

# ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

---

Εισαγωγή στη Java

# HelloWorld.java

```
class HelloWorld
{
    public static void main(String args[])
    {
        // print message
        System.out.println("Hello world!");
    }
}
```

➤ `javac HelloWorld.java`

➤ `java HelloWorld`

Χωρίς κανένα επίθεμα!

# Παράδειγμα 2

- Φτιάξτε ένα πρόγραμμα που τυπώνει το λόγο δύο ακεραίων.

# Division.java

```
class Division
{
    public static void main(String args[])
    {
        int enumerator = 32;
        int denominator = 10;
        double division;
        division = enumerator / (double) denominator;
        System.out.println("Result = " + division);
    }
}
```

# Division.java

```
class Division
{
    public static void main(String args[])
    {
        int enumerator = 32;
        int denominator = 10;
        double division;
        division = enumerator / (double) denominator;
        System.out.println("Result = " + division);
    }
}
```

- Ορισμός μεταβλητών
- Η Java είναι **strongly typed** γλώσσα: κάθε μεταβλητή θα πρέπει να έχει ένα **τύπο**.
- Οι τύποι **int** και **double** είναι **πρωταρχικοί (βασικοί) τύποι (primitive types)**
- Εκτός από τους βασικούς τύπους, όλοι οι άλλοι **τύποι** είναι **κλάσεις**

# Πρωταρχικοί τύποι

| Όνομα τύπου | Τιμή                 | Μνήμη   |
|-------------|----------------------|---------|
| boolean     | true/false           | 1 byte  |
| char        | Χαρακτήρας (Unicode) | 2 bytes |
| byte        | Ακέραιος             | 1 byte  |
| short       | Ακέραιος             | 2 bytes |
| int         | Ακέραιος             | 4 bytes |
| long        | Ακέραιος             | 8 bytes |
| float       | Πραγματικός          | 4 bytes |
| double      | Πραγματικός          | 8 bytes |

Όταν ορίζουμε μια μεταβλητή **δεσμεύεται** ο αντίστοιχος χώρος στη **μνήμη**. Το **όνομα της μεταβλητής** αντιστοιχίζεται με αυτό το χώρο στη **μνήμη**.

# Πρωταρχικοί τύποι

| Όνομα τύπου | Τιμή                 | Μνήμη   |
|-------------|----------------------|---------|
| boolean     | true/false           | 1 byte  |
| char        | Χαρακτήρας (Unicode) | 2 bytes |
| byte        | Ακέραιος             | 1 byte  |
| short       | Ακέραιος             | 2 bytes |
| int         | Ακέραιος             | 4 bytes |
| long        | Ακέραιος             | 8 bytes |
| float       | Πραγματικός          | 4 bytes |
| double      | Πραγματικός          | 8 bytes |

Όταν ορίζουμε μια μεταβλητή **δεσμεύεται** ο αντίστοιχος χώρος στη **μνήμη**. Το **όνομα της μεταβλητής** αντιστοιχίζεται με αυτό το χώρο στη **μνήμη**.

# Η μνήμη του υπολογιστή

- Η **κύρια μνήμη** (main memory) του υπολογιστή κρατάει τα **δεδομένα** (και τις εντολές) για την εκτέλεση των προγραμμάτων.
  - Η μνήμη είναι προσωρινή, τα δεδομένα χάνονται όταν ολοκληρωθεί το πρόγραμμα.
- Η μνήμη είναι χωρισμένη σε **bytes** (8 bits)
  - Ο χώρος που χρειάζεται για ένα **χαρακτήρα ASCII**.
- Το κάθε byte έχει μια **διεύθυνση**, με την οποία μπορούμε να προσπελάσουμε τη συγκεκριμένη θέση μνήμης
  - **Random Access Memory (RAM)**
  - Σε 32-bit συστήματα μια διεύθυνση είναι 32 bits, σε 64-bit συστήματα μια διεύθυνση είναι 64 bits.

| Διεύθυνση μνήμης | Περιεχόμενο μνήμης |
|------------------|--------------------|
| 0000             | 'a'                |
| 0001             | 'b'                |
| 0010             | 'c'                |
| 0011             | 'd'                |
| 0100             | 'e'                |
| 0101             | 'f'                |
| 0110             | 'g'                |
| 0111             | 'h'                |



# Αποθήκευση μεταβλητών

- Η **κύρια μνήμη** (main memory) του υπολογιστή κρατάει τις **μεταβλητές** ενός προγράμματος
- Μια μεταβλητή μπορεί να απαιτεί χώρο περισσότερο από 1 byte.
  - Π.χ., οι μεταβλητές τύπου double χρειάζονται 8 bytes.
  - Η μεταβλητή τότε αποθηκεύεται σε συνεχόμενα bytes στη μνήμη.
- Η **θέση μνήμης** (διεύθυνση) της μεταβλητής θεωρείται το **πρώτο byte** από το οποίο ξεκινάει η αποθήκευση του της μεταβλητής.
  - Στο παράδειγμα μας η μεταβλητή βρίσκεται στη θέση 0000
  - Αν ξέρουμε την αρχή και το μέγεθος της μεταβλητής μπορούμε να τη διαβάσουμε.
- Άρα μία **θέση μνήμης** αποτελείται από μία **διεύθυνση** και το **μέγεθος**.

| Διεύθυνση μνήμης | Περιεχόμενο μνήμης |
|------------------|--------------------|
| 0000             | 8.5                |
| 0001             |                    |
| 0010             |                    |
| 0011             |                    |
| 0100             |                    |
| 0101             |                    |
| 0110             |                    |
| 0111             |                    |

# Αποθήκευση μεταβλητών πρωταρχικού τύπου

- Για τις μεταβλητές **πρωταρχικού** τύπου (char, int, double,...) ξέρουμε εκ των προτέρων το μέγεθος της μνήμης που χρειαζόμαστε.
- Όταν ο μεταγλωττιστής δει τη **δήλωση** μιας μεταβλητής πρωταρχικού τύπου **δεσμεύει** μια θέση μνήμης αντίστοιχου μεγέθους
  - Η δήλωση μιας μεταβλητής ουσιαστικά **δίνει ένα όνομα** σε μία θέση μνήμης
  - Συχνά λέμε η **θέση μνήμης x** για τη μεταβλητή **x**.

