**HOME (μπορουν να τοποθετηθουν σε πλαισια, όχι σαν κατεβατο κειμενο)**

*The fancy way to learn how to code.*

What is the purpose of this website?

FunC aims to help people make their first steps into the programming community. It provides basic knowledge on various major programming languages. It’s up to the user to decide which one to begin with.

How do the lessons work?

Every language has a list of topics that describe a phenomenon in said language. Each topic consists of the theoretical - the use, the grammar and syntax - and the practical – examples – parts. There is also a testing part, provided only to the subscribers of the page. After the user has become familiar with the phenomenon, they can move one to the next one.

Why subscribe and create a profile?

When logged in your FunC profile, you have a series of advantages. First of all, you have access to personal tests in each topic of every lesson, for evaluation of your skills and further training. Also, you know at any point how far along are you in the course, and your progress is saved, so that you can continue at any moment from any device.

(Maybe premium, to be decided)

Who are we?

We are a team of undergraduate students, tasked to build up their own website, with a subject of our choosing. And what better way to promote your studies than sharing your knowledge. It is, indeed, amateur work, so feel free to give us your feedback with comments and reactions on our lessons, to help us help you get the best possible experience while learning how to code.

**Java Course**

Introduction

A few words about Java

Java is a high level, modern programming language designed in the early 1990s by Sun Microsystems, and currently owned by Oracle. It currently is one of the most wide-spread programming languages.

Java is an object-oriented languages, which means that it uses the terms of classes and objects. It is also portable, so you only have to write the program once to able to run it on a number of different platforms.

Why use Java?

* Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
* It is one of the most popular programming language in the world
* It is easy to learn and simple to use
* It is open-source and free
* It is secure, fast and powerful
* It has a huge community support (tens of millions of developers)
* Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
* As Java is close to [C++](https://www.w3schools.com/cpp/default.asp) and [C#](https://www.w3schools.com/cs/default.asp), it makes it easy for programmers to switch to Java or vice versa

Main

//In Java, every line of code that can actually run needs to be inside a Class.

In Java, each application has an entry point or a starting point which is a method called Main.

To run our program the main method must be identical to this signature:

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

Every single word in this signature has a meaning

-public: Everyone can access it.

-static: Method can be run without creating an instance of the class containing the main method.

-void: Method doesn’t return any value.

-main: The name of the method.

*Tip:* In some coding environments, if you type psvm it replaces itself with the signature of the main method. (αν γινεται σε συννεφακι ή κατι τετοιο)

When you open a coding environment, it is very possible that the following code will be written

*public class MyClass {*

*public static void main(String[] args) {*

*System.out.println("Hello World");*

*}*

*}*

If you run the program, the first instruction to be executed will be the first line inside the main, so the console will print the message *Hello world.*

Print Instructions

print() as a method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String.

//This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

Various print() methods are included in the *java.io.\** library:

* *void print(boolean b)* – Prints a boolean value.
* *void print(char c)* – Prints a character.
* *void print(char[] s)* – Prints an array of characters.
* *void print(double d)* – Prints a double-precision floating-point number.
* *void print(float f)* – Prints a floating-point number.
* *void print(int i)* – Prints an integer.
* *void print(long l)* – Prints a long integer.
* *void print(Object obj)* – Prints an object.
* *void print(String s)* – Prints a string.

For example the program:

|  |
| --- |
| *import java.io.\*;*  *class GFG {*  *public static void main(String[] args)*  *{*  *System.out.print("phrase 1");*    *System.out.print("phrase 2");*  *}*  *}* |

Will have an output in the console:

*phrase 1phrase 2*

How do we fix that?

We use the method *println()* instead.

The difference:

* print() method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.
* println() method prints the text on the console and the cursor remains at the start of the next line at the console. The next printing takes place from next line.

