

ΤΕΧΝΙΚΕΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Εισαγωγή στη Java

Ιστορία

- Ο **Patrick Naughton** απειλεί την Sun ότι θα φύγει.
- Τον βάζουν σε μία ομάδα αποτελούμενη από τους **James Gosling** και **Mike Sheridan** για να σχεδιάσουν τον προγραμματισμό των έξυπνων συσκευών της επόμενης γενιάς.
 - The **Green project**.
- Ο Gosling συνειδητοποιεί ότι η C++ δεν είναι αρκετά αξιόπιστη για να δουλεύει σε συσκευές περιορισμένων δυνατοτήτων και με διάφορες αρχιτεκτονικές.
 - Δημιουργεί τη γλώσσα **Oak**
- Το 1992 η ομάδα κάνει ένα demo μιας συσκευής **PDA, *7 (star 7)**
 - Δημιουργείται η θυγατρική εταιρία **FirstPerson Inc**
- Η δημιουργία των έξυπνων συσκευών αποτυγχάνει και η ομάδα (μαζί με τον **Eric Schmidt**) επικεντρώνεται στην εφαρμογή της πλατφόρμας στο **Internet**.
 - Ο Naughton φτιάχνει τον **WebRunner browser** (μετα **HotJava**)
 - Η γλώσσα μετονομάζεται σε **Java** και το ενδιαφέρον επικεντρώνεται σε εφαρμογές που τρέχουν μέσα στον browser.
- Ο **Marc Andersen** ανακοινώνει ότι ο **Netscape browser** θα υποστηρίζει Java μικροεφαρμογές (applets)

Ιστορία

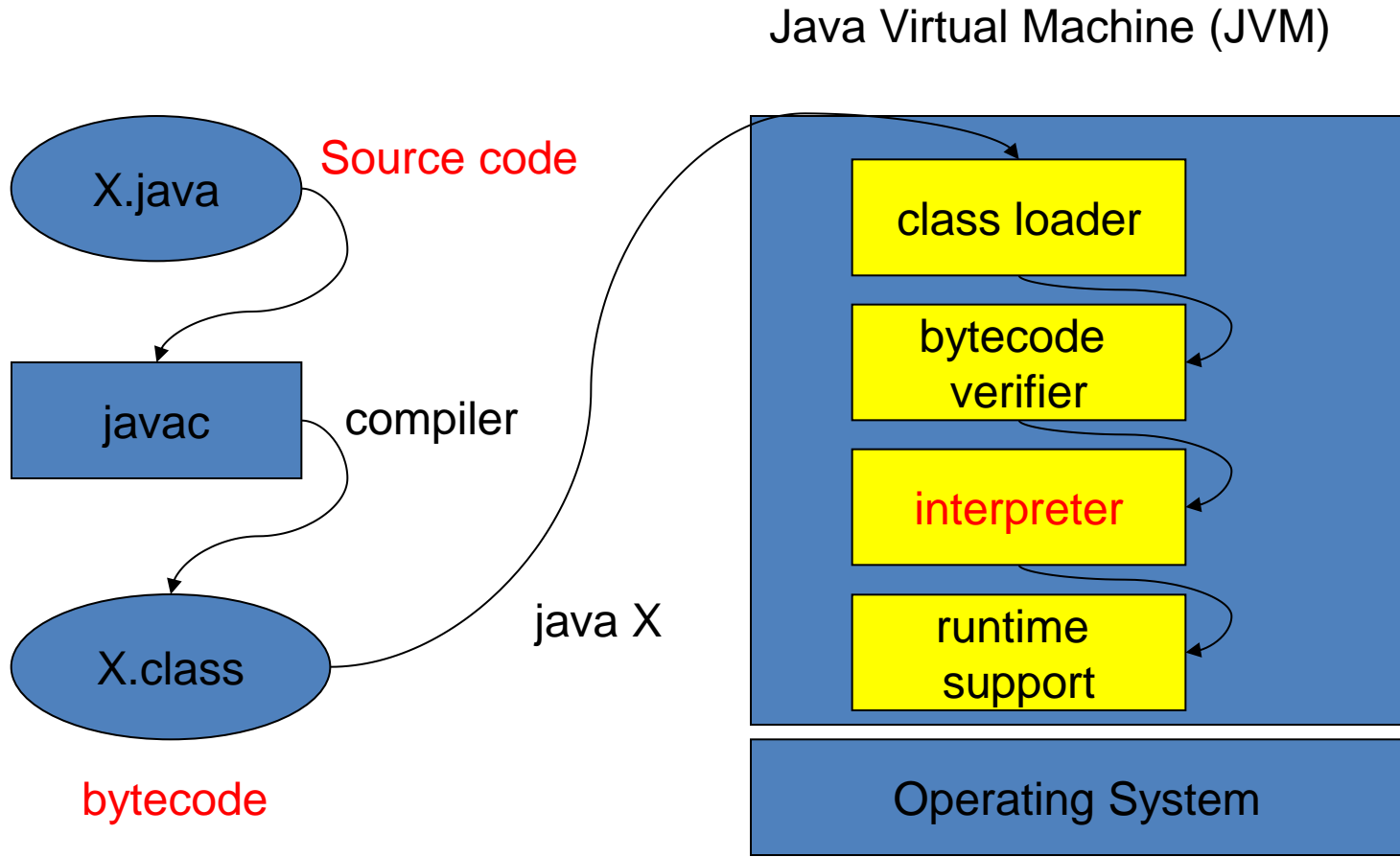
- Η Java είχε τους εξής στόχους:
 - "simple, object-oriented and familiar"
 - "robust and secure"
 - "architecture-neutral and portable"
 - "high performance"
 - "interpreted, threaded, and dynamic"

