2. ΕΙΣΑΓΩΓΗ ΣΤΗ JAVA

Η γλώσσα προγραμματισμού Java

Βασικό συντακτικό, ορισμός μεταβλητών, έλεγχος ροής

ΜΕΤΑΒΛΗΤΕΣ-ΑΝΤΙΚΕΙΜΕΝΑ

Η μεταβλητή αντικείμενο Scanner

- Στα προηγούμενα παραδείγματα ορίσαμε μεταβλητές τύπου Scanner. Οι μεταβλητές αυτές είναι αντικείμενα της κλάσης Scanner.
- Τις ορίσαμε για να χρησιμοποιήσουμε τις μεθόδους της κλάσης Scanner ώστε να διαβάσουμε από την είσοδο.
- Όταν τις ορίσαμε χρησιμοποιήσαμε και την εντολή new:

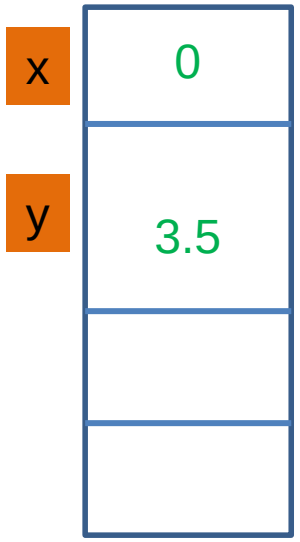
```
Scanner input = new Scanner(System.in);
```

- Τι κάνει η εντολή new? Δημιουργεί το αντικείμενο

Μεταβλητές πρωταρχικού τύπου

- Όταν ορίζουμε μια μεταβλητή πρωταρχικού τύπου (π.χ., **`int x`**, **`double y`**):
  - Δεσμεύουμε ένα κουτάκι κατάλληλου μεγέθους
    - 4 bytes για ένα int, 8 bytes για double
  - Το κουτάκι παίρνει το όνομα της μεταβλητής
    - Έχουμε πλέον τα κουτάκια **`x`**,**`y`**
- Όταν γίνει ανάθεση τιμής, μέσα στο κουτάκι αποθηκεύουμε την τιμή της μεταβλητής.
  - Το κουτάκι **`x`** περιέχει την τιμή 1, το κουτάκι **`y`** περιέχει την τιμή 3.5

```
int x = 1;
double y = 3.5;
```



Μεταβλητές-αντικείμενα

- Τα πράγματα είναι διαφορετικά με τις μεταβλητές-αντικείμενα.
- Μια μεταβλητή-αντικείμενο πρέπει να την ορίσουμε και μετά να δημιουργήσουμε το αντικείμενο καλώντας την new.
- Παράδειγμα: Η κλάση Scanner

Ορισμός αντικειμένου

- Όταν ορίζουμε μια μεταβλητή μίας κλάσης (π.χ., `Scanner input`), πριν καλέσουμε τη new:
  - Δεσμεύουμε ένα κουτάκι μεγέθους 4 ή 8 bytes (για 32-bit, ή 64-bit λειτουργικά) ανεξάρτητα από την κλάση
  - Το κουτάκι παίρνει το όνομα της μεταβλητής
    - Το κουτάκι με διεύθυνση 0000 έχει το όνομα `input`
  - Η τιμή στο κουτάκι αρχικοποιείται στην τιμή `null`
    - Η τιμή null είναι η τιμή ενός "κενού" αντικειμένου που δεν έχει δημιουργηθεί ακόμη.

```
Scanner input;
```

input

Διεύθυνση μνήμης	Περιεχόμενα μνήμης
0000	null
0001	
0010	
0011	
0100	
0101	
0110	
0111	

Δημιουργία αντικειμένου – κλήση της new

- Η κλήση της new δεσμεύει τον χώρο μνήμης που χρειάζεται για την αποθήκευση του αντικειμένου τύπου Scanner.
  - Ο χώρος μνήμης αυτός είναι στην διεύθυνση 0011

