ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Εισαγωγή στη Java II

Strings

- Η κλάση String είναι προκαθορισμένη κλάση της Java που μας επιτρέπει να χειριζόμαστε αλφαριθμητικά.
- Ο τελεστής “+” μας επιτρέπει την συνένωση
- Υπάρχουν πολλές χρήσιμες μέθοδοι της κλάσης String.
  - length(): μήκος του String
  - equals(String x): τσεκάρει για ισότητα
  - trim(): αφαιρεί κενά στην αρχή και το τέλος του string.
  - split(char delim): σπάει το string σε πίνακα από strings με βάσει το χαρακτήρα delim.
  - Μέθοδοι για να βρεθεί ένα υπο-string μέσα σε ένα string.
  - Κλπ.

Παράδειγμα με Strings

```
public class StringProcessingDemo
{
    public static void main(String[] args)
    {
        String sentence = "I hate text processing!";
        int position = sentence.indexOf("hate");
        String ending =
                sentence.substring(position + "hate".length( ));

        System.out.println("012345678901234567890123");
        System.out.println(sentence);
        System.out.println("The word \"hate\" starts at index "
                                                + position);

        sentence = sentence.substring(0, position) + "adore"
                                                + ending;
        System.out.println("The changed string is:");
        System.out.println(sentence);
    }
}
```

Τα Strings είναι αμετάβλητα (immutable) αντικείμενα
Όταν κάνουμε ανάθεση δημιουργούνται και αντιγράφονται από την αρχή

Ρευματα εισόδου/εξόδου

- Τι είναι ένα ρεύμα? Μια αφαίρεση που αναπαριστά μια πηγή (για την είσοδο), ή ένα προορισμό (για την έξοδο) χαρακτήρων
  - Αυτό μπορεί να είναι ένα αρχείο, το πληκτρολόγιο, η οθόνη.
  - Όταν δημιουργούμε το ρεύμα το συνδέουμε με την ανάλογη πηγή, ή προορισμό.

Είσοδος & Έξοδος

- Τα βασικά ρεύματα εισόδου/εξόδου είναι έτοιμα αντικείμενα τα οποία ορίζονται σαν πεδία (στατικά) της κλάσης System
  - System.out
  - System.in
  - System.err
- Μέσω αυτών και άλλων βοηθητικών αντικειμένων γίνεται η είσοδος και έξοδος δεδομένων ενός προγράμματος.
- Μια εντολή εισόδου/εξόδου έχει αποτέλεσμα το λειτουργικό να πάρει ή να στείλει χαρακτήρες από/προς την αντίστοιχη πηγή/προορισμό.

Έξοδος

- Μπορούμε να καλέσουμε τις συναρτήσεις:
  - println(String s): για να τυπώσουμε ένα αλφαριθμητικό s και τον χαρακτήρα ‘\n’
  - print(String s): τυπώνει το s αλλά δεν αλλάζει γραμμή
  - printf: Formatted output
    - printf(“%d”,myInt); // τυπώνει ένα ακέραιο
    - printf(“%f”,myDouble); // τυπώνει ένα πραγματικό
    - printf(“%.2f”,myDouble); // τυπώνει ένα πραγματικό με δύο δεκαδικά

Είσοδος

- Χρησιμοποιούμε την κλάση Scanner της Java
  - import java.util.Scanner;
- Αρχικοποιείται με το ρεύμα εισόδου:
  - Scanner in = new Scanner(System.in);
- Μπορούμε να καλέσουμε μεθόδους για να διαβάσουμε κάτι από την είσοδο
  - nextLine(): διαβάζει μέχρι να βρει τον χαρακτήρα '\n'
  - next(): διαβάζει το επόμενο String
  - nextInt(): διαβάζει τον επόμενο int
  - nextDouble(): διαβάζει τον επόμενο double.