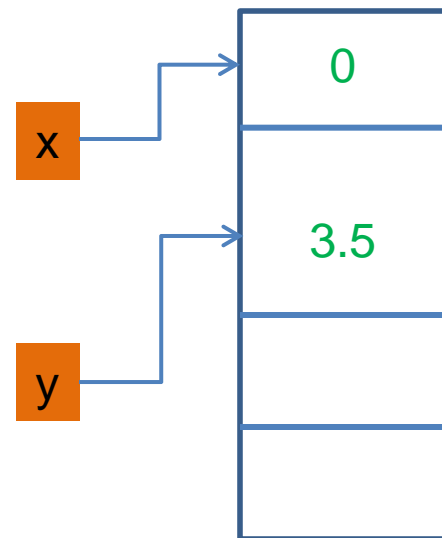
```
int x = 5;  
int y = 3;
```

|          | Διεύθυνση μνήμης | Περιεχόμενο μνήμης |
|----------|------------------|--------------------|
| <b>x</b> | 0000             | 5                  |
|          | 0001             |                    |
|          | 0010             |                    |
|          | 0011             |                    |
| <b>y</b> | 0100             | 3                  |
|          | 0101             |                    |
|          | 0110             |                    |
|          | 0111             |                    |

# Αποθήκευση μεταβλητών

- Μπορούμε να σκεφτόμαστε την μνήμη του υπολογιστή σαν μια σειρά από «**κουτάκια**» διαφόρων μεγεθών στα οποία μπορούμε να αποθηκεύουμε δεδομένα
  - Το κάθε κουτάκι έχει **διεύθυνση** και **περιεχόμενα**
- Όταν **ορίζουμε** μια μεταβλητή πρωταρχικού τύπου (π.χ., **int x**) αυτό που γίνεται είναι ότι:
  - **Δεσμεύουμε** ένα κουτάκι κατάλληλου μεγέθους
    - 4 bytes για ένα int
  - Δίνουμε στο κουτάκι το **όνομα** της μεταβλητής
    - Το κουτάκι **x** αντί για το κουτάκι **0110**
  - Γι αυτό και η μεταβλητή **ορίζεται** μόνο μια φορά.
- Με την ανάθεση αλλάζουμε τα **περιεχόμενα** του κουτιού
- Αν δεν αρχικοποιήσουμε μια μεταβλητή η Java την αρχικοποιεί στο μηδέν (ή το αντίστοιχο του μηδέν για μη αριθμητικές μεταβλητές)

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



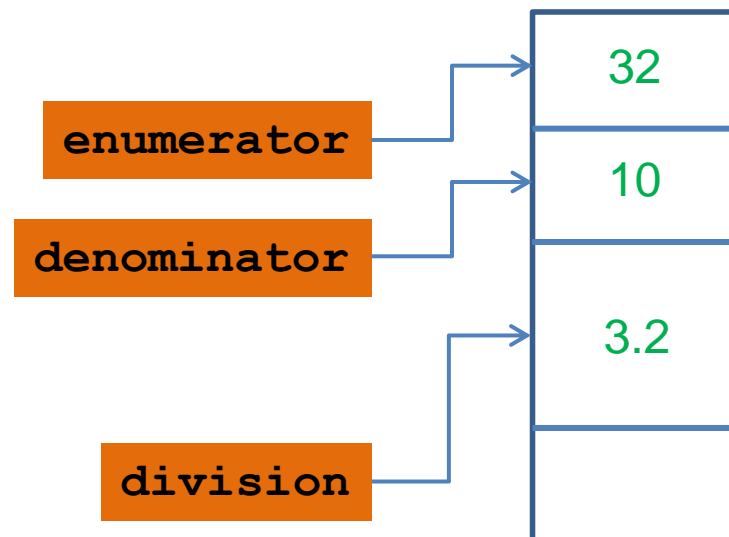
# Division.java

```
class Division
{
    public static void main(String args[])
    {
        int enumerator = 32;
        int denominator = 10;
        double division;
        division = enumerator / (double) denominator;
        System.out.println("Result = " + division);
    }
}
```

**Ανάθεση:** αποτίμηση της τιμής της έκφρασης στο δεξιό μέλος του “=” και μετά ανάθεση της τιμής στην μεταβλητή στο αριστερό μέλος  
Σε μια ανάθεση το αριστερό μέλος πρέπει να είναι μια μεταβλητή

# Division.java

```
class Division
{
    public static void main(String args[])
    {
        int enumerator = 32;
        int denominator = 10;
        double division;
        division =
            enumerator / (double) denominator;
        System.out.println(
            "Result = " + division);
    }
}
```



**Ανάθεση:** Διαβάζουμε τα περιεχόμενα των μεταβλητών **enumerator** και **denominator** κάνουμε τον υπολογισμό και αλλάζουμε τα περιεχόμενα της μεταβλητής **division** αποθηκεύοντας το αποτέλεσμα της διαίρεσης.