Ιστορία

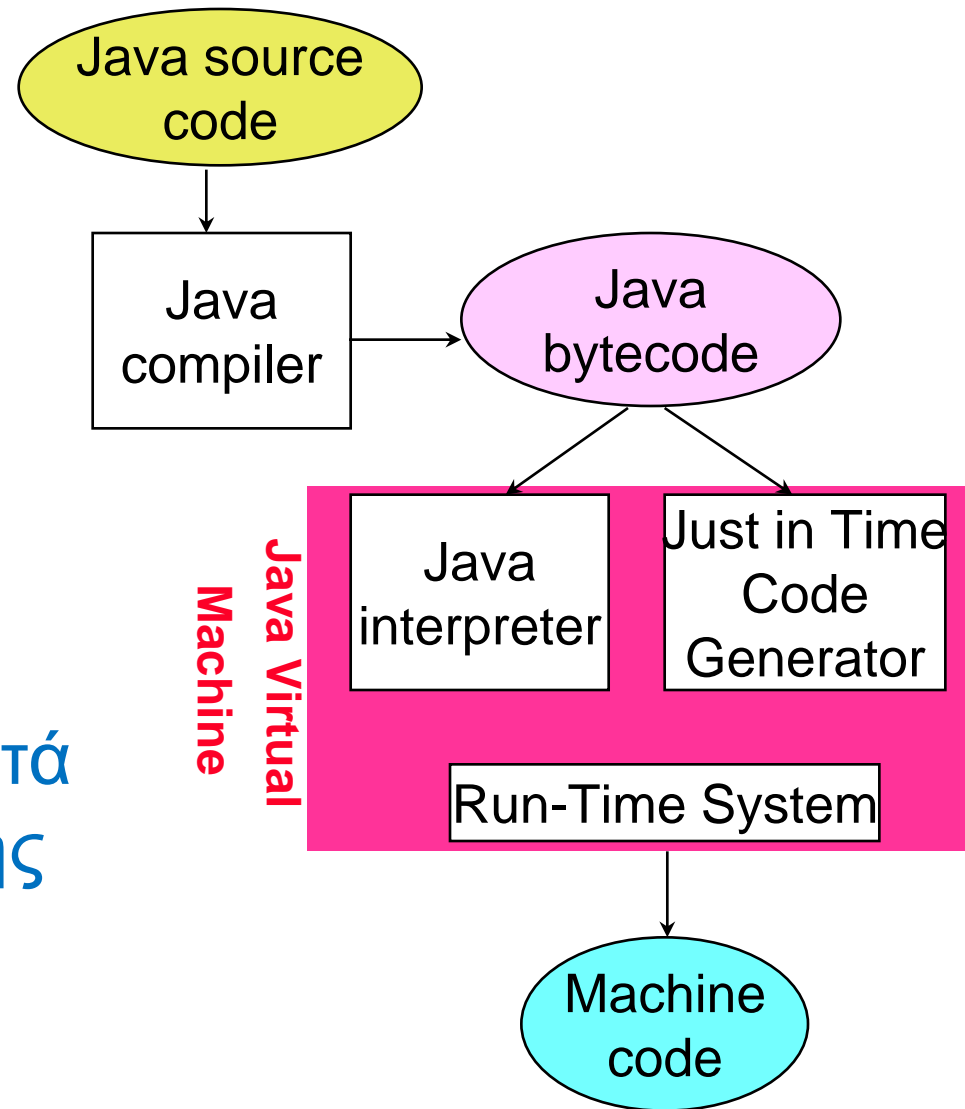
- Η Java είχε τους εξής στόχους:
 - "simple, object-oriented and familiar"
 - "robust and secure"
 - "architecture-neutral and portable"
 - "high performance"
 - "interpreted, threaded, and dynamic"