Παράδειγμα

```
import java.util.Scanner;

class TestIO
{
	public static void main(String args[])
	{
		Scanner input = new Scanner(System.in);
		String line = input.nextLine();
		System.out.println(line);
	}
}
```

Παράδειγμα

```
import java.util.Scanner;

class TestIO2
{
	public static void main(String args[])
	{
		Scanner input = new Scanner(System.in);
		Double d = input.nextDouble();
		System.out.println("Division = " + d/3);
		System.out.println("1+ Division = " +1 + d/3);
		System.out.printf("1+ Division = %.2f", 1+d/3);
	}
}
```

Λογικοί τελεστές

- Έλεγχος για βασικούς τύπους A,B:
  - Ισότητας: (A == B)
  - Ανισότητας: (A != B)
  - Μεγαλύτερο/Μικρότερο ή ίσο: (A <= B), (A >= B)

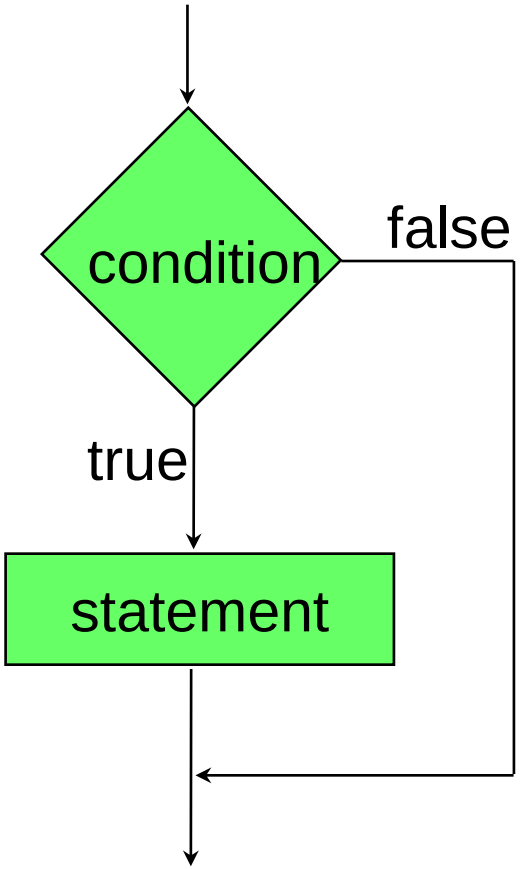
- Λογικοί τελεστές για λογικές εκφράσεις
  - Άρνηση: !B
  - ΚΑΙ: (A && B)
  - Ή: (A || B)

Διακλαδώσεις– Το if Statement

- Στη Java το `if` statement έχει το ακόλουθο συντακτικό:

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

  - Αν η λογική συνθήκη είναι αληθής το statement εκτελείται; Αν όχι, το statement προσπερνιέται

