Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

<https://docs.oracle.com/en/java/javase/17/docs/api/allclasses-index.html>

here are the classes.

Tutorials:

<https://docs.oracle.com/javase/tutorial/java/>

Bytecodes are executed by the JVM and the bytecode instructions are translated into native machine code by the \_\_\_JRE\_\_\_\_\_\_\_ Java Runtime Environment.

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

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación, Teams

Descripción generada automáticamente Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente Interfaz de usuario gráfica, Sitio web

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

Diagrama

Descripción generada automáticamente

Texto

Descripción generada automáticamente

<https://www.coursera.org/learn/java-introduction/ungradedLab/a6oUZ/experience-the-lab-environment/lab?path=%2F>

Tabla

Descripción generada automáticamente

Every line of code that runs in Java must be inside a class. In our example, we named the class **Main**. A class should always start with an uppercase first letter.

Java is case-sensitive: "MyClass" and "myclass" has different meaning.

The name of the java file **must match** the class name.

Variables are containers for storing data values.

In Java, there are different **types** of variables, for example:

* String - stores text, such as "Hello". String values are surrounded by double quotes
* int - stores integers (whole numbers), without decimals, such as 123 or -123
* float - stores floating point numbers, with decimals, such as 19.99 or -19.99
* char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
* boolean - stores values with two states: true or false
* Texto

  Descripción generada automáticamente

Data types are divided into two groups:

* Primitive data types - includes byte, short, int, long, float, double, boolean and char
* Non-primitive data types - such as [String](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp) (you will learn more about these in a later chapter)
* Numbers
* Primitive number types are divided into two groups:
* **Integer types** stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are byte, short, int and long. Which type you should use, depends on the numeric value.
* **Floating point types** represents numbers with a fractional part, containing one or more decimals. There are two types: float and double.
* Even though there are many numeric types in Java, the most used for numbers are int (for whole numbers) and double (for floating point numbers). However, we will describe them all as you continue to read.

## Non-Primitive Data Types

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **primitive** and **non-primitive** data types are:

* Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
* Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
* A primitive type has always a value, while non-primitive types can be null.
* A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
* The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Tabla

Descripción generada automáticamenteInterfaz de usuario gráfica, Aplicación

Descripción generada automáticamenteInterfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

<https://www.coursera.org/degrees/maestria-ingenieria-software-uniandes/admisiones#fechas-de-inscripcin>

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

## Parameters and Arguments

Information can be passed to methods as parameter. Parameters act as variables inside the method.

Parameters are specified after the method name, inside the parentheses. You can add as many parameters as you want, just separate them with a comma.

When a **parameter** is passed to the method, it is called an **argument**. So, from the example above: fname is a **parameter**, while Liam, Jenny and Anja are **arguments**.

Note that when you are working with multiple parameters, the method call must have the same number of arguments as there are parameters, and the arguments must be passed in the same order.

Java - What is OOP?

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

**Tip:** The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

So, a class is a template for objects, and an object is an instance of a class.

A Class is like an object constructor, or a "blueprint" for creating objects.

When the individual objects are created, they inherit all the variables and methods from the class.

## Java Class Attributes

In the previous chapter, we used the term "variable" for x in the example (as shown below). It is actually an **attribute** of the class. Or you could say that class attributes are variables within a class:

### **Example**

Create a class called "Main" with two attributes: x and y:

public class Main {

int x = 5;

int y = 3;

}

Another term for class attributes is **fields**.