“architecture-neutral and portable”

- Το μεγαλύτερο πλεονέκτημα της Java είναι η **μεταφερισιμότητα (portability)**: ο κώδικας μπορεί να τρέξει πάνω σε οποιαδήποτε πλατφόρμα.
 - **Write-Once-Run-Anywhere** μοντέλο, σε αντίθεση με το σύνηθες **Write-Once-Compile-Anywhere** μοντέλο.
- Αυτό επιτυγχάνεται δημιουργώντας ένα **ενδιάμεσο κώδικα (bytecode)** ο οποίος μετά τρέχει πάνω σε μια **εικονική μηχανή (Java Virtual Machine)** η οποία το μεταφράζει σε **γλώσσα μηχανής**.
 - Οι προγραμματιστές πλέον γράφουν κώδικα για την εικονική μηχανή, η οποία δημιουργείται **για οποιαδήποτε πλατφόρμα**.



- **Just in Time (JIT) code generator (compiler)** βελτιώνει την απόδοση των Java Applications μεταφράζοντας (compiling) bytecode σε machine code **πριν ή κατά τη διάρκεια της εκτέλεσης**

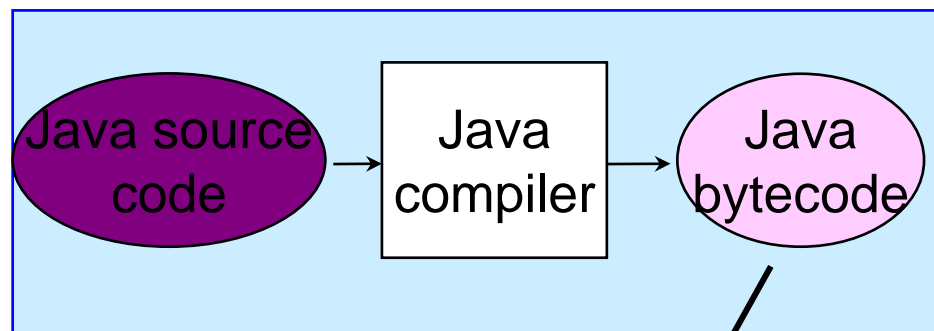


Java και το Internet

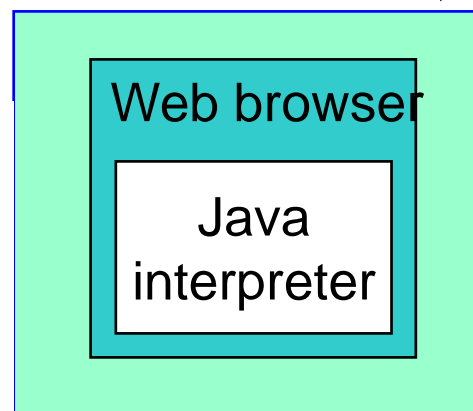
- Η προσέγγιση της Java είχε μεγάλη επιτυχία για **Web εφαρμογές**, όπου έχουμε ένα τεράστιο κατακευματισμένο **client-server** μοντέλο με πολλές διαφορετικές αρχιτεκτονικές
 - **Client-side programming**: Αντί να κάνει όλη τη δουλειά ο server για την δημιουργία της σελίδας κάποια από την επεξεργασία των δεδομένων γίνεται στη μηχανή του client.
 - **Web Applets**: κώδικας ο οποίος κατεβαίνει μαζί με τη Web σελίδα και τρέχει στη μηχανή του client. Είναι πολύ σημαντικό στην περίπτωση αυτή ο κώδικας να είναι portable.
 - **Server-side programming**: μία web σελίδα μπορεί να είναι το αποτέλεσμα ενός προγράμματος που συνδυάζει δυναμικά δεδομένα και είσοδο του χρήστη.
 - **Java Service Pages (JSPs)**: Η λύση της Java. Γίνεται compiled σε **servlets** και τρέχει στη μεριά του server.

Java Applets

- Το Web Browser software περιλαμβάνει ένα **JVM**
 - ◆ **Φορτώνει** τον java byte code από τον remote υπολογιστή
 - ◆ **Τρέχει** τοπικά το Java πρόγραμμα μέσα στο παράθυρο του Browser



Remote computer



Local computer



"simple, object-oriented and familiar"

- **Familiar:** Η Java είχε ως έμπνευση της την C++, και δανείζεται αρκετά από τα χαρακτηριστικά της.
- **Object-oriented:** Η Java είναι «**πιο αντικειμενοστραφής**» από την C++ η οποία προσπαθεί να μείνει συμβατή με την C
 - Στην Java **τα πάντα** είναι **αντικείμενα**
- **Simple:** Η Java δίνει λιγότερο έλεγχο στο χρήστη, αλλά κάνει τη ζωή του πιο εύκολη. Η **διαχείριση της μνήμης** γίνεται **αυτόματα**.
 - Η γλώσσα φροντίζει να κάνει πιο γρήγορο και πιο σταθερό (robust) τον προγραμματισμό παρότι αυτό μπορεί να έχει αποτέλεσμα τα προγράμματα να γίνονται **πιο αργά**.

HELLO WORLD

Το πρώτο μας πρόγραμμα σε Java

Δομή ενός απλού Java προγράμματος

- Το **όνομα** του αρχείου που κρατάει το πρόγραμμα είναι **X.java** (όπου **X** το όνομα του προγράμματος)
 - Στο παράδειγμα μας ονομάζουμε το πρόγραμμα μας: **HelloWorld.java**
- Μέσα στο πρόγραμμα μας πρέπει να έχουμε μια **κλάση** με το όνομα **X**.
 - **class X**
- Η κλάση **X** θα πρέπει να περιέχει μια **μέθοδο main** η οποία είναι το **σημείο εκκίνησης** του προγράμματος μας
 - **public static void main(String[] args)**

File 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!

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

Λέξεις σε κόκκινο: δεσμευμένες λέξεις

Ορίζει την
κλάση

Όνομα της κλάσης

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

```
class HelloWorld
```

```
{
```

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

```
    {
```

```
        // print message
```

```
        System.out.println("Hello world!");
```

```
    }
```

```
}
```

Τα άγκιστρα { ... } ορίζουν ένα **λογικό block** του κώδικα

- Αυτό μπορεί να είναι **μία κλάση**, **μία συνάρτηση**, **ένα if statement**
- Οι μεταβλητές που ορίζουμε μέσα σε ένα λογικό block, έχουν **εμβέλεια** μέσα στο block
- Αντίστοιχο των tabs στην Python, εδώ δεν χρειάζονται αλλά είναι καλό να τα βάζουμε για να διαβάζεται ο κώδικας πιο εύκολα.