# Division.java

```
class Division
{
    public static void main(String args[])
    {
        int enumerator = 32;
        int denominator = 10;
        double division;
        division = enumerator / (double)denominator;
        System.out.println("Result = " + division);
    }
}
```

**Μετατροπή τύπου (type casting):** `(double) denominator` μετατρέπει την τιμή της μεταβλητής `denominator` σε `double`.

Αν δεν γίνει η μετατροπή, η διαίρεση μεταξύ ακεραίων μας δίνει **πάντα** ακέραιο.

# Αναθέσεις

- Στην ανάθεση κατά κανόνα, η τιμή του δεξιού μέρους θα πρέπει να είναι **ίδιου τύπου** με την μεταβλητή του αριστερού μέρους.
- Υπάρχουν εξαιρέσεις όταν υπάρχει **συμβατότητα** μεταξύ τύπων
- **byte** → **short** → **int** → **long** → **float** → **double**
  - Μια τιμή τύπου **T** μπορούμε να την αναθέσουμε σε μια μεταβλητή τύπου που εμφανίζεται **δεξιά του T**.
- (Σε αντίθεση με την C) ο τύπος **boolean** δεν είναι συμβατός με τους ακέραιους.

# Division.java

```
class Division
{
    public static void main(String args[])
    {
        int enumerator = 32;
        int denominator = 10;
        double division;
        division = enumerator / (double) denominator;
        System.out.println("Result = " + division);
    }
}
```

Ο τελεστής “+” μεταξύ αντικείμενων της κλάσης String **συνενώνει** (concatenates) τα δύο String.

Μεταξύ ενός String και ενός βασικού τύπου, ο βασικός τύπος **μετατρέπεται** σε String και γίνεται η συνένωση



# 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. Η πιο χρήσιμη αυτή τη στιγμή είναι:
  - **equals(String x)**: ελέγχει για ισότητα του αντικειμένου που κάλεσε την μέθοδο και του ορίσματος x.
- Άλλες μέθοδοι:
  - **length()**: μήκος του String
  - **trim()**: αφαιρεί κενά στην αρχή και το τέλος του string.
  - **split(char delim)**: σπάει το string σε πίνακα από strings με βάση το χαρακτήρα delim.
  - Επίσης, μέθοδοι για να βρεθεί ένα υπο-string μέσα σε ένα string, κλπ.

# Escape sequences

- Για να τυπώσουμε κάποιους ειδικούς χαρακτήρες (π.χ., τον χαρακτήρα “) χρησιμοποιούμε τον χαρακτήρα \ και μετά τον χαρακτήρα που θέλουμε να τυπώσουμε
  - Π.χ., ακολουθία \”
- Αυτό ισχύει γενικά για ειδικούς χαρακτήρες.

|          |                  |
|----------|------------------|
| • \b     | Backspace        |
| • \t     | Tab              |
| • \n     | New line         |
| • \f     | Form feed        |
| • \r     | Return (ENTER)   |
| • \”     | Double quote     |
| • \’     | Single quote     |
| • \\     | Backslash        |
| • \ddd   | Octal code       |
| • \uxxxx | Hex-decimal code |

# Ρεύματα 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 μεθόδους του `System.out`:
  - `println(String s)`: για να τυπώσουμε ένα αλφαριθμητικό `s` και τον χαρακτήρα `'\n'` (αλλαγή γραμμής)
  - `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 της Scanner για να διαβάσουμε κάτι από την είσοδο
  - `nextLine()`: διαβάζει **μέχρι** να βρει τον χαρακτήρα `'\n'`
  - `next()`: διαβάζει το επόμενο **String** μέχρι να βρει **λευκό χαρακτήρα**
  - `nextInt()`: διαβάζει τον επόμενο **int**
  - `nextDouble()`: διαβάζει τον επόμενο **double**.
  - `nextBoolean()`: διαβάζει τον επόμενο **boolean**.

# Παράδειγμα

Με την εντολή αυτή φέρνουμε την κλάση Scanner μέσα στο πρόγραμμά μας ώστε να μπορούμε να φτιάξουμε αντικείμενα τύπου Scanner

```
import java.util.Scanner;
```

```
class TestIO
```

```
{
```

```
    public static void main(String args[])
```

```
    {
```

```
        System.out.println("Say something:");
```

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

```
        String line = input.nextLine();
```

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

```
    }
```

```
}
```

**new**: δημιουργεί ένα αντικείμενο τύπου **Scanner** (μία μεταβλητή) με το οποίο μπορούμε πλέον να διαβάζουμε από την είσοδο.

- Το αντικείμενο αυτό αναπαριστά το **πληκτρολόγιο** στο πρόγραμμά μας. Ένα αντικείμενο φτάνει για να διαβάσουμε πολλαπλές τιμές.

# 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 by 4 = " + d/4);
        System.out.println("1+ (division by 4) = " +1+d/4);
        System.out.printf("1+ (division of %.2f by 4) = %.2f",d, 1+d/4);
    }
}
```

Το + λειτουργεί ως **concatenation** τελεστής μεταξύ Strings, άρα μετατρέπει τους αριθμούς σε Strings

Τι θα τυπώσει αυτό το πρόγραμμα?



# Λογικοί τελεστές

- **Λογικοί τελεστές** για λογικές εκφράσεις
  - Άρνηση: `!B`
  - ΚΑΙ: `(A && B)`
  - Ή: `(A || B)`
- Έλεγχος για βασικούς τύπους A,B:
  - Ισότητας: `(A == B)`
  - Ανισότητας: `(A != B)` ή `(!(A == B))`
  - Μεγαλύτερο/Μικρότερο ή ίσο: `(A <= B)` , `(A >= B)`
- Έλεγχος για μεταβλητές (**αντικείμενα**) οποιουδήποτε άλλου τύπου γίνεται με την μέθοδο `equals` (πρέπει να έχει οριστεί):
  - Ισότητας: `(A.equals(B))`
  - Ανισότητας: `(!A.equals(B))`
- Λογικές σταθερές:
  - `true`: αληθές
  - `false`: ψευδές

# 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);
```

