Java

Αντικειμενοστρεφής Προγραμματισμός

Βασικά Χαρακτηριστικά OOP

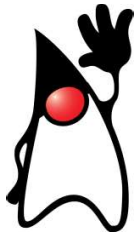
- Object-oriented programming (OOP) is a method of programming based on a hierarchy of classes, and well-defined and cooperating objects
- Ο αντικειμενοστρεφής προγραμματισμός είναι μια μέθοδος προγραμματισμού βασισμένη σε μια ιεραρχία τάξεων και καλά ορισμένα αντικείμενα, τα οποία αλληλεπιδρούν μεταξύ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

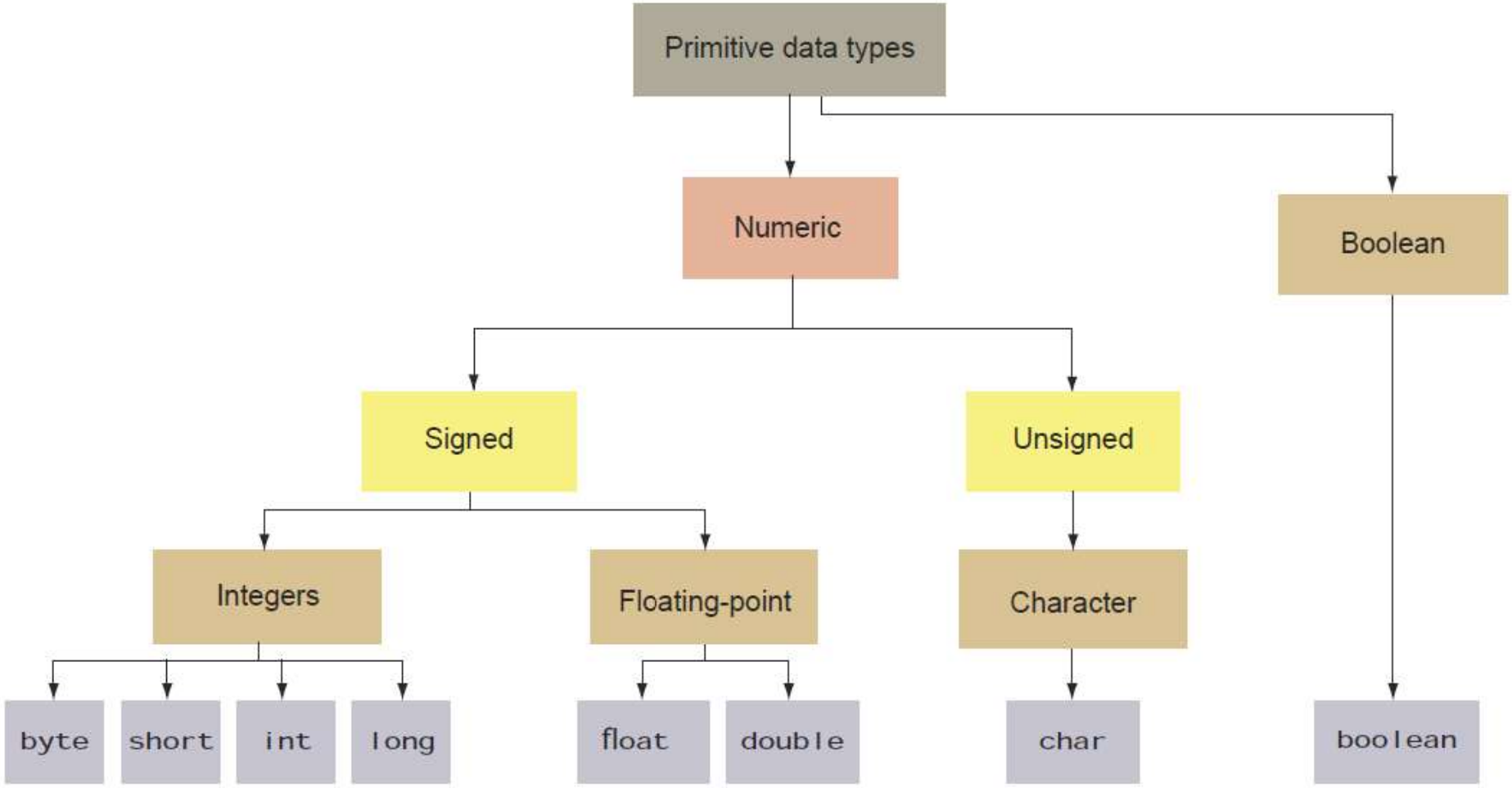
Data Types

Java Data Types

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

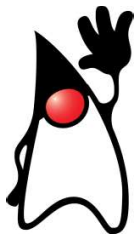
Primitive Data Types

- Στη Java υπάρχουν 8:
  - `char`
  - `byte`
  - `short`
  - `int`
  - `long`
  - `float`
  - `double`
  - `boolean`