For example the program:

|  |
| --- |
| *import java.io.\*;*  *class GFG {*  *public static void main(String[] args)*  *{*  *System.out.println("phrase 1");*    *System.out.println("phrase 2");*  *}*  *}* |

Will have an output in the console:

*phrase 1*

*phrase 2*

Comments

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

* Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

For example:

// This is a comment

System.out.println("Hello World");

The console will only print:

*Hello world*

* Multi-line comments start with /\* and ends with \*/.

Any text between /\* and \*/ will be ignored by Java.

For example:

/\* The code below will print the words Hello World

to the screen, and it is amazing \*/

System.out.println("Hello World");

The console will only print:

*Hello world*

* Documentation Comments  
    
  Documentation comments are special comments that have the appearance of multi-line comments, with the difference being that they generate external documentation of your source code. These begin with a forward slash followed by two asterisks, and end with an asterisk followed by a forward slash.

Example:

*/\*\* This is a documentation comment \*/*

Example:

*/\*\* This is also a  
documentation comment \*/*

Javadoc is a tool which comes with JDK and it is used for generating Java code documentation in HTML format from Java source code which has required documentation in a predefined format.  
  
When a documentation comment begins with more than two asterisks, Javadoc assumes that you want to create a "box" around the comment in the source code. It simply ignores the extra asterisks.

For example:

*/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  
  
This is the start of a method  
  
\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/*

*System.out.print(“Hello world”);*

The console will only print:

*Hello world*

Variables & Types

Variables

Variables store data for processing.  
A variable is given a name (or identifier), such as area, age, height, and the like. The name uniquely identifies each variable, assigning a value to the variable and retrieving the value stored.

For example:

*int number = 5;*

In the example above, *number* is the name of the variable which is given the value *5*.  
  
Types

Variables have types. The most important examples:

* int: for integers (whole numbers) such as 123 and -456
* double: for floating-point or real numbers with optional decimal points and fractional parts in fixed or scientific notations, such as 3.1416, -55.66.
* String: for texts such as "Hello" or "Good Morning!". Text strings are enclosed within double quotes.
* boolean : has either the value true or false

If-Loops

An **if** statement consists of a Boolean expression followed by one or more statements.

Syntax:

*if(Boolean\_expression) {*

*// Statements will execute if the Boolean expression is true*

*}*

If the Boolean expression evaluates to true then the block of code inside the if statement will be executed. If not, the first set of code after the end of the if statement (after the closing curly brace) will be executed.

For example the code:

*int x = 10;*

*if( x < 20 ) {*

*System.out.print("This is an if statement");*

*}*

*}*

The console will print:

*This is an if statement*

But if we set the value of x to equal 30:

*int x = 30;*

*if( x < 20 ) {*

*System.out.print("This is an if statement");*

*}*

*}*

The console will print nothing

Though there are more intricate if-statements:

* An **if** statement can be followed by an optional **else** statement, which executes when the Boolean expression is false.

For Example:

*public class Test {*

*public static void main(String args[]) {*

*int x = 30;*

*if( x < 20 ) {*

*System.out.print("This is an if statement");*

*}else {*

*System.out.print("This is an else statement");*

*}*

*}*

*}*

Console output:

*This is an else statement*

* It is always legal to nest if-else statements which means you can use one if or else if statement inside another if or else if statement.

For example:

*int x = 30;*

*int y = 10;*

*if( x == 30 ) {*

*if( y == 10 ) {*

*System.out.print("X = 30 and Y = 10");*

*}*

*}*

Console output:

*X = 30 and Y = 10*

Switch Case

A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case, and the variable being switched on is checked for each case.

