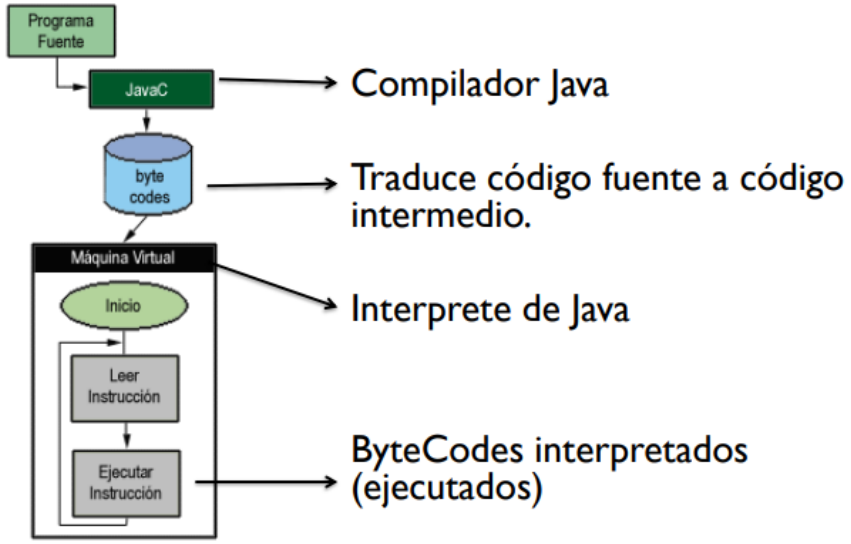
Resumen Java

Cómo funciona Java

1. Se escribe el **código fuente**.
2. Con el **compilador** se convierte el código fuente en **Bytecode.**
3. La JVM ejecuta el **código objeto** leyendo el Bytecode.



Tipos de datos

boolean	int
double	String

Inicialización de variables

```
int variable = valor;
String variable = valor;
// Si ya la he inicializado (L1, únicamente asigno valor)
variable = valor2;

int var1, var2;
```

!! Siempre finalizar instrucciones con *punto y coma* (;). De no hacerlo reventará.

Operadores aritméticos

– Simple arithmetic

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 - 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2 7 / 2	0.25 3
%	Resto de la división entera	20 % 7	6

– Combined arithmetic

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
*=	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

El operador **suma** también se puede usar en Strings para la concatenación:

```
// Declaración de variable y asignación de valor en 1 instrucción.
String str1 = "Valor 1";

// Declaración de variables
String str2, strresul;

// Asignación de valor
str2 = "Valor 2";

// Concateno str1, "//" y str2 en la variable strresul
strresul = str1 + " // "+ str2;

System.out.println(strresul);
```

Operadores lógicos

– Simple arithmetic

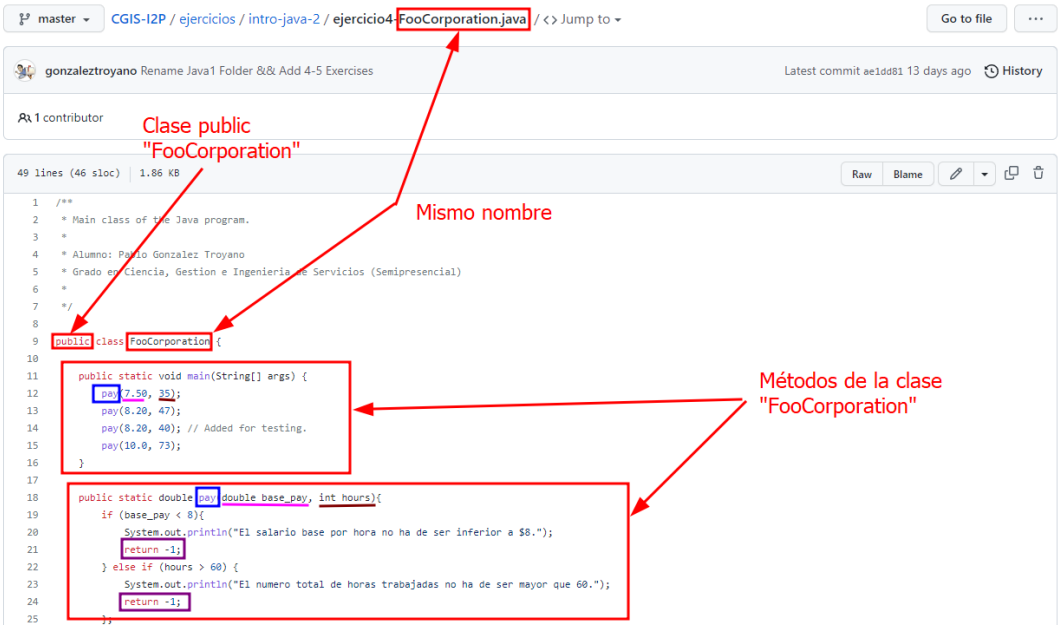
Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
-	operador unario de cambio de signo	-4	-4
+	Suma	2.5 + 7.1	9.6
-	Resta	235.6 - 103.5	132.1
*	Producto	1.2 * 1.1	1.32
/	División (tanto entera como real)	0.050 / 0.2 7 / 2	0.25 3
%	Resto de la división entera	20 % 7	6