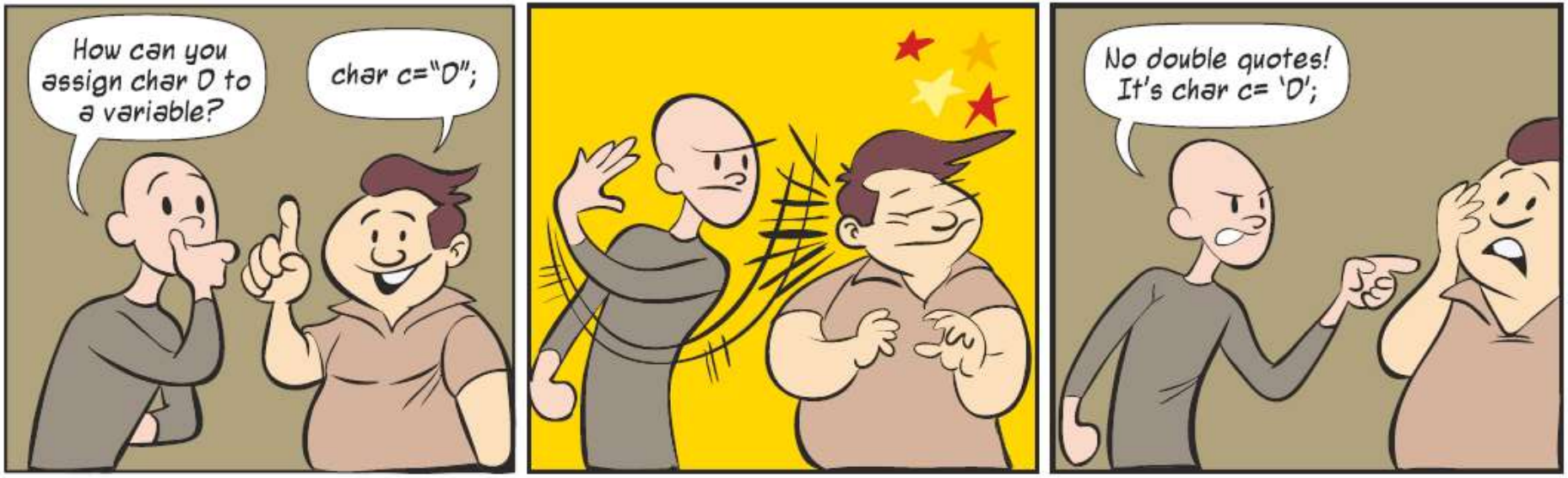

Primitive data types
Numeric
Boolean
Signed
Unsigned
Integers
Floating-point
Character
byte
short
int
long
float
double
char
boolean

