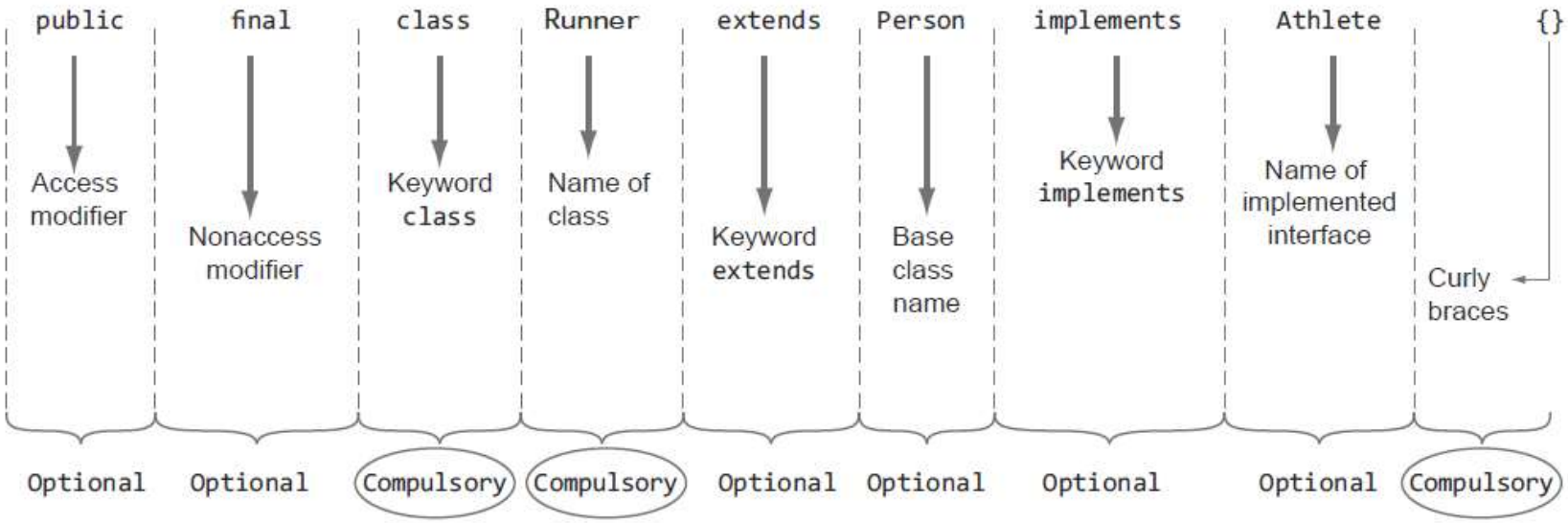
Class declaration

- 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

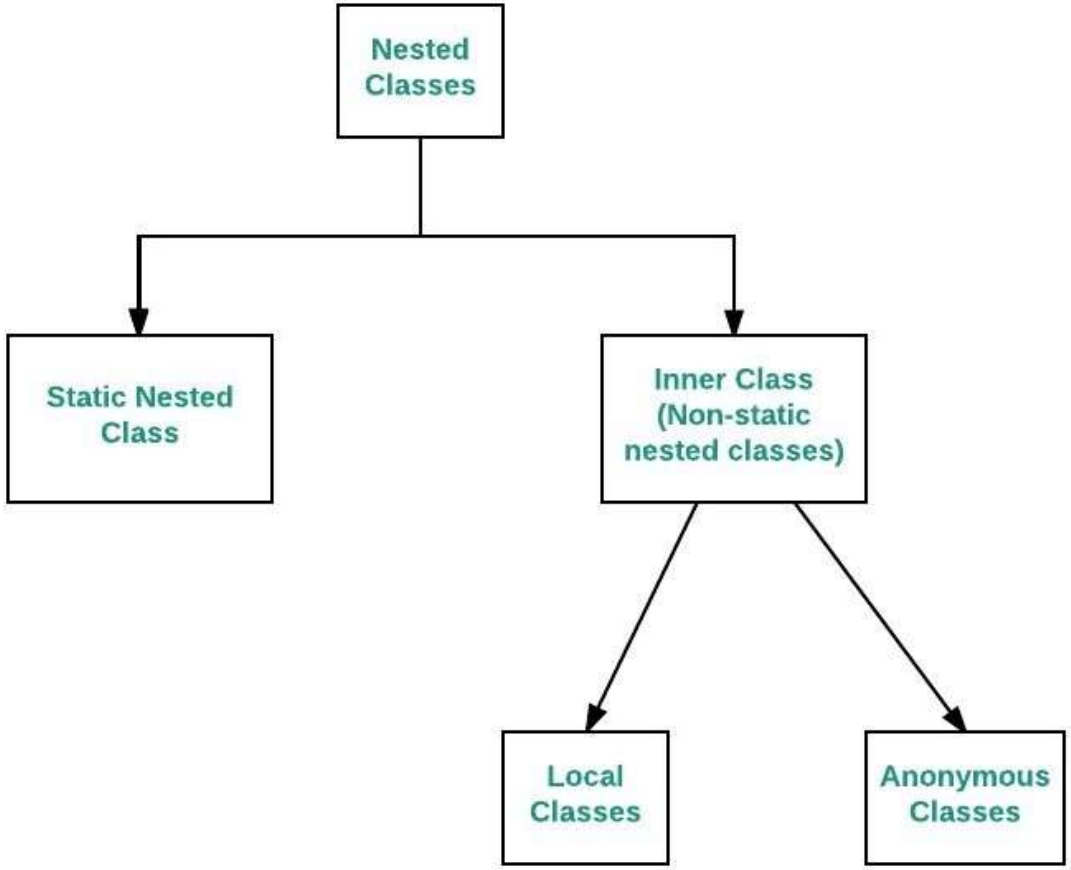

public
final
class
Runner
extends
Person
implements
Athlete
{}
Access modifier
Nonaccess modifier
Keyword class
Name of class
Keyword extends
Base class name
Keyword implements
Name of implemented interface
Curly braces
Optional
Optional
Compulsory
Compulsory
Optional
Optional
Optional
Optional
Compulsory

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 Inner

- 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 inner class ή αλλιώς 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

