# Java

Αντικειμενοστρεφής Προγραμματισμός

# Βασικά Χαρακτηριστικά OOP

- Object-oriented programming (OOP) is a method of programming based on a hierarchy of classes, and well-defined and cooperating objects
- Ο αντικειμενοστρεφής προγραμματισμός είναι μια μέθοδος προγραμματισμού βασισμένη σε μια ιεραρχία τάξεων και καλά ορισμένα αντικείμενα, τα οποία αλληλεπιδρούν μεταξύ τους

# Class (Τάξη)

- A class is a structure that defines the data and the methods to work on that data
- When you write programs in the Java language, all program data is wrapped in a class, whether it is a class you write or a class you use from the Java platform API libraries
- Τάξη είναι μια δομή που ορίζει δεδομένα και τις μεθόδους που επιδρούν πάνω στα δεδομένα

# Objects (Αντικείμενα)

- An instance is an executable copy of a class
- Another name for instance is object
- There can be any number of objects of a given class in memory at any one time
- Στιγμιότυπο είναι ένα εκτελέσιμο αντίγραφο μιας κλάσης
- Τα στιγμιότυπα τα λέμε και αντικείμενα
- Μπορεί να υπάρξει ένας οποιοσδήποτε αριθμός από αντικείμενα μιας τάξης στη μνήμη, ανά πάσα στιγμή

# Class vs Object

- Class:  A class is a blueprint that describes the states and/or behaviors that objects of its type support
- Object:  An object is an instance of a class. Objects have states and behaviors. E.g. Object states have values and/or Objects call their methods

# Interface

- In the Java programming language, an *interface* is a reference type, similar to a class, that can contain *only* constants, method signatures, default methods, static methods, and nested types
- Method bodies exist only for default methods and static methods. Interfaces cannot be instantiated—they can only be implemented by classes or extended by other interfaces
- Στην Java, το interface είναι reference type, με ομοιότητες με την τάξη, το οποίο μπορεί να περιέχει μόνο σταθερές, υπογραφές μεθόδων, default μεθόδους, στατικές μεθόδους και εμφωλευμένους τύπους
- Σώμα μεθόδων υπάρχει μόνο σε αυτές που είναι δηλωμένες ως static ή default. Τα interfaces δεν παράγουν από μόνα τους αντικείμενα. Τα interfaces υλοποιούνται από τάξεις, ή επεκτείνονται από άλλα interfaces

# Προγραμματιστικά Συμβόλαια

- Implementing an interface allows a class to become more formal about the behavior it promises to provide
- Interfaces form a contract between the class and the outside world, and this contract is enforced at build time by the compiler
- If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile
- Το interface παρέχει ένα είδος «συμβολαίου» το οποίο περιμένουμε να «τηρήσει» μια τάξη και είναι κάτι που ελέγχεται κατά τη μεταγλώττιση

# Data Types

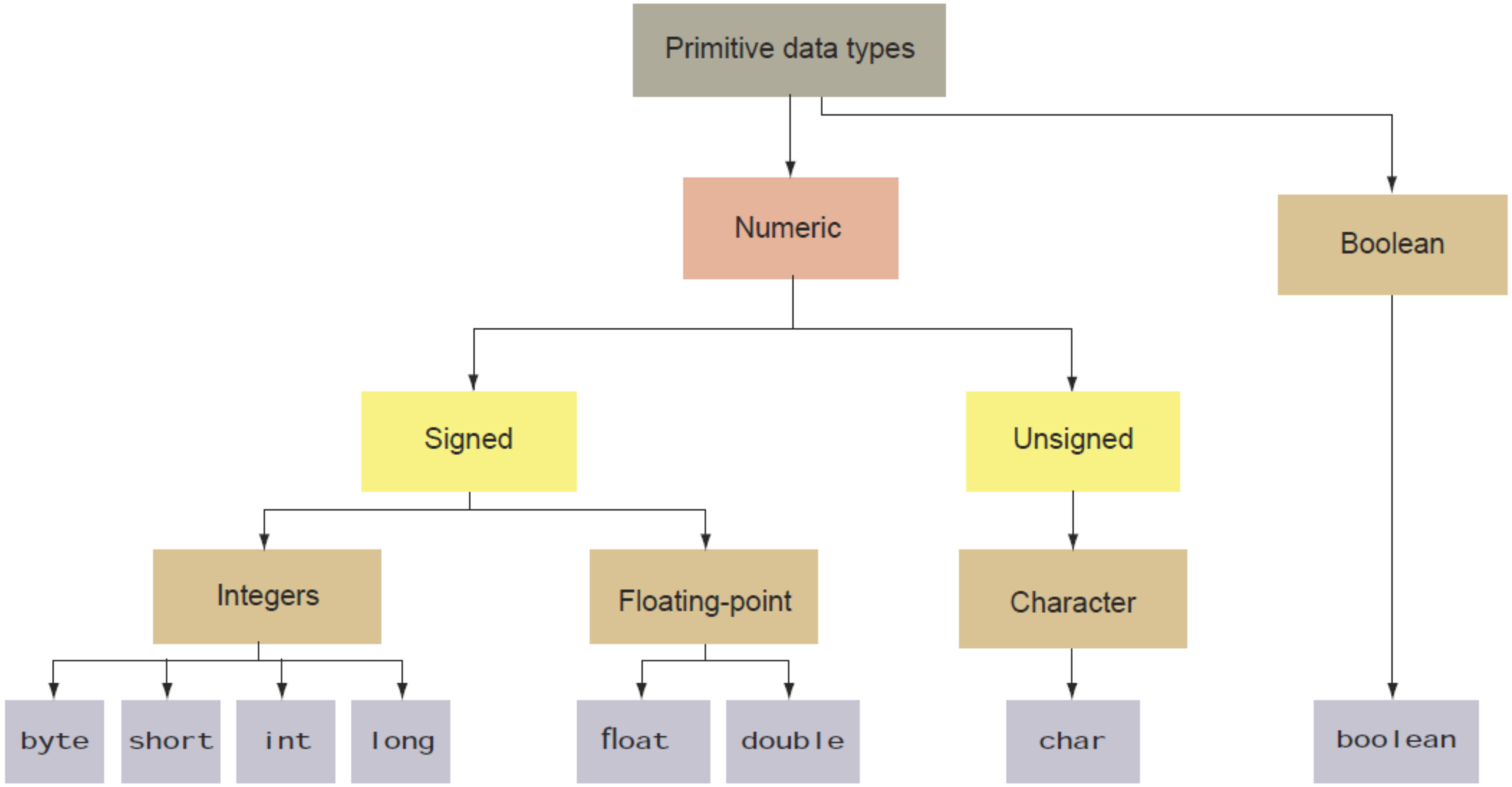
# Java Data Types

- Οι 2 μεγάλες κατηγορίες μεταβλητών στη Java είναι οι:
  - Primitive variables
  - Reference variables

# Primitive Data Types

- Στη Java υπάρχουν 8:

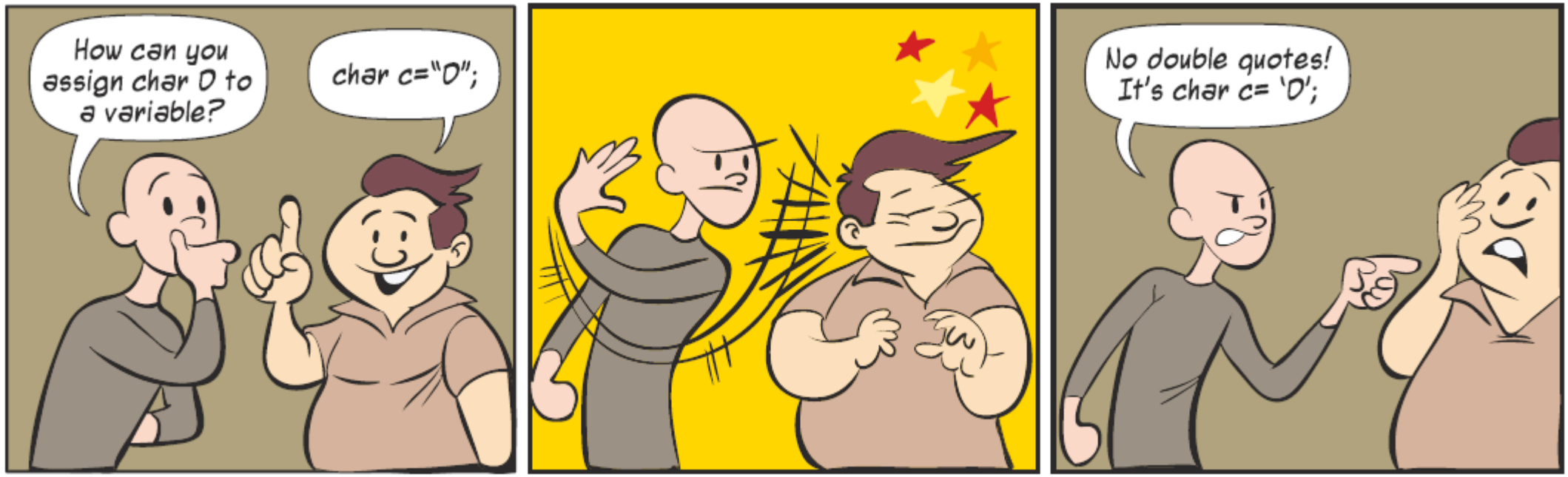
  - `char`
  - `byte`
  - `short`
  - `int`
  - `long`
  - `float`
  - `double`
  - `boolean`


Primitive data types
Numeric
Boolean
Signed
Unsigned
Integers
Floating-point
Character
byte
short
int
long
float
double
char
boolean

# Ενδεικτικός πίνακας εύρους τιμών

| Data type | Size | Range of values |
|---|---|---|
| `byte` | 8 bits | –128 to 127, inclusive |
| `short` | 16 bits | –32,768 to 32,767, inclusive |
| `int` | 32 bits | –2,147,483,648 to 2,147,483,647, inclusive |
| `long` | 64 bits | –9,223,372,036,854,775,808 to 9,223,372,036,854,775,807, inclusive |

# Προσοχή στους char!



# Ονοματοδοσία για identifiers

| Properties of valid identifiers | Properties of invalid identifiers |
|---|---|
| Unlimited length | Same spelling as a Java reserved word or keyword |
| Starts with a letter ( a–z, upper- or lowercase), a currency sign, or an underscore | Uses special characters: !, @, #, %, ^, &, *, (, ), ', :, ;, [, /, \, } |
| Can use a digit (not at the starting position) | Starts with a Java digit (0–9) |
| Can use an underscore (at any position) | |
| Can use a currency sign (at any position): ¤, $, £, ¢, ¥, and others | |

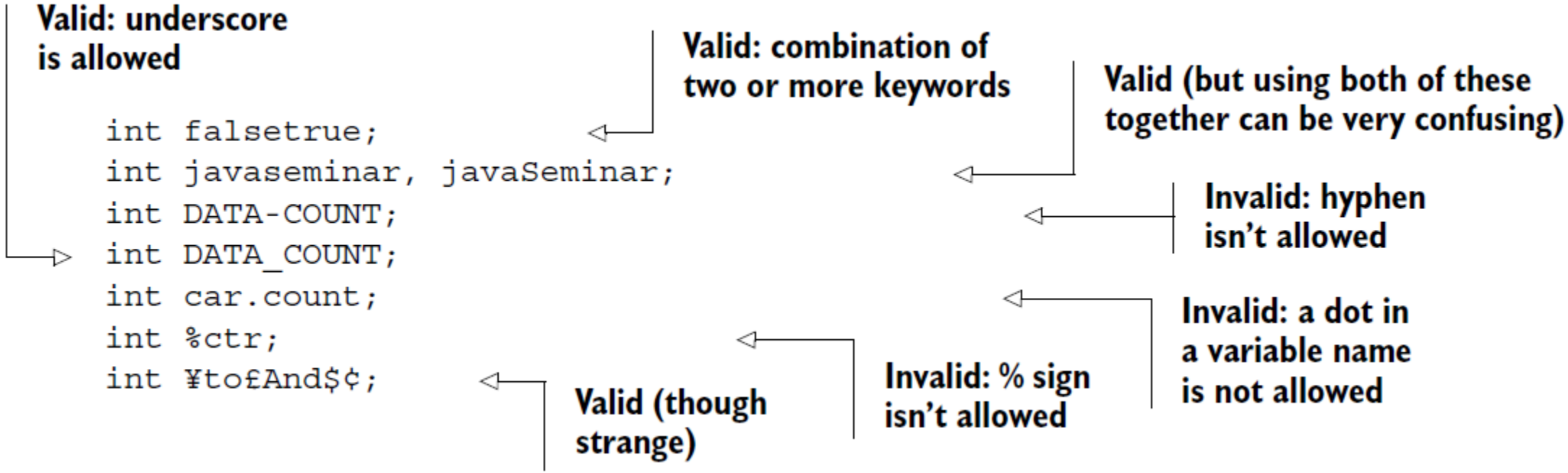
# Παραδείγματα

| Examples of valid identifiers | Examples of invalid identifiers |
|---|---|
| `customerValueObject` | `7world` (identifier can't start with a digit) |
| `$rate, £Value, _sine` | `%value` (identifier can't use special char `%`) |
| `happy2Help, nullValue` | `Digital!, books@manning` (identifier can't use special char `!` or `@`) |
| `Constant` | `null, true, false, goto` (identifier can't have the same name as a Java keyword or reserved word) |

# Δεσμευμένες λέξεις Java

| | | | | |
|---|---|---|---|---|
| abstract | default | goto | package | this |
| assert | do | if | private | throw |
| boolean | double | implements | protected | throws |
| break | else | import | public | transient |
| byte | enum | instanceof | return | true |
| case | extends | int | short | try |
| catch | false | interface | static | void |
| char | final | long | strictfp | volatile |
| class | finally | native | super | while |
| const | float | new | switch | |
| continue | for | null | synchronized | |

# Παραδείγματα

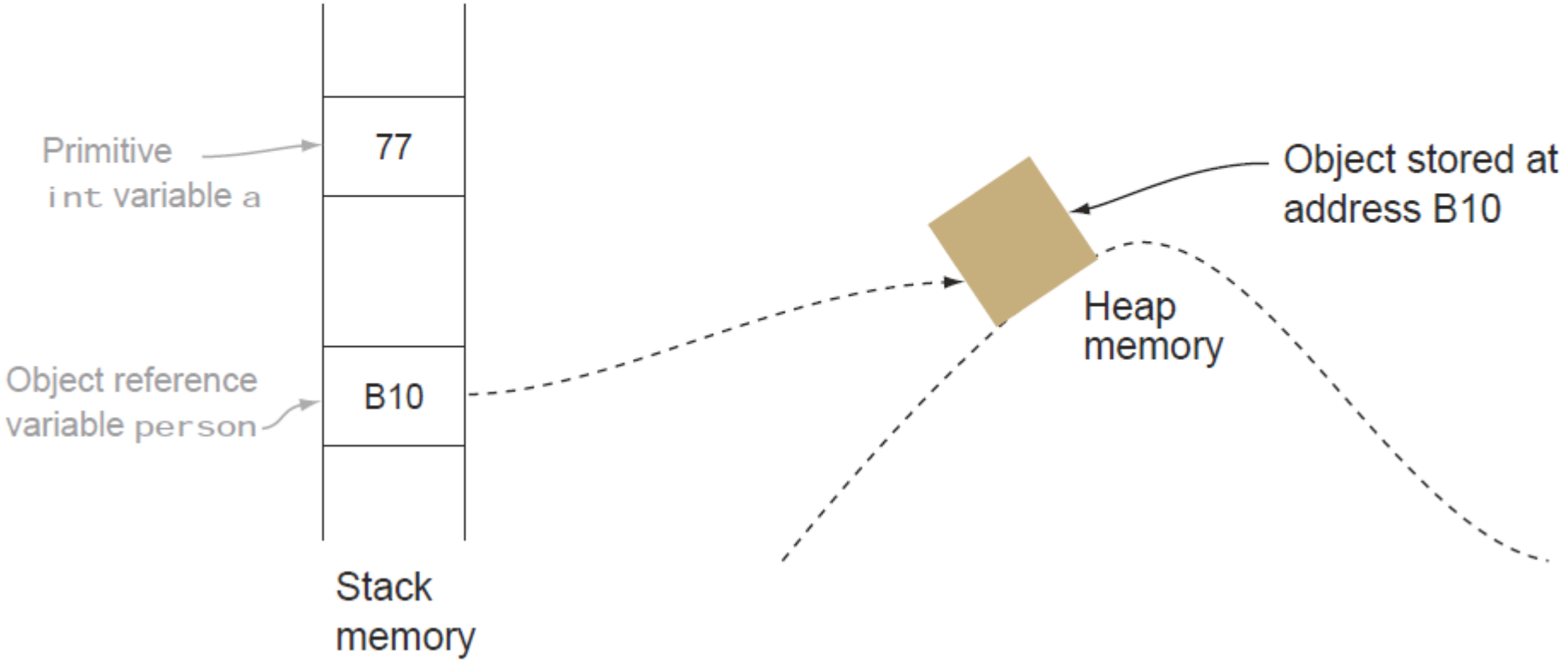


# Reference Variables

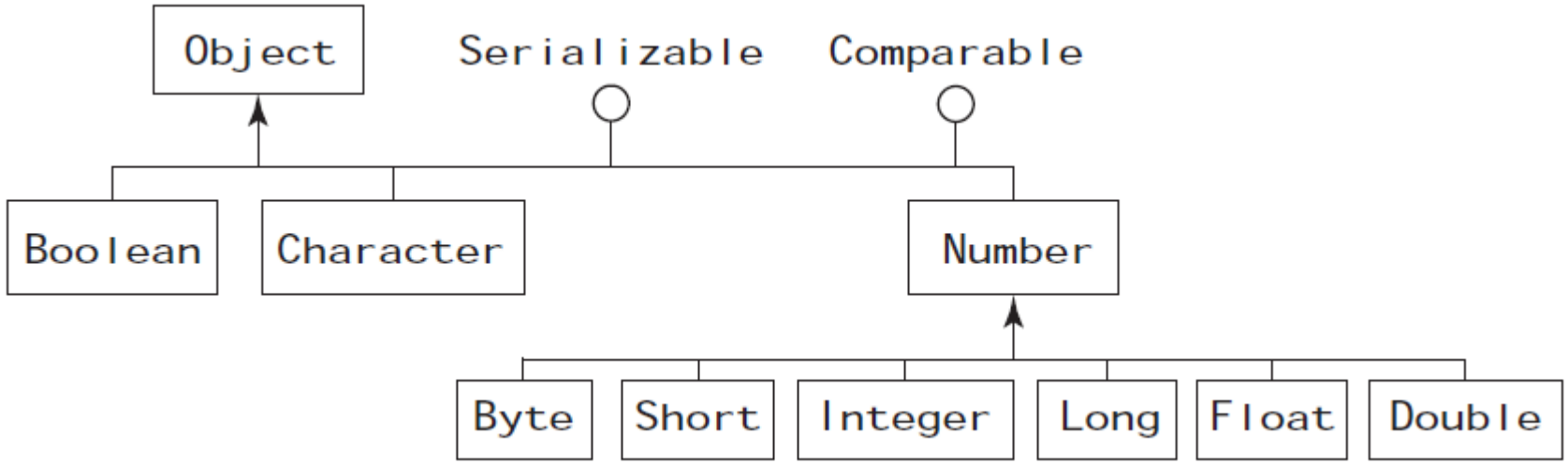
- Αλλιώς Object Variables
- Default τιμή -> null