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	

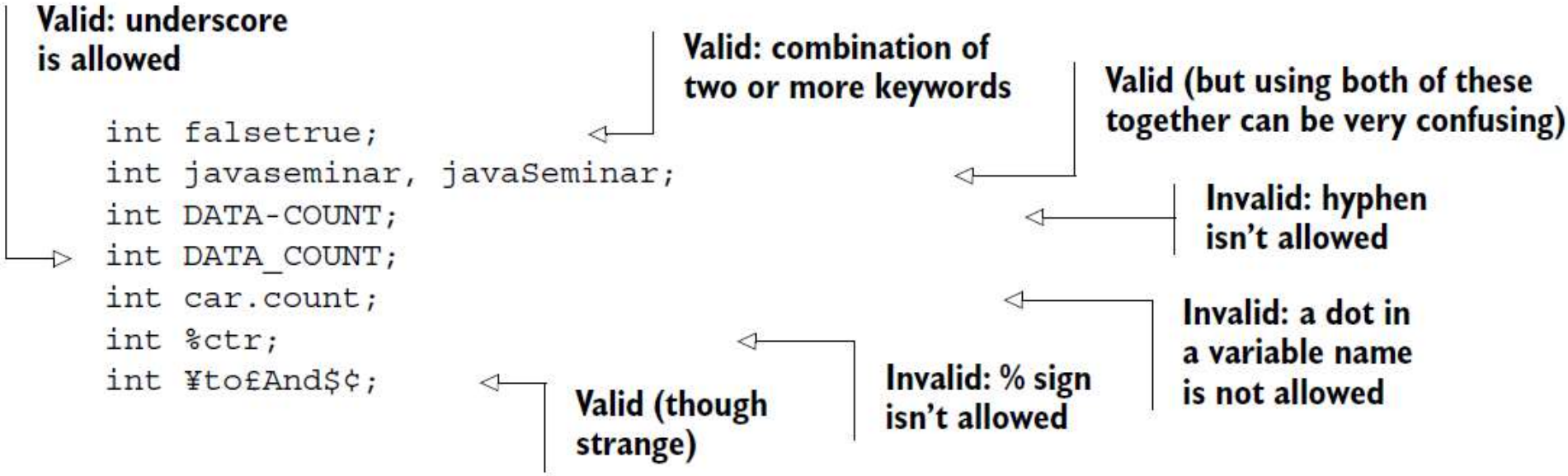
Παραδείγματα

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

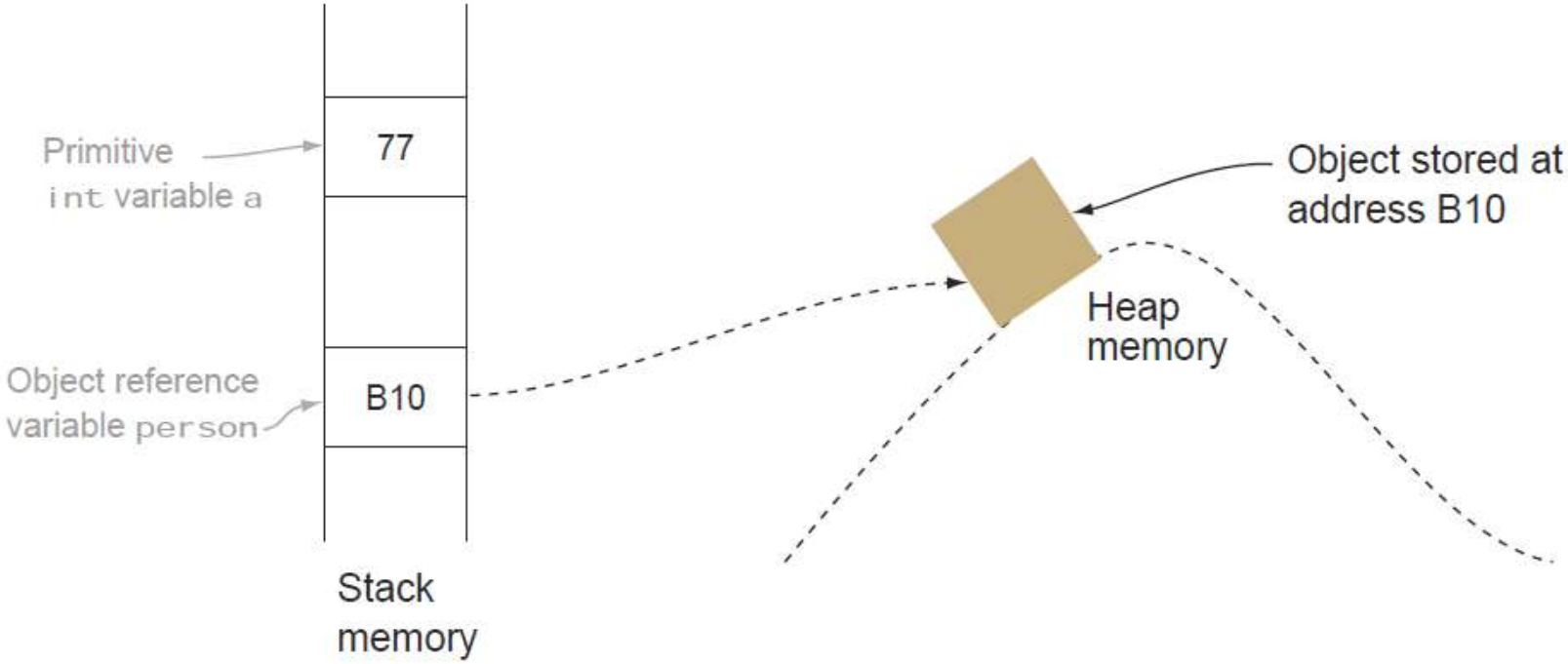


Reference Variables

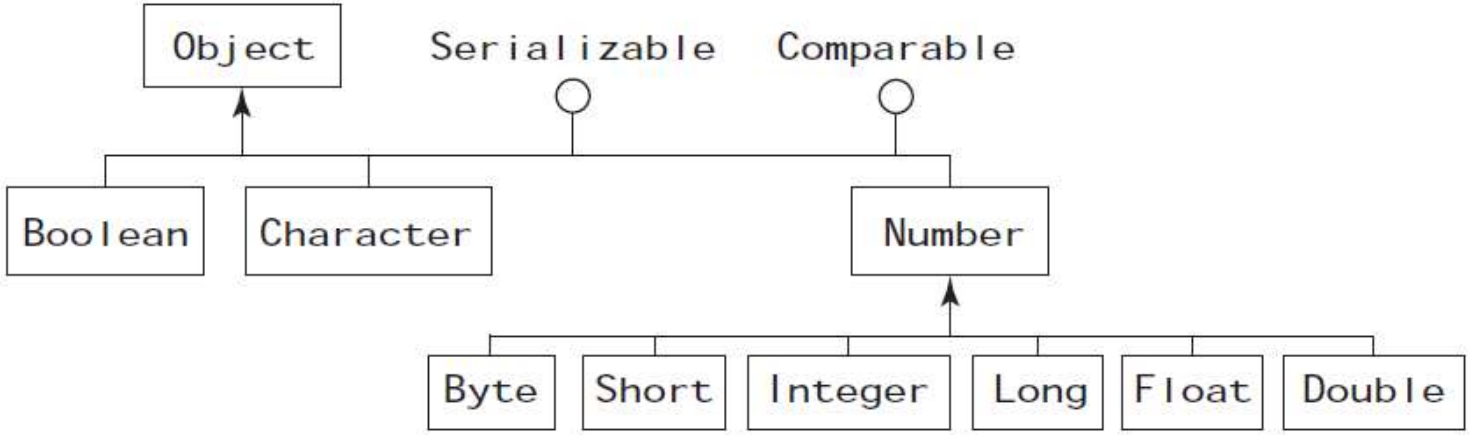
- Αλλιώς Object Variables
- Default τιμή -> null