Ορισμός της συνάρτησης main

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

Ορισμός της συνάρτησης main

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

public, static: θα τα εξηγήσουμε αργότερα

Ορισμός της συνάρτησης main

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

Το τι επιστρέφει η μέθοδος

void: Η μέθοδος δεν επιστρέφει τίποτα.

Ορισμός της συνάρτησης main

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

Το όνομα της μεθόδου

- **main**: ειδική περίπτωση που σηματοδοτεί το σημείο εκκίνησης του προγράμματος.

Ορισμός της συνάρτησης main

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

Ορίσματα της μεθόδου

- Ένας **πίνακας** από **Strings** που αντιστοιχούν στις **παραμέτρους** με τις οποίες τρέχουμε το πρόγραμμα.

Η κλάση String

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

- **String**: κλάση που χειρίζεται τα **αλφαριθμητικά**.
- Στη Java χρειάζεται να ορίσουμε τον **τύπο** της κάθε μεταβλητής
- **Strongly typed language**

Σχόλια!

```
/**  
 * A class that prints a message "hello world"  
 **/
```

```
class HelloWorld
```

```
{
```

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

```
    {
```

```
        // print message
```

```
        System.out.println("Hello world!");
```

```
    }
```

```
}
```

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

Κάθε εντολή στη Java πρέπει να τερματίζει με το ;


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

Αντικείμενο **System.out**
Ορίζει το ρεύμα εξόδου

Μέθοδος println:
Τυπώνει το String αντικείμενο που
δίνεται ως όρισμα και αλλάζει γραμμή

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

Το "Hello World" είναι ένα αντικείμενο της κλάσης String

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**.