# References Vs Values

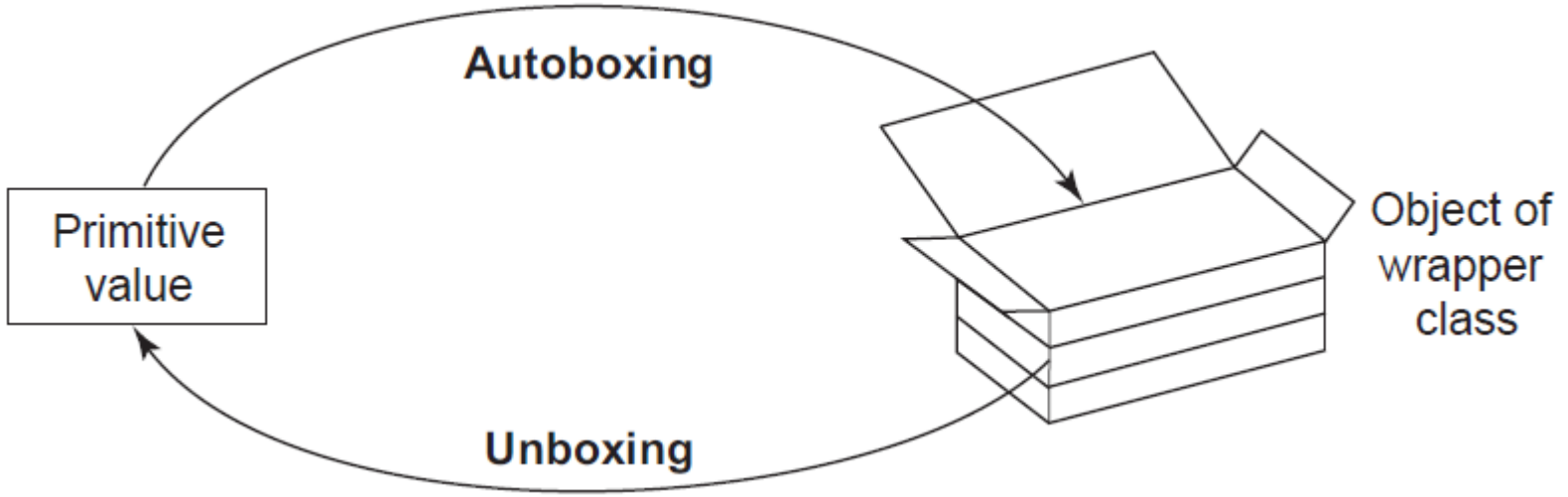
```
int a = 77;
Person person = new Person();
```



# Wrapper Classes

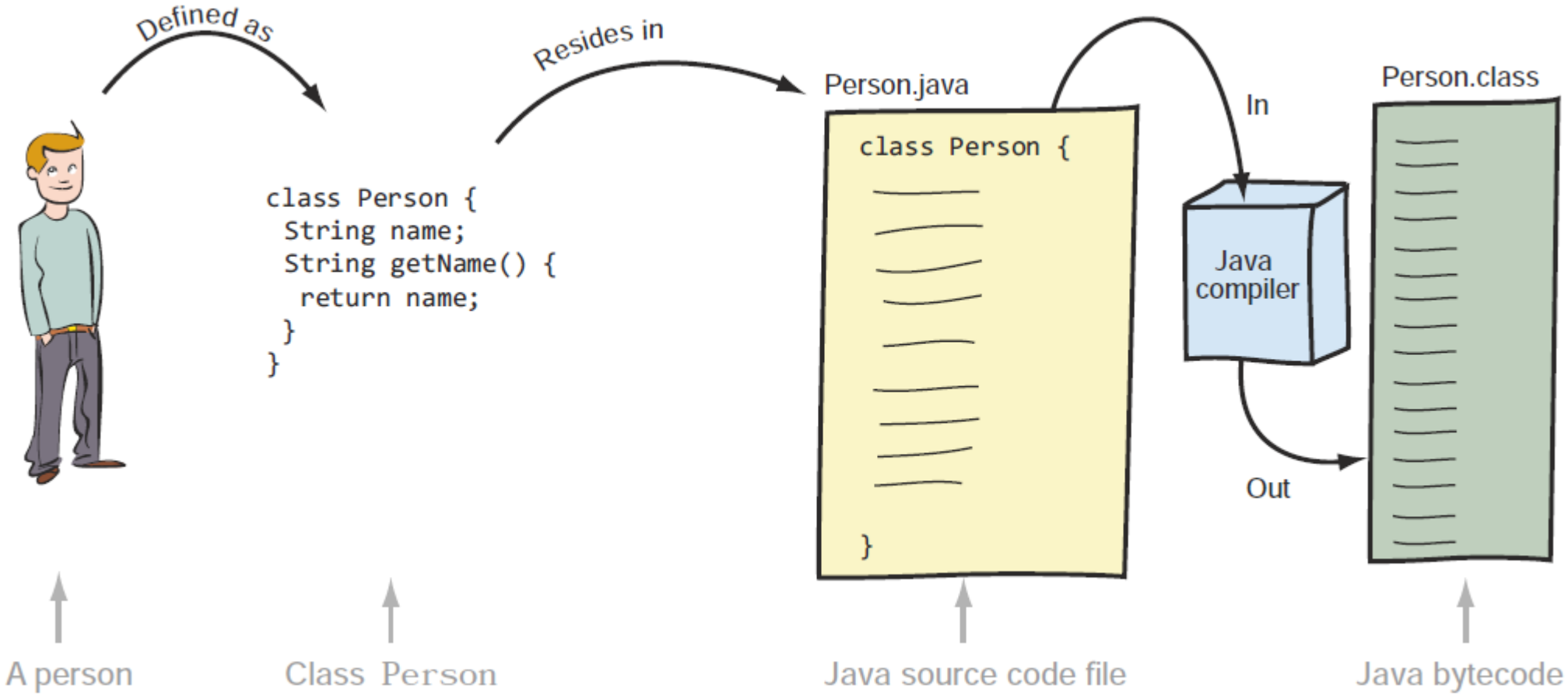


# Autoboxing - Unboxing



# Java Basics

# Java program execution



# Java – Compile and Run

- Compile one java file: javac Student.java
- Compile two java files: javac Student.java StudentTester.java
- Compile all java files in current directory: javac *.java
- Run a Java class file: java StudentTester

# Important Notes

- The Java compiler is smart enough to compile all source files that are needed by the source files that you specify. For example, the StudentTester class requires the Student class. When you compile StudentTester.java, the compiler automatically compiles Student.java.
- The javac command takes file names as input. The java command takes a class name, without the .java or .class extension.

# Java class

Package statement
Import statements
Comments
Class declaration {
    Variables
    Comments
    Constructors
    Methods
    Nested classes
    Nested interfaces
    Enum
}

# Package

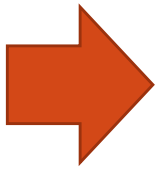
- Κάθε τάξη ανήκει σε ένα πακέτο
- Η δήλωση μιας τάξης σε ένα πακέτο μπορεί να γίνει άμεσα με τη δήλωση του πακέτου, ή έμμεσα, χωρίς τη δήλωση ενός πακέτου, όπου στην περίπτωση αυτή δηλώνεται το default package (το οποίο δεν διαθέτει όνομα)
- Εάν δηλωθεί πακέτο σε μια τάξη, τότε η εν λόγω δήλωση θα πρέπει να είναι η πρώτη δήλωση στο αρχείο (πριν από την τάξη) (με μοναδική «εξαίρεση» τα σχόλια)
- Εάν υπάρχει δήλωση πακέτου, τότε αυτή είναι και η μοναδική για την τάξη που το δηλώνει (δεν επιτρέπεται άλλη δήλωση στο ίδιο αρχείο)

# Packages and Directories

- Η ιεραρχία των τάξεων και των interfaces εντός των packages θα πρέπει να διατηρηθεί και σε επίπεδο καταλόγων/φακέλων
- Δεν έχει σημασία που θα τοποθετηθεί ο αρχικός κατάλογος του package, αρκεί να τηρηθεί η ιεραρχία των sub-packages εφόσον υπάρχουν

# Example

```
package com.test;

class Test {
     public static void main(String[] args){
          System.out.println("Hello World!");
     }
}
```



```
└── com
    └── test
        └── Test.java
```

# Comments

- Σχόλια μπορούν να εμφανιστούν σε πολλά σημεία στον κώδικα, πριν και μετά τη δήλωση του package, πριν και μετά τη δήλωση τάξεων, πριν, μετά και εντός τη δήλωση μεθόδων
- Υπάρχουν σχόλια μια γραμμής και πολλαπλών γραμμών

```
class MyClass {
    /*
      comments that span multiple
      lines of code
    */
}
```

**Multiline comments start with /* and end with */.**

```
class MyClass {
    /*
       Multi-line comments with
       special characters &%^*{}|\|:;"'
       ?/>.<,!@#$%^&*()
    */
}
```

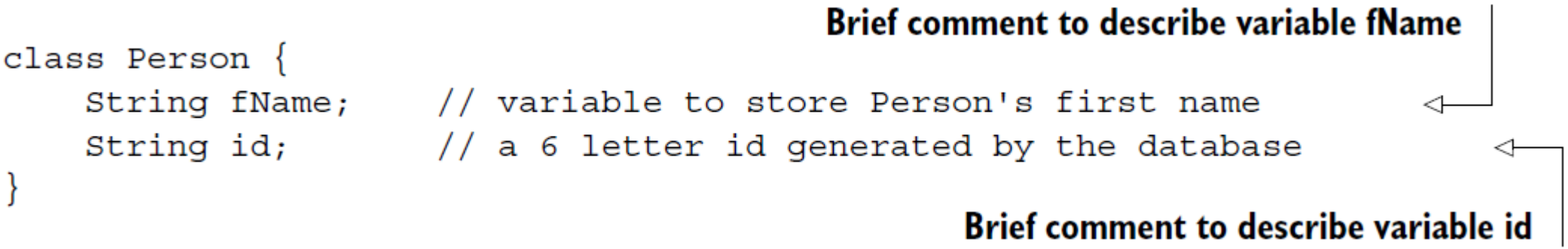
**Multiline comment with special characters in it**

```
class MyClass {
    /*
     * comments that span multiple
     * lines of code
     */
}
```

**Multiline comments that start with * on a new line—don't they look well organized? The usage of * isn't mandatory; it's done for aesthetic reasons.**

# End of line comments

```
class Person {
    String fName;     // variable to store Person's first name
    String id;        // a 6 letter id generated by the database
}
```



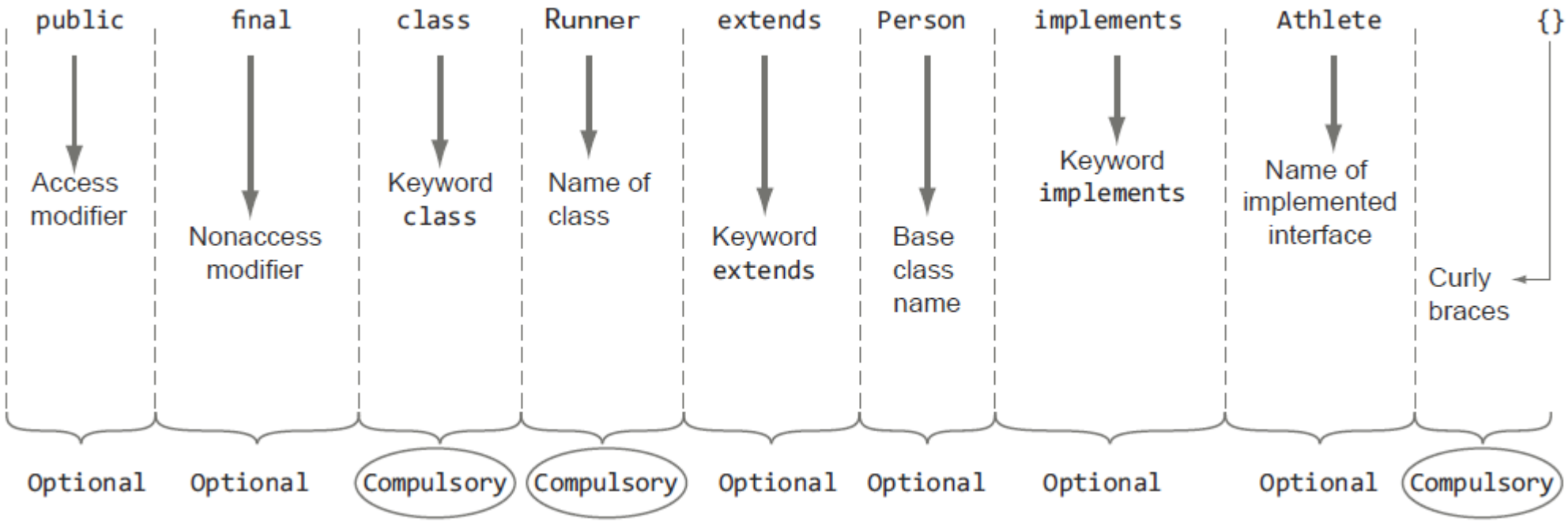
# Question

- Comments inside code?

# Class declaration

- Access modifiers
- Non-access modifiers
- Class name
- Extended class if present
- Implemented Interfaces (all) if any
- Class body:
  - Fields (if any)
  - Methods (if any)
  - Constructors (if any)
  - Inner classes (if any)

# Example

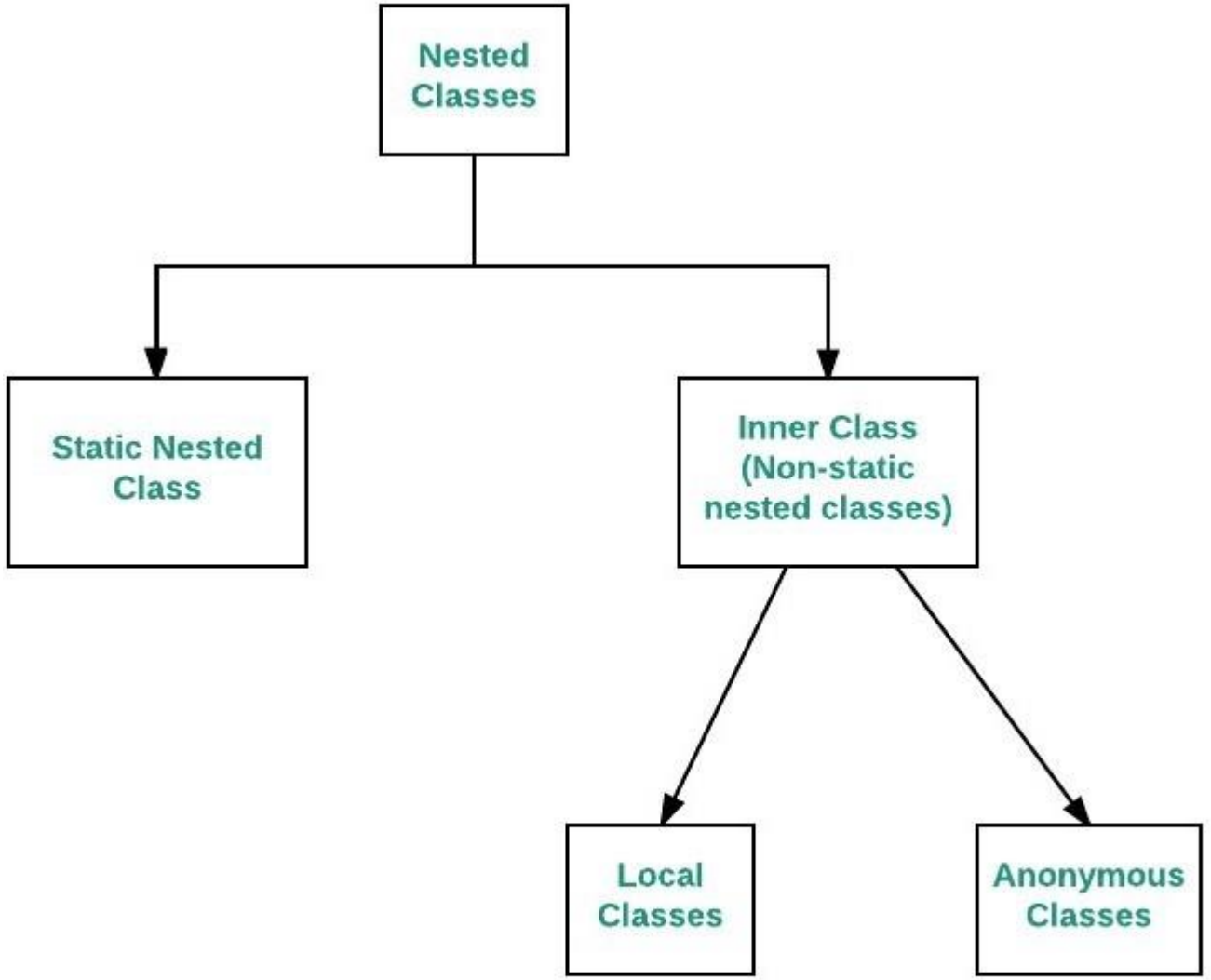

public
final
class
Runner
extends
Person
implements
Athlete
{}
Access modifier
Nonaccess modifier
Keyword class
Name of class
Keyword extends
Base class name
Keyword implements
Name of implemented interface
Curly braces
Optional
Optional
Compulsory
Compulsory
Optional
Optional
Optional
Optional
Compulsory

# Example

```
class Phone {
    String model;
    String company;
    Phone(String model) {
        this.model = model;
    }
    double weight;
    void makeCall(String number) {
        // code
    }
    void receiveCall() {
        // code
    }
}
```

# Top level Class Vs Nested

- Μια τάξη η οποία δεν περιέχεται σε κάποια άλλη ονομάζεται «Top level» class
- Μια τάξη η οποία περιέχεται (ο κώδικάς της βρίσκεται εντός του κώδικα μιας άλλης τάξης) ονομάζεται nested class (εμφωλευμένη)


Nested Classes
Static Nested Class
Inner Class (Non-static nested classes)
Local Classes
Anonymous Classes

# Take a note!

- Κάθε αρχείο πηγαίου κώδικα Java μπορεί να περιέχει μέχρι μία Top level public class ή interface
- Ένα αρχείο πηγαίου κώδικα Java μπορεί να περιέχει πολλές τάξεις ή/και διεπαφές (intefaces) και μάλιστα με οποιαδήποτε σειρά εμφάνισης

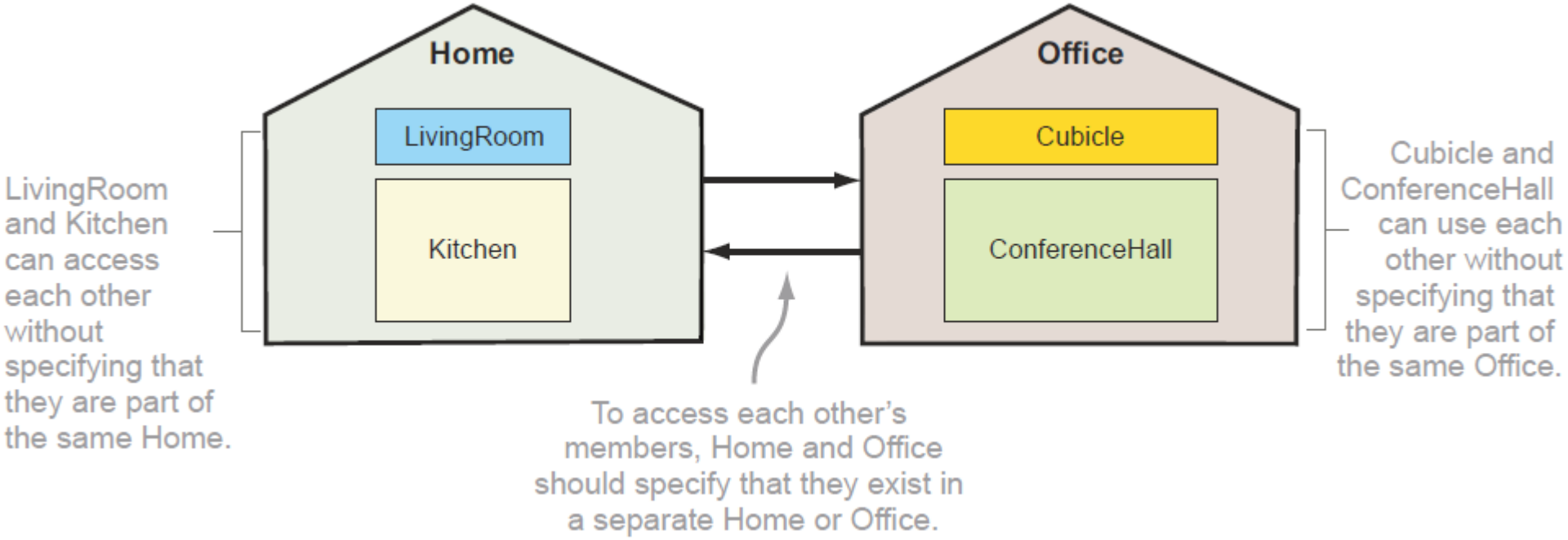
# Fully qualified names

- Για μια τάξη ή ένα interface το fully qualified name τους είναι ο συνδυασμός του package name στο οποίο ανήκουν με το όνομα της τάξης ή του interface, το ένα μετά το άλλο (πρώτα το package name) με την προσθήκη . (dot) ανάμεσά τους