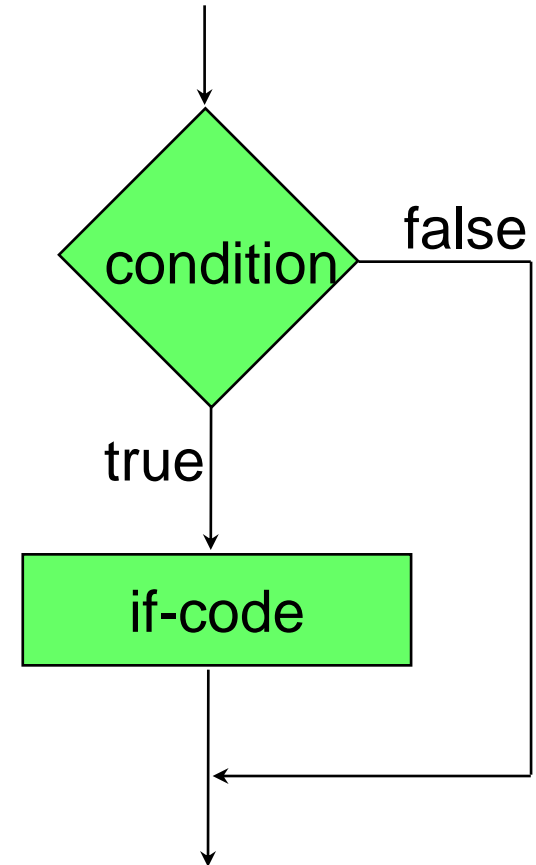
- Περνάει από τον compiler και σε κάποιες περιπτώσεις θα δουλέψει αλλά **δεν κάνει αυτό που θέλουμε**.

# Βρόγχοι – Το if-then Statement

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

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

- Αν η **συνθήκη** είναι **αληθής** τότε εκτελείται το block κώδικα if-code
- Αν η **συνθήκη** είναι **ψευδής** τότε το κομμάτι αυτό προσπερνιέται και συνεχίζεται η εκτέλεση.







```
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 ισότητα

# 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 isPositive = (x > 0);
boolean isNegative = (x < 0);
boolean isNotPositive = !isPositive;
```

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

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

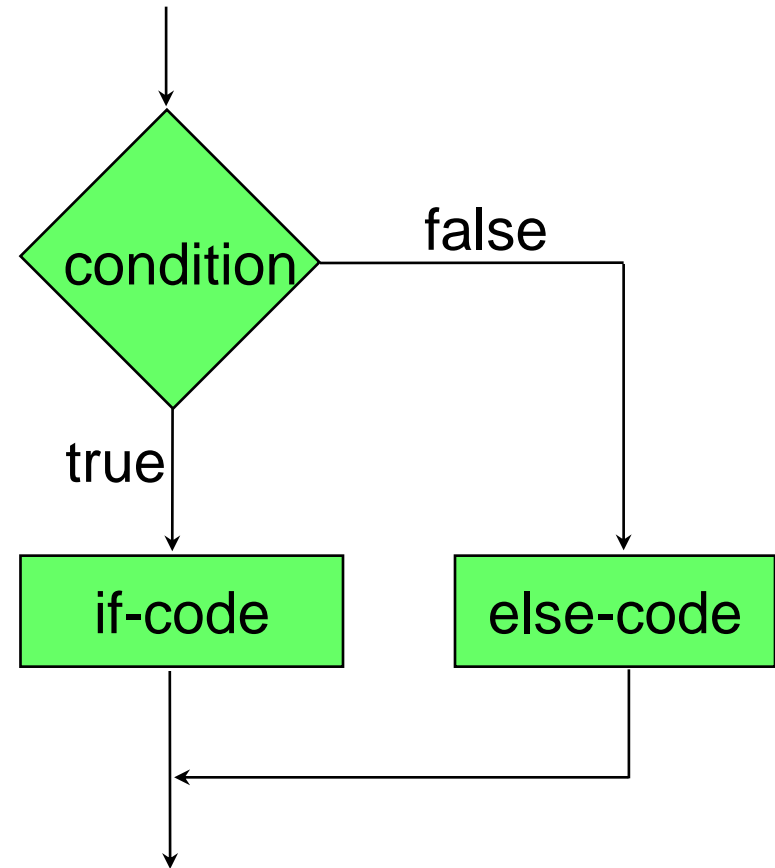
- Το ίδιο ισχύει και όταν αργότερα θα ορίζουμε **μεθόδους** που επιστρέφουν λογικές τιμές
  - Π.χ., για τα Strings υπάρχει η μέθοδος **equals** που γίνεται true όταν έχουμε ισότητα και η μέθοδος **isEmpty** που είναι true όταν έχουμε άδειο String.