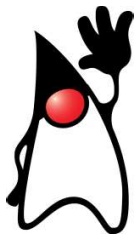
References Vs Values

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

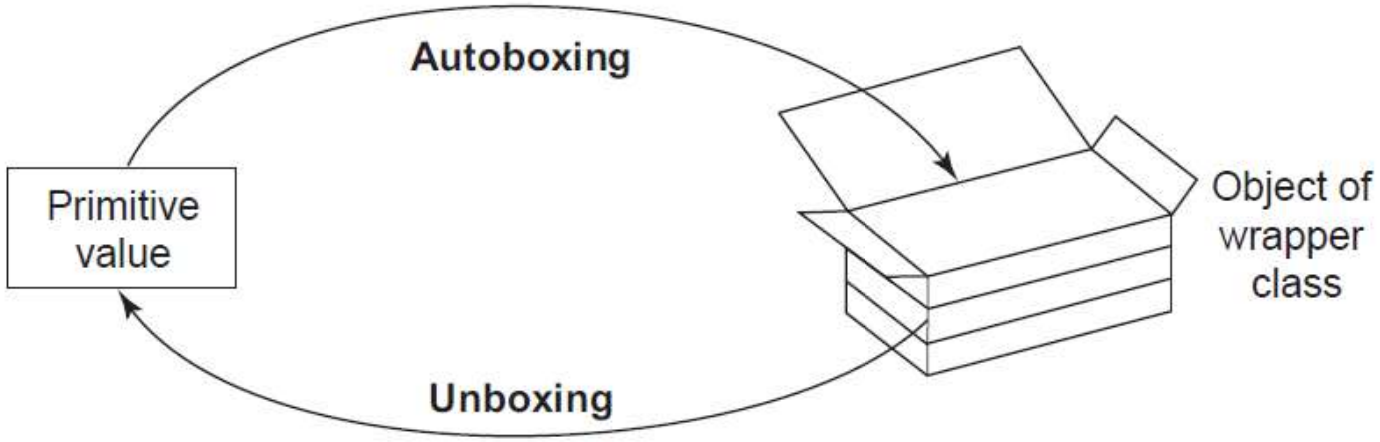


Wrapper Classes



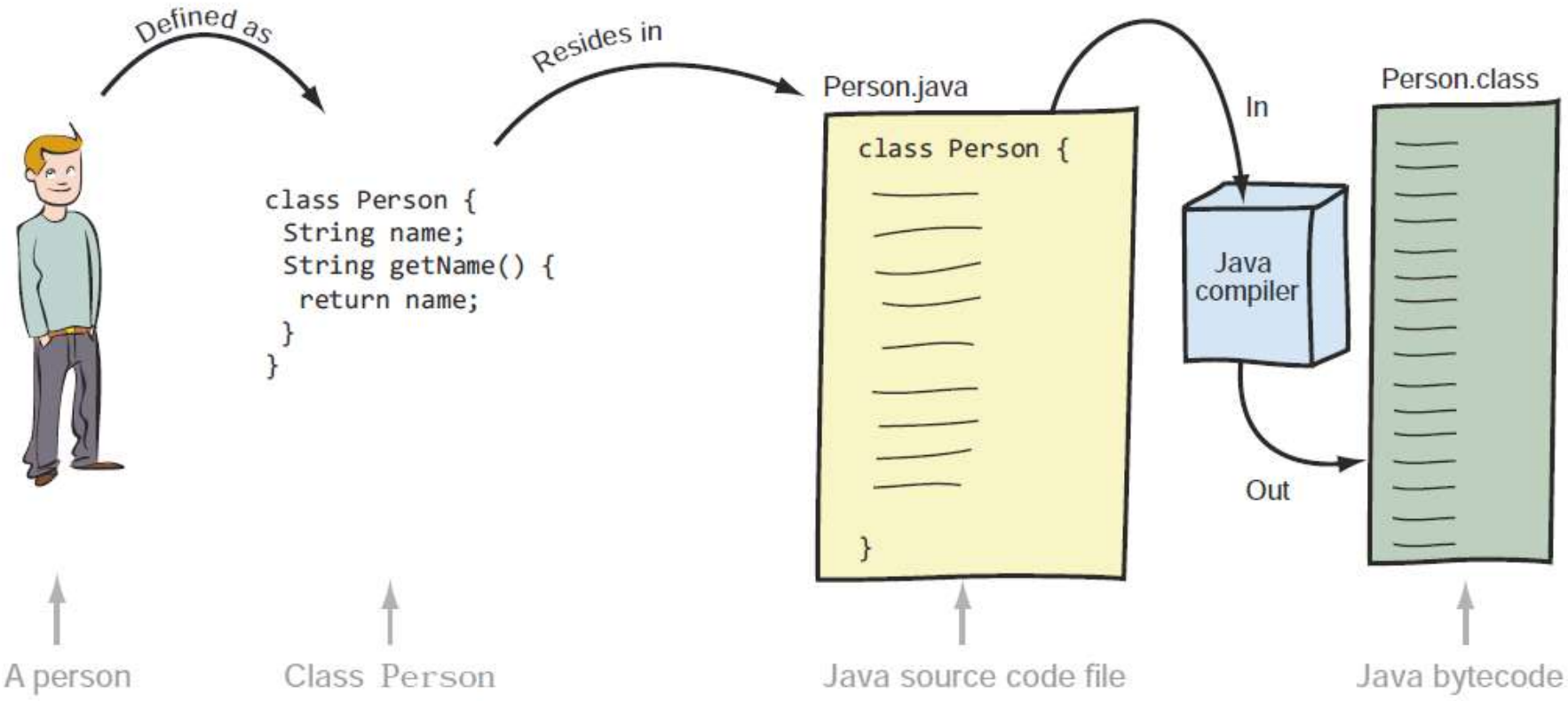


Autoboxing - Unboxing



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 &%^*{}|\|:;"'
       ?/>.<,!@#$%^&*()
    */
}
```

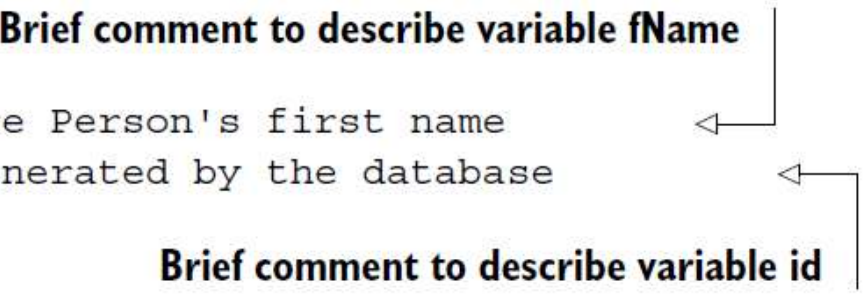
Multiline comment with special characters in it