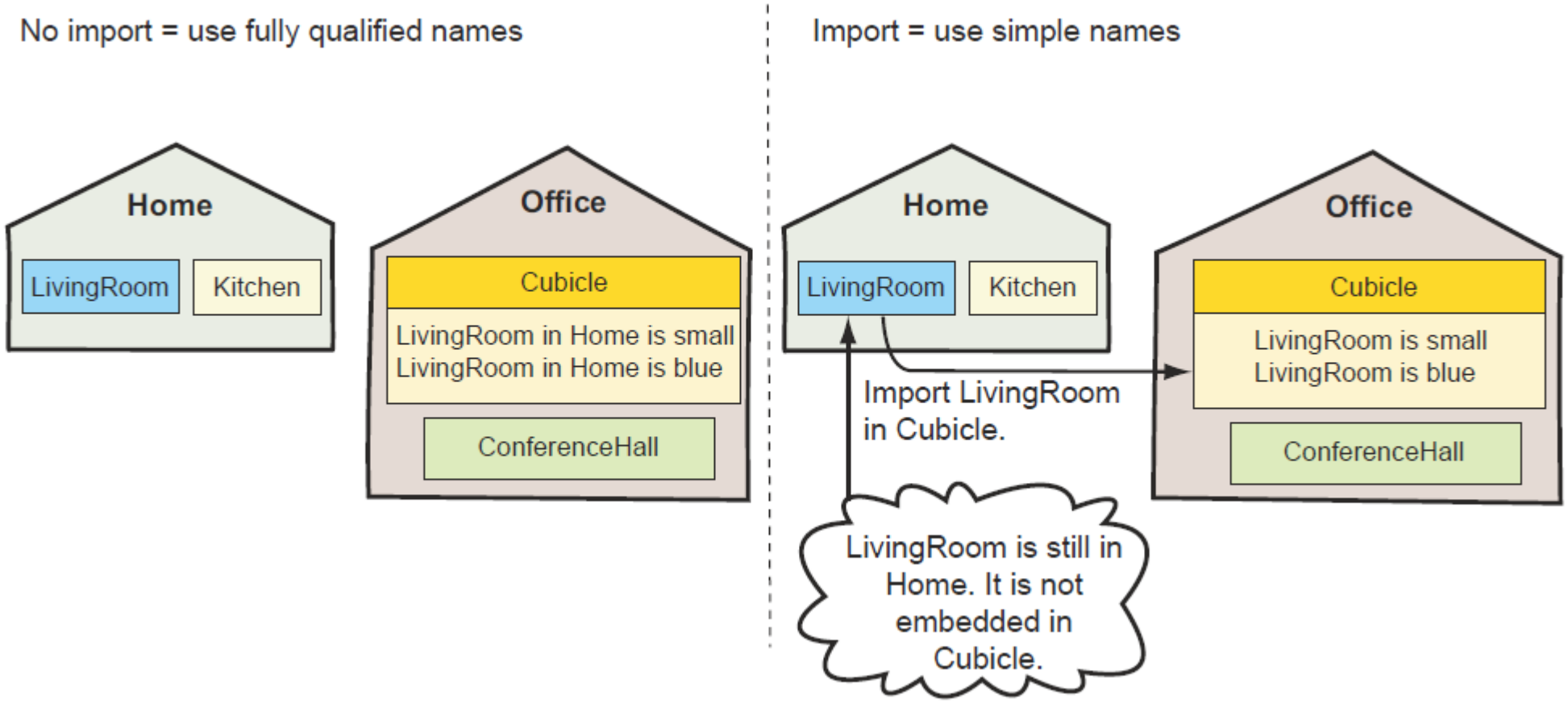
# Import statements

- Εντός του ιδίου πακέτου, οι τάξεις και οι διεπαφές μπορούν να χρησιμοποιήσουν τα μεταξύ τους στοιχεία χωρίς κάποιο πρόθεμα
- Για να μπορέσουμε να χρησιμοποιήσουμε τάξεις και διεπαφές από άλλα πακέτα, πρέπει να δηλώσουμε το «πλήρες» όνομά τους (fully qualified name)
- Επειδή η δήλωση των πλήρων ονομάτων πολλές φορές «μπερδεύει» μπορούμε να προχωρήσουμε στη δήλωση import η οποία θα μας επιτρέψει να χρησιμοποιήσουμε τα «απλά» ονόματα των τάξεων και των διεπαφών (χωρίς το fully qualified name)
- Ένα import statement βρίσκεται πάντα μετά τη δήλωση του package (αν υπάρχει) και πριν τη δήλωση της τάξης ή της διεπαφής

# Example


Home
LivingRoom
Kitchen
Office
Cubicle
ConferenceHall
LivingRoom and Kitchen can access each other without specifying that they are part of the same Home.
Cubicle and ConferenceHall can use each other without specifying that they are part of the same Office.
To access each other's members, Home and Office should specify that they exist in a separate Home or Office.

# Import usage 1/2



# Import usage 1/2

```
package office;
class Cubicle {
    home.LivingRoom livingRoom;
}
```

In the absence of an import statement, use the fully qualified name to access class LivingRoom.

VS

```
package office;
import home.LivingRoom;
class Cubicle {
    LivingRoom livingRoom;
}
```

import statement

No need to use the fully qualified name of class LivingRoom

# Importing Packages and Classes

```
import package.name.ClassName;    // To import a certain class only
import package.name.*    // To import the whole package
```

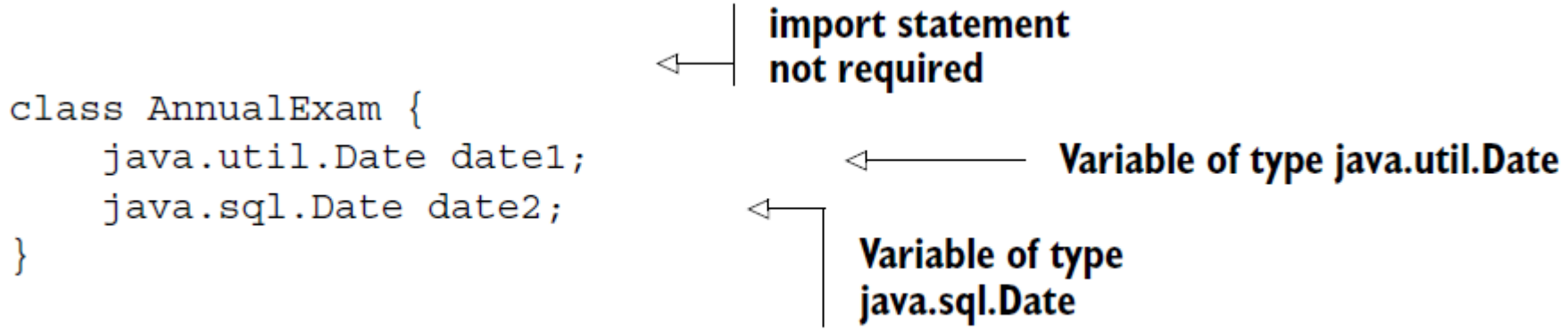
# Υπόδειξη

- Χρησιμοποιώντας τον ειδικό χαρακτήρα * μπορούμε να κάνουμε import σε όλες τις τάξεις και τα interfaces εντός του εν λόγω πακέτου
- Ωστόσο, η χρήση του * δεν υποδηλώνει ότι έτσι κάνουμε import και σε όλες τις τάξεις ή/και τα interfaces που βρίσκονται εντός των όποιων εμπλεκόμενων sub-packages

# Υπόδειξη

- Η μοναδική περίπτωση όπου δεν χρειαζόμαστε import για να χρησιμοποιήσουμε μέλη από άλλα packages είναι η περίπτωση του package «java.lang»
- Όλες οι τάξεις και τα interfaces εντός του java.lang package γίνονται «αυτόματα» import σε όλες τις τάξεις και τα interfaces

# Προσοχή σε ειδικές περιπτώσεις



# Τι ισχύει με το default package?

```
class Person {
    // code
}
class Office {
    Person p;
}
```

Not defined in an explicit package

Class Person accessible in class Office

# Static imports

- Εκτός από τα imports τάξεων και interfaces υπάρχει και η δυνατότητα να κάνουμε import σε μεταβλητές (χαρακτηριστικά) ή/και σε μεθόδους τάξεων
- Για να συμβεί αυτό χρειάζεται:
  - Τα εν λόγω μέλη να έχουν δηλωθεί ως static
  - Στο αντίστοιχο import να περιλάβουμε τη διαδρομή που μας ενδιαφέρει με τη χρήση του «import static»
- Π.χ. import static com.unipi.talepis.MyClass.AFM;
- ή όλα τα στατικά μέλη με import static com.unipi.talepis.*

# Example

```
package certification;
public class ExamQuestion {
    static public int marks;
    public static void print() {
        System.out.println(100);
    }
}
```

public static variable marks

public static method print

```
package university;
import static certification.ExamQuestion.marks;
class AnnualExam {
    AnnualExam() {
        marks = 20;
    }
}
```

Correct statement is import static, not static import

Access variable marks without prefixing it with its class name

# Naming conventions 1/6

| | | |
|---|---|---|
| Packages | The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981.<br><br>Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names. | com.sun.eng<br><br>com.apple.quicktime.v2<br><br>edu.cmu.cs.bovik.cheese |

# Naming conventions 2/6

| | | |
|---|---|---|
| Classes | Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML). | class Raster;<br>class ImageSprite; |

# Naming conventions 3/6

| | | |
|---|---|---|
| Interfaces | Interface names should be capitalized like class names. | interface RasterDelegate;<br>interface Storing; |

# Naming conventions 4/6

| | | |
|---|---|---|
| Methods | Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized. | run();<br>runFast();<br>getBackground(); |

# Naming conventions 5/6

| | | |
|---|---|---|
| Variables | Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.<br><br>Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are `i`, `j`, `k`, `m`, and `n` for integers; `c`, `d`, and `e` for characters. | `int i;`<br>`char c;`<br>`float myWidth;` |

# Naming conventions 6/6

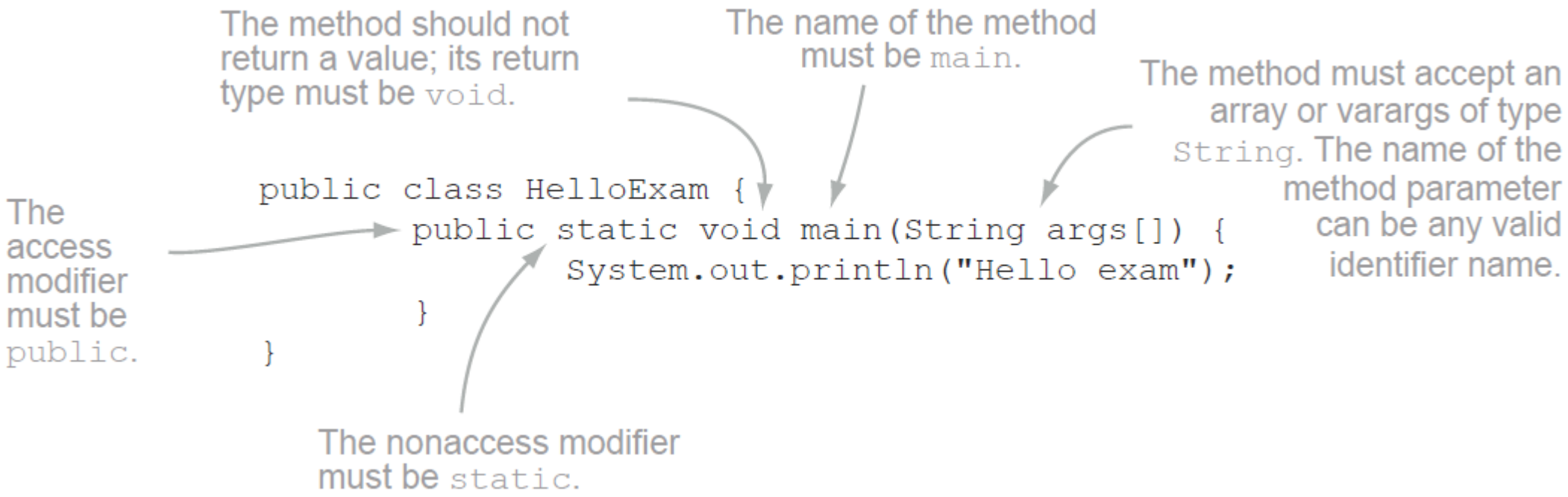
| | | |
|---|---|---|
| Constants | The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.) | static final int MIN_WIDTH = 4;<br><br>static final int MAX_WIDTH = 999;<br><br>static final int GET_THE_CPU = 1; |

# Executable Vs Non-executable Java classes

- What is the difference?
- Think about it!

# The main method

- The method must be marked as a `public` method.
- The method must be marked as a `static` method.
- The name of the method must be `main`.
- The return type of this method must be `void`.
- The method must accept a method argument of a `String` array or a variable argument (varargs) of type `String`.


The method should not return a value; its return type must be void.
The name of the method must be main.
The method must accept an array or varargs of type String. The name of the method parameter can be any valid identifier name.
public class HelloExam {
public static void main(String args[]) {
System.out.println("Hello exam");
}
}
The access modifier must be public.
The nonaccess modifier must be static.

# Did you know?

```
public static void main(String... args)
```

It's valid to define args as a variable argument.

```
public static void main(String[] arguments)
public static void main(String[] HelloWorld)
```

The names of the method arguments are arguments and HelloWorld, which is acceptable.

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

The square brackets [] can follow either the variable name or its type.

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

The placements of the keywords public and static are interchangeable.

# Τι θα συμβεί;

```
public class HelloExam {
    public static void main(String args) {
        System.out.println("Hello exam 2");
    }
    public static void main(String args[]) {
        System.out.println("Hello exam");
    }
    public static void main(int number) {
        System.out.println("Hello exam 3");
    }
}
```

# Access Modifiers

# Access modifiers

- Οι access modifiers ουσιαστικά, όπως φαίνεται από το όνομά τους, ορίζουν την πρόσβαση στοιχείων της Java
- Οι access modifiers είναι συνολικά 4 στον αριθμό ωστόσο, δεν μπορούν να χρησιμοποιηθούν με τον ίδιο τρόπο από όλα τα στοιχεία του κώδικα Java
- Access modifiers:
  - private
  - default (package)
  - protected
  - public

# Access modifiers explanation

| Modifier | Description |
| --- | --- |
| Private | Declarations are visible within the class only |
| Default | Declarations are visible only within the package (package private) |
| Protected | Declarations are visible within the package or and all sub classes |
| Public | Declarations are visible everywhere |

# Access modifiers Table 1

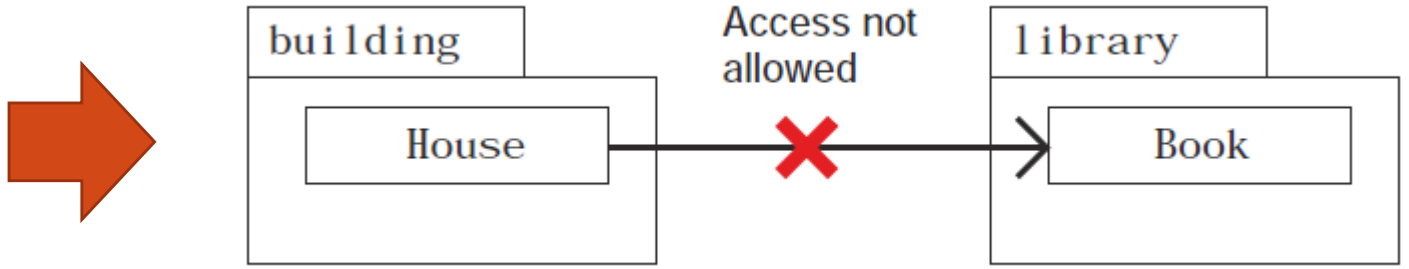
| | private | default | protected | public |
|---|---|---|---|---|
| Class | No | Yes | No | Yes |
| Nested Class | Yes | Yes | Yes | Yes |
| Constructor | Yes | Yes | Yes | Yes |
| Method | Yes | Yes | Yes | Yes |
| Field | Yes | Yes | Yes | Yes |

# Access modifiers Table 2

| Entity name | public | protected | private |
|---|---|---|---|
| Top-level class, interface, enum | ✓ | ✗ | ✗ |
| Class variables and methods | ✓ | ✓ | ✓ |
| Instance variables and methods | ✓ | ✓ | ✓ |
| Method parameter and local variables | ✗ | ✗ | ✗ |

# Παράδειγμα 1

```
package building;
class House {}
package library;
class Book {}
```



# Παράδειγμα 2

```
package library;
public class Book {
    public String isbn;
    public void printBook() {}
}
```

public class Book

public variable isbn

public method printBook

```
package building;
import library.Book;
public class House {
    House() {
        Book book = new Book();
        String value = book.isbn;
        book.printBook();
    }
}
```

Class Book is accessible to class House.

Variable isbn is accessible in House.

Method printBook is accessible in House.

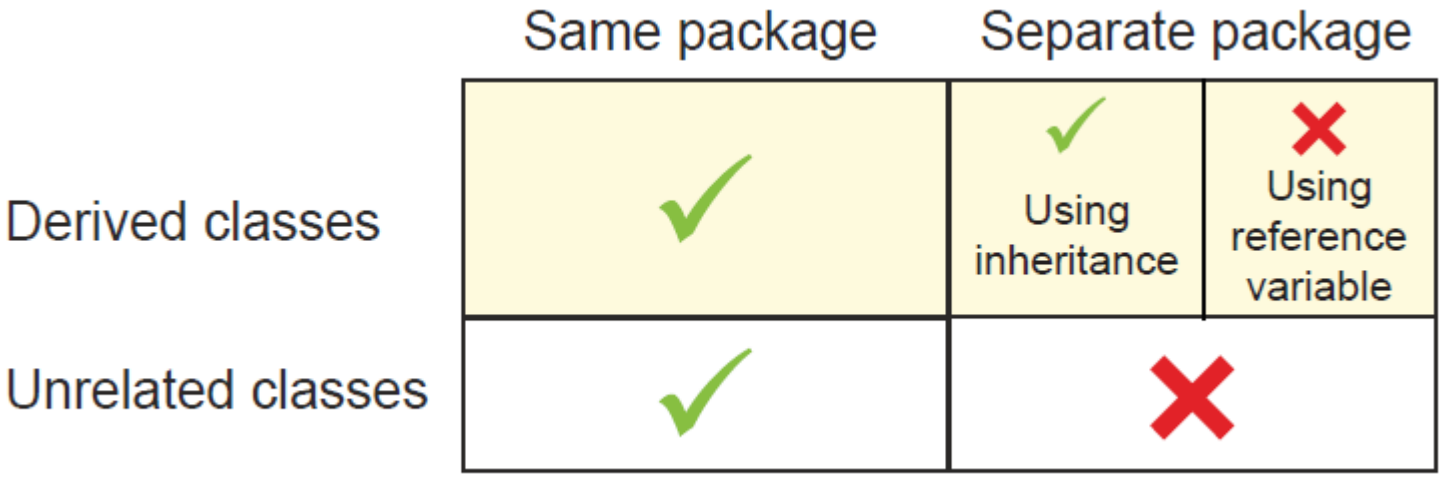
# Private Access Modifier

| | Same package | Separate package |
|---|---|---|
| Derived classes | ✗ | ✗ |
| Unrelated classes | ✗ | ✗ |

# Default Access Modifier

| | Same package | Separate package |
|---|---|---|
| Derived classes | ✓ | ✗ |
| Unrelated classes | ✓ | ✗ |

# Protected Access Modifier



# Public Access Modifier

| | Same package | Separate package |
|---|---|---|
| Derived classes | ✓ | ✓ |
| Unrelated classes | ✓ | ✓ |

# Non-access Modifiers

# Non-access modifiers

- Δεσμευμένες λέξεις-κλειδιά οι οποίες δεν σχετίζονται με την προσβασιμότητα
- Είναι οι εξής:
  - abstract
  - static
  - final
  - synchronized
  - volatile
  - strictfp
  - transient
  - native
- Προς το παρόν θα αναλύσουμε μερικές σε βάθος και άλλες απλώς αναφορικά

# Synchronized

- Αφορά μόνο μεθόδους
- Υποδηλώνει ότι μια μέθοδος δεν μπορεί να προσπελαστεί από διαφορετικά threads ταυτόχρονα

# Volatile

- Αφορά μεταβλητές
- Υποδηλώνει ότι μια μεταβλητή μπορεί να προσπελαστεί με «ασφάλεια» από διαφορετικά threads
- Η μεταβλητή βρίσκεται στη μνήμη μόνο

# Strictfp

- Αφορά τάξεις, interfaces και μεθόδους (όχι μεταβλητές)
- Υποδηλώνει ότι οι υπολογισμοί που θα γίνουν και αφορούν αριθμούς κινητής υποδιαστολής, θα είναι ίδιοι σε όλες τις πλατφόρμες

# Transient

- Αφορά μεταβλητές
- Υποδηλώνει ότι μια μεταβλητή δε θα γίνει serialized όταν το αντικείμενό της υποστεί serialization

# Native

- Αφορά μόνο μεθόδους
- Υποδηλώνει ότι μια μέθοδος μπορεί να χρησιμοποιήσει βιβλιοθήκες και μεθόδους σε άλλες γλώσσες προγραμματισμού, όπως C και C++

# Abstract

- Εφαρμόζεται σε τάξεις και μεθόδους
- Τα interfaces είναι abstract εξ ορισμού (οπότε δεν το γράφουμε, το προσθέτει ο compiler, ωστόσο δεν είναι λάθος και να το γράψουμε)
- Τάξεις:
  - Δε μπορούμε να δημιουργήσουμε instance (αντικείμενο) αυτής
  - Χρησιμοποιείται κυρίως για να δηλώσουμε τάξεις που θέλουμε να επεκτείνουμε (extend) μέσω κληρονομικότητας
- Μέθοδοι:
  - Είναι οι μέθοδοι που έχουν υπογραφή αλλά όχι σώμα
  - Το σώμα των μεθόδων το παρέχουν συνήθως οι υπο-τάξεις
  - Μια μέθοδος που έχει σώμα αλλά κενό (…) δεν είναι abstract method
- Μια τάξη μπορεί να δηλωθεί abstract και να μην έχει abstract methods
- Αν μια μέθοδος δηλωθεί abstract τότε πρέπει να δηλωθεί abstract και η τάξη που την περιέχει

# Σημειώσεις για abstract

- A class cannot be declared both abstract and final
- If a class contains one or more abstract methods then the class should be declared abstract, or else a compile error will be thrown
- An abstract class may contain both abstract methods as well normal methods
- An abstract class does not need to contain abstract methods
- Abstract methods can never be final
- Any class that extends an abstract class must implement all the abstract methods of the super class unless the subclass is also an abstract class
- Abstract methods end with a semicolon

# Final

- Εφαρμόζεται σε τάξεις, μεθόδους και μεταβλητές
- Τα interfaces είναι abstract εξ ορισμού, οπότε δεν μπορούν να δηλωθούν final
- Τάξεις:
  - Μια final class δεν μπορεί να την επεκτείνει (μέσω κληρονομικότητας) καμία τάξη
- Μέθοδοι:
  - Μια final method δεν μπορεί να υποσκελιστεί (override) από μια τάξη κληρονόμο (sub class) (υπό τάξη)
- Μεταβλητές:
  - Σε μια final μεταβλητή δεν μπορούμε να ξάνα-δώσουμε τιμή. Δηλαδή θα πάρει τιμή μόνο μια φορά (μόνο μια φορά αρχικοποίηση) (Προσοχή: Τι συμβαίνει με τις reference variables?)

