κατασκευή του πίνακα κάνουμε τα εξής :

```
        <μεταβλητή πίνακα> = new <τύπος ή κλάση> [ <μέγεθος> ];
```

όπου το <μέγεθος> είναι τύπου byte, short, int ή long.

Παραδείγματα

```
        int[]   K;
        K = new int[200];
        byte    buff[] = new byte[1024];
        A[]     a = new A[4];

        float   matrix[][];
```

```
matrix = new float[30][];
matrix = new float[0][40]; ...  matrix = new float[29][40];
```
ή μπορούμε και απευθείας να δηλώσουμε:
```
matrix = new float[30][40];
```

Αναφορά σε στοιχείο του πίνακα :
```
a[2] = 5;
matrix[0][8] = (float) (buff[19] + buff[25]);
```

Όταν ένας πίνακας είναι μεγέθους N τότε τα έγκυρα indexes θα είναι από 0 έως N-1. Κάθε άλλη τιμή θα προκαλέσει compile-time ή run-time error.

Οι μέθοδοι επιτρέπεται να παίρνουν σαν όρισμα ένα πίνακα ή να επιστρέφουν ένα πίνακα. Αυτό γίνεται ως εξής :
```
int[]  give_ints() {
        int        I[] = new int[3];
        I[0] = I[1] = I[2] = 7;
        return I;
}
```
ή
```
int give_ints() [] {....}

void get_ints(int[] i) {
        i[0] = i[1]+i[2];
}
```
ή
```
void get_ints(int i[]) { ... }
```

Σε αυτό το σημείο να πούμε ότι στην JAVA ένας πίνακας είναι ένα αντικείμενο. Η κλάση ενός τέτοιου αντικειμένου (πίνακα) δημιουργείται αυτόματα από την γλώσσα και είναι υποκλάση της κλάσης Array. Δηλαδή όταν ορίζουμε μία κλάση, η JAVA ορίζει αυτόματα μία νέα κλάση, (υποκλάση της Array), η οποία θα δίνει πίνακες με στοιχεία αντικείμενα της κλάσης που εμείς ορίσαμε.

ΠΡΟΣΟΧΗ : όταν γράφουμε int a[]; τότε το αντικείμενο - πίνακας θα είναι το a. Αντίθετα το στοιχείο a[0], ... κλπ θα είναι απλοί int που περιέχει ο πίνακας όπως ακριβώς τους ξέρουμε.

Τέλος κάθε πίνακας περιέχει μια μεταβλητή length η οποία δίνει το μέγεθος του πίνακα. Αυτή η μεταβλητή χρησιμοποιείται ως εξής :
```
a.length            //η δήλωση αυτή επιστρέφει το μέγεθος του πίνακα
if  (a.length < 1024) {...}
for (int i=0; i<a.length; i++) {
        System.out.println(a[i]);
}
```