```
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
}
```



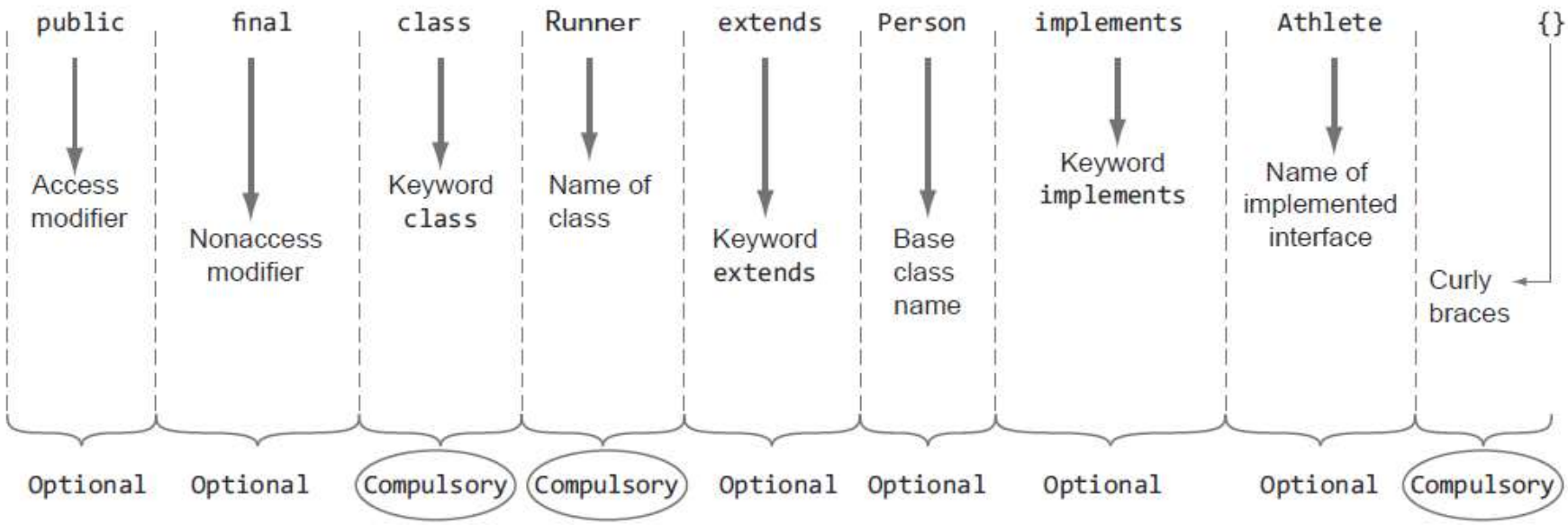
Question

- 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

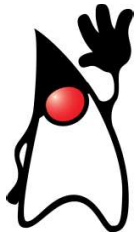

public
Access modifier
Optional
final
Nonaccess modifier
Optional
class
Keyword class
Compulsory
Runner
Name of class
Compulsory
extends
Keyword extends
Optional
Person
Base class name
Optional
implements
Keyword implements
Optional
Athlete
Name of implemented interface
Optional
{}
Curly braces
Compulsory

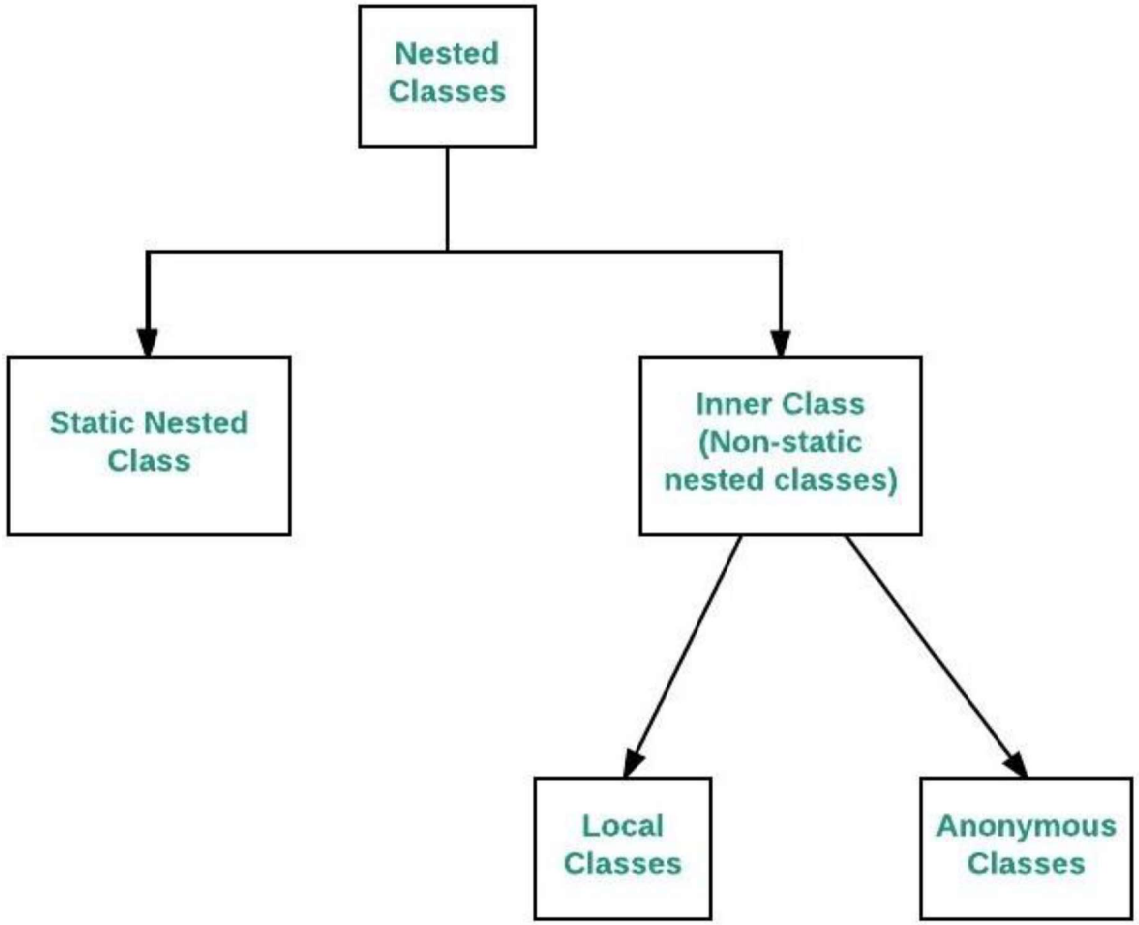
Example

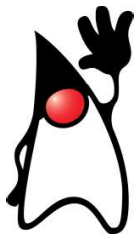
```
class Phone {
    String model;
    String company;
    Phone(String model) {
        this.model = model;
    }
    double weight;