# Static 1/2

- Εφαρμόζεται σε nested τάξεις, nested interfaces, μεθόδους και μεταβλητές
- Εμφωλευμένες τάξεις:
  - Δηλώνοντες ως στατικές εντός μιας top level class
  - Δεν μπορούν να προσπελάσουν μη στατικά μέλη (non static members)
  - Αναφερόμαστε σε αυτές χρησιμοποιώντας το όνομα της outer class
- Εμφωλευμένα interfaces:
  - Είναι by default δηλωμένα ως static, οπότε δεν έχει σημασία αν θα τα δηλώσουμε static ή όχι

# Static 1/2

- Μέθοδοι:
  - Δηλώνονται ως στατικές εντός μιας class και ανήκουν σε αυτή (όχι στα στιγμιότυπά της)
  - Δεν μπορούν να προσπελάσουν μη στατικά μέλη (non static members) (προσοχή)
  - Αναφερόμαστε σε αυτές χρησιμοποιώντας το όνομα της outer class
  - Static methods μπορούν να υπάρχουν τόσο σε τάξεις, όσο και σε interfaces (Java 8+)
- Μεταβλητές:
  - Δηλώνονται ως στατικές εντός μιας class και ανήκουν σε αυτή (όχι στα στιγμιότυπά της)
  - Είναι κοινές για όλα τα instances της τάξης. Δεν χρειάζεται object instantiation για την ύπαρξή τους
  - Όλες οι μεταβλητές ενός interface είναι εξ ορισμού static
  - Στη Java για να δηλώσουμε μια σταθερά (δεν υπάρχει const) συνήθως χρησιμοποιούμε final μεταβλητές και πολλές φορές και static
  - Η προσπέλαση γίνεται τόσο από την τάξη, αλλά και από τα αντικείμενα (προσοχή στον 2ο τρόπο, δεν ενδείκνυται)

# Προσπέλαση μεταξύ static και non-static

- Μια στατική μέθοδος και μια στατική μεταβλητή δεν μπορούν να προσπελάσουν μη στατικές μεθόδους και μεταβλητές (Προσοχή: Τα στατικά μέλη προϋπάρχουν με την τάξη. Τα μη στατικά όμως;)
- Το αντίθετο ισχύει: Μη στατικές μέθοδοι και μεταβλητές μπορούν να προσπελάσουν στατικές μεθόδους και μεταβλητές
- Πολύ μεγάλη Προσοχή!:
  - Ακόμα και αν ένα αντικείμενο είναι null, μπορεί να προσπελάσει μια στατική μεταβλητή ή/και μέθοδο. Αυτό συμβαίνει διότι τα εν λόγω μέλη δεν ανήκουν σε αυτό, αλλά στην τάξη. Για το λόγο αυτό, δεν θα έχουμε null pointer exception. Ωστόσο, τέτοιος κώδικας πρέπει να αποφεύγεται!..

# Static Vs Non-static

| Member type | Can access `static` attribute or method? | Can access non-`static` attribute or method? |
|---|---|---|
| `static` | Yes | No |
| Non-`static` | Yes | Yes |

# Variables in Java

# Variable scope

- Οι μεταβλητές στη Java χαρακτηρίζονται και από την «εμβέλειά» τους (scope)
- Βάσει του scope έχουμε τις εξής 4 κατηγορίες μεταβλητών:
  - Local variables (τοπικές μεταβλητές εντός μεθόδων)
  - Method parameters (οι μεταβλητές που χρησιμοποιούνται ως παράμετροι στις μεθόδους)
  - Instance Variables (Global μεταβλητές που ανήκουν σε κάθε αντικείμενο ξεχωριστά. Οι συγκεκριμένες αναφέρονται και ως τα «πεδία» ή τα «χαρακτηριστικά» του αντικειμένου)
  - Class Variables (Global μεταβλητές που ανήκουν στην τάξη. Συγκεκριμένα οι static variables)

# Local variables – Method parameters

- Έχουν το μικρότερο «χρόνο ζωής» αφού ορίζονται εντός μεθόδων και πολλές φορές σε τμήματα μεθόδων (π.χ μέσα σε ένα loop)
- Πάντα κοιτάζουμε το block ( { } ) μέσα στο οποίο ορίζονται
- Φυσικά δεν υπάρχει πρόσβαση σε αυτές έξω από τη μέθοδο, ούτε και έξω από το block

```
class Student {
    private double marks1, marks2, marks3;
    private double maxMarks = 100;
    public double getAverage() {
        double avg = 0;
        avg = ((marks1 + marks2 + marks3) / (maxMarks*3)) * 100;
        return avg;
    }
    public void setAverage(double val) {
        avg = val;
    }
}
```

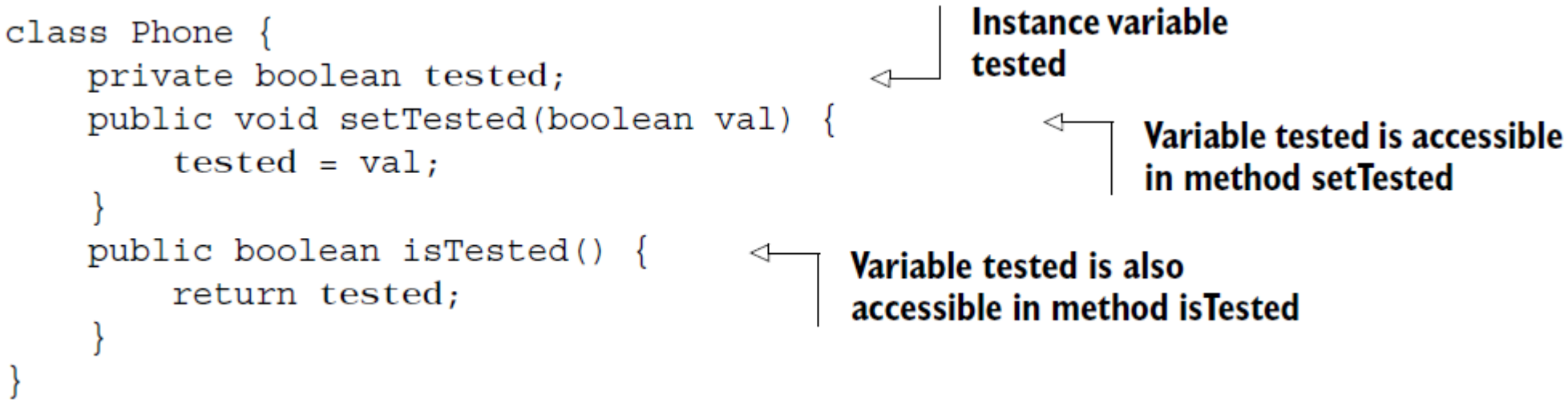
Instance variables

Local variable avg

This code won't compile because avg is inaccessible outside the method getAverage.

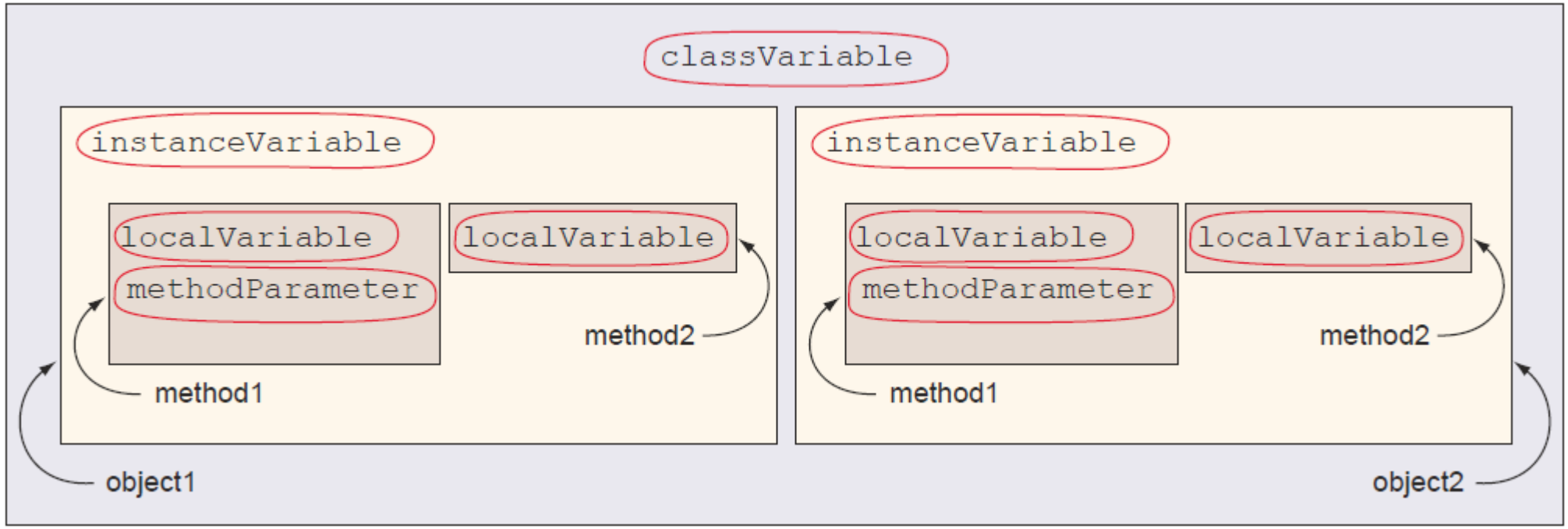
# Instance variables

- Εντός της τάξης, έξω από μεθόδους
- Χωρίς τη χρήση του keyword: static
- Σε αυτές έχουν πρόσβαση όλες οι μη στατικές μέθοδοι
- Κάθε αντικείμενο έχει τις δικές του



# Class variables

- Εντός της τάξης, έξω από μεθόδους
- Με τη χρήση του keyword: static
- Σε αυτές έχουν πρόσβαση όλες οι μη μέθοδοι, στατικές και μη (σκεφτείτε γιατί!)
- Ανήκουν στην τάξη, οπότε όλα τα αντικείμενα έχουν πρόσβαση στις ίδιες static variables. Δεν μπορεί ένα αντικείμενο να έχει τις δικές του


classVariable
instanceVariable
localVariable
methodParameter
localVariable
method2
method1
object1
instanceVariable
localVariable
methodParameter
localVariable
method2
method1
object2

# Ίδια ονόματα μεταβλητών

- Δε μπορούμε να έχουμε instance και static variable με το ίδιο όνομα
- Δε μπορούμε να έχουμε παράμετρο μεθόδου και local variable με το ίδιο όνομα
- Μπορούμε να έχουμε είτε instance και local variable με το ίδιο όνομα, είτε static και local variable με το ίδιο όνομα. Όμως θέλει πολύ μεγάλη προσοχή!

# Συμπληρωματικές γνώσεις 1

# Method return statement

- Είναι υποχρεωτικό σε μεθόδους που «επιστρέφουν» τιμές
- Δεν είναι υποχρεωτικό σε μεθόδους που επιστρέφουν void
- Πολλές φορές χρησιμοποιείται μέσα σε συνθήκη ελέγχου για τον τερματισμό της μεθόδου (δείτε το οπωσδήποτε με παράδειγμα, καθώς και μέσα σε loop)

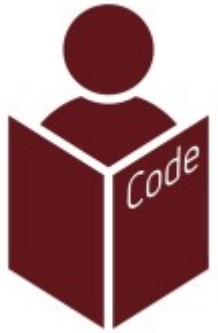
- For a method that returns a value, the `return` statement must be followed immediately by a value.
- For a method that doesn't return a value (return type is `void`), the `return` statement must *not* be followed by a return value.
- If the compiler determines that a `return` statement isn't the last statement to *execute* in a method, the method will fail to compile.

# Variable Arguments (varargs)

- Java 5+
- Είναι ουσιαστικά πίνακες
- Μας επιτρέπουν να «στείλουμε» σε μια μέθοδο ένα οποιονδήποτε αριθμό από ορίσματα, του ιδίου τύπου
- Μια μέθοδος μπορεί να έχει έως μια παράμετρο varargs
- Αν υπάρχει παράμετρος varargs τότε πρέπει να είναι η τελευταία παράμετρος

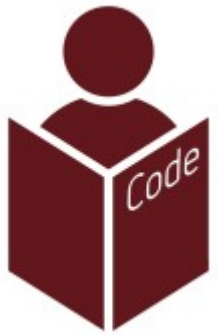
```
public static void main(String[] args) {
    varargTest( ...args: "one","two","three");
}
public static void varargTest(String... args){
    for (int i=0;i<args.length;i++)
        System.out.println(args[i]);
}
```

# Java instanceof Operator



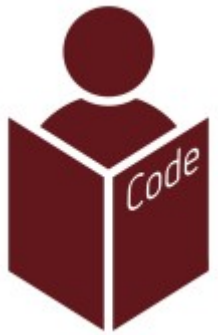

- Δυαδικός τελεστής
- true – false
- Εφαρμόζεται σε αντικείμενα
- Εξετάζει αν ένα αντικείμενο «είναι» κάποιου τύπου, συμπεριλαμβάνοντας της «υλοποίησης» των interfaces
- Χρησιμοποιείται ιδιαίτερα αν δεν είναι γνωστός ο «τύπος» ενός αντικειμένου για την αποφυγή ClassCastException κατά τη διάρκεια του Casting
- Γενικός τύπος:
  - (object) instanceof (type)

# Παράδειγμα 1/2




```
interface IStudy{
    void study(String message);
}
public class Human{
    String name;
}
class Student extends Human implements IStudy{
    int AM;
    @Override
    public void study(String message) {
        System.out.println("I am reading "+message);
    }
}
class Professor extends Human{
    int officeNumber;
}
```

# Παράδειγμα 2/2

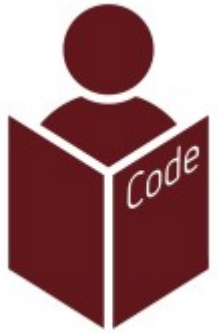



```
public class Main {
    public static void main(String[] args) {
        Human human = new Human();
        Student student = new Student();
        Professor professor = new Professor();
        System.out.println(human instanceof Human);
        System.out.println(student instanceof Human);
        System.out.println(student instanceof IStudy);
        System.out.println(professor instanceof IStudy);
        System.out.println(human instanceof Professor);
    }
}
```

# Αποτέλεσμα

```
true
true
true
false
false

Process finished with exit code 0
```

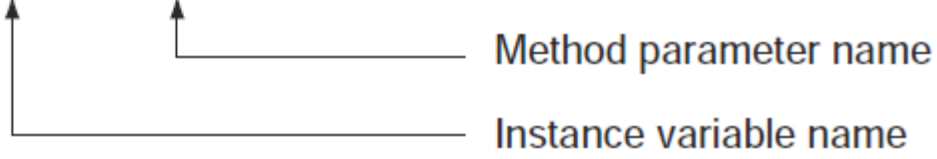
# this and super



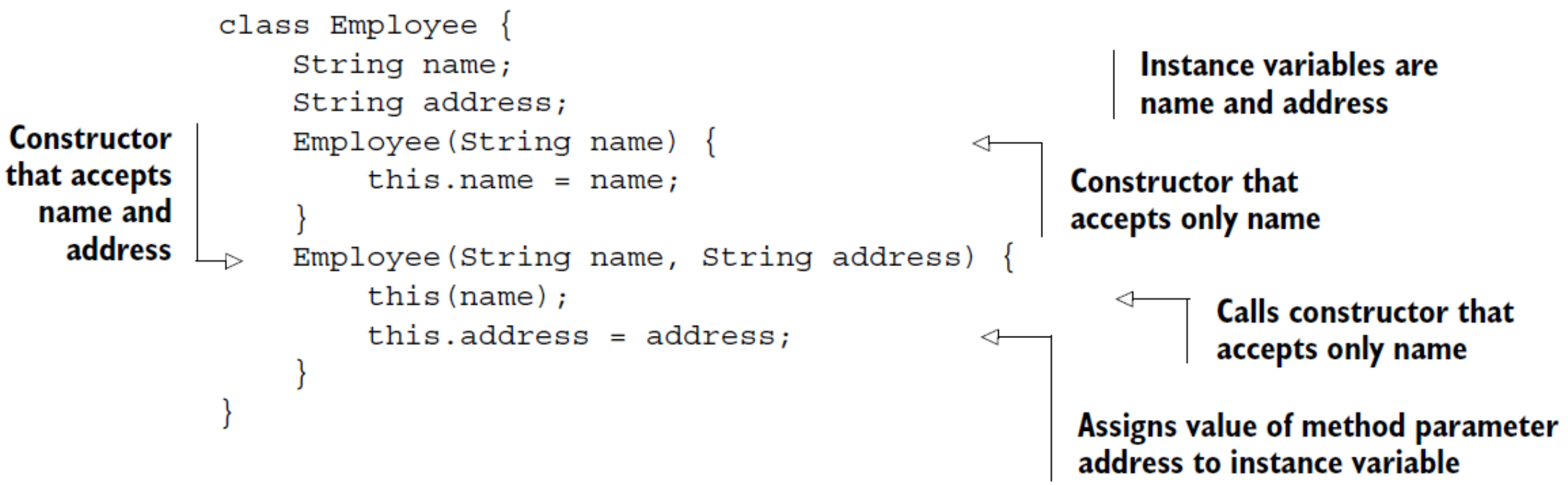

- Χρησιμοποιούμε το this για να αποκτήσουμε πρόσβαση στα μέλη (members) της «τρέχουσας» τάξης (μέσα στην οποία βρισκόμαστε τη στιγμή που γράφουμε τη λέξη this), ή και για να χρησιμοποιήσουμε το τρέχον instance (π.χ. για να το στείλουμε ως παράμετρο μεθόδου)
- Χρησιμοποιούμε το super για να καλέσουμε μέλη (members) της υπερτάξης (της τάξης από την οποία έχουμε κληρονομήσει)
- Και οι 2 λέξεις μπορούν να χρησιμοποιηθούν και για static και για instance members
- Επιπλέον μπορούν να χρησιμοποιηθούν και στους constructors (Advanced Java)