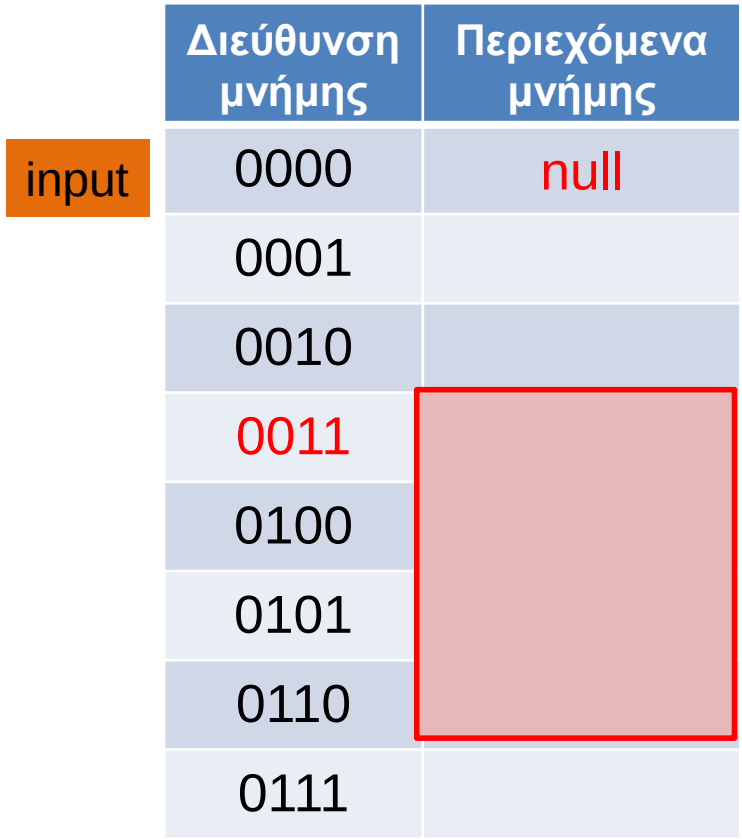
```
Scanner input;
input =
    new Scanner(System.in);
```

	Διεύθυνση μνήμης	Περιεχόμενα μνήμης
input	0000	null
	0001	
	0010	
	0011	
	0100	
	0101	
	0110	
	0111	

Δημιουργία αντικειμένου – κλήση της new

- Η κλήση της new δεσμεύει τον χώρο μνήμης που χρειάζεται για την αποθήκευση του αντικειμένου τύπου Scanner.
  - Ο χώρος μνήμης αυτός είναι στην διεύθυνση 0011
- Η new επιστρέφει την διεύθυνση της χώρου μνήμης που δέσμευσε
  - Στο παράδειγμα επιστρέφει 0011

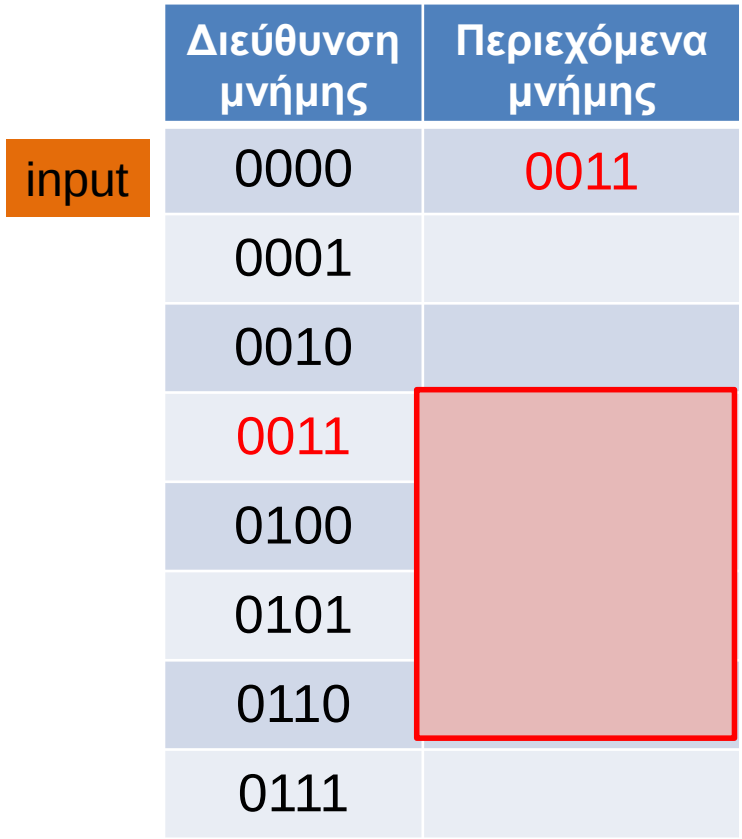
```
Scanner input;
input =
    new Scanner(System.in);
```