# 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 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");
}
```

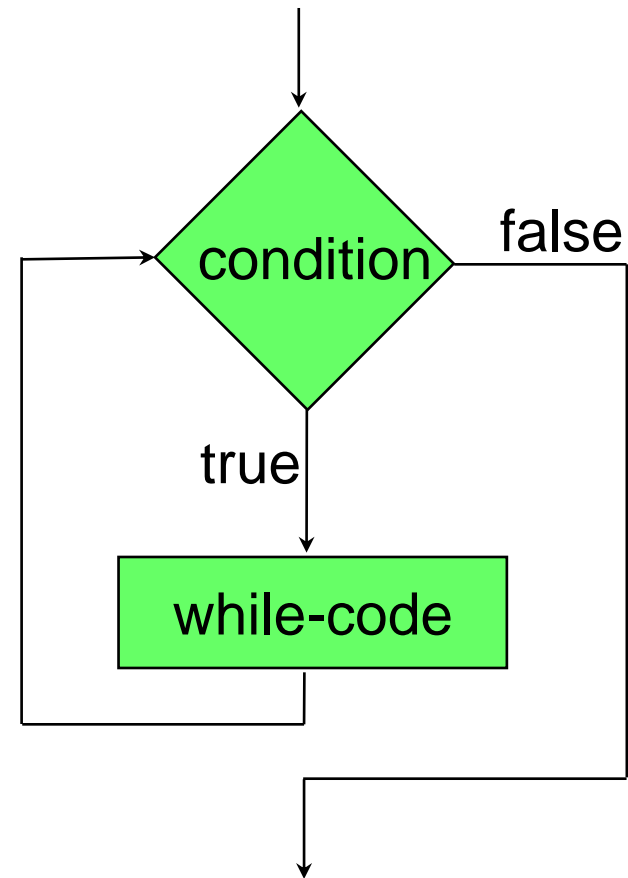
**Πάντα** να βάζετε `{ }` στο σώμα των if-then-else statements.  
Πάντα να στοιχίζετε σωστά τον κώδικα.

# 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**.



# Παράδειγμα

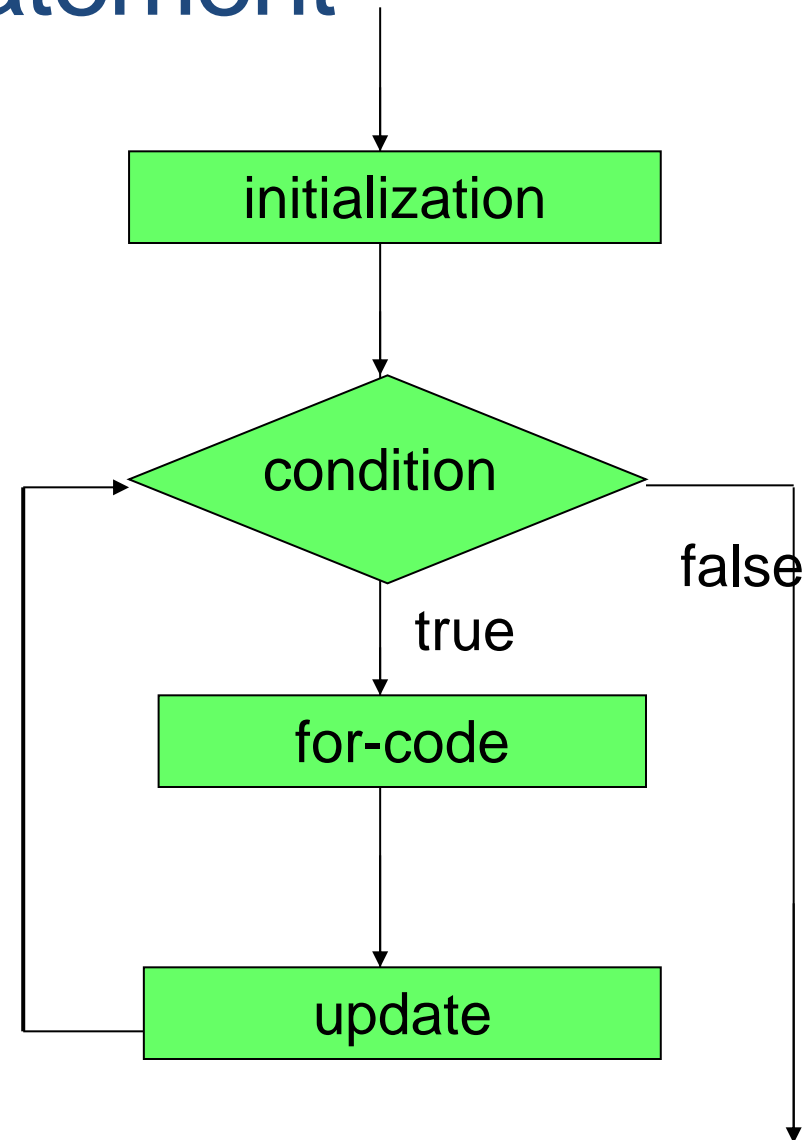
```
Scanner inputReader = new Scanner(System.in);  
String input = inputReader.next();  
  
while(input.equals("Yes"))  
{  
    System.out.println("Do you want to continue?");  
    input = inputReader.next();  
}
```

# Επαναλήψεις – for statement

- Στην Java το **for statement** έχει το εξής ΣΥΝΤΑΚΤΙΚΟ

```
for (initialization;  
    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

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

Ορισμός της μεταβλητής *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 ++;  
}
```

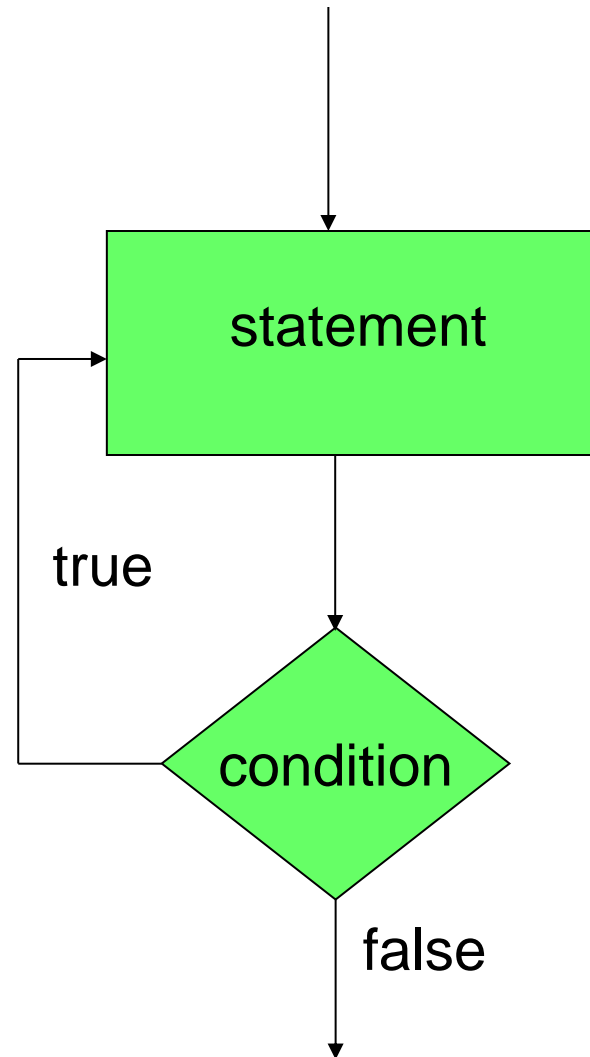


# To Do-While statement

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