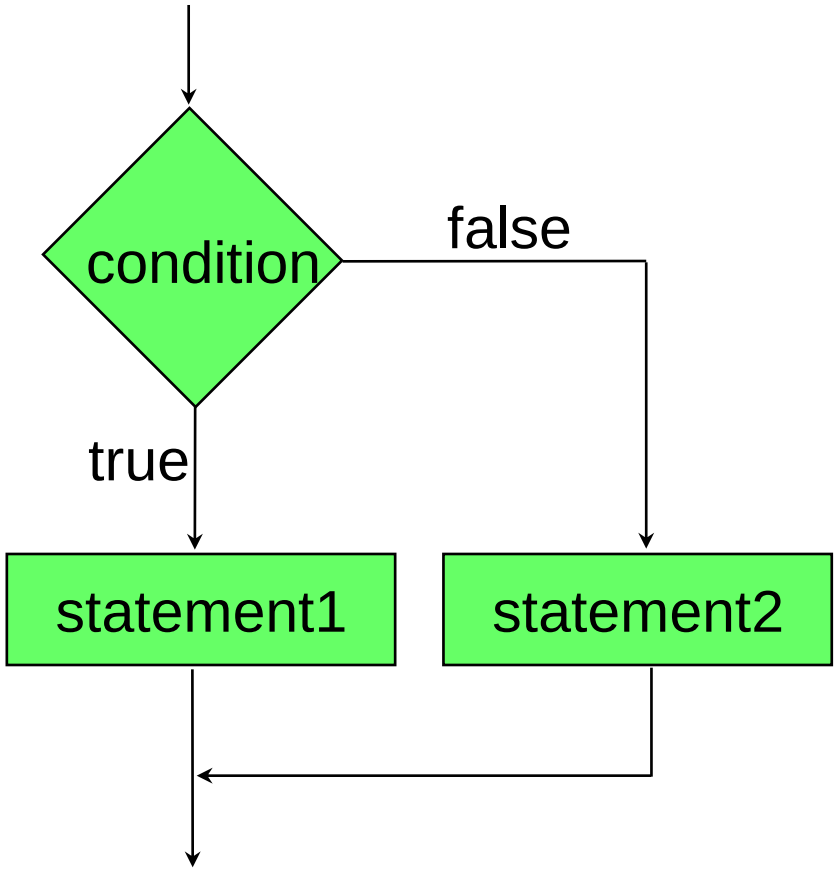


Βρόγχοι – Το if-else statement

- Προσθέτοντας ένα **`else`** clause στο **`if`** statement παίρνουμε το**`if-else`** statement:

```
if (condition) {
    statement(s)1;}
else {
    statement(s)2;}
```

  - Αν η συνθήκη είναι αληθής τότε το statement1 εκτελείται; Αν η συνθήκη είναι ψευδής τότε το statement2 εκτελείται



- Το σώμα του `if` statement ή του `else` statement μπορεί να είναι ένα άλλο `if` statement (φωλιασμένο (nested) **`if`** statement)
- Προσοχή: ένα **`else`** clause ταιριάζει με το τελευταίο ελεύθερο **`if`** (indentation implies)

Προσοχή!

WRONG!

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

CORRECT!

```
if( i == j ) {
 if ( j == k )
 System.out.print(
      “i equals k”);
}
else
 System.out.print(“i
 is not equal to j”);
    // Correct!
```

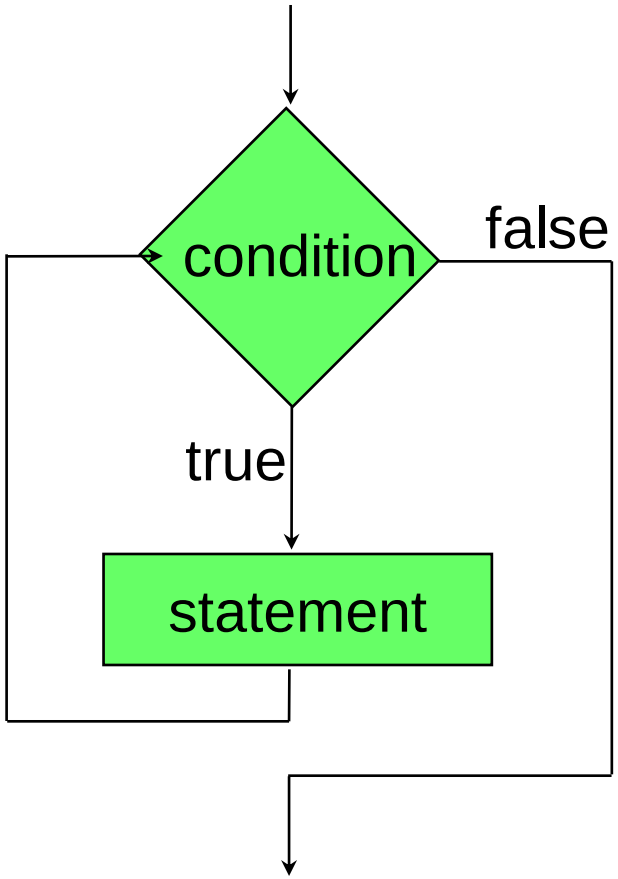
Πάντα να βάζετε { } στο σώμα των if-else statements.

Βρόγχοι - While statement

- Ένα **while** statement έχει το εξής συντακτικό:

```
Initialize the conditions
while (condition){
        statement(s);
 }
```

  - Αν η συνθήκη είναι αληθής, το statement εκτελείται; Μετά η συνθήκη αξιολογείται εκ νέου
  - Τα Statements υλοποιούν τις επαναλήψεις και αλλάζουν την συνθήκη.
- Το statement εκτελείται μέχρι η συνθήκη να γίνει ψευδής