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

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: `(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.
 - **length()**: μήκος του String
 - **equals(String x)**: ελέγχει για ισότητα του αντικειμένου που κάλεσε την μέθοδο και του ορίσματος x.
 - **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.
 - 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**
 - `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);
    }
}
```

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

Λογικοί τελεστές

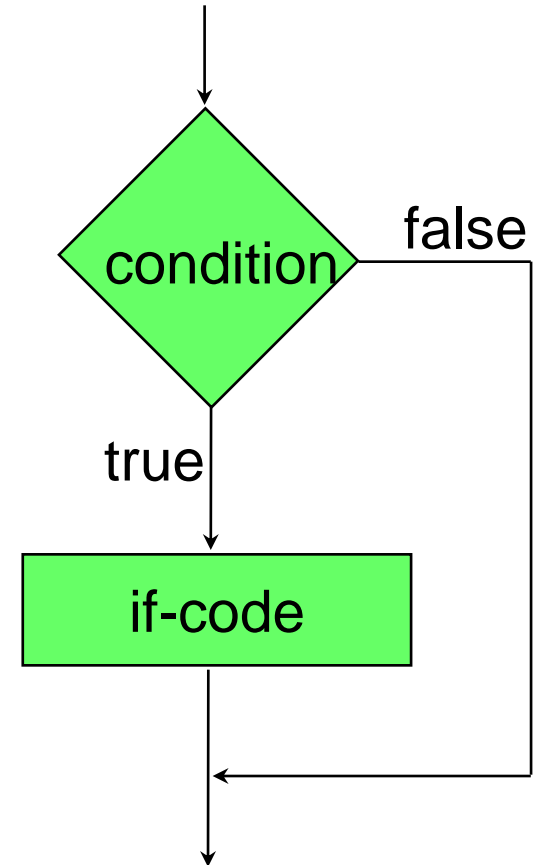
- **Λογικοί τελεστές** για λογικές εκφράσεις
 - Άρνηση: **!B**
 - ΚΑΙ: **(A && B)**