Δημιουργία αντικειμένου – κλήση της new

- Η κλήση της new δεσμεύει τον χώρο μνήμης που χρειάζεται για την αποθήκευση του αντικειμένου τύπου Scanner.
  - Ο χώρος μνήμης αυτός είναι στην διεύθυνση 0011
- Η new επιστρέφει την διεύθυνση της χώρου μνήμης που δέσμευσε
  - Στο παράδειγμα επιστρέφει 0011
- Η διεύθυνση αποθηκεύεται στο κουτάκι input

```
Scanner input;
input =
    new Scanner(System.in);
```



Αναφορές

- Οι μεταβλητές-αντικείμενα κρατάνε την διεύθυνση του χώρου που έχει δεσμευτεί για το αντικείμενο.
- Η διεύθυνση αυτή λέγεται αναφορά.
- Λέμε ότι η μεταβλητή δείχνει στον χώρο μνήμης του αντικειμένου.
- Συγκριτικά οι μεταβλητές πρωταρχικού τύπου κρατάνε την τιμή της μεταβλητής.

```
Scanner input;
input =
    new Scanner(System.in);
int x = 2;
```

	Διεύθυνση μνήμης	Περιεχόμενα μνήμης
input	0000	0011
	0001	
	0010	
	0011	
	0100	
	0101	
	0110	
x	0111	2

BPONXOI
IF-THEN-ELSE

Λογικές εκφράσεις και Λογικοί τελεστές

- Λογικές σταθερές/τιμές:
  - true: αληθές, false: ψευδές

- Λογικοί τελεστές για λογικές σταθερές και λογικές εκφράσεις
  - Άρνηση: `!B`
  - Λογικό ΚΑΙ: `(A && B)`
  - Λογικό Ή: `(A || B)`

- Λογική έκφραση: μια έκφραση που αποτιμάται σε μια λογική τιμή
- Παραδείγματα:
  - Έλεγχος ισότητας δύο μεταβλητών x,y πρωταρχικού τύπου: `(x == y)`
  - Χρήση του ΚΑΙ: `(x > 10 && x < 20)`
  - Χρήση του Ή: `(x < 10 || x > 20)`
  - Χρήση του NOT: `(!(x > 10 && x < 20))`
  - Κλήση λογικής συνάρτησης: `s.equals('hello");`

Έλεγχος ισότητας για πρωταρχικούς τύπους

- Για δύο μεταβλητές x,y πρωταρχικού τύπου (int, double, boolean)
  - Έλεγχος ισότητας: `(x == y)`
  - Έλεγχος ανισότητας: `(x != y)` ή `(!(x == y))`
  - Μεγαλύτερο/Μικρότερο ή ίσο: `(x <= y)`,`(x >= y)`
- Παραδείγματα:

```
int x = 10;
int y = 20;
boolean test1 = (x == y);
boolean test2 = (x != y);
boolean test3 = (x >= 10);
boolean test4 = (x%10 == 0 && y%10 == 0);
boolean test5 = (2*x+3*y > 50);
```