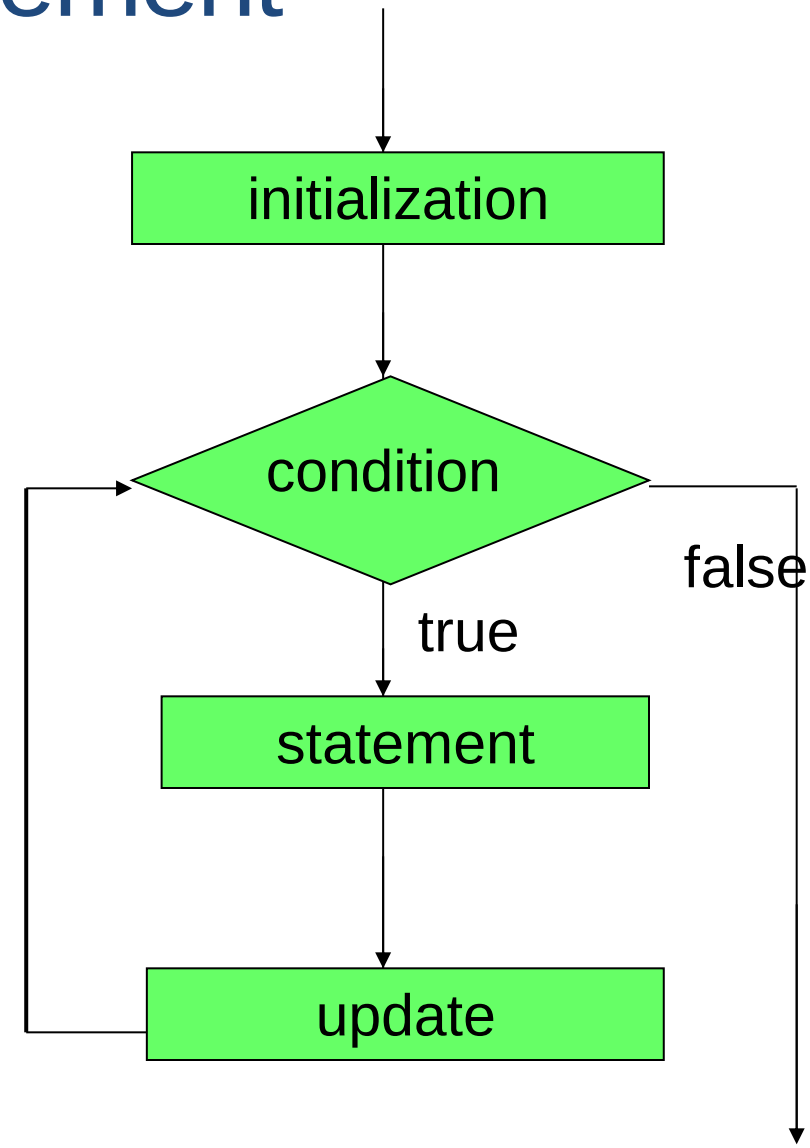


Επανάληψη – for statement

- Ένα **for** statement έχει το ακόλουθο συντακτικό:

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

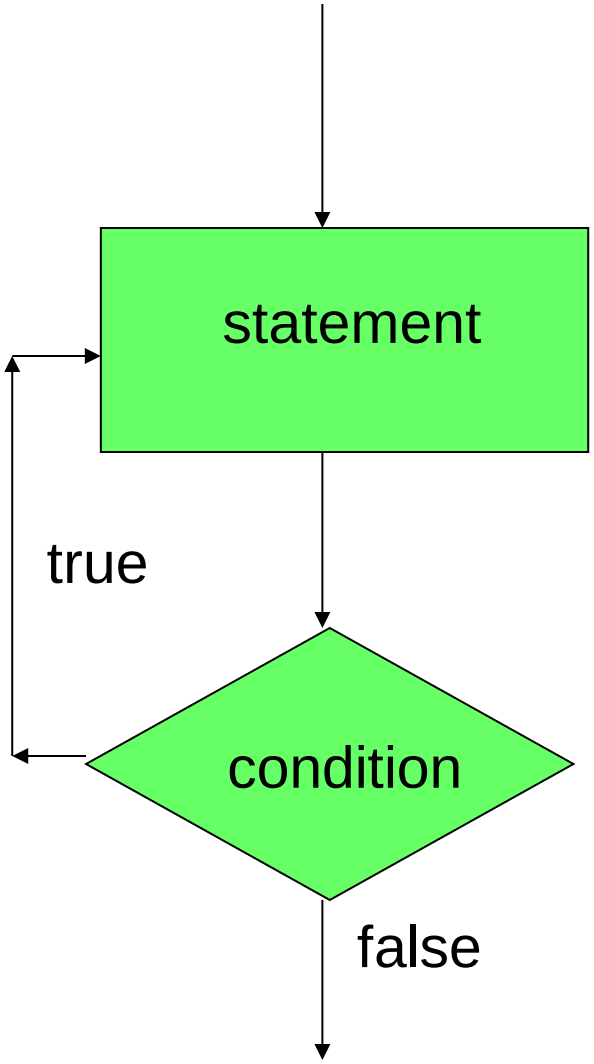
- Το όρισμα του for έχει 3 κομμάτια χωρισμένα με ;
  - Την αρχικοποίηση (initialization section): εκτελείται πάντα μία μόνο φορά
  - Τη λογική συνθήκη (condition): εκτιμάται πριν από κάθε επανάληψη.
  - Την ενημέρωση (update expression): υπολογίζεται μετά το κυρίως σώμα της επανάληψης.