# Παράδειγμα

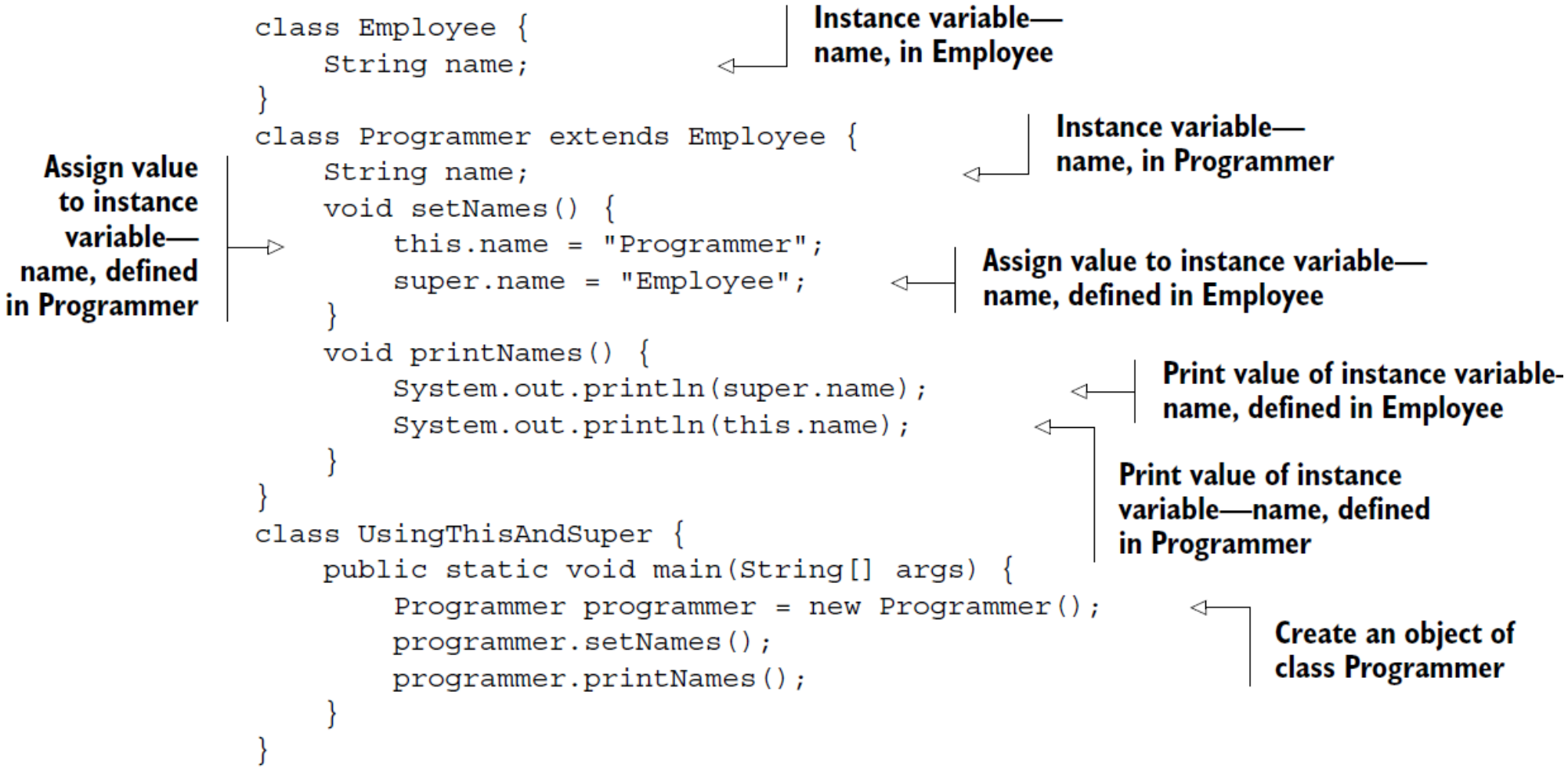
```
class Employee {
     String name;
     Employee(String name) {
         this.name = name;
     }
}
```



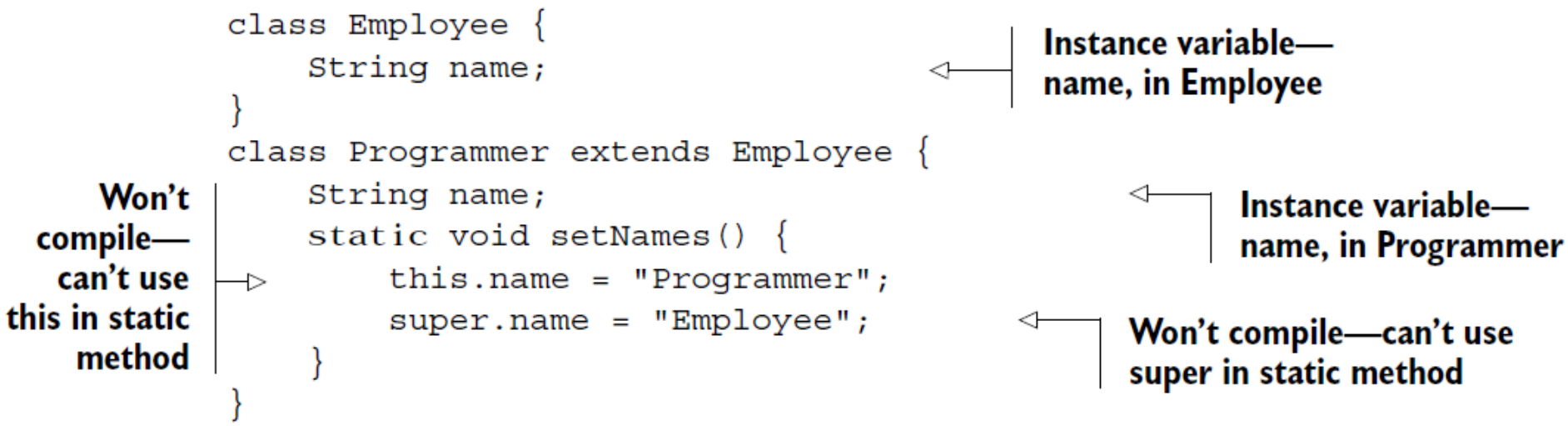
# Παράδειγμα


class Employee {
String name;
String address;
Employee(String name) {
this.name = name;
}
Employee(String name, String address) {
this(name);
this.address = address;
}
}
Instance variables are name and address
Constructor that accepts only name
Constructor that accepts name and address
Calls constructor that accepts only name
Assigns value of method parameter address to instance variable

# Παράδειγμα

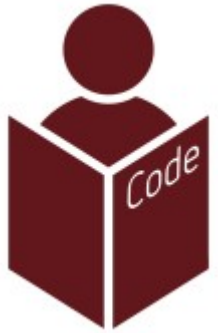

class Employee {
String name;
}
class Programmer extends Employee {
String name;
void setNames() {
this.name = "Programmer";
super.name = "Employee";
}
void printNames() {
System.out.println(super.name);
System.out.println(this.name);
}
}
class UsingThisAndSuper {
public static void main(String[] args) {
Programmer programmer = new Programmer();
programmer.setNames();
programmer.printNames();
}
}
Instance variable—name, in Employee
Instance variable—name, in Programmer
Assign value to instance variable—name, defined in Programmer
Assign value to instance variable—name, defined in Employee
Print value of instance variable-name, defined in Employee
Print value of instance variable—name, defined in Programmer
Create an object of class Programmer

# Παράδειγμα – Προσοχή!



| 'this' keyword | 'super' keyword |
| --- | --- |
| It represents the current instance of a class. | It represents the current instance of a parent class. |
| It is used to call default constructor of the same class. | It is used to call default constructor of the parent class. |
| It is used to access methods of the current class. | It is used to access methods of the base class. |
| It is used for pointing the current class instance. | It is used for pointing the super class instance. |

# Method overloading

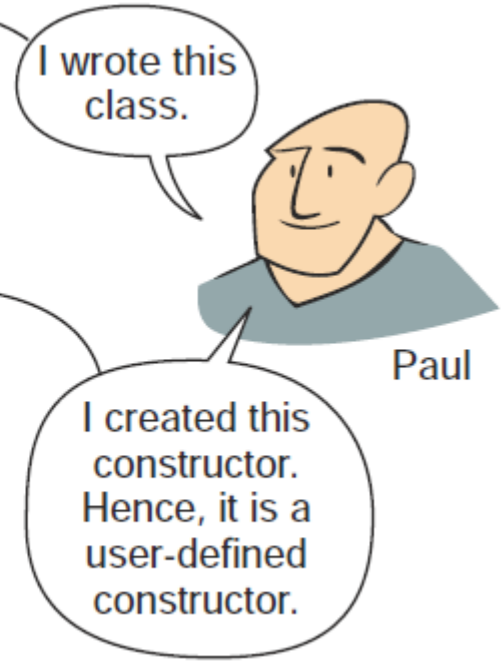



- Overloaded methods must have method parameters different from one another.
- Overloaded methods may or may not define a different return type.
- Overloaded methods may or may not define different access levels.
- Overloaded methods can't be defined by only changing their return type or access modifiers or both.

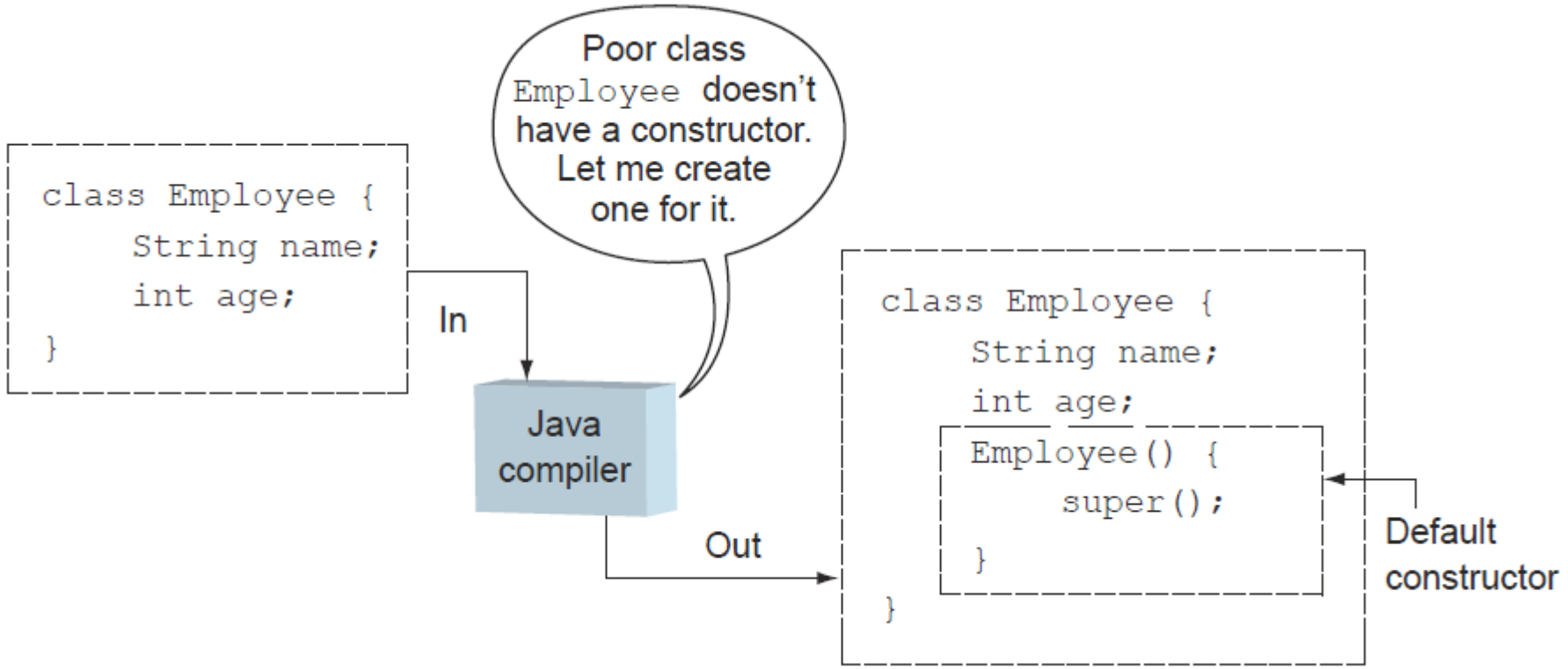
# Constructors

- Χρησιμοποιούνται για την αρχικοποίηση των αντικειμένων
- Έχουν το ίδιο όνομα με την τάξη, όμως το σημαντικό χαρακτηριστικό τους είναι ότι ΔΕΝ επιστρέφουν τίποτα (ούτε void)
- Δέχονται overloading
- Υποστηρίζουν και τα 4 επίπεδα των access modifiers

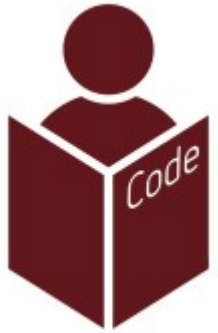
```
class Employee {
    Employee () {
        System.out.println("Constructor");
    }
}
```



# Default constructors



# Invoking constructor from another constructor…




- Με τη χρήση του keyword **this**

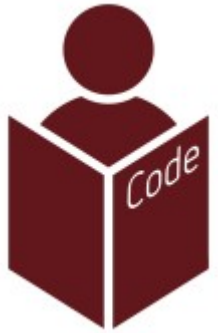
```
class Employee {
    String name;
    int age;
    Employee() {
        this(null, 0);
    }
    Employee(String newName, int newAge) {
        name = newName;
        age = newAge;
    }
}
```

1 **No-argument constructor**

2 **Invokes constructor that accepts two method arguments**

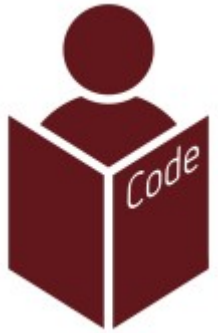
3 **Constructor that accepts two method arguments**

# Constructors Σημαντικά




- Overloaded constructors must be defined using different argument lists.
- Overloaded constructors can't be defined by just a change in the access levels.
- Overloaded constructors may be defined using different access levels.
- A constructor can call another overloaded constructor by using the keyword `this`.
- A constructor can't invoke a constructor by using its class's name.
- If present, the call to another constructor must be the first statement in a constructor.
- You can't call multiple constructors from a constructor.
- A constructor can't be invoked from a method (except by instantiating a class using the `new` keyword).

# Initializer Block (Advanced)




- Το initializer block είναι ένα τμήμα κώδικα που εκτελείται πάντα όταν δημιουργείται ένα νέο αντικείμενο
- Διαφέρει από τους constructors διότι αυτοί μπορεί να είναι πολλοί (constructor overloading) και δεν είναι σίγουρο ποιος θα εκτελεστεί. Ενώ το initializer block θα εκτελεστεί σίγουρα
- Εκτελείται χρονικά πριν τον constructor

```
public class Main {
    // Initializer block starts..
    {
        // This code is executed before every constructor.
        System.out.println("Common part of constructors invoked !!");
    }
    // Initializer block ends
```

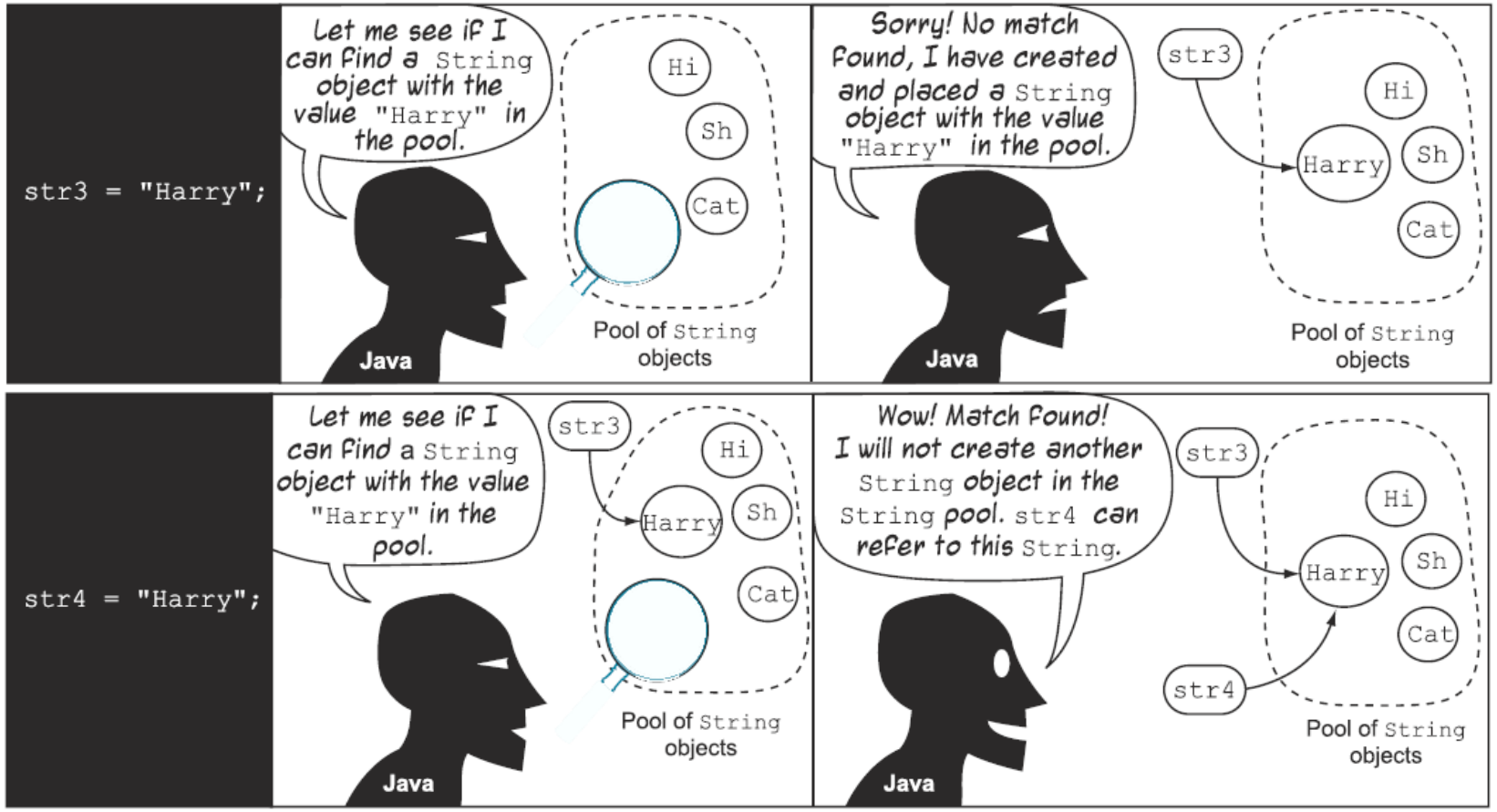
# Java API

# Java Strings

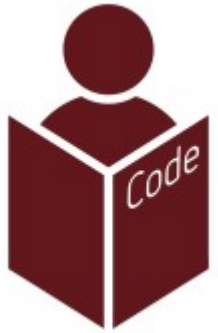
- Αλληλουχία χαρακτήρων
- Δήλωση μέσα σε " "
- Κάθε String είναι ένα αντικείμενο
- Εκτός από τον συνηθισμένο τρόπο δήλωσης ενός αλφαριθμητικού (String s = "Hello";) , υπάρχουν πάρα πολλοί ακόμα
- Ο σωστός χειρισμός των Strings μπορεί να βοηθήσει:
  - Σε αποφυγή λαθών (π.χ. σύγκρισης)
  - Στην βελτιστοποίηση του κώδικα!
  - Σε περιβάλλοντα multithreading

# String pool

- Για λόγους βελτιστοποίησης η Java δημιουργεί μια «δεξαμενή» από Strings και επανα-χρησιμοποιεί όσα βρίσκονται εκεί μέσα



# Διαφορετικοί τρόποι δημιουργίας String




```
String girl = new String("Shreya");
char[] name = new char[]{'P','a','u','l'};
String boy = new String(name);
```