Syntax:

*switch(expression) {*

*case value :*

*// Statements*

*break; // optional*

*case value :*

*// Statements*

*break; // optional*

*// You can have any number of case statements.*

*default : // Optional*

*// Statements*

*}*

The following rules apply to a **switch** statement −

* + The variable used in a switch statement can only be integers, convertable integers (byte, short, char), strings and enums.
  + You can have any number of case statements within a switch. Each case is followed by the value to be compared to and a colon.
  + The value for a case must be the same data type as the variable in the switch and it must be a constant or a literal.
  + When the variable being switched on is equal to a case, the statements following that case will execute until a *break* statement is reached.
  + When a *break* statement is reached, the switch terminates, and the flow of control jumps to the next line following the switch statement.
  + Not every case needs to contain a break. If no break appears, the flow of control will *fall through*to subsequent cases until a break is reached.
  + A *switch* statement can have an optional default case, which must appear at the end of the switch. The default case can be used for performing a task when none of the cases is true. No break is needed in the default case.

For example:

*char grade = 'C';*

*switch(grade) {*

*case 'A' :*

*System.out.println("Excellent!");*

*break;*

*case 'B' :*

*case 'C' :*

*System.out.println("Well done");*

*break;*

*case 'D' :*

*System.out.println("You passed");*

*case 'F' :*

*System.out.println("Better try again");*

*break;*

*default :*

*System.out.println("Invalid grade");*

*}*

*System.out.println("Your grade is " + grade);*

Console output:

*Well done*

*Your grade is C*

While & Do-While Loops

While

A while loop statement in Java programming language repeatedly executes a target statement as long as a given condition is true.

Syntax:

*while(Boolean\_expression) {*

*// Statements*

*}*

Here, **statement(s)** may be a single statement or a block of statements. The **condition** may be any expression, and true is any non zero value.

When executing, if the *boolean\_expression* result is true, then the actions inside the loop will be executed. This will continue as long as the expression result is true.

When the condition becomes false, program control passes to the line immediately following the loop.

For example:

*int x = 10;*

*while( x < 20 ) {*

*System.out.print("value of x : " + x );*

*x++;*

*System.out.print("\n");*

*}*

Console output:

value of x : 10

value of x : 11

value of x : 12

value of x : 13

value of x : 14

value of x : 15

value of x : 16

value of x : 17

value of x : 18

value of x : 19

Tip: Always make sure that there is a part inside the loop where the value of the Boolean expression can and will be at some point altered, so that you avoid to fall into an infinity loop.

Do – while

A **do...while** loop is similar to a while loop, except that a do...while loop is guaranteed to execute at least one time.

Syntax:

*do {*

*// Statements*

*}while(Boolean\_expression);*

The Boolean expression appears at the end of the loop, so the statements in the loop is executed once before the expression is tested.

If the Boolean expression is true, the control jumps back up to do statement, and the statements in the loop execute again. This process repeats until the Boolean expression is false.

*int x = 10;*

*do {*

*System.out.print("value of x : " + x );*

*x++;*

*System.out.print("\n");*

*}while( x < 20 );*

Console output:

*value of x : 10*

*value of x : 11*

*value of x : 12*

*value of x : 13*

*value of x : 14*

*value of x : 15*

*value of x : 16*

*value of x : 17*

*value of x : 18*

*value of x : 19*

For-Loop

A **for** loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times.

A **for** loop is useful only when you know how many times a task is to be repeated.

Syntax:

*for(initialization; Boolean\_expression; update) {*

*// Statements*

*}*

Here is the flow of control in a **for** loop:

* The **initialization** step is executed first, and only once. This step allows you to declare and initialize any loop control variables and this step ends with a semi colon (;).
* Next, the **Boolean expression** is evaluated. If it is true, the body of the loop is executed. If it is false, the body of the loop will not be executed and control jumps to the next statement past the for loop.
* After the **body** of the for loop gets executed, the control jumps back up to the **update** statement. This statement allows you to update any loop control variables. This statement can be left blank with a semicolon at the end.
* The Boolean expression is now evaluated again. If it is true, the loop executes and the process repeats (body of loop, then update step, then Boolean expression). After the Boolean expression is false, the for loop terminates.