Το Do-While statement

- Ένα **`do while`** statement έχει το εξής συντακτικό:

```
Initialize the conditions
   do {
        statement(s);
      } while (conditions);
```



- Το statement εκτελείται τουλάχιστον μία φορά; Μετά αν η συνθήκη είναι αληθής το statement εκτελείται ξανά.
- Τα statements εκτελούν το βρόγχο και αλλάζουν την συνθήκη.

```
import java.util.Scanner;

class FlowTest
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        while (inputInt != 0)
        {
            if (inputInt < 0 ){
                for (int i = inputInt; i < 0; i ++)
                {
                    System.out.println("Counter = " + i);
                }
            } else if (inputInt > 0){
                for (int i = inputInt; i > 0; i --)
                {
                    System.out.println("Counter = " + i);
                }
            }
            inputInt = reader.nextInt();
        }
    }
}
```

Scope μεταβλητών

- Η κάθε μεταβλητή που ορίζουμε έχει εμβέλεια (scope) μέσα στο block το οποίο ορίζεται.
- Μόλις βγούμε από το block η μεταβλητή χάνεται
  - Ο compiler δημιουργεί στο stack ένα χώρο για το block το οποίο μετά εξαφανίζεται όταν το block τελείωσει.

Παράδειγμα με το scope μεταβλητών

```
public static void main(String[] args)
{
        int y = 1;
        int x = 2;
        for (int i = 0; i < 3; i ++)
        {
           y = i;
           int x = i+1;
           int z = x+y;
           System.out.println("i = " + i);
           System.out.println("y = " + y);
           System.out.println("z = " + z);
        }
        System.out.println("i = " + i);
        System.out.println("z = " + z);
        System.out.println("y = " + y);
        System.out.println("x = " + x);
}
```

Ο κώδικας έχει λάθη σε τρία σημεία

Οι εντολές break και continue

- continue: Επιστρέφει τη ροή του προγράμματος στον έλεγχο της συνθήκης σε ένα βρόγχο.
  - Βολικό για τον έλεγχο συνθηκών πριν ξεκινήσει η εκτέλεση του βρόγχου
  - Π.χ., πώς θα τυπώναμε μόνο τους άρτιους αριθμούς?

- break: Μας βγάζει έξω από την εκτέλεση του βρόχου από οποιοδήποτε σημείο μέσα στον κώδικα.
  - Κάποιοι θεωρούν ότι χαλάει το μοντέλο του δομημένου προγραμματισμού.
  - Βολικό για να σταματάμε το βρόγχο όταν κάτι δεν πάει καλά.

Παράδειγμα

```
while (…)
{
  if (everything is ok){

    < rest of code>

  }// end of if
} // end of while loop
```

```
while (… && !StopFlag)
{
  < some code>

  if (I should stop){
    StopFlag = true;
  }else{
    < some code>
  }
} // end of while loop
```

```
while (…)
{
  if (I don't like something){
      continue;
  }

  < rest of code>

} // end of while loop
```

```
while (…)
{
  < some code>

  if (I should stop){
      break;
  }

  < some code>
} // end of while loop
```