String constructor that accepts a String

String constructor that accepts a char array

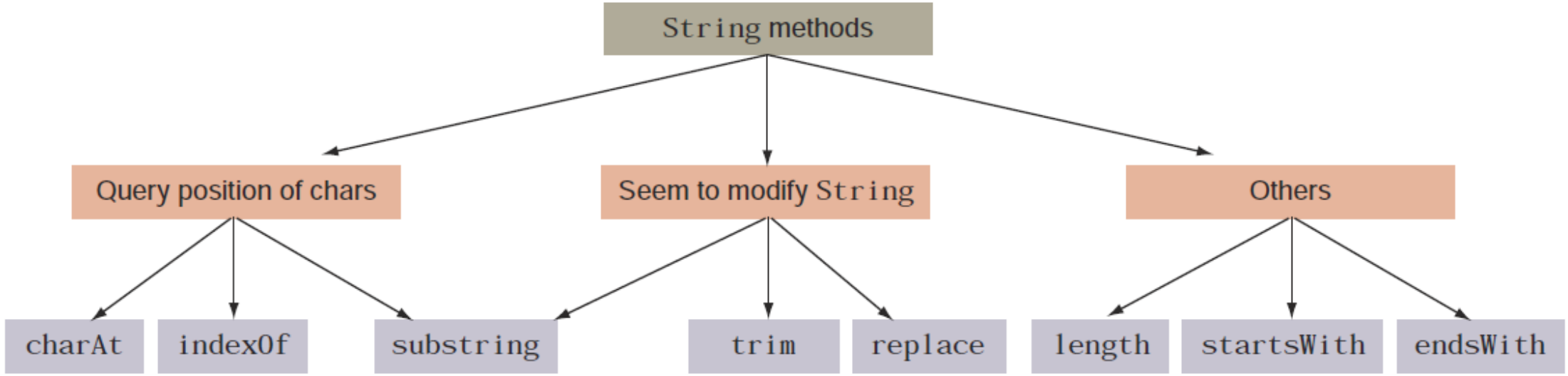
```
StringBuilder sd1 = new StringBuilder("String Builder");
String str5 = new String(sd1);
StringBuffer sb2 = new StringBuffer("String Buffer");
String str6 = new String(sb2);
```

String constructor that accepts object of StringBuilder

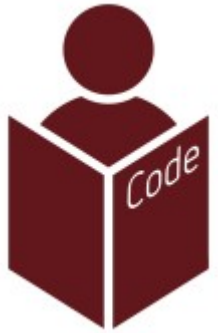
String constructor that accepts object of StringBuffer

- Ανεξαρτήτως της ύπαρξης του String Pool, εάν δημιουργήσετε String με το keyword new, θα δημιουργηθεί νέο String

# String handling methods

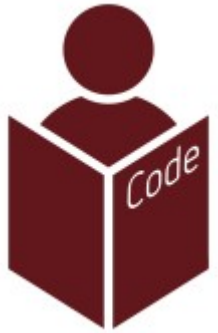


# Σημαντικά!



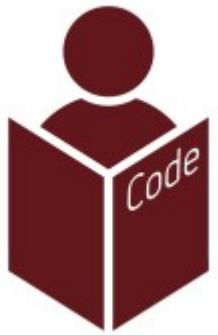

- String is Immutable
- Οι μέθοδοι της τάξης String δεν μεταβάλουν το περιεχόμενο του String

# Σύγκριση Αλφαριθμητικών




- Χρειάζεται πολύ μεγάλη προσοχή. Γίνονται πολλά λάθη και από φοιτητές και από προγραμματιστές
- Ο τελεστής == χρησιμοποιείται για τη σύγκριση των διευθύνσεων σε reference variables (όπως είναι αναμενόμενο)
- Αρκετές φορές ο τελεστής == μπορεί να μας δώσει ισότητα αλφαριθμητικών, αλλά αυτό συμβαίνει λόγω της χρήσης του String Pool. Πρέπει να αποφεύγεται!
- Για τη σύγκριση της πραγματικής «τιμής» του αλφαριθμητικού ενδείκνυται η χρήση της μεθόδου equals()

# Προσοχή!




```
String var3 = "code";
String var4 = "code";
System.out.println(var3.equals(var4));
System.out.println(var3 == var4);

String var1 = new String("Java");
String var2 = new String("Java");
System.out.println(var1.equals(var2));
System.out.println(var1 == var2);
```

Prints true

Prints true

Prints true

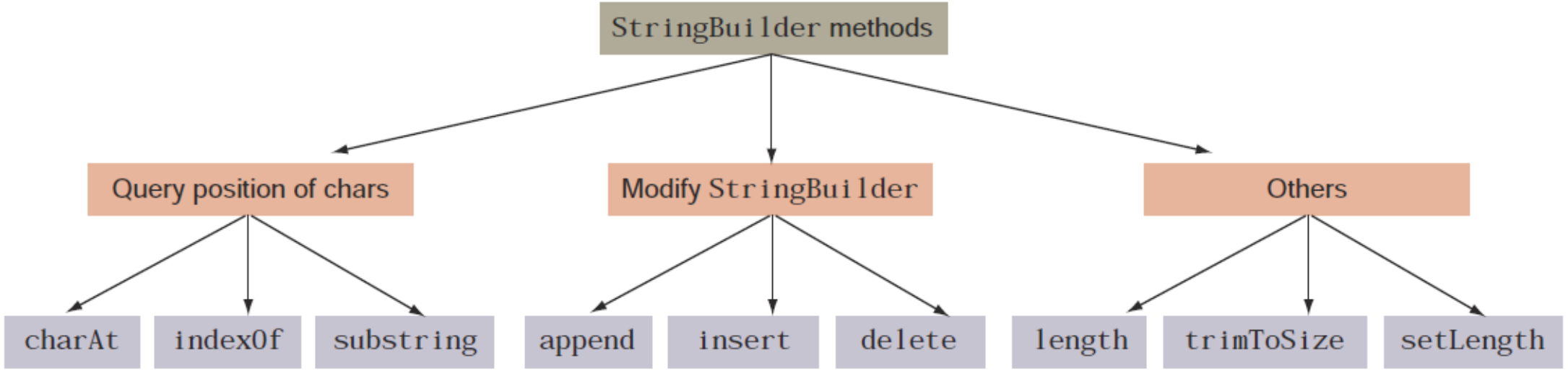
Prints false

- (Άσκηση) Ελέγξτε αν τα αλφαριθμητικά που επιστρέφουν οι μέθοδοι της τάξης String αποθηκεύονται στο String Pool

# Mutable Strings

- Μια από τις πιο βασικές τάξεις: StringBuilder
- Ανήκει στο package: java.lang
- Χρησιμοποιείται συχνά όταν έχουμε μεγάλα Strings ή συχνές αλλαγές πάνω σε ένα String
- Στην προηγούμενη περίπτωση μπορούμε να έχουμε σημαντικές βελτιώσεις στην «επίδοση» του κώδικά μας
- Ο constructor της τάξης δέχεται ένα μεγάλο αριθμό από overloads, οπότε μπορεί να χρησιμοποιηθεί με πολλούς και διαφορετικούς τρόπους!

# StringBuilder Methods



# StringBuilder Vs StringBuffer

- Η τάξη StringBuffer προσφέρει περίπου την ίδια λειτουργικότητα με την τάξη StringBuilder
- Η βασική διαφορά έγκειται στο γεγονός ότι οι μέθοδοι της τάξης StringBuffer είναι synchronized και ως αποτέλεσμα «thread safe»
- Ωστόσο, η χρήση synchronized μεθόδων συνοδεύεται από ένα αναμενόμενο πρόσθετο προγραμματιστικό κόστος

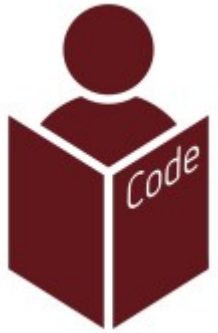
# Java ArrayList

- Η τάξη ArrayList είναι μια από τις πιο γνωστές και ευρέως χρησιμοποιούμενες τάξεις της Java (και άλλως γλωσσών βέβαια)
- Προσπαθεί να συνδυάσει τα καλύτερα χαρακτηριστικά από τον κόσμο των πινάκων (Arrays) και των λιστών (Lists)
- Έχει μεταβλητό μέγεθος
- Γνωστές/Βασικές λειτουργίες:
  - Προσθήκη στοιχείων στη λίστα
  - Διαγραφή στοιχείων από τη λίστα
  - Αλλαγή στοιχείων της λίστας
  - Προσπέλαση όλων των στοιχείων της λίστας

# ArrayList Properties

- It implements the interface `List`.
- It allows `null` values to be added to it.
- It implements all list operations (add, modify, and delete values).
- It allows duplicate values to be added to it.
- It maintains its insertion order.
- You can use either `Iterator` or `ListIterator` to iterate over the items of an `ArrayList`.
- It supports generics, making it type safe. (You have to declare the type of the elements that should be added to an `ArrayList` with its declaration.)

# ArrayList Creation

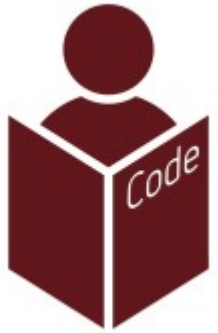



```
import java.util.ArrayList;
public class CreateArrayList {
    public static void main(String args[]) {
        ArrayList<String> myArrList = new ArrayList<String>();
    }
}
```

1 Import java.util.ArrayList

2 Declare an ArrayList object

# ArrayList – Add Items

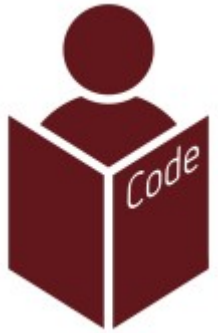



```
import java.util.ArrayList;
public class AddToArrayList {
    public static void main(String args[]) {
        ArrayList<String> list = new ArrayList<>();
         list.add("one");
         list.add("two");
         list.add("four");
         list.add(2, "three");
    }
}
```

1 Add element at the end

2 Add element at specified position

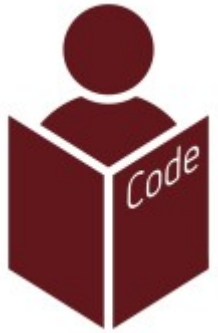
# ArrayList – Item Iteration




```
import java.util.ArrayList;
public class AccessArrayList {
    public static void main(String args[]) {
        ArrayList<String> myArrList = new ArrayList<>();
        myArrList.add("One");
        myArrList.add("Two");
        myArrList.add("Four");
        myArrList.add(2, "Three");
        for (String element : myArrList) {
            System.out.println(element);
        }
    }
}
```

❶ **Code to access ArrayList elements**

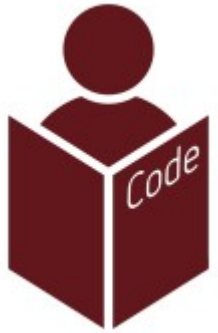
# ArrayList – Item Iteration v2 using ListIterator




```
import java.util.ArrayList;
import java.util.ListIterator;
public class AccessArrayListUsingListIterator {
    public static void main(String args[]) {
        ArrayList<String> myArrList = new ArrayList<String>();
        myArrList.add("One");
        myArrList.add("Two");
        myArrList.add("Four");
        myArrList.add(2, "Three");
        ListIterator<String> iterator = myArrList.listIterator();
        while (iterator.hasNext()) {
            System.out.println(iterator.next());
        }
    }
}
```

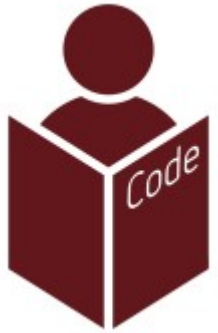
1. Get the iterator
2. Use hasNext() to check whether more elements exist
3. Call next() to get the next item from iterator

# Άσκηση




- Με τους παραπάνω 2 τρόπους και ίσως δοκιμάζοντας και κάποιον ακόμα (π.χ. με απλό loop?) δείτε αν μπορείτε να τροποποιήσετε (αλλαγή – προσθήκη – διαγραφή) τα στοιχεία που βρίσκονται εντός ενός ArrayList κατά τη διάρκεια των επαναλήψεων!
- Καταγράψτε τι παρατηρείτε!

# ArrayList - addAll() method



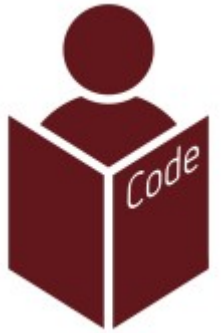

```
ArrayList<String> myArrList = new ArrayList<String>();
myArrList.add("One");
myArrList.add("Two");
ArrayList<String> yourArrList = new ArrayList<String>();
yourArrList.add("Three");
yourArrList.add("Four");
myArrList.addAll(1, yourArrList);
for (String val : myArrList)
    System.out.println(val);
```

**Add elements of yourArrList to myArrList**

# ArrayList – Access elements

- `get(int index)`—This method returns the element at the specified position in this list.
- `size()`—This method returns the number of elements in this list.
- `contains(Object o)`—This method returns `true` if this list contains the specified element.
- `indexOf(Object o)`—This method returns the index of the first occurrence of the specified element in this list, or `-1` if this list doesn't contain the element.
- `lastIndexOf(Object o)`—This method returns the index of the last occurrence of the specified element in this list, or `-1` if this list doesn't contain the element.

# Παραδείγματα




```
ArrayList<StringBuilder> myArrList =
                                new ArrayList<StringBuilder>();
StringBuilder sb1 = new StringBuilder("Jan");
StringBuilder sb2 = new StringBuilder("Feb");
myArrList.add(sb1);
myArrList.add(sb2);
myArrList.add(sb2);
System.out.println(myArrList.contains(new StringBuilder("Jan")));
System.out.println(myArrList.contains(sb1));
System.out.println(myArrList.indexOf(new StringBuilder("Feb")));
System.out.println(myArrList.indexOf(sb2));
System.out.println(myArrList.lastIndexOf(
                                        new StringBuilder("Feb")));
System.out.println(myArrList.lastIndexOf(sb2));
}
```

Adds sb1 to the ArrayList

Adds sb2 to the ArrayList

Adds sb2 to the ArrayList again

Prints false

Prints true

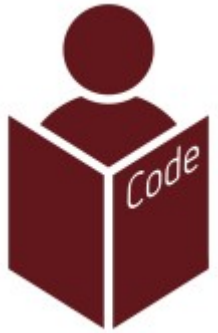
Prints –1

Prints 1

Prints –1

Prints 2

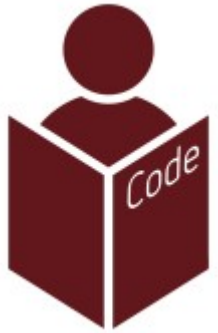
# Άσκηση




- Ελέγξτε τη χρήση των 2 παρακάτω μεθόδων της τάξης ArrayList:
  - toArray()
  - clone()

# Java Extras

# Ternary Operator ?:




| Ternary construct | if-else construct |
| --- | --- |
| `int bill = 2000;`<br>`int discount = (bill > 2000)? 15 : 10;` | `int bill = 2000;`<br>`int discount`<br>`if (bill > 2000)`<br>`  discount = 15;`<br>`else`<br>`  discount = 10;` |

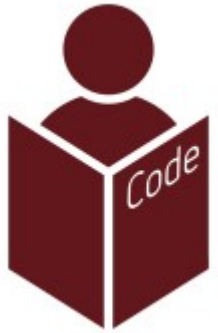
# ?:

- In computer programming, ?: is a ternary operator that is part of the syntax for a basic conditional expression
- It is commonly referred to as the conditional operator, inline if (iif), or ternary if
- Basic syntax:

  condition ? value_if_true : value_if_false

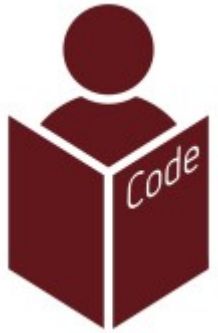
- The condition is evaluated true or false as a Boolean expression
- The entire expression returns value_if_true if condition is true, but value_if_false otherwise

# examples

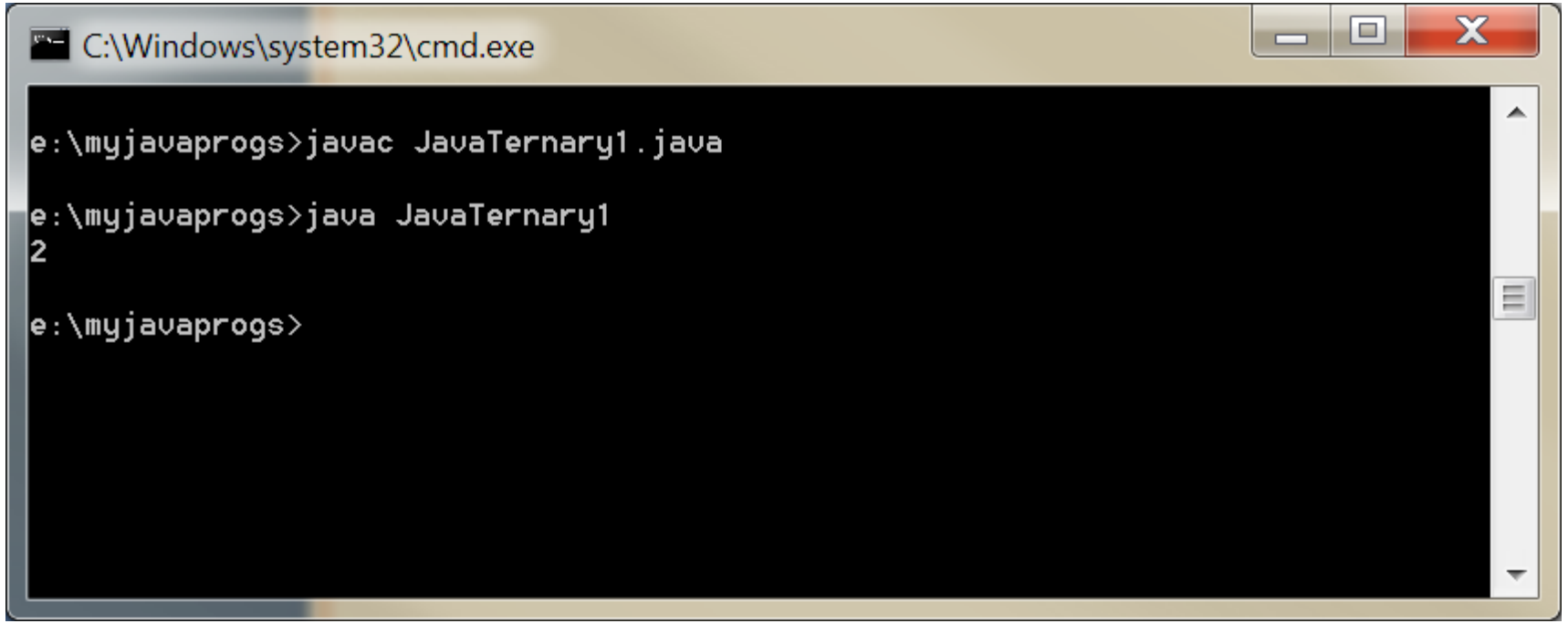



- result = testCondition ? value1 : value2

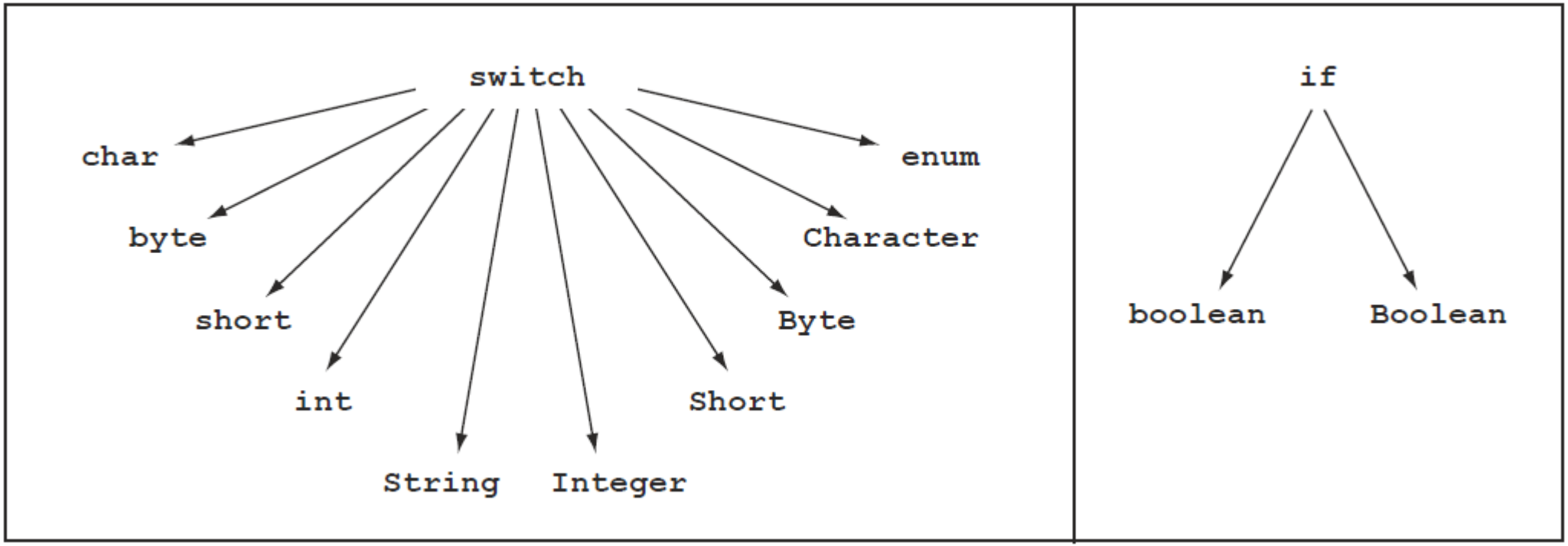
- int minVal = (a < b) ? a : b;
- String str1 = false ? “Yes, its ok" : “I am sorry";