For example:

*for(int x = 10; x < 20; x = x + 1) {*

*System.out.print("value of x : " + x );*

*System.out.print("\n");*

*}*

Console output:

*value of x : 10*

*value of x : 11*

*value of x : 12*

*value of x : 13*

*value of x : 14*

*value of x : 15*

*value of x : 16*

*value of x : 17*

*value of x : 18*

*value of x : 19*

Arrays

An array is a group of like-typed variables that are referred to by a common name.

Syntax:

*type[] var-name;*

Where *type* is any datatype as mentioned in a previous course, and *var-name* is the name given to the array. Arrays can contain primitive data types as well as objects of a class.

When an array is declared, only a reference of array is created. To actually create or give memory to array, you create an array like this:

*var-name = new type [size];*

Here,  *size* specifies the number of elements in the array. That is, to use *new* to allocate an array, you must specify the type and number of elements to allocate.

For example:

*int intArray[]; //declaring array*

*intArray = new int[20]; // allocating memory to array*

Since arrays are objects in Java, we can find their length using the method *length*.

For example:

*int[] numbers = new int[]{1,3,5,7,9};*

*System.out.print(numbers.lenght);*

Console output:

*5*

The variables in the array are ordered and each have an index beginning from 0.

Multidimensional arrays are arrays of arrays with each element of the array holding the reference of other array. A multidimensional array is created by appending one set of square brackets ([]) per dimension.

For example:

*int[][] twoDArray = new int[10][20]; //a 2D array or matrix*

The twoDArray[] is an array depicted as a table with 10 rows and 20 columns.

Example of use of a 2D array:

*int arr[][] = { {2,7,9},{3,6,1},{7,4,2} };*

*for (int i=0; i< 3 ; i++)*

*{*

*for (int j=0; j < 3 ; j++)*

*System.out.print(arr[i][j] + " ");*

*System.out.println();*

*}*

Console output:

*2 7 9*

*3 6 1*

*7 4 2*

ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.

The difference between a built-in array and an ArrayList in Java, is that the size of an array cannot be modified (if you want to add or remove elements to/from an array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.

Syntax (Create an ArrayList object called **cars** that will store strings):

*import java.util.ArrayList; // import the ArrayList class*

*ArrayList<String> cars = new ArrayList<String>();*

The ArrayList class has many useful methods:

* To add elements to the ArrayList, use the add() method.

For example:

*import java.util.ArrayList;*

*public class MyClass {*

*public static void main(String[] args) {*

*ArrayList<String> cars = new ArrayList<String>();*

*cars.add("Volvo");*

*cars.add("BMW");*

*cars.add("Ford");*

*cars.add("Mazda");*

*System.out.println(cars);*

*}*

*}*

* To access an element in the ArrayList, use the get() method and refer to the index number.

For example:

*cars.get(0);*

Console output:

*Volvo*

* To modify an element, use the set() method and refer to the index number.

For example:

*cars.set(0, "Opel");*

* To remove an element, use the remove() method and refer to the index number.

For example:

*cars.remove(0);*

* To remove all the elements in the ArrayList, use the clear() method.

For example:

*cars.clear();*

* To find out how many elements an ArrayList have, use the size method.

For example:

cars.size();

For-Each

For-each is another array traversing technique like for loop, while loop, do-while loop. It’s commonly used to iterate over an array or a Collections class (eg, ArrayList)

Syntax:

*for (type var : array)*

*{*

*statements using var;*

*}*

* It starts with the keyword for like a normal for-loop.
* Instead of declaring and initializing a loop counter variable, you declare a variable that is the same type as the base type of the array, followed by a colon, which is then followed by the array name.
* In the loop body, you can use the loop variable you created rather than using an indexed array element.

For example:

*int[] marks = { 2, 3, 1, 4, 5 };*

*int maxSoFar = numbers[0];*

*for (int num : numbers)*

*{*

*if (num > maxSoFar)*

*{*

*maxSoFar = num;*

*}*

*}*

*System.out.println("The maximum number is " + maxSoFar);*

Console output:

*The maximum number is 5*

HashMap

HashMap stores the data in (Key, Value) pairs. To access a value one must know its key. HashMap is known as HashMap because it uses a technique called Hashing. Hashing is a technique of converting a large String to small String that represents the same String. A shorter value helps in indexing and faster searches.