Έλεγχος ισότητας για αντικείμενα: equals

- Έλεγχος για μεταβλητές (αντικείμενα) που δεν είναι πρωταρχικού τύπου γίνεται με την μέθοδο equals :
  - Έλεγχος Ισότητας: `(x.equals(y))`
  - Ανισότητας: `(!x.equals(y))`
- Την μέθοδο equals ορίζεται όταν δημιουργούμε την κλάση.
- Αν δεν έχει οριστεί, θα την «ορίσει» η Java για σας, οπότε μπορείτε να την καλέσετε χωρίς να πάρετε compile error.
- Δεν κάνει αυτό που θέλουμε όμως. Ελέγχει αν οι δύο μεταβλητές αποθηκεύουν την ίδια αναφορά.

Έλεγχος ισότητας για Strings

- Αν έχουμε δύο μεταβλητές String για να ελέγξουμε αν έχουν την ίδια τιμή πρέπει να χρησιμοποιήσουμε την μέθοδο equals.
- Παράδειγμα:

```
String firstString = "abc";
String secondString = "ABC";
boolean test1 = firstString.equals(secondString);
boolean test2 = firstString.equals("abc");
```

- Η παρακάτω εντολή δεν είναι σωστή

```
boolean test3 = (firstString == secondString);
```

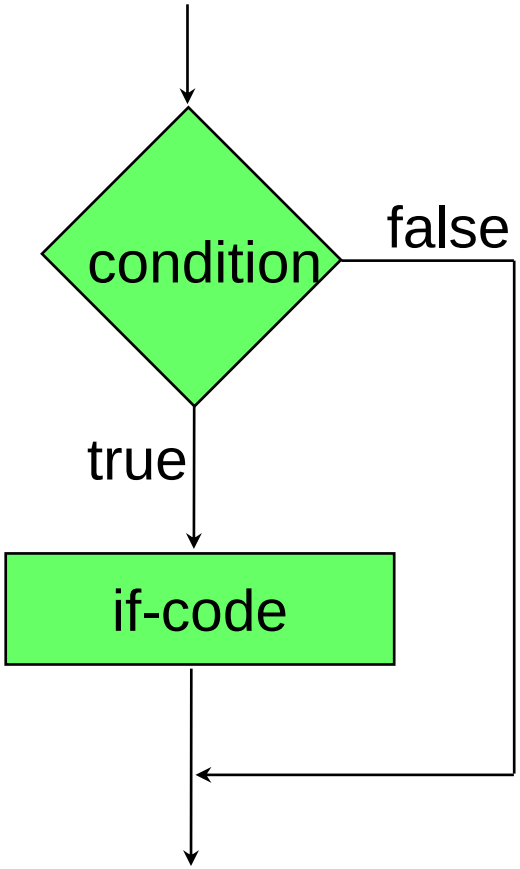
- Περνάει από τον compiler και σε κάποιες περιπτώσεις θα δουλέψει αλλά δεν κάνει αυτό που θέλουμε.

Βρόγχοι – Το if-then Statement

- Στην Java το if-then statement έχει το εξής συντακτικό

Η παρένθεση είναι απαραίτητη

```
if (condition)
{
    …if-code block…
}
```



- Αν η συνθήκη είναι αληθής τότε εκτελείται το block κώδικα if-code. Εκτελείται ότι είναι μέσα στα άγκιστρα
- Αν η συνθήκη είναι ψευδής τότε το κομμάτι αυτό προσπερνιέται και συνεχίζεται η εκτέλεση.