    void makeCall(String number) {
        // code
    }
    void receiveCall() {
        // code
    }
}
```

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

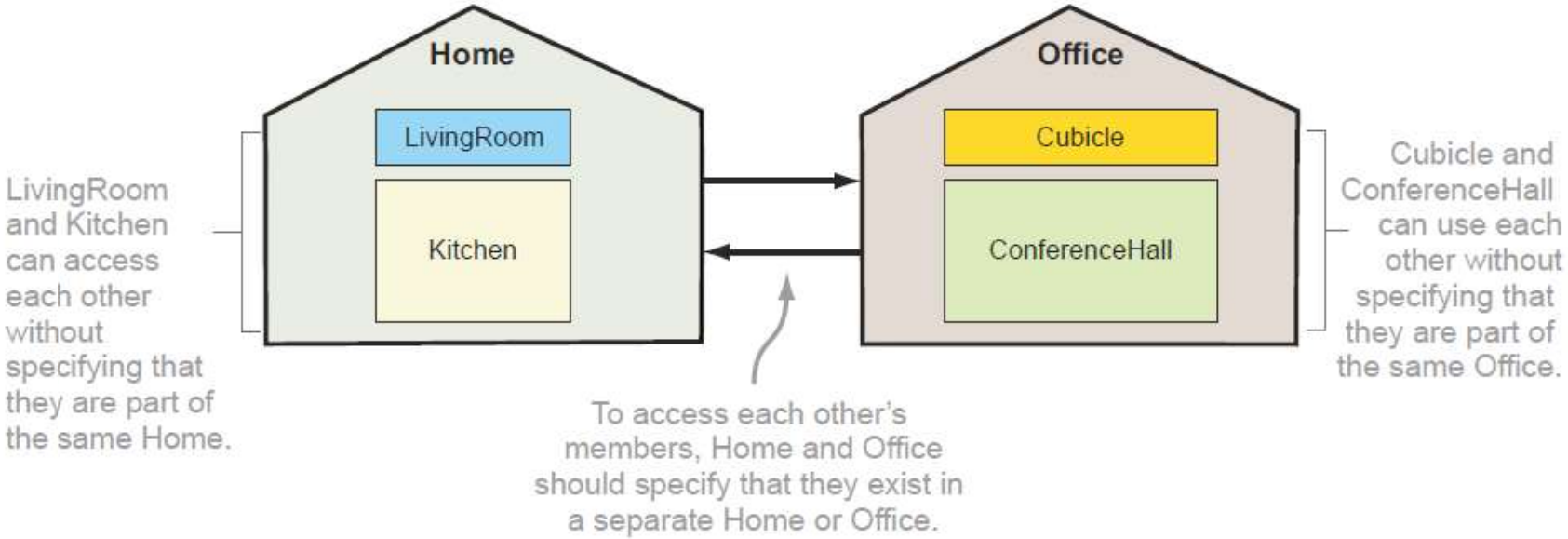
Fully qualified names

- 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

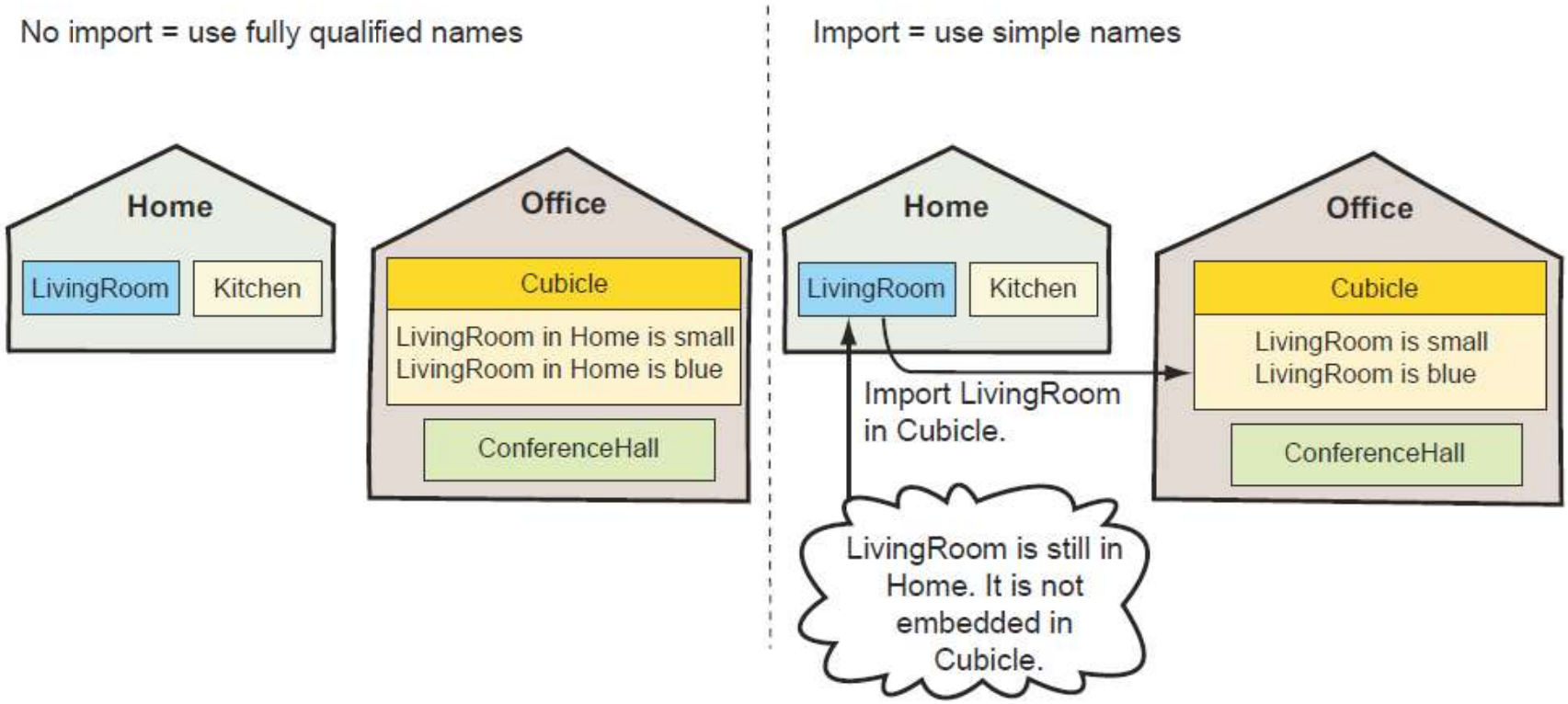
Import statements

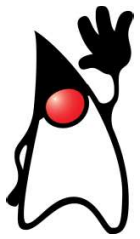
- Εντός του ιδίου πακέτου, οι τάξεις και οι διεπαφές μπορούν να χρησιμοποιήσουν τα μεταξύ τους στοιχεία χωρίς κάποιο πρόθεμα
- Για να μπορέσουμε να χρησιμοποιήσουμε τάξεις και διεπαφές από άλλα πακέτα, πρέπει να δηλώσουμε το «πλήρες» όνομά τους (fully qualified name)
- Επειδή η δήλωση των πλήρων ονομάτων 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