 - Ή: **(A || B)**
- Έλεγχος για βασικούς τύπους A,B:
 - Ισότητας: **(A == B)**
 - Ανισότητας: **(A != B)** ή **(!(A == B))**
 - Μεγαλύτερο/Μικρότερο ή ίσο: **(A <= B)** , **(A >= B)**
- Έλεγχος για μεταβλητές οποιουδήποτε άλλου τύπου γίνεται με την μέθοδο **equals**:
 - Ισότητας: **(A.equals(B))**
 - Ανισότητας: **(!A.equals(B))**

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.

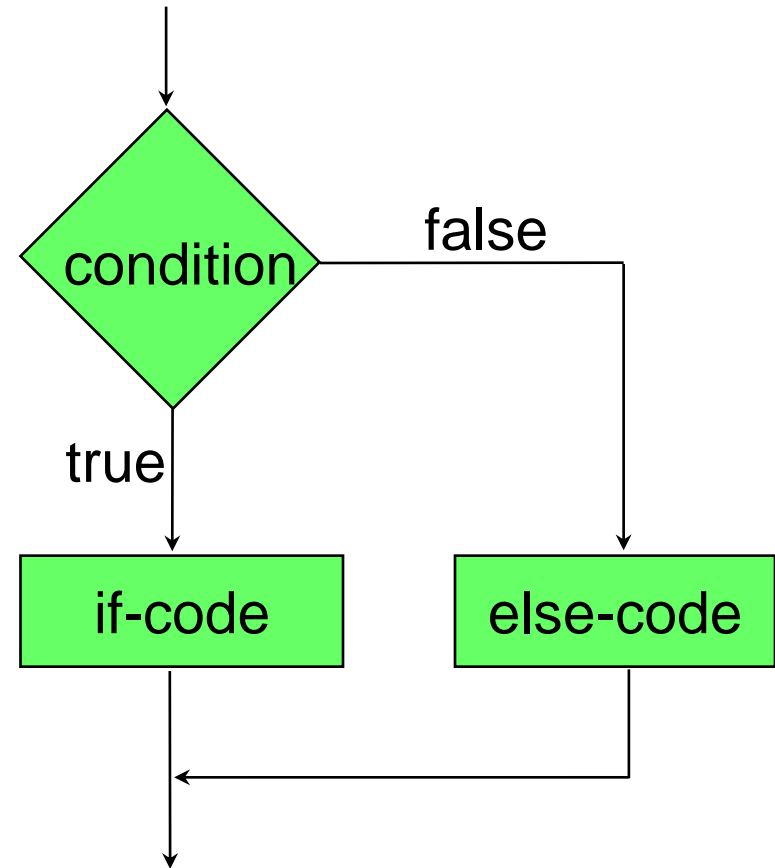


Βρόγχοι – Το if-then-else Statement

- 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.



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");
}
```

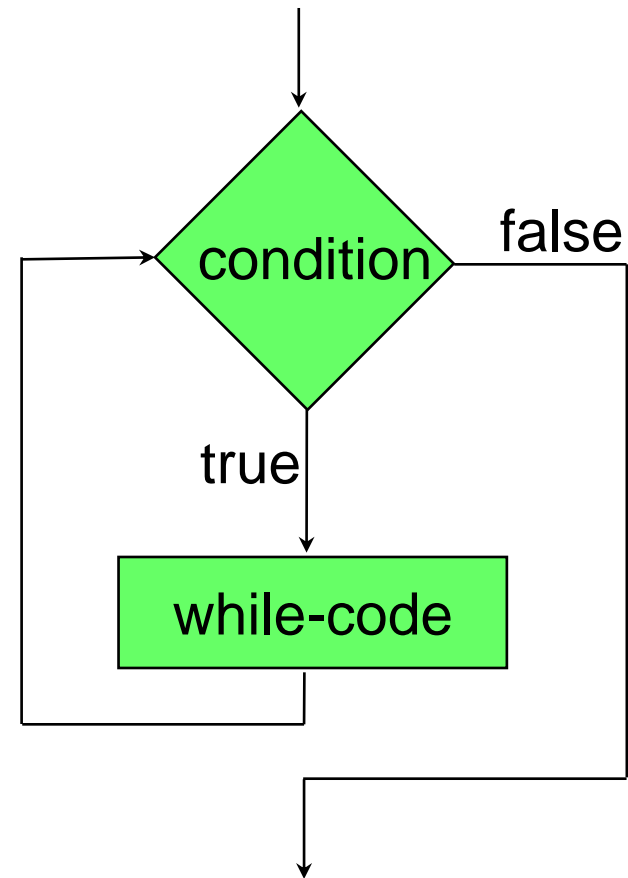
Πάντα να βάζετε { } στο σώμα των if-then-else statements.