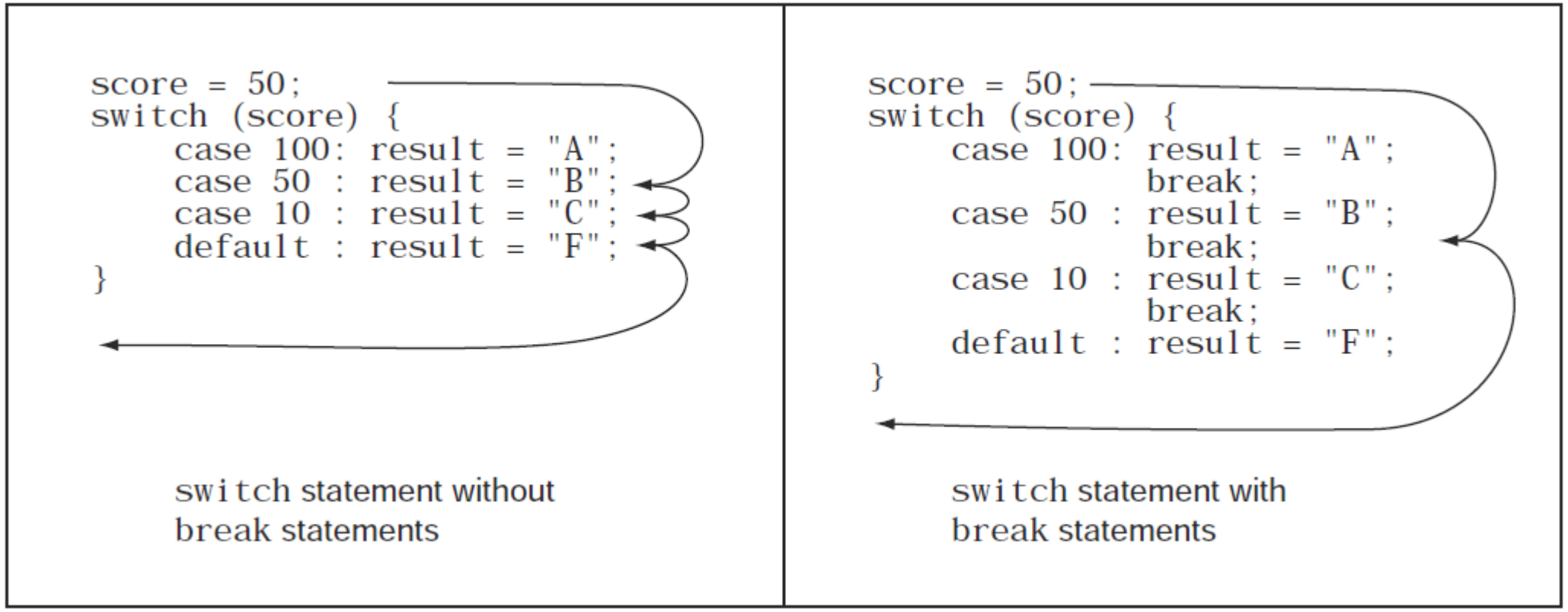

```
public class JavaTernary1
{
public static void main(String[] args)
   {
      int minVal, a=5, b=2;
      minVal = a < b ? a : b;
      System.out.println(minVal);
   }
}
```


C:\Windows\system32\cmd.exe
e:\myjavaprogs>javac JavaTernary1.java
e:\myjavaprogs>java JavaTernary1
2
e:\myjavaprogs>

# Switch Vs If


switch
char
byte
short
int
String
Integer
Short
Byte
Character
enum
if
boolean
Boolean

# Προσοχή με τη χρήση Switch



# Προαιρετικά πεδία στο for loop

```
int a = 10;
for(; a < 5; ++a) {
    System.out.println(a);
}
```

Valid for loop without any code in the initialization block

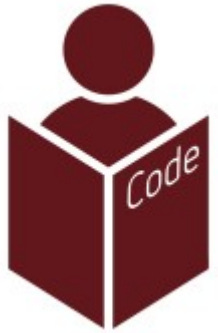
```
for(int a = 10; ; ++a) {
    System.out.println(a);
}
```

Missing termination condition implies infinite loop

```
for(int a = 10; a > 5; ) {
    System.out.println(a);
}
```

Missing update clause

# Enhanced for loop (*for-each*)




```
int total = 0;
int primeNums[] = {2, 3, 7, 11};
for (int num : primeNums)
    total += num;
```

# Java for-each

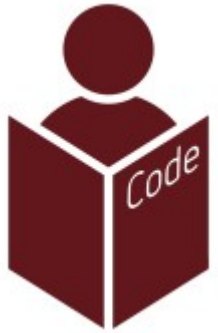
- Since Java 1.5

```
for (Type item : iterableCollection) {
// Do something to item }
```

# Examples (without foreach loop)

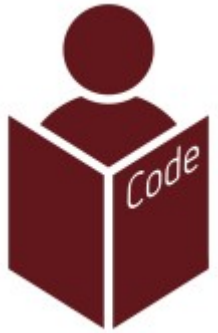
```
int[] intarray = new int[]{2,4,6,8};

for(int i=0;i<intarray.length;i++)
        System.out.println(intarray[i]);
```

# Examples (with foreach loop)




```
int[] intarray = new int[]{2,4,6,8};

for(int i : intarray)
        System.out.println(i]);
```

# Some limitations

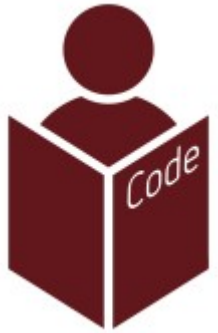



```
int[] intarray = new int[]{2,4,6,8};

    for(int i=0;i<intarray.length;i+=2)
        System.out.println(intarray[i]);
```

# More examples 1/2

```
String[] strarray = new String[]{"Maria","Pavlos","George"};

    for(String s : strarray)
        System.out.println(s);
```

# More examples 2/2 (even better…)

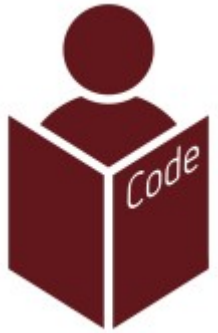



```
List<String> mylist = new
ArrayList<String>();
        mylist.add("Hallo");
        mylist.add("Students");
        mylist.add("of");
        mylist.add("Unipi");
        for(String s : mylist)
            System.out.println(s);
```

Wait! What was that strange <String>
in
List<String> mylist = new ArrayList<String>();
?

It is called Java Generics
And we will talk about this too..!

# for Vs for-each



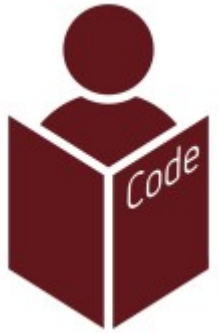

- The enhanced `for` loop can't be used to initialize an array and modify its elements.
- The enhanced `for` loop can't be used to delete the elements of a collection.
- The enhanced `for` loop can't be used to iterate over multiple collections or arrays in the same loop.

# Java – Labeled Statements

- Στη Java μπορούμε να χρησιμοποιήσουμε labels στις παρακάτω περιπτώσεις τμημάτων κώδικα:

  - A code block defined using { }
  - All looping statements (`for`, enhanced `for`, `while`, `do-while`)
  - Conditional constructs (`if` and `switch` statements)
  - Expressions
  - Assignments
  - `return` statements
  - `try` blocks
  - `throws` statements

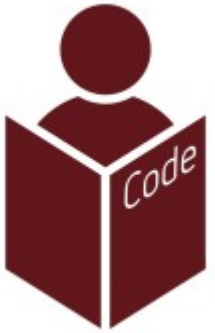
# Labeled break




```
String[] programmers = {"Outer", "Inner"};
outer:
for (String outer : programmers) {
    for (String inner : programmers) {
        if (inner.equals("Inner"))
            break outer;
        System.out.print(inner + ":");
    }
}
```

Exits the outer loop, marked with label outer

# Labeled continue




```
String[] programmers = {"Paul", "Shreya", "Selvan", "Harry"};
outer:
for (String name1 : programmers) {
    for (String name : programmers) {
        if (name.equals("Shreya"))
            continue outer;
        System.out.println(name);
    }
}
```

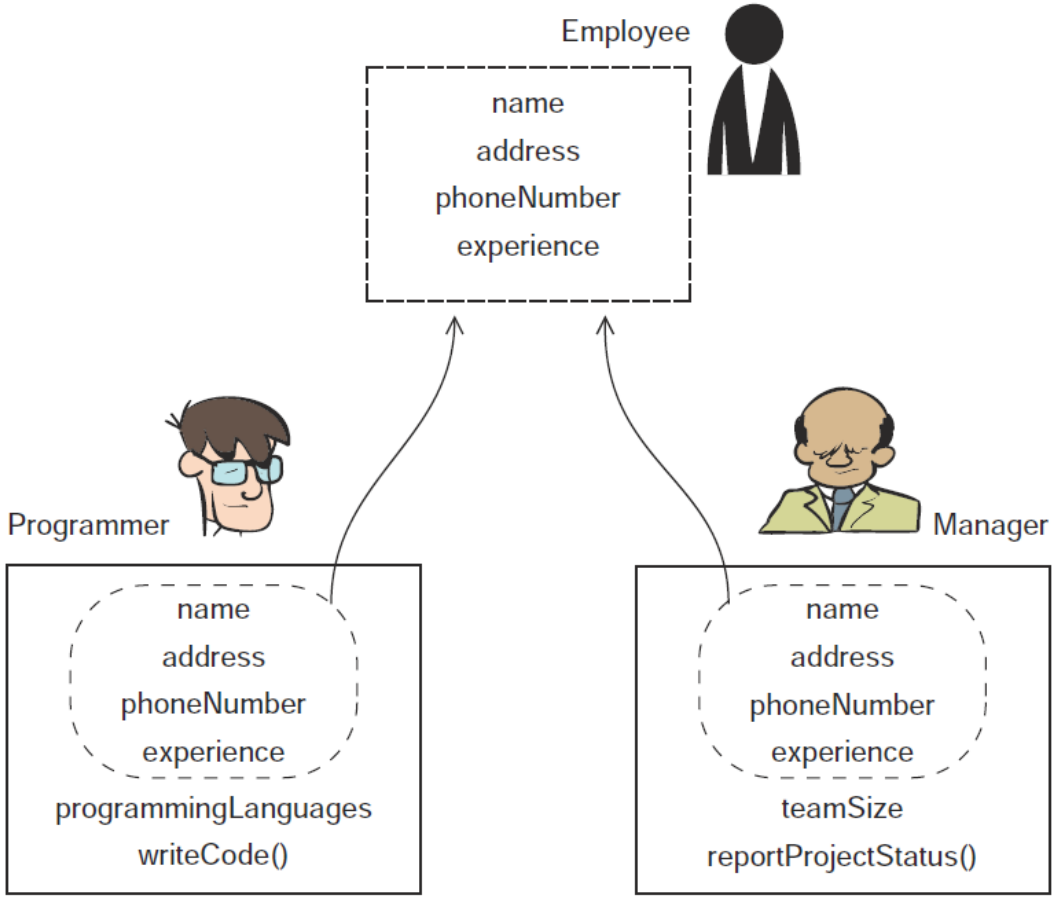
**Skips remaining code for current iteration of outer loop and starts with its next iteration**

# Σύνοψη – Branching statements

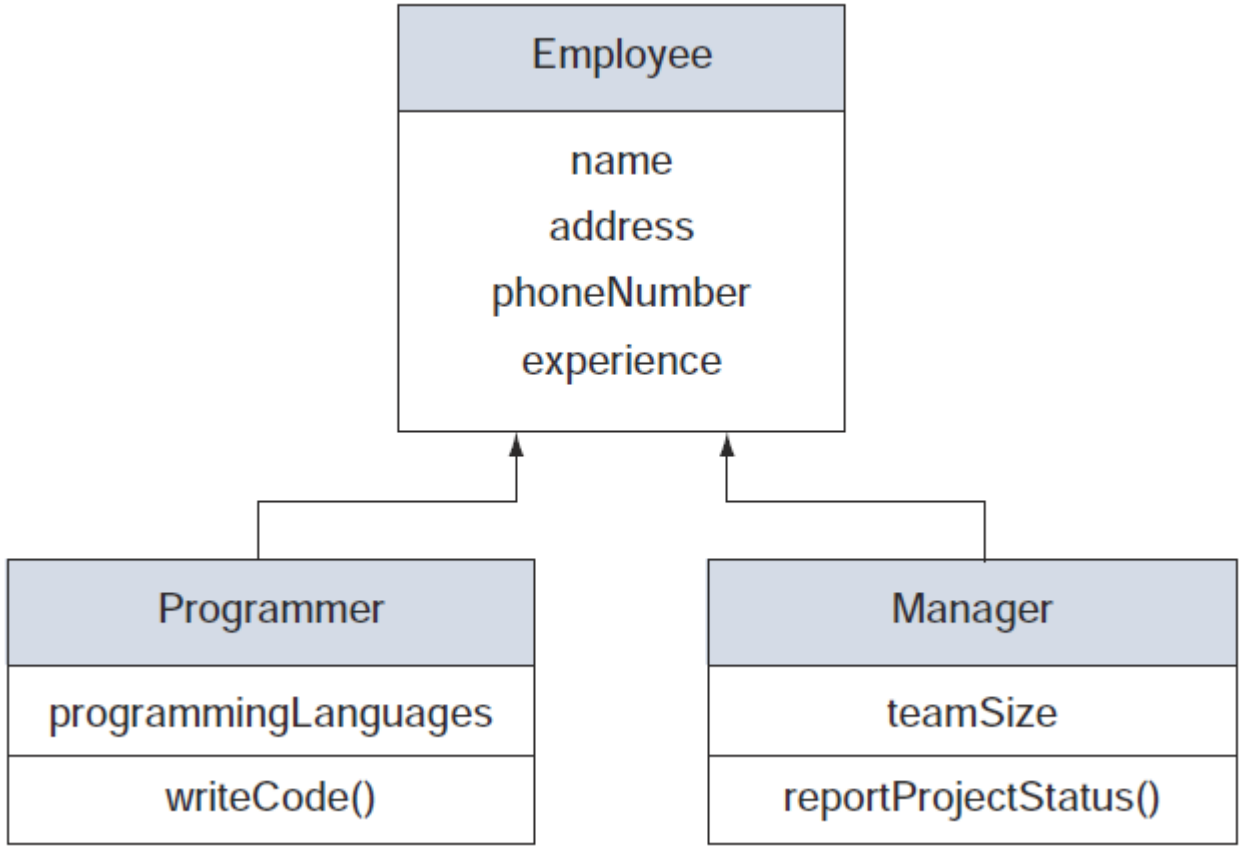
- Break
  - Unlabeled
  - Labeled
- Continue
  - Unlabeled
  - Labeled
- Return
  - With value
  - No value

# Java Inheritance

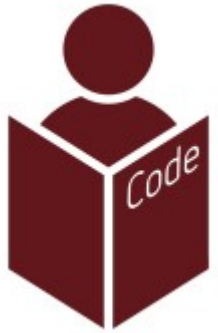
# Inheritance 1/3



# Inheritance 2/3



# Inheritance 3/3




```
class Employee {
    String name;
    String address;
    String phoneNumber;
    float experience;
}
class Programmer extends Employee {
    String[] programmingLanguages;
    void writeCode() {}
}
class Manager extends Employee {
    int teamSize;
    void reportProjectStatus() {}
}
```

# Inheritance Σημειώσεις

- In the Java language, classes can be derived from other classes, thereby inheriting fields and methods from those classes
- A class that is derived from another class is called a subclass
- The class from which the subclass is derived is called a superclass

# What can a Subclass do?

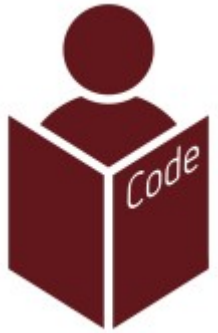
- The inherited fields can be used directly, just like any other fields
- You can declare new fields in the subclass that are not in the superclass
- The inherited methods can be used directly as they are
- You can write a new instance method in the subclass that has the same signature as the one in the superclass, thus overriding it
- You can declare new methods in the subclass that are not in the superclass

# Inheritance Example

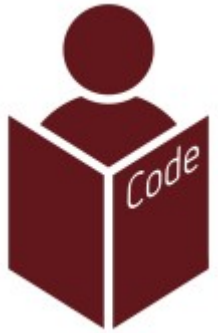
```
public class Animal{
}

public class Mammal extends Animal{
}

public class Reptile extends Animal{
}

public class Dog extends Mammal{
}
```

# IS-A vs HAS-A

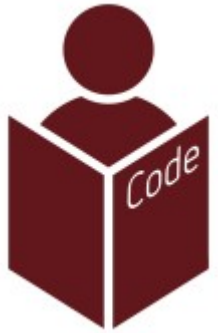
- IS-A σημαίνει «είναι του τύπου» και υποδηλώνει κληρονομικότητα (Inheritance) μεταξύ αντικειμένων
- HAS-A σημαίνει «έχει» ή «διαθέτει» και υποδηλώνει σχέση σύνθεσης (Composition) μεταξύ αντικειμένων



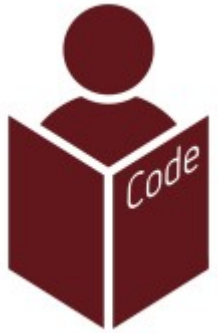

```java
package com.unipi.talepis;

class Car {
    // Methods implementation and class/Instance members
    private String color;
    private int maxSpeed;
    void carInfo(){
        System.out.println("Car Color= "+color + " Max Speed= " + maxSpeed);
    }
    void setColor(String color) {
        this.color = color;
    }
    void setMaxSpeed(int maxSpeed) {
        this.maxSpeed = maxSpeed;
    }
}
```




```java
package com.unipi.talepis;

public class Engine {
    void start(){
        System.out.println("Engine Started:");
    }
    public void stop(){
        System.out.println("Engine Stopped:");
    }
}
```