```
Initialize  
do  
{  
    ...while-code block...  
}while (condition)
```

- Το while code εκτελείται **τουλάχιστον μία φορά**; Μετά αν η συνθήκη είναι αληθής ο κώδικας εκτελείται ξανά.
- Το while code εκτελούν το βρόγχο και αλλάζουν την συνθήκη.



# Παράδειγμα

- Κάνετε πρόγραμμα που παίρνει σαν είσοδο ένα αριθμό και υλοποιεί μια αντίστροφη μέτρηση. Αν ο αριθμός είναι θετικός η αντίστροφη μέτρηση γίνεται προς τα κάτω μέχρι το μηδέν, αν είναι αρνητικός γίνεται προς τα πάνω μέχρι το μηδέν. Η διαδικασία επαναλαμβάνεται μέχρι ο χρήστης να δώσει την τιμή μηδέν.

```
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();
        }
    }
}
```

```
import java.util.Scanner;

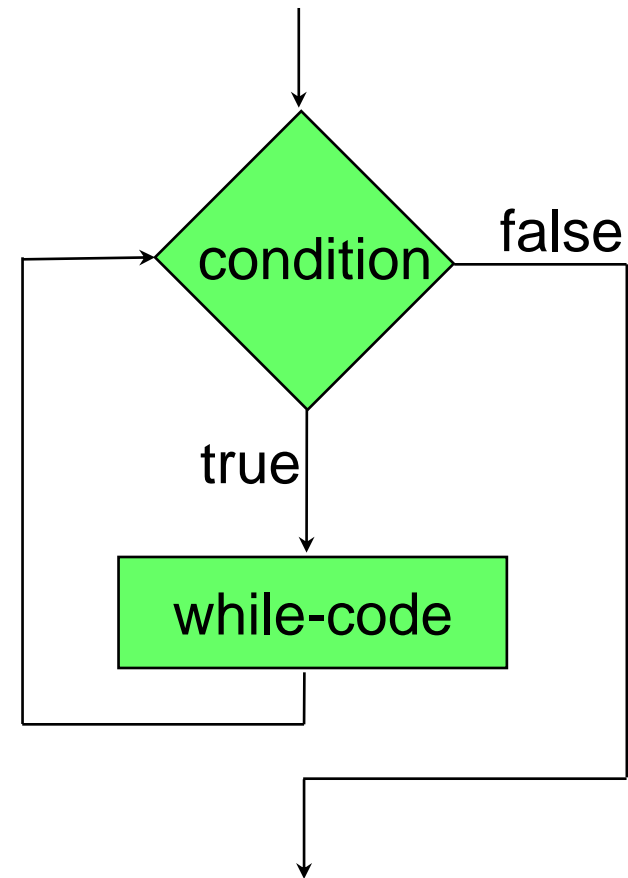
class FlowTest2
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt;
        do
        {
            inputInt = reader.nextInt();
            if (inputInt < 0 ){
                for (int i = inputInt; i < 0; i ++){
                    System.out.println("Counter = " + i);
                }
            } else if (inputInt > 0){
                for (int i = inputInt; i > 0; i --){
                    System.out.println("Counter = " + i);
                }
            }
        } while (inputInt != 0)
    }
}
```

# Επαναλήψεις - While statement

- Στην Java το **while statement** έχει το εξής συντακτικό

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

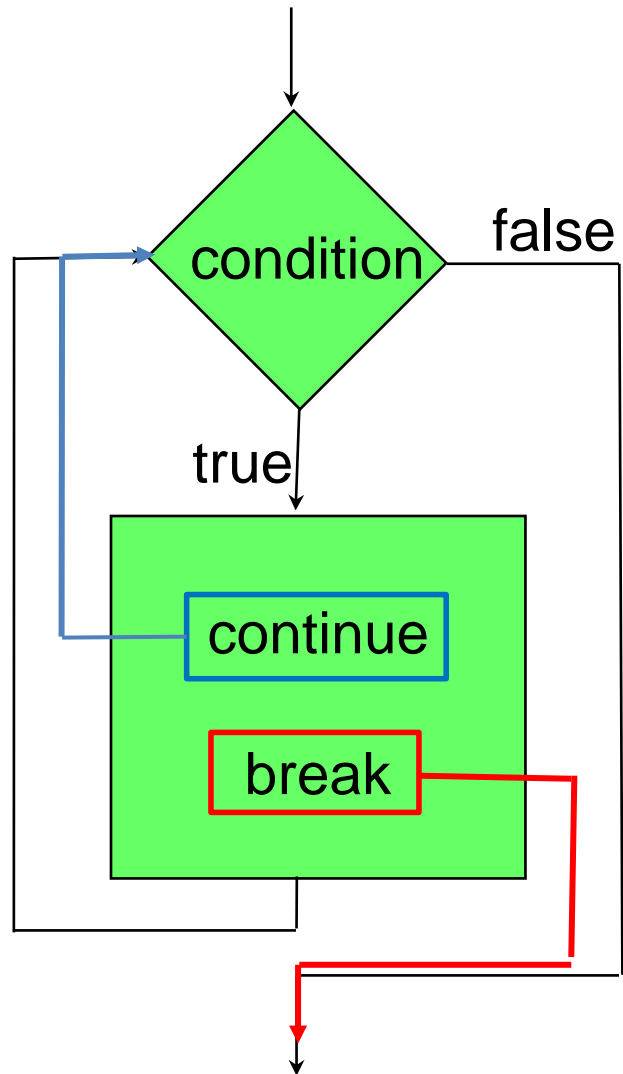
- Αν η **συνθήκη** είναι **αληθής** τότε εκτελείται το block κώδικα **while-code**
- Ο **while-code block** κώδικας υλοποιεί τις επαναλήψεις και **αλλάζει την συνθήκη**.
- Στο **τέλος του while-code block** η συνθήκη **αξιολογείται εκ νέου**
- Ο κώδικας επαναλαμβάνεται **μέχρι** η συνθήκη να γίνει **ψευδής**.



# 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, ή για πρόορη επιστροφή στον έλεγχο της συνθήκης
- **`break`**: Μας βγάζει έξω από την εκτέλεση του βρόγχου από οποιοδήποτε σημείο μέσα στον κώδικα.
  - Βολικό για να σταματάμε το βρόγχο όταν κάτι δεν πάει καλά.
- Κάποιοι θεωρούν οι εντολές αυτές χαλάνε το μοντέλο του δομημένου προγραμματισμού.

# Οι εντολές break και continue



# 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 more 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
```



```

import java.util.Scanner;

class FlowTestContinue
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt = reader.nextInt();
        while (inputInt != 0)
        {
            if (inputInt%2 == 0){
                inputInt = reader.nextInt();
                continue;
            }
            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();
        }
    }
}