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

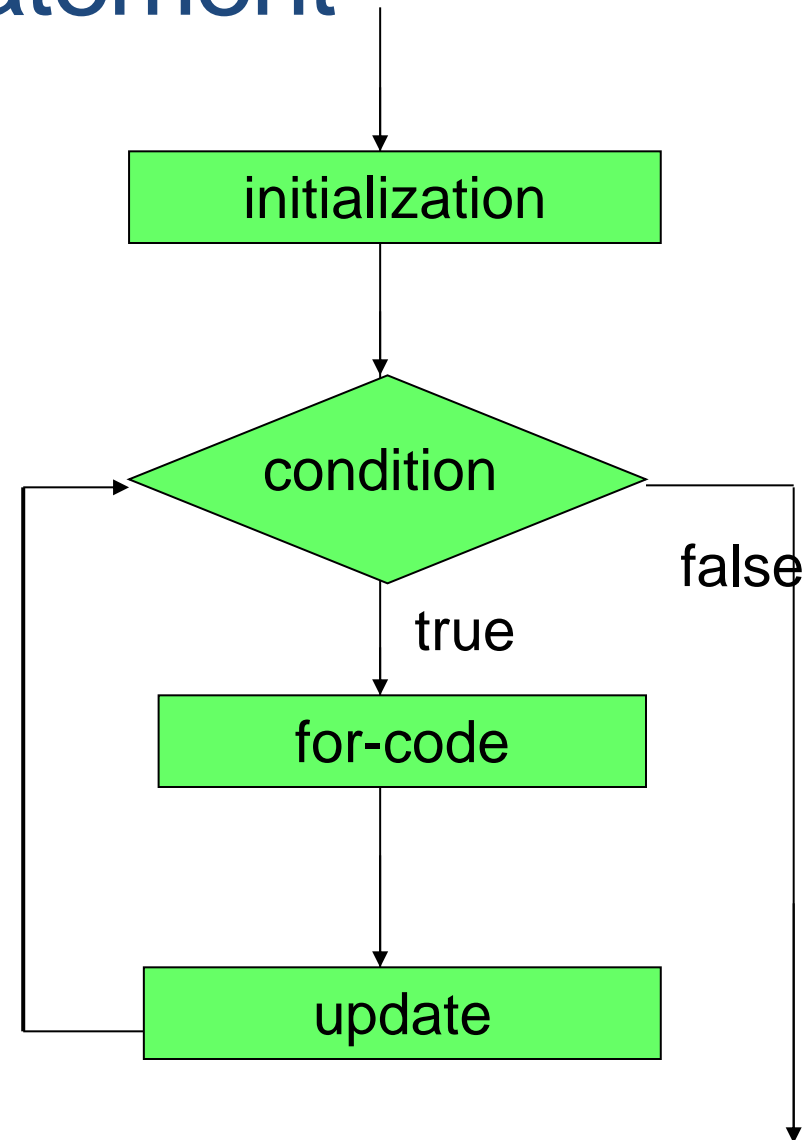
```
Scanner inputReader = new Scanner(System.in);  
String input = inputReader.next();  
  
while (input.equals("Yes"))  
{  
    System.out.println("Do you want to continue?");  
    input = inputReader.next();  
}
```


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 ++;  
}
```