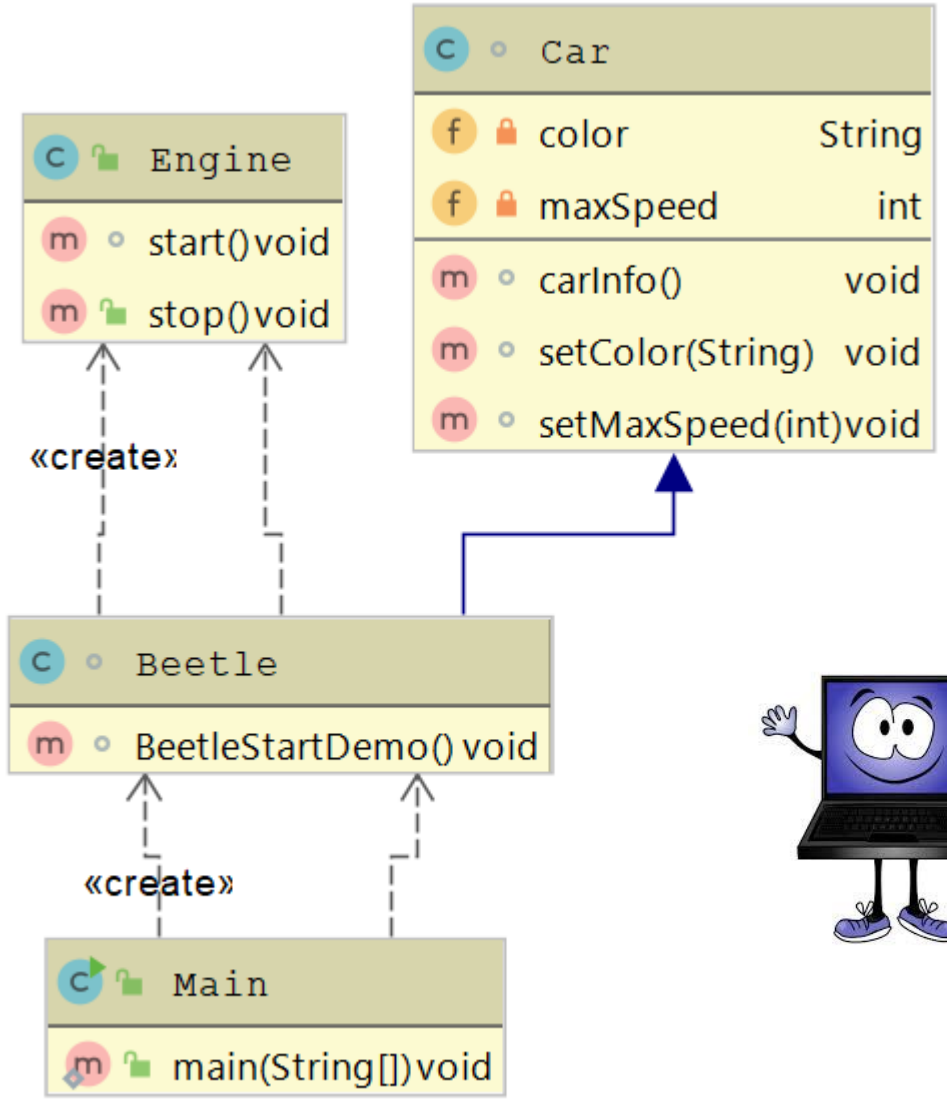

```
package com.unipi.talepis;

class Beetle extends Car{
    //Beetle extends Car and thus inherits all methods
    //from Car (except final and static)
    //Beetle can also define all its specific functionality
    void BeetleStartDemo(){
        Engine beetleEngine = new Engine();
        beetleEngine.start();
    }
}
```




```java
package com.unipi.talepis;

public class Main {

    public static void main(String[] args) {
        Beetle myBeetle = new Beetle();
        myBeetle.setColor("Blue");
        myBeetle.setMaxSpeed(200);
        myBeetle.carInfo();
        myBeetle.BeetleStartDemo();
    }
}
```

```
Main
"C:\Program Files\Java\jdk-11.0.1\bin\java.exe" "-javaage
Car Color= Blue Max Speed= 200
Engine Started:

Process finished with exit code 0
```

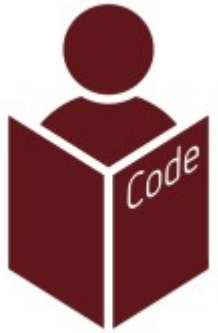
# Class Diagram



# Java Method Overriding

- A method defined in a super class may be overridden by a method of the same name defined in a sub class
- The overriding method has the same name, number and type of parameters, and return type as the method that it overrides

# Πλεονεκτήματα Κληρονομικότητας



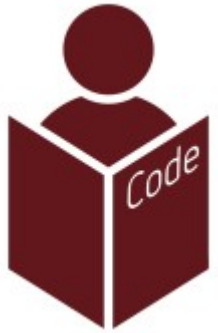

- Μικρότερο μέγεθος για τις τάξεις – παιδιά
- Εύκολη τροποποίηση των κοινών μεθόδων και ιδιοτήτων
- Επεκτασιμότητα
- Base class code testing
- Λογική «τμηματοποίηση» του κώδικα

# Χρήση μεταβλητών αντικειμένων

# Μεταβλητές

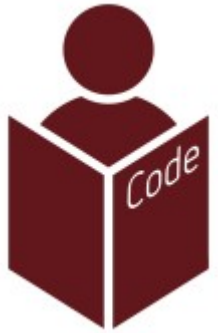
- Μπορούμε να αναφερθούμε σε μια μεταβλητή αντικειμένου στη Java με περισσότερους από έναν τρόπους:
  - Απευθείας με την τάξη στην οποία ανήκει
  - Μέσω μιας υπερ-τάξης της τάξης που ανήκει
  - Μέσω ενός interface το οποίο υλοποιεί
  - Μέσω της τάξης Object (Γιατί?)
- Σε κάθε περίπτωση, ωστόσο, πρέπει να προσέξουμε τις διαφορές που προκύπτουν

# Παράδειγμα



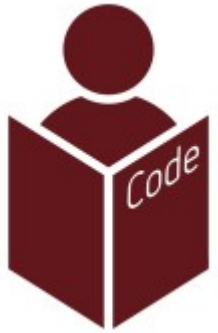

```
class Human{
    int age;
    String name;
}
interface ISpeak{
    void speak(String s);
}
class Student extends Human implements ISpeak{
    int am;
    @Override
    public void speak(String s) {
        System.out.println("Hello "+s);
    }
}
class Professor extends Human implements ISpeak{
    int officeNumber;
    @Override
    public void speak(String s) {
        System.out.println("Hi "+s);
    }
}
```

# Μέσω της τάξης του




```
Student student = new Student();
student.name = "George";
student.am = 12345;
student.speak( s: "Unipi");
```

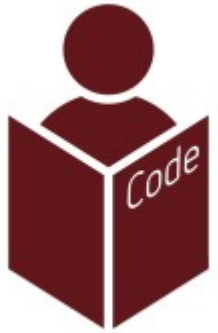
# Μέσω της super class




```
        Human human = new Student();
        human.name = "Maria";
        human.am = 23456;
        human.speak("Informatics");
    }
```

Cannot resolve method 'speak(java.lang.String)'

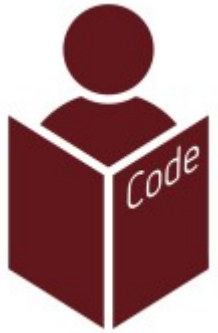
# Μέσω του interface που υλοποιεί




```
ISpeak speaker = new Student();
speaker.speak( s: "Piraeus");
speaker.name = "Peter";
speaker.am = 34567;
```

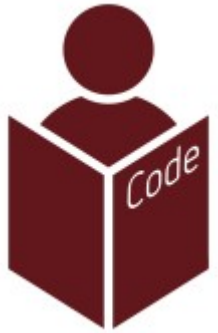
Cannot resolve symbol 'am'

# Μέσω της τάξης Object




```
Object o = new Student();
o.name = "Costas";
o.am = 45678;
o.speak("Greece");
```

# Γιατί υπάρχει τότε αυτή η δυνατότητα;




```
ISpeak[] speakers = new ISpeak[2];
speakers[0] = new Student();
speakers[1] = new Professor();
for (ISpeak iSpeak : speakers)
    iSpeak.speak( s: "University of Piraeus!");
```

# Casting with objects

# What is Casting

- Casting ονομάζουμε τη διαδικασία κατά την οποία «αναγκάζουμε» μια μεταβλητή να «συμπεριφερθεί» ως μια άλλη μεταβλητή
- Σε επίπεδο τάξεων το casting λαμβάνει χώρα όταν ένα αντικείμενο είναι ενός άλλου τύπου (IS-A), δηλαδή έχουμε σχέση κληρονομικότητας

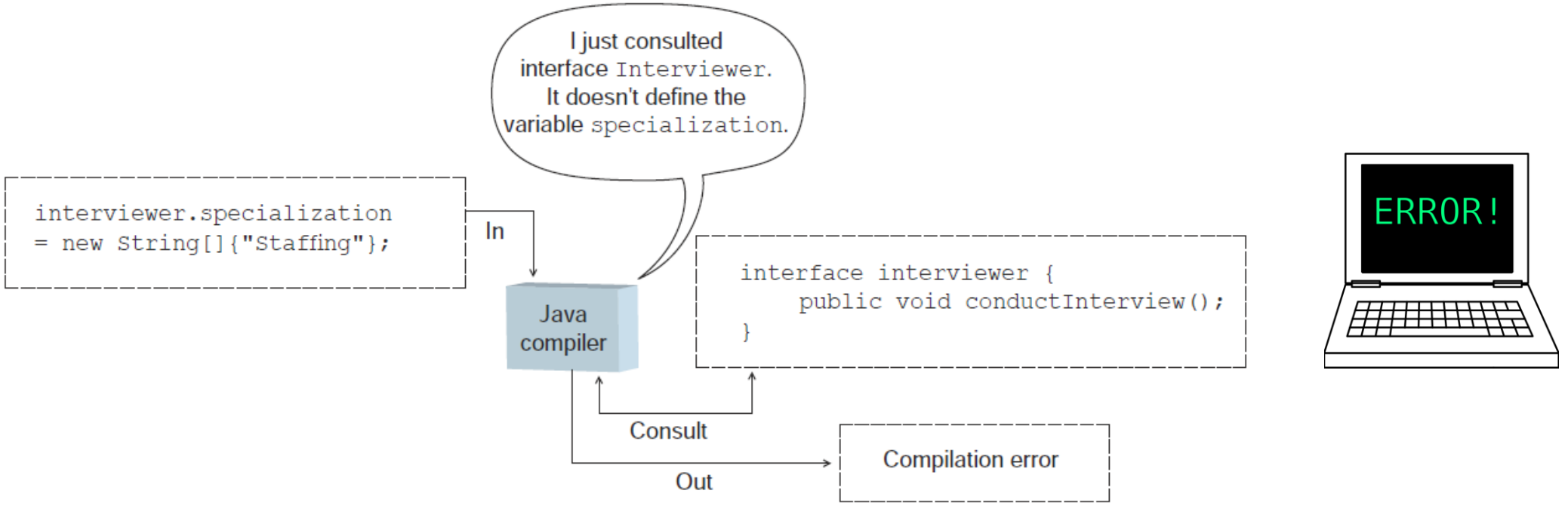
# Παράδειγμα

```
class Employee {}
interface Interviewer {
    public void conductInterview();
}
class HRExecutive extends Employee implements Interviewer {
    String[] specialization;
    public void conductInterview() {
        System.out.println("HRExecutive - conducting interview");
    }
}
class Manager implements Interviewer{
    int teamSize;
    public void conductInterview() {
        System.out.println("Manager - conducting interview");
    }
}
```

# Δοκιμές 1/2

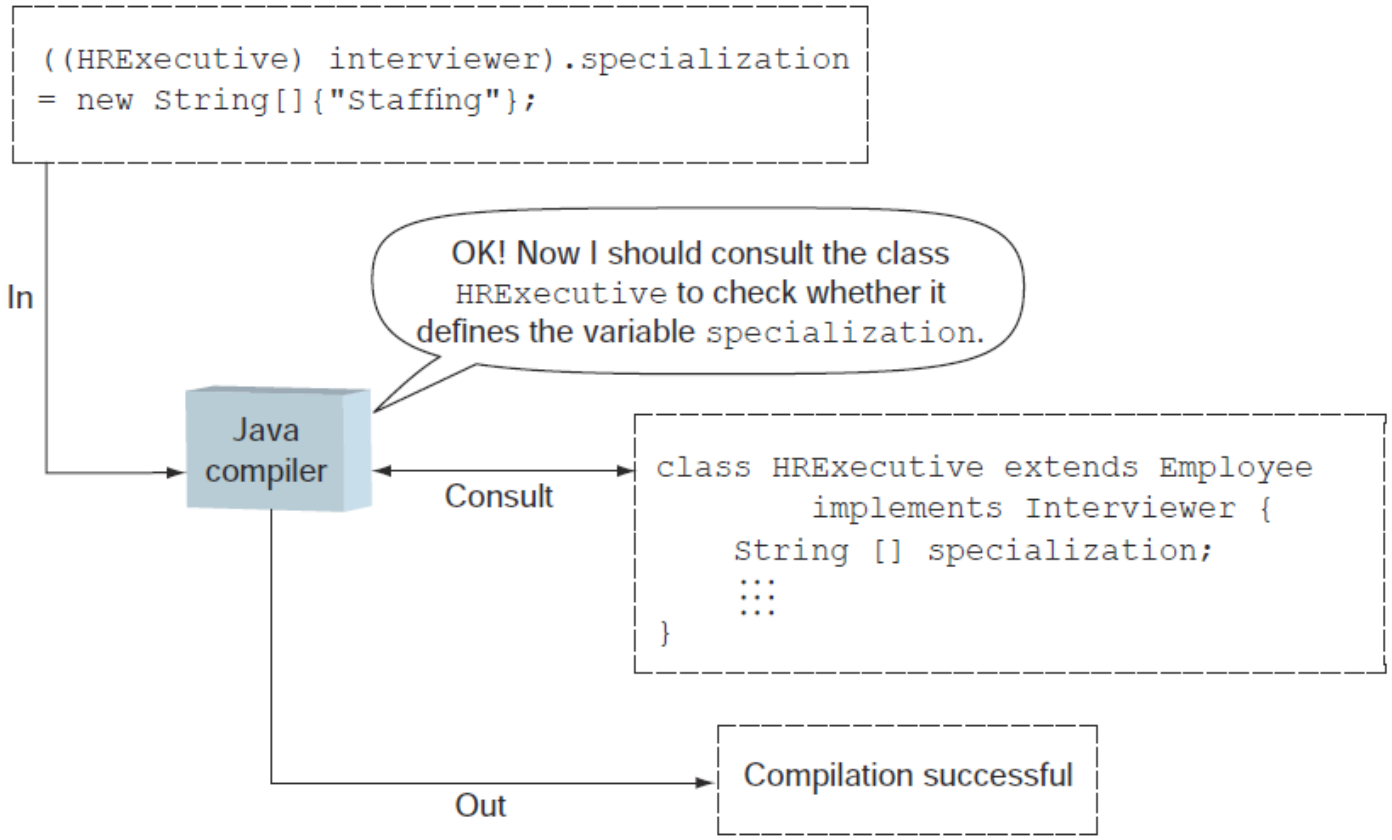
```
Interviewer interviewer = new HRExecutive();

interviewer.specialization = new String[] {"Staffing"};
```

⟵ **Won't compile**



# Δοκιμές 2/2

```
((HRExecutive)interviewer).specialization = new String[] {"Staffing"};
```




# Java Anonymous Classes

# Τι είναι anonymous class?

- Anonymous class είναι μια ανώνυμη κλάση, χωρίς όνομα, για την οποία δημιουργούμε και αντικείμενο αμέσως με τη δήλωσή της
- Δεν χρησιμοποιούνται για τη δημιουργία πολλαπλών αντικειμένων
- Χρησιμοποιούνται για τη δήλωση και την αρχικοποίηση ενός αντικειμένου μέσα σε ένα expression
- Μπορούν να οριστούν με 2 (+1) τρόπους:
  - Extending an existing class
  - Implementing an existing interface
  - Με τον «κλασικό» τρόπο δημιουργίας στιγμιότυπου, χωρίς την ύπαρξη μεταβλητής. Έτσι βέβαια δεν θα δημιουργηθεί νέα κλάση, αλλά μόνο ένα ανώνυμο αντικείμενο. Συμπερασματικά, anonymous class έχουμε με 2 τρόπους, ενώ για anonymous object μπορούμε να χρησιμοποιήσουμε ακόμα έναν τρόπο

# Extending an existing class

# Implementing an existing interface

```
new InterfaceName() {...}
```

name of the interface to implement — methods' implementations

# Using new without a variable

- Ο 3ος τρόπος είναι ουσιαστικά να χρησιμοποιηθεί ένα αντικείμενο απλώς χωρίς την ύπαρξη μεταβλητής, μόνο με τη χρήση του στιγμιοτύπου
- new Object()
- Εδώ βέβαια δεν δημιουργείται νέα τάξη, μόνο ένα νέο αντικείμενο
- Αν γίνει τροποποίηση οποιουδήποτε χαρακτηριστικού του νέου αντικειμένου κατά το instantiation, τότε θα ισχύει η πρώτη περίπτωση -> **extending an existing class**

# Παραδείγματα

```java
package com.unipi.talepis;

public class Human {
    private int age;
    private String name;
    public void eat(){
        System.out.println("I am eating!");
    }
}
```

```java
package com.unipi.talepis;

public interface ISee {
    void see(String s);
}
```

# Extending an existing Class

```
package com.unipi.talepis;

public class Main {

    public static void main(String[] args) {
        new Human(){
            @Override
            public void eat() {
                System.out.println("Hello, I am eating now!");
            }
        }.eat();
    }
}
```

```
C:\Java\jdk-9.0.4\bin\java.exe "-javaagent:
Hello, I am eating now!

Process finished with exit code 0
```

# Implementing an existing Interface

```
package com.unipi.talepis;

public class Main {

    public static void main(String[] args) {
        new ISee(){
            @Override
            public void see(String s) {
                System.out.println("I am watcing "+s);
            }
        }.see( s: "Unipi");
    }
}
```

```
C:\Java\jdk-9.0.4\bin\java.exe "-javaagent:
I am watcing Unipi

Process finished with exit code 0
```

# Με τη χρήση απλού “new”

```
package com.unipi.talepis;

public class Main {

    public static void main(String[] args) {
        new Human().eat();
    }
}
```

```
C:\Java\jdk-9.0.4\bin\java.exe "-javaagent:
I am eating!

Process finished with exit code 0
```

- Για τη χρήση ενός απλού anonymous object μπορεί να χρησιμοποιηθεί και οποιαδήποτε συνάρτηση επιστρέφει το εν λόγω object

```
package com.unipi.talepis;

public class Human {
    private int age;
    private String name;
    public void eat(){
        System.out.println("I am eating!");
    }
    public static Human getAHuman(){
        return new Human();
    }
}
```

```java
package com.unipi.talepis;

public class Main {

    public static void main(String[] args) {
        //new Human().eat();
        Human.getAHuman().eat();
    }
}
```

```
C:\Java\jdk-9.0.4\bin\java.exe "-javaagent:
I am eating!

Process finished with exit code 0
```

# Χρήση μεταβλητών από anonymous classes

- Μια ανώνυμη τάξη έχει πλήρη πρόσβαση σε member variables της τάξης στην οποία ανήκει (οι anonymous classes θα είναι «αναγκαστικά» εσωτερικές τάξεις)
- Μια ανώνυμη τάξη ΔΕΝ μπορεί να έχει πρόσβαση σε τοπικές μεταβλητές (εντός μεθόδων), οι οποίες δεν είναι δηλωμένες ως final, ή έστω να είναι ουσιαστικά τελικές (effectively final)

# Παραδείγματα

```
public static void main(String[] args) {
    /*new ISee(){
        @Override
        public void see(String s) {
            System.out.println("I am watching "+s);
        }
    }.see("Unipi");*/
    String temp = "John";
    final String t = "Jim";
    new Human(){
        int i = 5;
        String s = args[0];
        String g = t;
        //String v = temp; //will compile if it is effective final
        private void go(){
            //temp = "Bob"; //problem
            System.out.println(s);
        }
    }.go();
}
```

# Anonymous class with no default constructor

```java
public class Student extends Human{
    private int AM;
    protected String email;
    public String department;

    public Student(int AM) {
        this.AM = AM;
    }
    public void speak(){
        System.out.println("My am is "+AM);
    }
}
```

```java
new Student( AM: 432){
}.speak();
```

# Anonymous class with instance initializer

```
public class Student extends Human{
    private int AM;
    protected String email;
    public String department;

    public Student(int AM) {
        this.AM = AM;
    }
    public void speak(){
        System.out.println("My am is "+AM);
    }
}
```

```
new Student( AM: 53){
    {
        department="Informatics";
        email = "me@unipi.gr"; //why?...
        //AM = 54; //of course not...
    }
}.speak();
```

# Τι μπορούμε να δηλώσουμε εντός μιας anonymous class

- Fields (instance, static need to be final and initialized)
- Extra methods (even if they do not implement any methods of the supertype)
- Instance initializers
- Local classes (εσωτερικές τάξεις)
- ΔΕΝ επιτρέπεται εσωτερικό interface

# Χρήση anonymous object

- Η χρήση των anonymous class/object δεν γίνεται συνήθως όπως στα προηγούμενα παραδείγματα (τα οποία δημιουργήθηκαν για εκπαιδευτικούς λόγους), αλλά μέσω παραμέτρων σε μεθόδους
- Οι μέθοδοι αυτοί επιβάλλουν χρήση:
  - είτε αντικειμένων που είναι υποτάξεις μια συγκεκριμένης τάξης
  - είτε αντικειμένων που υλοποιούν κάποιο συγκεκριμένο Interface
- Επίσης, εξίσου πολλές φορές τα anonymous objects αποτελούν μέρος ενός expression απόδοσης instance σε μια μεταβλητή

# x 32 => Java Projects