```
class AnnualExam {
    java.util.Date date1;
    java.sql.Date date2;
}
```

import statement not required

Variable of type java.util.Date

Variable of type java.sql.Date

Τι ισχύει με το default package?

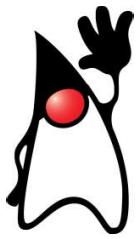
```
class Person {
    // code
}
class Office {
    Person p;
}
```

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;
    }
}
```

Access variable marks without prefixing it with its class name

Correct statement is import static, not static import

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. 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 com.apple.quicktime.v2 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; class ImageSprite;

Naming conventions 3/6

Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate; 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();
		runFast();
		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.
	
	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;`
		`char c;`
		`float myWidth;`

Naming conventions 6/6

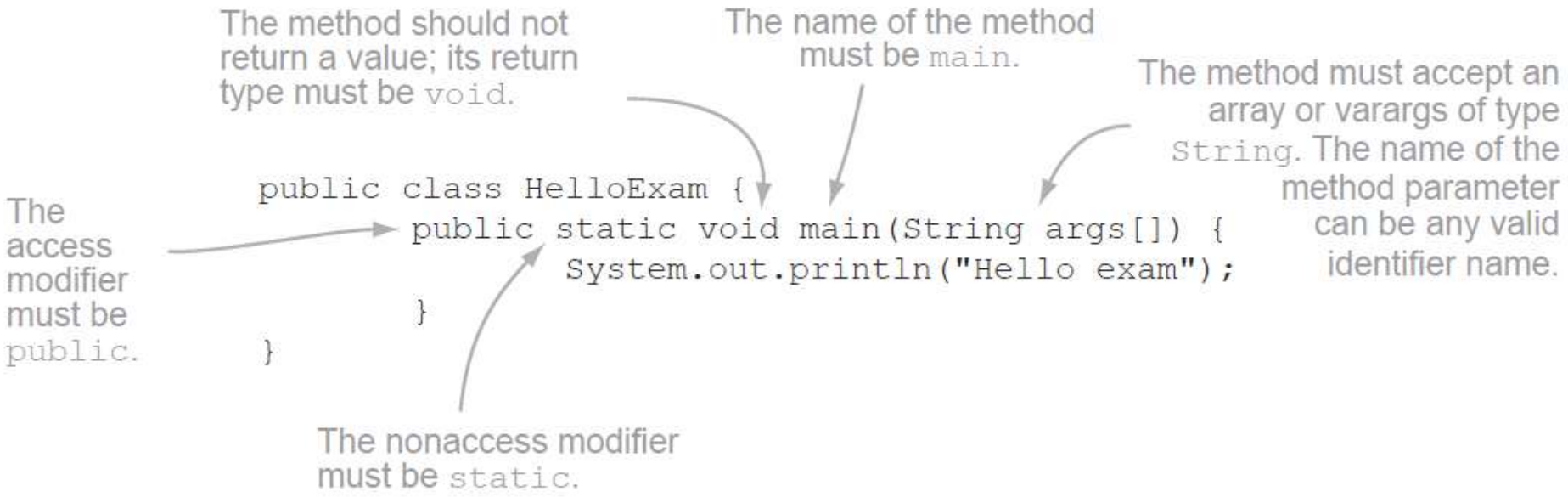
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		
		static final int MAX_WIDTH = 999;
		
		static final int GET_THE_CPU = 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.
The access modifier must be public.
public class HelloExam {
public static void main(String args[]) {
System.out.println("Hello exam");
}
}
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.