Το condition είναι μια λογική έκφραση που αποτιμάται σε true ή false

```
import java.util.Scanner;

class IfTest1
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        if (inputInt > 0){
            System.out.println(inputInt +
                                    " is positive");
        }
    }
}
```

```
import java.util.Scanner;

class IfTest1b
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        boolean inputIsPositive = (inputInt > 0);
        if (inputIsPositive == true){
            System.out.println(inputInt +
                                    " is positive");
        }
    }
}
```

```
import java.util.Scanner;

class IfTest1b
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        boolean inputIsPositive = (inputInt > 0)
        if (inputIsPositive){
            System.out.println(inputInt +
                                    " is positive");
        }
    }
}
```

Ακόμη και αν δεν το προσδιορίσουμε ελέγχει για true

Programming Style: Λογικές μεταβλητές

- Συνηθίζεται όταν ορίζουμε λογικές μεταβλητές το όνομα τους να είναι αυτό που εκφράζει την περίπτωση που η μεταβλητή αποτιμάται true.

```
int x = 10;
boolean xIsPositive = (x > 0);
boolean xIsNegative = (x < 0);
boolean xIsNotPositive = !xIsPositive;
```

- Αυτό βολεύει για την εύκολη ανάγνωση του προγράμματος όταν χρησιμοποιούμε την μεταβλητή

```
if(xIsPositive){
    System.out.println("Variable x is positive");
}
```

- Το ίδιο ισχύει και όταν αργότερα θα ορίζουμε μεθόδους που επιστρέφουν λογικές τιμές
  - Π.χ., για τα Strings υπάρχει η μέθοδος equals που γίνεται true όταν έχουμε ισότητα και η μέθοδος isEmpty που είναι true όταν έχουμε άδειο String.

```
import java.util.Scanner;

class IfTest1b
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        boolean inputIsPositive = (inputInt > 0)
        if (!inputIsPositive){
              System.out.println(inputInt +
                                  " is negative");
       }
    }
}
```

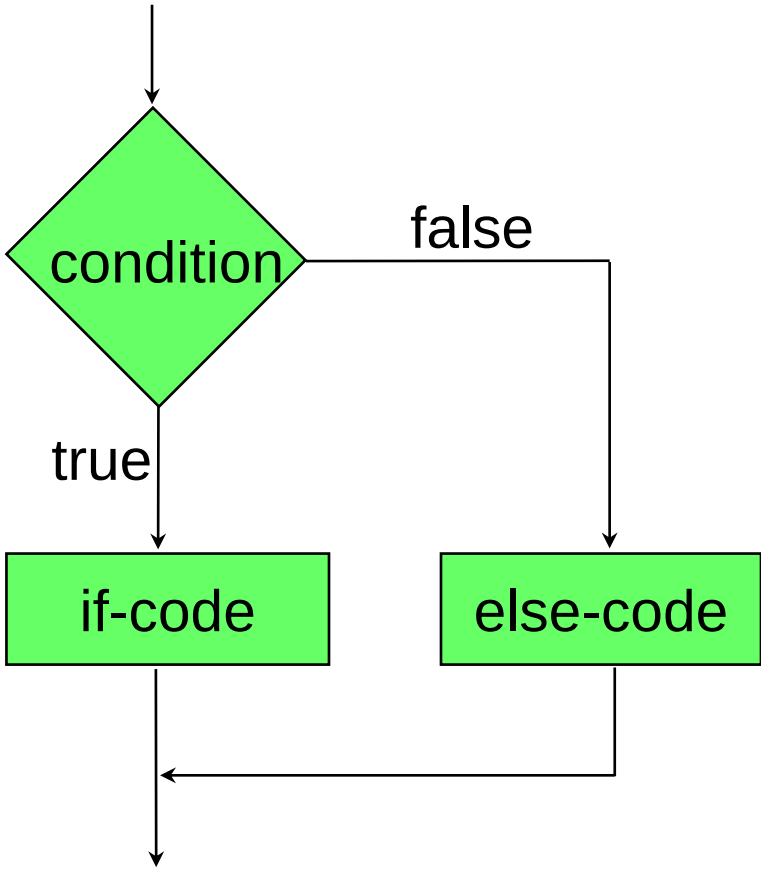
Έλεγχος για false

Βρόγχοι – Το if-then-else Statement

- Στην Java το if-then-else statement έχει το εξής συντακτικό

```
if (condition){
     …if-code block…
}else{
     …else-code block…
}
```