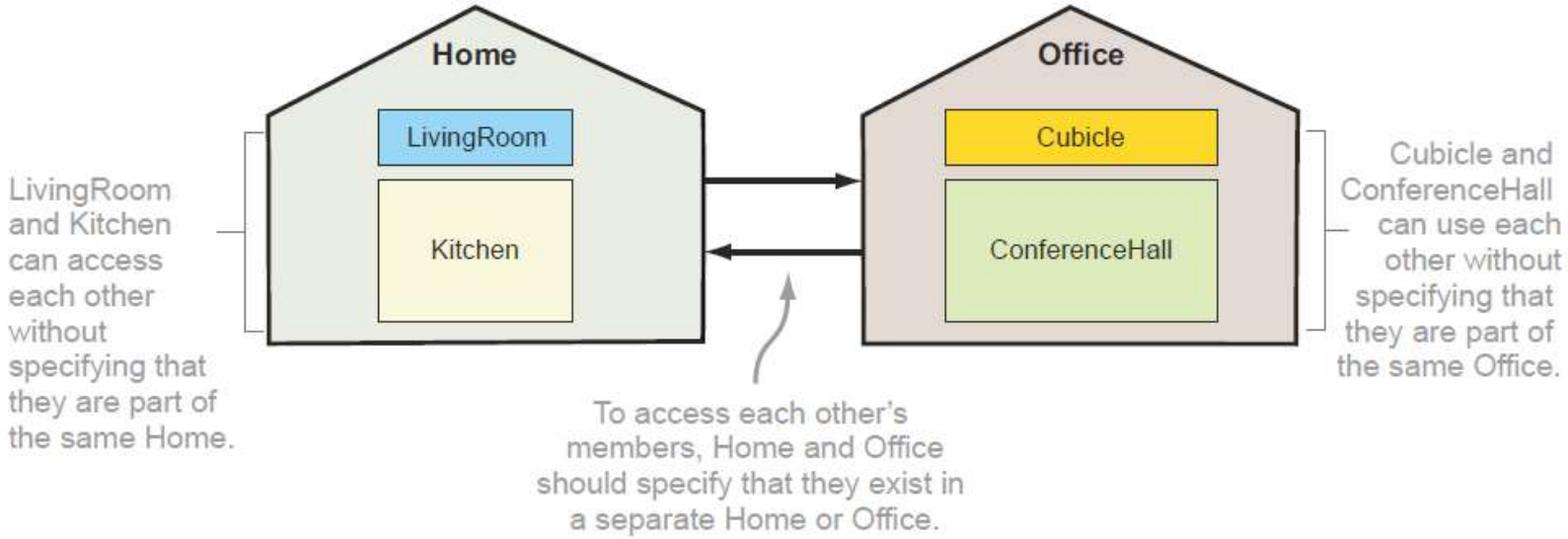
Fully qualified names

- 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

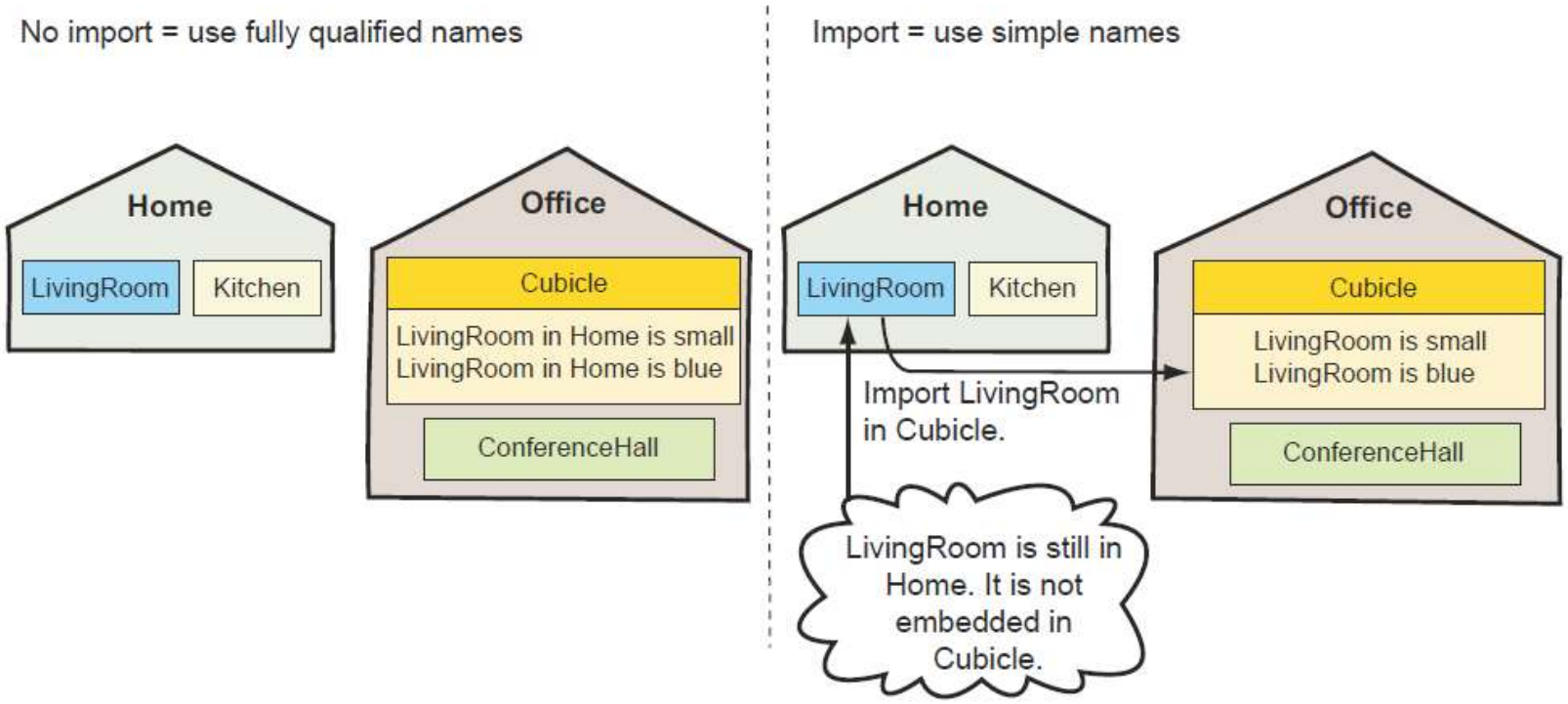
Import statements

- 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

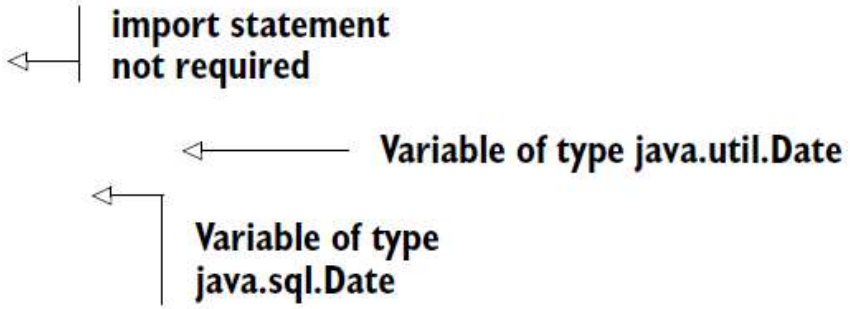
- Χρησιμοποιώντας τον ειδικό χαρακτήρα * μπορούμε να κάνουμε import σε όλες τις τάξεις και τα interfaces εντός του εν λόγω πακέτου
- Ωστόσο, η χρήση του * δεν υποδηλώνει ότι έτσι κάνουμε import και σε όλες τις τάξεις ή/και τα interfaces που βρίσκονται εντός των όποιων εμπλεκόμενων sub-packages