```

Η αντίστροφη μέτρηση εκτελείται μόνο για περιττούς αριθμούς

```
import java.util.Scanner;

class FlowTest2
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        int inputInt;
        do
        {
            inputInt = reader.nextInt();
            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);
                }
            }
        }while (inputInt != 0)
    }
}
```

```
import java.util.Scanner;

class FlowTestBreak
{
    public static void main(String[] args)
    {
        Scanner reader = new Scanner(System.in);
        do
        {
            int inputInt = reader.nextInt();
            if (inputInt == 0){
                break;
            }
            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);
                }
            }
        } while (true)
    }
}
```

# Εμβέλεια (scope) μεταβλητών

- Προσέξτε ότι η μεταβλητή `int i` πρέπει να οριστεί **σε** **κάθε** `for`, ενώ η `inputInt` πρέπει να οριστεί **έξω** από το `while-loop` αλλιώς ο compiler διαμαρτύρεται.
  - Προσπαθούμε να χρησιμοποιήσουμε μια μεταβλητή εκτός της **εμβέλειας** της
- Η κάθε μεταβλητή που ορίζουμε έχει **εμβέλεια (scope)** μέσα στο `block` το οποίο ορίζεται.
  - **Τοπική μεταβλητή** μέσα στο `block`.
- Μόλις **βγούμε** από το `block` η μεταβλητή χάνεται
  - Ο compiler δημιουργεί ένα χώρο στη μνήμη για το `block` το οποίο εκτελούμε, ο οποίος εξαφανίζεται όταν το `block` τελειώσει.
- Ένα `block` μπορεί να περιλαμβάνει κι άλλα **φωλιασμένα blocks**
  - Η μεταβλητή έχει **εμβέλεια** και μέσα στα **φωλιασμένα blocks**
  - **Δεν μπορούμε** να ορίσουμε μια άλλη **μεταβλητή με το ίδιο όνομα** σε ένα φωλιασμένο `block`

# 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;
        double 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

```
public static void main(String[] args)
{
    ... ..
    {
        ... ..
        {
            int y = 1;
            ... ..
            {
                ... ..
            }
            ... ..
        }
        ... ..
    }
    ... ..
}
```