- Αν η συνθήκη είναι αληθής τότε εκτελείται το block κώδικα if-code
- Αν η συνθήκη είναι ψευδής τότε εκτελείται το block κώδικα else-code.
- Ο κώδικας του if-code block ή του else-code block μπορεί να περιέχουν ένα άλλο (φωλιασμένο (nested)) if statement
- Προσοχή: ένα `else` clause ταιριάζεται με το τελευταίο ελεύθερο `if` ακόμη κι αν η στοίχιση του κώδικα υπονοεί διαφορετικά.



```
import java.util.Scanner;

class IfTest2
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        if (inputInt > 0){
              System.out.println(inputInt +
                                  " is positive");
        }else{
              System.out.println(inputInt +
                              " is not positive");
        }
    }
}
```

If-Then-Else block

- Στα παραδείγματα που είδαμε ορίσαμε το if-code block και το else-code block χρησιμοποιώντας τα άγκιστρα { }
- Αν ο κώδικας μέσα στο block είναι μόνο μια εντολή τότε μπορούμε να μην βάλουμε τα άγκιστρα
- Αυτοί οι δύο κώδικες είναι ισοδύναμοι:

```
if (x > 0)
   s = "positive";
else
   s = "not posive";
```

```
if (x > 0){
   s = "positive";
} else {
   s = "not posive";
}
```

- Αυτό είναι εύκολο να προκαλέσει λάθη, και δεν θα το χρησιμοποιούμε ποτέ στα προγράμματα μας. Θα βάζουμε πάντα τα άγκιστρα
  - Εκτός από μια πολύ ειδική περίπτωση: else if

else if: έλεγχος πολλαπλών επιλογών

- Για να ελέγξουμε πολλαπλά ενδεχόμενα, μπορούμε να χρησιμοποιήσουμε την δομή:
  - if, else if, else if, …, else if, else
- Δεν ύπάρχει ειδική εντολή elif όπως στην python. Απλά χρησιμοποιούμε μια φωλιασμένη if μετά την else.
- Ο κώδικας στα αριστερά είναι ουσιαστικά ίδιος με αυτόν στα δεξιά.
- Χρησιμοποιούμε το αριστερό συντακτικό γιατί είναι πιο κατανοητό

```
if (x == 0){
    s = "zero";
}else if (x == 1){
    s = "one";
}else{
    s = "non binary";
}
```

```
if (x == 0){
    s = "zero";
}else{
      if (x == 1){
          s = "one";
      }else{
          s = "non binary";
      }
}
```

Έλεγχος πολλαπλών επιλογών

```
import java.util.Scanner;

class IfTest3
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        if (inputInt > 0){
              System.out.println(inputInt +
                                      " is positive");
        }else if (inputInt < 0){
              System.out.println(inputInt +
                                  " is not positive");
        }else{
              System.out.println(inputInt +  " is zero");
        }

    }
}
```

Προσοχή!

ΛΑΘΟΣ!

```
if( i == j )
    if ( j == k )
        System.out.print(
                "i equals k");
else
    System.out.print(
        "i is not equal to j");
```

Το else μοιάζει σαν να πηγαίνει με το μπλε else αλλά ταιριάζεται με το τελευταίο (πράσινο) if

ΣΩΣΤΟ!

```
if( i == j ){
    if ( j == k ){
        System.out.print(
                "i equals k");
    }
}
else {
    System.out.print(
        "i is not equal to j");
}
```