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;
}
```



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. 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

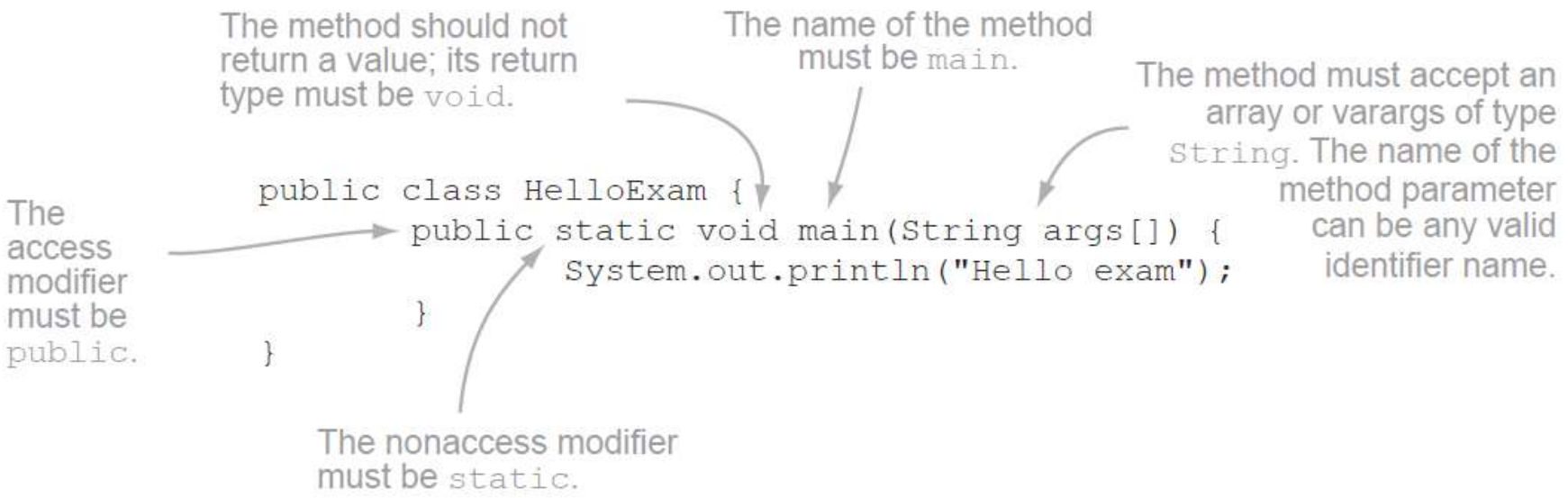
Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		
		static final int MAX_WIDTH = 999;
		
		static final int GET_THE_CPU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

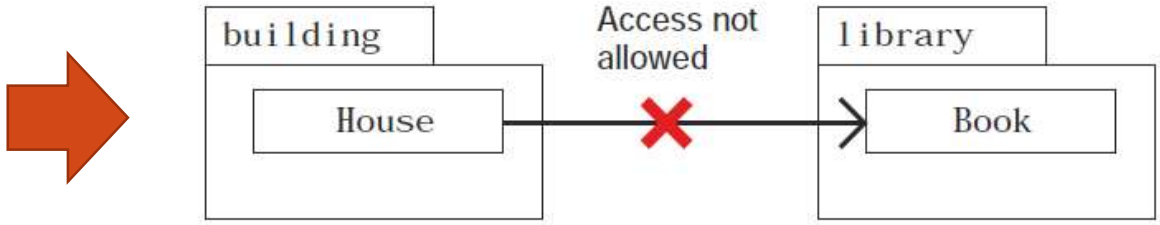
	private	default	protected	public
Class	No	Yes	No	Yes
Nested Class	Yes	Yes	Yes	Yes
Constructor	Yes	Yes	Yes	Yes
Method	Yes	Yes	Yes	Yes
Field	Yes	Yes	Yes	Yes

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

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++