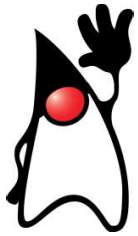
```
public static void main(String[] args)
static public void main(String[] args)
```

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

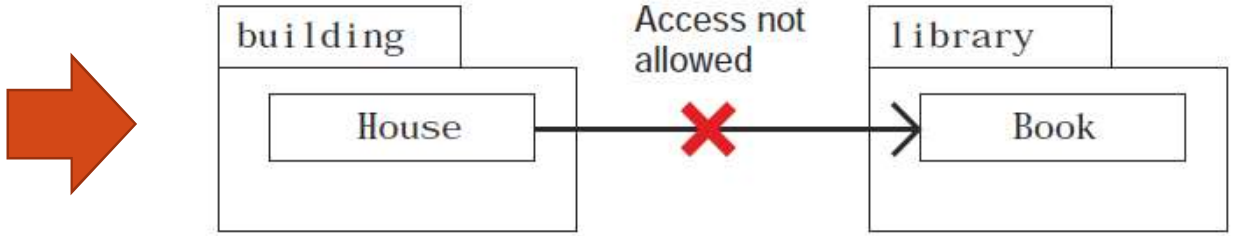
	private	default	protected	public
Class	No	Yes	No	Yes
Nested Class	Yes	Yes	Yes	Yes
Constructor	Yes	Yes	Yes	Yes
Method	Yes	Yes	Yes	Yes
Field	Yes	Yes	Yes	Yes

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

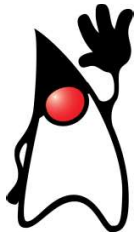
- Αφορά τάξεις, interfaces και μεθόδους (όχι μεταβλητές)
- Υποδηλώνει ότι οι υπολογισμοί που θα γίνουν και αφορούν αριθμούς κινητής υποδιαστολής, θα είναι ίδιοι σε όλες τις πλατφόρμες

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++

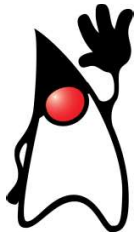


Abstract

- 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

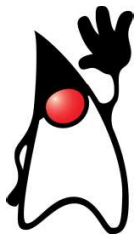
Σημειώσεις για abstract

- 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?)

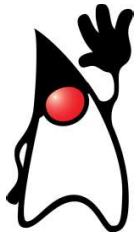


Static 1/2

- 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)



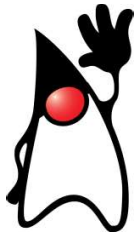
Προσπέλαση μεταξύ static και non-static

- 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 Phone {
    private boolean tested;
    public void setTested(boolean val) {
        tested = val;
    }
    public boolean isTested() {
        return tested;
    }
}
```

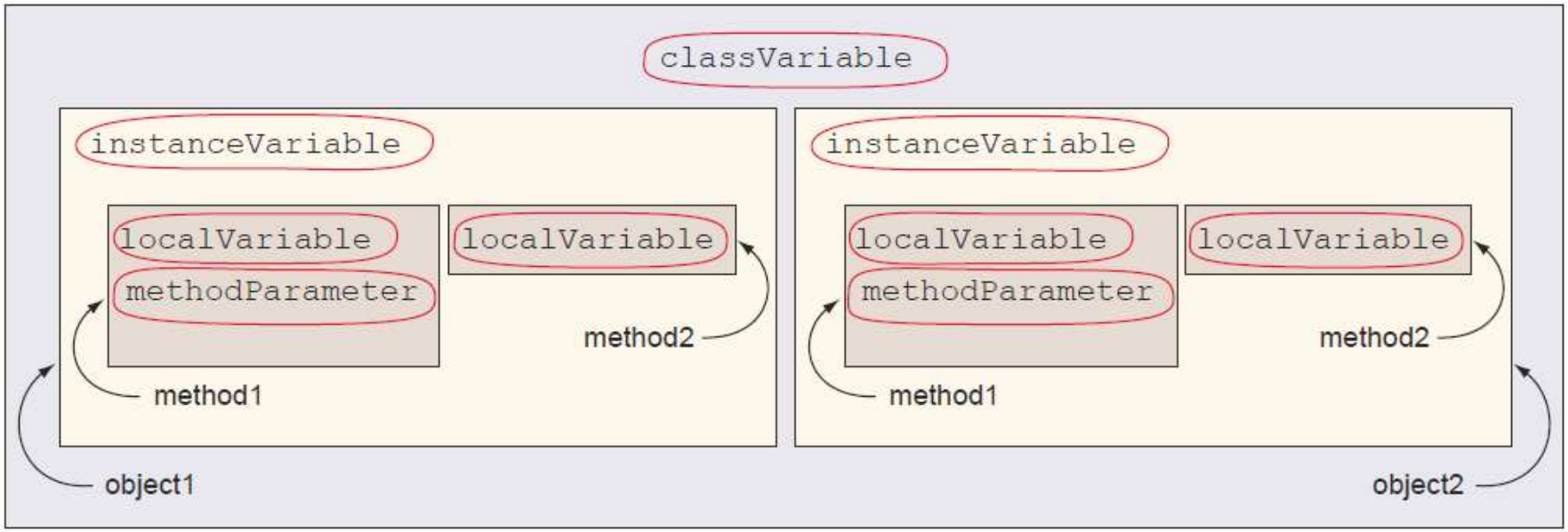
Instance variable tested

Variable tested is accessible in method setTested

Variable tested is also accessible in method isTested

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!