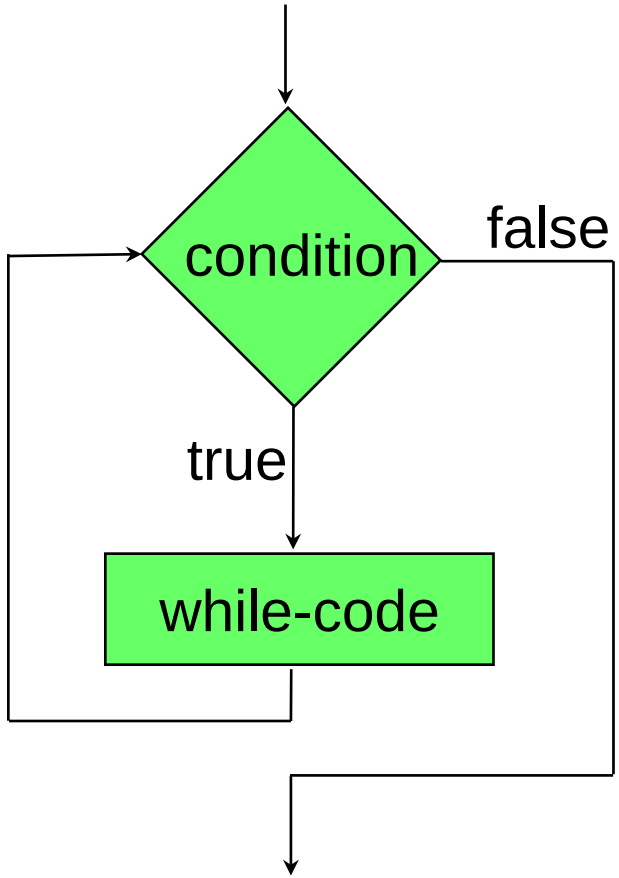
Πάντα να βάζετε { } στο σώμα των if-then-else statements.
Πάντα να στοιχίζετε σωστά τον κώδικα.

ΕΠΑΝΑΛΗΨΗ WHILE, FOR

Επαναλήψεις - While statement

- Στην Java το while statement έχει το εξής συντακτικό

```
while(condition)
{
    …while-code block…
}
```



- Αν η συνθήκη είναι αληθής τότε εκτελείται το block κώδικα while-code
- Ο while-code block κώδικας υλοποιεί τις επαναλήψεις και αλλάζει την συνθήκη.
- Στο τέλος του while-code block η συνθήκη αξιολογείται εκ νέου
- Ο κώδικας επαναλαμβάνεται μέχρι η συνθήκη να γίνει ψευδής.
- Μέσα στο while-code block πρέπει να αλλάζει κάτι ώστε να γίνει ψευδής η συνθήκη.