Η διαφορά του κόκκινου από το μπλε είναι ο χώρος εκτός της εμβελείας του **y**

Έξω από το μπλε δεν μπορούμε να **χρησιμοποιήσουμε** τη μεταβλητή **y**

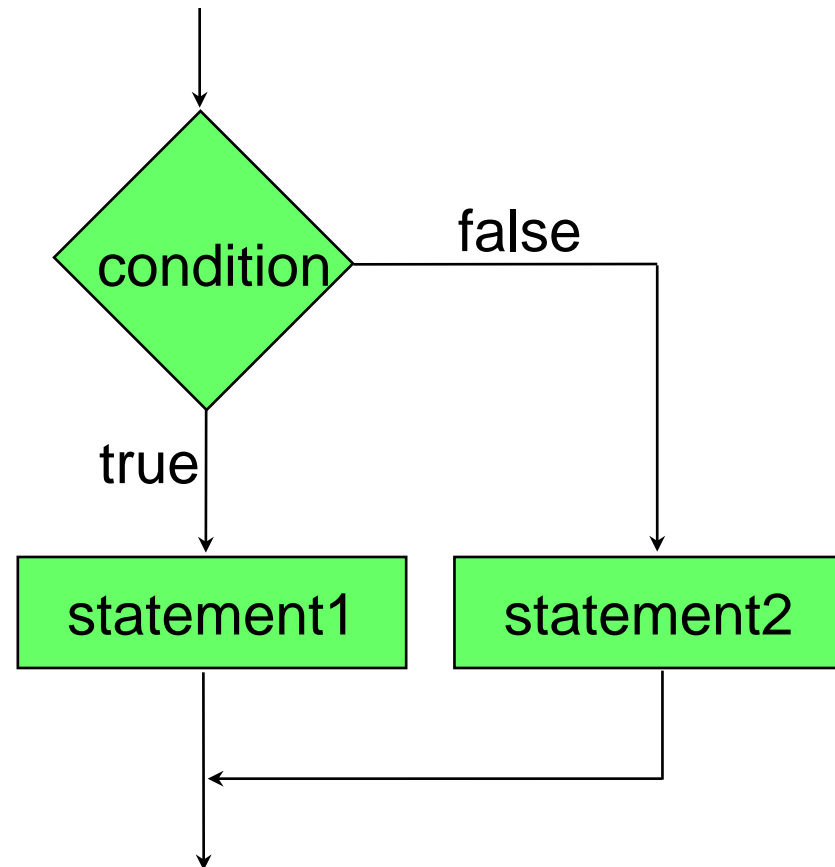
Η εμβέλεια του **y**

Μέσα στο μπλε μπορούμε να χρησιμοποιήσουμε την μεταβλητή **y**, αλλά δεν μπορούμε να **ορίσουμε** άλλη μεταβλητή με το όνομα **y**

Προχωρημένο: Κάθε block έχει το δικό του χώρο μνήμης. Σε ένα χώρο μνήμης μια μεταβλητή μπορεί να οριστεί μόνο μία φορά. Τα φωλιασμένα blocks έχουν και τις μεταβλητές των προγόνων.

# To if-else statement

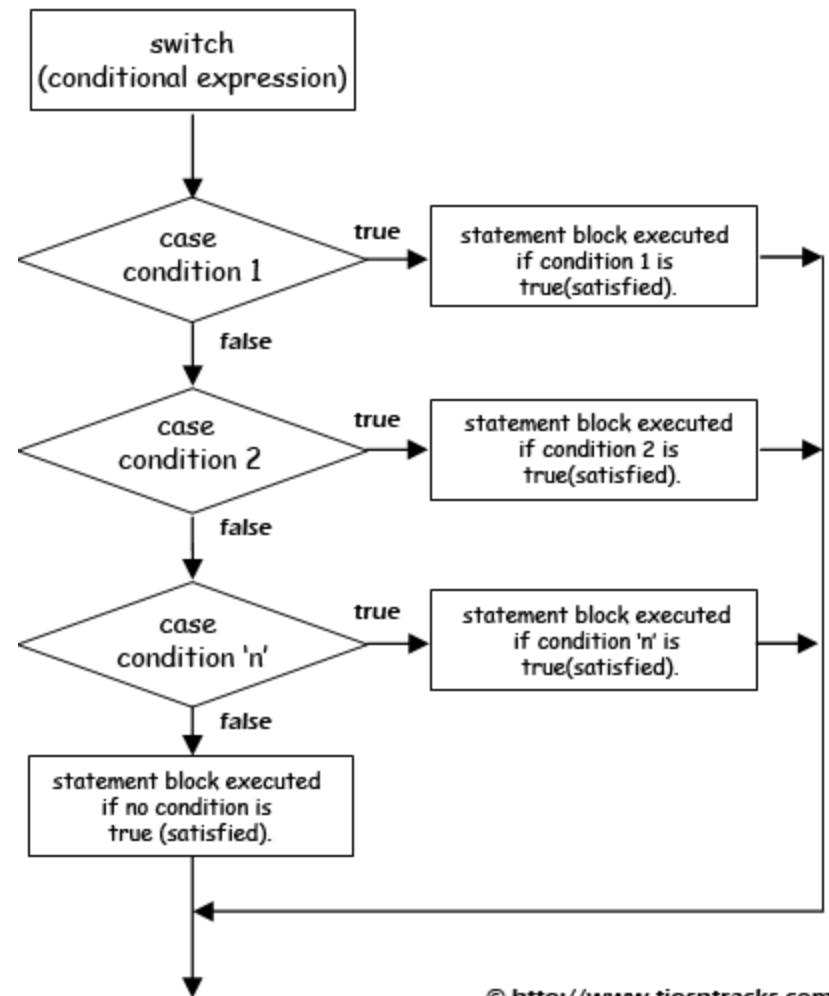
- Το if-else statement δουλεύει καλά όταν στο condition θέλουμε να περιγράψουμε μια επιλογή με **δύο** πιθανά ενδεχόμενα.
- Τι γίνεται αν η συνθήκη μας έχει πολλά ενδεχόμενα?



# Switch statement

ΣΥΝΤΑΚΤΙΚΟ:

```
switch (<condition expression>) {  
  case <condition 1>:  
    code statements 1  
    break;  
  case <condition 2>:  
    code statements 2  
    break;  
  case <condition 3>:  
    code statements 3  
    break;  
  default:  
    default statements  
    break;  
}
```





# Παράδειγμα

- Ένα πρόγραμμα που να εύχεται καλημέρα σε τρεις διαφορετικές γλώσσες ανάλογα με την επιλογή του χρήστη.

```
import java.util.Scanner;

class SwitchTest{
    public static void main(String args[])
    {
        Scanner input = new Scanner(System.in);
        String option = input.next();

        switch(option) {
            case "GR":
            case "gr":
                System.out.println("kalimera");
                break;
            case "EN":
            case "en":
                System.out.println("good morning");
                break;
            case "FR":
            case "fr":
                System.out.println("bonjour");
                break;
            default:
                System.out.println("I do not speak this language.\n" +
                    "Greek, English, French only");
        }
    }
}
```