– Combined arithmetic

Operador	Descripción	Ejemplo de expresión	Resultado del ejemplo
+=	Suma combinada	a+=b	a=a+b
-=	Resta combinada	a-=b	a=a-b
*=	Producto combinado	a*=b	a=a*b
/=	División combinada	a/=b	a=a/b
%=	Resto combinado	a%=b	a=a%b

Clases, métodos y esas cosas

- Every Java program must contains at least one Class
- A Java file could contain several classes, but just one of them being Public
- File name must be that of the Public class
- The main method
  - Every program needs a main method
  - It is the starting point of the program
  - Always the same heading
  - Always invoked when the program is run



Casting de datos

```
int a = 2; // a = 2
double a = 2; // a = 2.0 Implicit

int a = 18.7; // ERROR
```

```java
int a = (int)18,7; // a = 18 # FUERZA INT

double a = 2/3; // a = 0.0
double a = (double)2/3; // a = 0.666 ...

double d = 5.25;
int i = (int) d; // d = 5 (Explicit) DOWNCAST

int d = 5;
double i = d; // i = 5.0 (Implicit) UPCAST
```

Condicionales y bucles

Condicional *if*

```java
if (base_pay < 8){
            System.out.println("El salario base por hora no ha de ser
inferior a $8.");
            return -1;
        } else if (hours > 60) {
            System.out.println("El numero total de horas trabajadas
no ha de ser mayor que 60.");
            return -1;
        } else {
            System.out.println("Fallback")
        };
```

Loop *while*

```java
int i = 0;
while (i < 3) {
  System.out.println(“Rule #“ + i);
  i = i+1;
};
```

Loop for

```java
for (initialization; condition; update){
  STATEMENTS;
};

for (int i = 0; i < 3; i = i + 1){
  System.out.println(“Rule #” + i);
};
```

do -while

At least once

```
do {
  STATEMENTS;
} while (condicion);
```

!! ***break*** terminates a for or while loop !! ***continue*** skips the current iteration of a loop and proceeds directly to the next iteration

Arrays

Definition

```
int [] values; // array of int values
int [][] values; // array of array of int values
```

Array of given size:

```
int[] values = new int[5];
// using a variable
int size = 12;
int[] values = new int[size];
```

Curly braces can be used to initialize an array. It can **ONLY** be used when you declarethe variable.

```
int[] values = {12, 24, -23, 47};
```

Acceder a elementos del array y añadir nuevos:

```
int[] values = {12, 24, -23, 47};
values[3] = 18; // {12, 24, -23, 18}
```

Each array has a length variable built-in that contains the length of the array

```
int[] values2 = {1,2,3,4,5}
int size2 = values2.length; // size = 5

// method to print an Array
public static void printArray(int[] array) {
  for (int i = 0; i < array.length; i++) {
  System.out.print(array[i] + " ");
  };
};
```

Array utils

Array Utils

– Arrays copy → `System.arrayCopy(...)`

```
public static void arraycopy(Object src, int srcPos,
                             Object dest, int destPos, int length)
```

```
class ArrayCopyDemo {
       public static void main(String[] args) {
              char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
                                  'i', 'n', 'a', 't', 'e', 'd' };
              char[] copyTo = new char[7];

              System.arraycopy(copyFrom, 2, copyTo, 0, 7);

              System.out.println(Arrays.toString(copyTo));
              // output: [c, a, f, f, e, i, n]
              System.out.println(new String(copyTo));
              // output: caffein
       }
}
```



```
int[] valores = new int[5];

for (int i=0; i < valores.length; i++) {
  valores[i] = i;
  int y = valores[i] * valores[i];
  System.out.println(y);
};


int[] valores = new int[5];
int i = 0;

while (i < valores.length) {
  valores[i] = i;
  int y = valores[i] * valores[i];
  System.out.println(y);
  i++;
};
```

Array Utils (III)

- java.util.Arrays

```
int binarySearch(tipo[] a, tipo key)
// returns the position of 'key' in 'a' array

boolean equals(tipo[] a, tipo[] a2)
// yields true if 'a' and 'a2' contain the same values

void fill(tipo[] a, tipo val)
// set every position of array 'a' to 'val'

void sort(tipo[] a)
// orders 'a' array (ASC)
```

Strings

- int length()
  - Length of the string
- int indexOf(String cad)
  - This method returns the index within this string of the first occurrence of the specified character or -1, if the character does not occur.
- char charAt(int ind)
  - This method returns the character located at the String's specified index. The string indexes start from zero
- Boolean equals(String cad)
  - This method compares this string to the specified object. The result is true if and only if the argument is not null and is a String object that represents the same sequence of characters as this object
- String replaceAll(String oldCad, String newCad)
  - This method replaces each substring of this string that matches the given regular expression with the given replacement
- String toLowerCase()
- String toUpperCase()
- String substring(int indexInicial, int indexFinal)
  - This method has two variants and returns a new string that is a substring of this string.
  - The substring begins with the character at the specified index and extends to
    - the end of this string or
    - up to endIndex – 1, if the second argument is given