HashMap is a part of java.util package.

For example (Create a HashMap object called capitalCities that will store String **keys** and String **values)**:

*import java.util.HashMap; // import the HashMap class*

*HashMap<String, String> capitalCities = new HashMap<String, String>();*

Important info:

* HashMap doesn’t allow duplicate keys but allows duplicate values. That means A single key can’t contain more than 1 value but more than 1 key can contain a single value.
* HashMap allows null key also but only once and multiple null values.

The HashMap class has many useful methods:

* To add items to it, use the put() method:

For example:

*capitalCities.put("England", "London");*

*capitalCities.put("Germany", "Berlin");*

* To access a value in the HashMap, use the get() method and refer to its key:

For example:

*capitalCities.get("England");*

* To remove an item, use the remove() method and refer to the key:

For example:

*capitalCities.remove(“England”);*

* To find out how many items there are, use the size method:

For example:

*capitalCities.size();*

To loop through the items of a HashMap we use a **for-each** loop.

For example:

// Print keys

for (String i : capitalCities.keySet()) {

System.out.println(i);

}

Tip: Use the keySet() method if you only want the keys, and use the values() method if you only want the values:

Classes and Objects

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

A class is a user defined blueprint or prototype from which objects are created.  It represents the set of properties or methods that are common to all objects of one type.

For example:

In real life, a dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating.

If we were to create a class to describe dogs, the states would be used as the fields of the class, and the behaviors as methods.

*public class Dog {*

*String breed;*

*int age;*

*String color;*

*void barking() {*

*}*

*void hungry() {*

*}*

*void sleeping() {*

*}*

*}*

An object is a basic unit of Object Oriented Programming and represents the real life entities.  It’s like a variable, only more complex, with more values of different types. A typical Java program creates many objects, which interact by invoking methods.

For example, if we have a class Dog, an object – instance of that class would be of terrier breed, 5 age, white color. If we were to see those attributes as a package, which is the point of an object, we would have to assign a name to it, say Billy, as we would do to a simple variable.

The initialization of a new object must mention the class it belongs to as Datatype before the name, followed by the word *new* and a constructor.

For example:

*Dog Billy = new Dog();*

Methods

A **method** is a block of code which only runs when it is called. You can pass data, known as parameters, into a method. It is used to perform certain actions, and they are also known as **functions**.

Methods allow us to **reuse** the code without retyping it each time.

In general, method declarations has six components :

* **Modifier**-: Defines **access type** of the method i.e. from where it can be accessed in your application. In Java, there 4 type of the access specifiers.
  + public: accessible in all class in your application.
  + protected: accessible within the class in which it is defined and in its **subclass(es)**
  + private: accessible only within the class in which it is defined.
  + default (declared/defined without using any modifier) : accessible within same class and package within which its class is defined.
* **The return type** : The data type of the value returned by the method or void if does not return a value.
* **Method Name** : the rules for field names apply to method names as well, but the convention is a little different.
* **Parameter list**: Comma separated list of the input parameters are defined, preceded with their data type, within the enclosed parenthesis. If there are no parameters, you must use empty parentheses ().
* **Exception list**: The exceptions you expect by the method can throw, you can specify these exception(s). (usually missing)
* **Method body**: it is enclosed between braces. The code you need to be executed to perform your intended operations.

For example:

public int addTwoInt(int a, int b){

        // adding two integer value.

        sum = a + b;

        //returning summation of two values.

        return sum;

    }

For given a = 1 and b = 2, the sum value that the method will return will equal 3.