Παράδειγμα

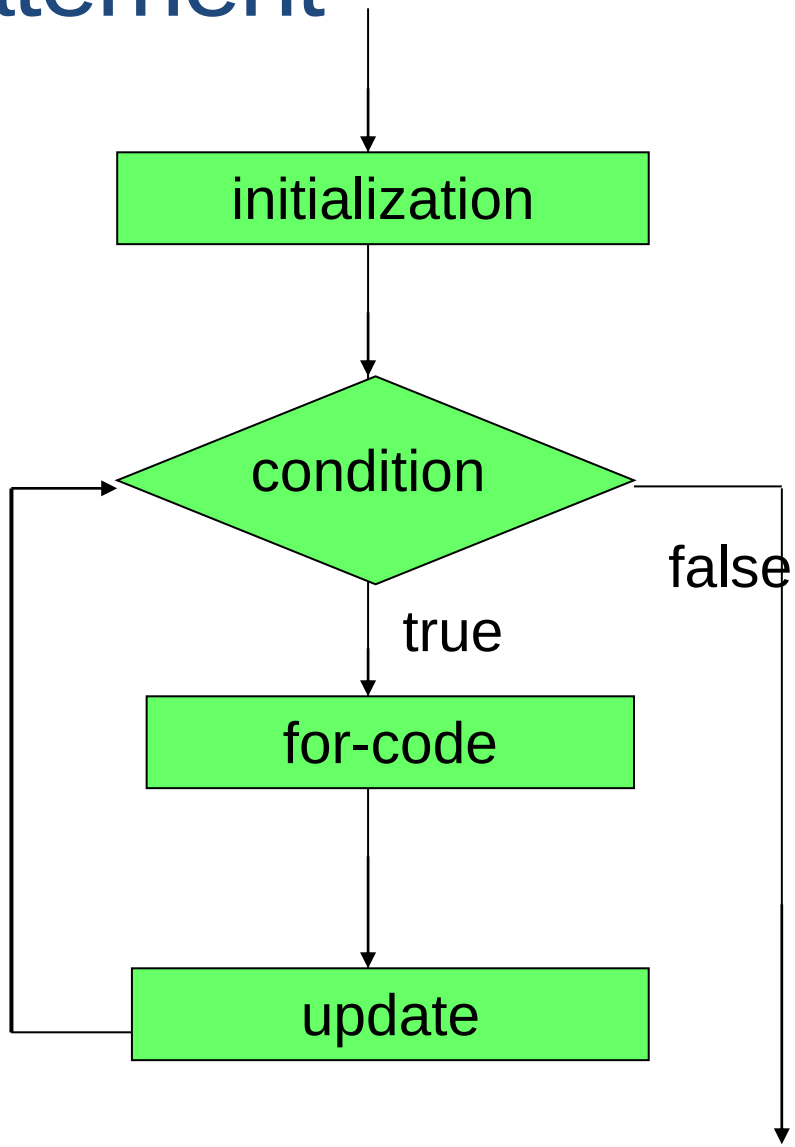
```
Scanner inputReader = new Scanner(System.in);
String input = inputReader.next();

while(input.equals("yes"))
{
    System.out.println("Do you want to continue?");
    input = inputReader.next();
}
```

Επαναλήψεις – for statement

- Στην Java το for statement έχει το εξής συντακτικό

```
for(intialization;
    condition;
    update)
{
    …for-code block…
}
```

- Το όρισμα του for έχει 3 κομμάτια χωρισμένα με ;
  - Την αρχικοποίηση (initialization section): εκτελείται πάντα μία μόνο φορά
  - Τη λογική συνθήκη (condition): εκτιμάται πριν από κάθε επανάληψη.
  - Την ενημέρωση (update expression): υπολογίζεται μετά το κυρίως σώμα της επανάληψης.
  - Ο κώδικας επαναλαμβάνεται μέχρι η συνθήκη να γίνει ψευδής.



Παράδειγμα

Ορισμός της τοπικής μεταβλητής `i`

```
for(int i = 0; i < 10; i = i+1)
{
     System.out.println("i = " + i);
}
```

Ανάθεση: υπολογίζεται η τιμή του `i+1` και ανατίθεται στη μεταβλητή `i`.

- Ισοδύναμο με while

```
int i = 0;
while(i < 10)
{
     System.out.println("i = " + i);
     i = i+1;
}
```

Παράδειγμα

```
for(int i = 0; i < 10; i ++)
{
     System.out.println("i = " + i);
     i = i+1;
}
```

i++ ισοδύναμο με το i = i+1

- Ισοδύναμο με while

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

Χρήση του for-loops

- Το for-loop χρησιμοποιείται για την υλοποίηση ενός μετρητή, ώστε να κάνουμε ένα συγκεκριμένο αριθμό από επαναλήψεις
- Αν θέλουμε να κάνουμε K επαναλήψεις ο κώδικας μας θα είναι:

```
for(int i = 0; i < K; i++)
{
    …
}
```

- Ξεκινάμε τον μετρητή `i` από το μηδέν και πάμε μέχρι K-1.
- Αυτό μας βοηθάει και όταν διατρέχουμε τις θέσεις ενός πίνακα.
- Αυτή είναι μια άλλη συχνή χρήση του for-loop