So if in our program, we assign the method addTwoInt on a new variable, say int summary:

*int a = 1;*

*int b = 2;*

*int summary = addTwoInt(a,b);*

The variable summary will have the outcome of the method, which is 3.

Tip: The datatype of the variable must be the same as the datatype of the outcome of the method.

This way of method use is only acceptable when used inside the class where the method belongs.

To use the method in an other class, we need to create an object od the method’s class and call it via the object.

For example:

|  |
| --- |
| *class Addition {*  *int sum = 0;*    *public int addTwoInt(int a, int b){*  *sum = a + b;*  *return sum;*  *}*  *}*    *class Action {*  *public static void main (String[] args) {*    *// creating an instance of Addition class*  *Addition add = new Addition();*    *// calling addTwoInt() method*  *int s = add.addTwoInt(1,2);*  *System.out.println("Sum of two integer values :"+ s);*    *}*  *}* |

Console output:

*Sum of two integer values :3*

Constructor

A constructor in Java is a **special method** that is used to initialize objects. The constructor is called when an object of a class is created. It can be used to set initial values for object attributes.

Every class has at least constructor, but it can have more. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

Each time a new object is created, at least one constructor will be invoked.

The main rules of constructors are that they should have the same name as the class and must always have public access.

For example:

*public class Dog {*

*public Dog() {*

*//actions*

*}*

*public Dog(String name) {*

*// This constructor has one parameter, name.*

*//actions*

*}*

*}*

Tip: The constructors of a specific class all have the same name. the only way to distinguish them is by the number an types of their parameters.

The program will determine which constructor to call according to the parameters given, or lack of them.

For example:

*Dog unnamed = new Dog();*

Will invoke the parameter-less constructor:

*public Dog() {*

*//actions*

*}*

While:

*Dog named = new Dog(Billy);*

Will invoke the constructor:

*public Dog(Billy) {*

*// This constructor has one parameter, name.*

*//actions*

*}*

Setters & Getters

Encapsulation

The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users. To achieve this, you must:

* declare class variables/attributes as private
* provide public **get** and **set** methods to access and update the value of a private variable

Private variables can only be accessed within the same class (an outside class has no access to it). However, it is possible to access them if we provide public **get** and **set** methods.

The main idea is that we use a (void) setter to assign (set) a value to a private field, and access said value via a getter method that returns it to the outside class.

Syntax for both is that they start with either get or set, followed by the name of the variable, with the first letter in upper case:

For example:

public class Person {

private String name; // private = restricted access

// Getter

public String getName() {

return name;

}

// Setter

public void setName(String newName) {

this.name = newName;

}

}

Example explained:

The get method returns the value of the variable name.

The set method takes a parameter (newName) and assigns it to the name variable. The this keyword is used to refer to the current object.

Packages

A package in Java is used to group related classes. Think of it as a folder in a file directory. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

* Built-in Packages (packages from the Java API)
* User-defined Packages (create your own packages)

The Java API is a library of prewritten classes, that are free to use, included in the Java Development Environment.

The library is divided into packages and classes. Meaning you can either import a single class (along with its methods and attributes), or a whole package that contain all the classes that belong to the specified package.

To use a class or a package from the library, you need to use the import keyword.

Syntax:

*import package.name.Class; // Import a single class*

*import package.name.\*; // Import the whole package*

To create your own package, you need to understand that Java uses a file system directory to store them. Just like folders on your computer:

For example:

└── root

└── mypack

└── MyPackageClass.java

To create a package, we just have to use the package keyword.

For example:

*package mypack;*

*class MyPackageClass {*

*public static void main(String[] args) {*

*System.out.println("This is my package!");*

*}*

*}*

Now the class *MyPackageClass* belongs to the *package* mypack. We can add as mamy classes as we want to our packages.

Tip: When classes, fields or methods have the default access, it means they can be accessed by every class that belongs in the same package as them.

Websites for copyrights:

<https://www.tutorialspoint.com/>

[www.w3schools.com/](http://www.w3schools.com/)