V: Clases y objetos

Para crear un objeto:

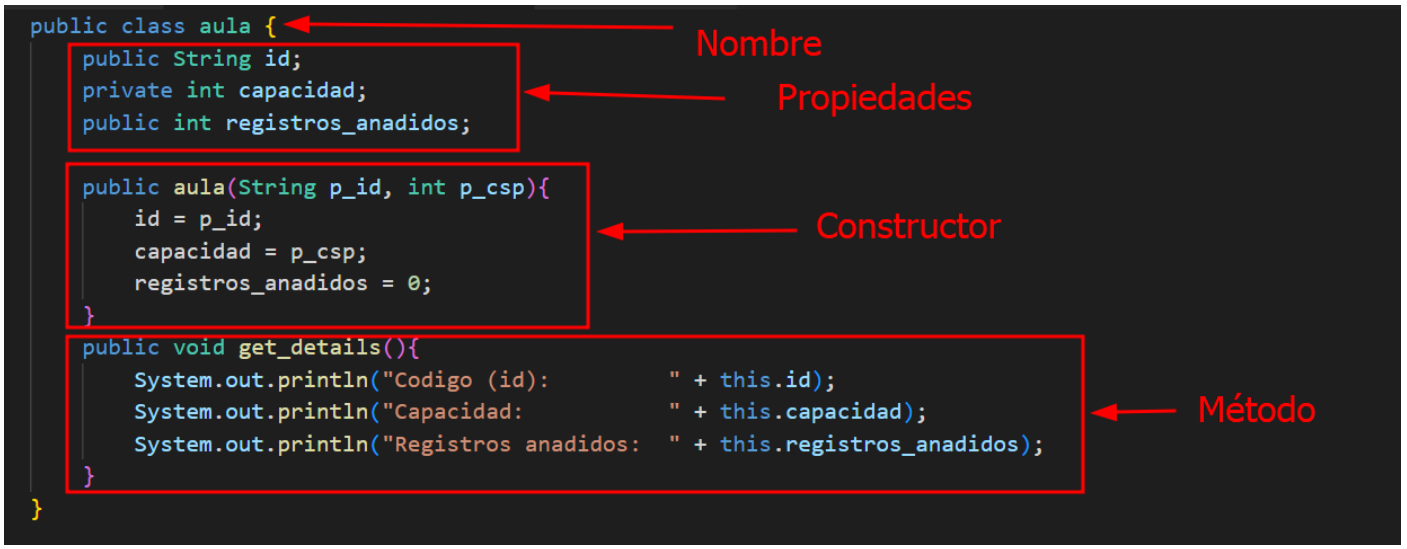
public class NOMBRE

Por convención, a continuación de la apertura del corchete para la definición de la clase (que estará en su archivo individual) se ponen las propiedades. Estas propiedades pueden ser de tipo public (pueden ser accedidas desde cualquier punto de programa), o private (solo podrán ser accedidas desde dentro del objeto).

Constructor es lo que se ejecuta cuando creamos un nuevo objeto de la clase. Pueden existir varios constructores, para permitir la creación usando diferentes variables de entrada.

Los objetos de la clase tienen una serie de métodos (funciones) definidas. Para crear un método:

```
(public|private) $RETURN_TYPE $NAME($IN_args+) {
        ---DESARROLLO de la clase---
        return XX (si no es void)
}
```



Para acceder desde fuera a una propiedad ⇒ OBJECT.PROPERTY

Para ejecutar desde fuera un método ⇒ OBJECT.METHOD();

Primitives vs Reference

- Primitive types: modifications over the formal parameter does not affect the actual one
- Reference types: modifications over the location of the formal parameter does not affect the actual one but … altering the object referenced by the formal parameter does affect the one referenced by the actual one (since there both the same)

*** Primitive types: value is copied

*** Reference types: location is copied. The object is not copied (a new alias is created)

- =
  - Copy the content from the variable on the right to the one on the left
    - Primitive types: actual value is copied
    - Reference types: object location is copied
      - The object is not duplicated but a new alias to access it is created
- ==
  - Compares the content of both variables
    - Primitie values: actual values are compared
    - Reference types: locations are compared
      - Objects' state is not compared

Imagine
– Following directions to a house
– Moving the furniture around
Analogous to
– Following the reference to an object
– Changing fields in the object

- [] | .
  - Navigates the reference until the referenced object
  - Can update the object state (FIELDs' values) but not the reference

Static Methods and Types

STATIC:

- Is not unique for each instance
- Is defined for the class declaration
- Commonly used for Carrying out COMMON operations and/or data storage that apply to every object of the class
- We refer to them:
  - In the class in which they are declared: using their name
  - In another class: preceding their name with that of the class in which they were declared

```
public static void main(String[] arguments) { }
```

VI: Access control, Class Scope, Packages, Java API

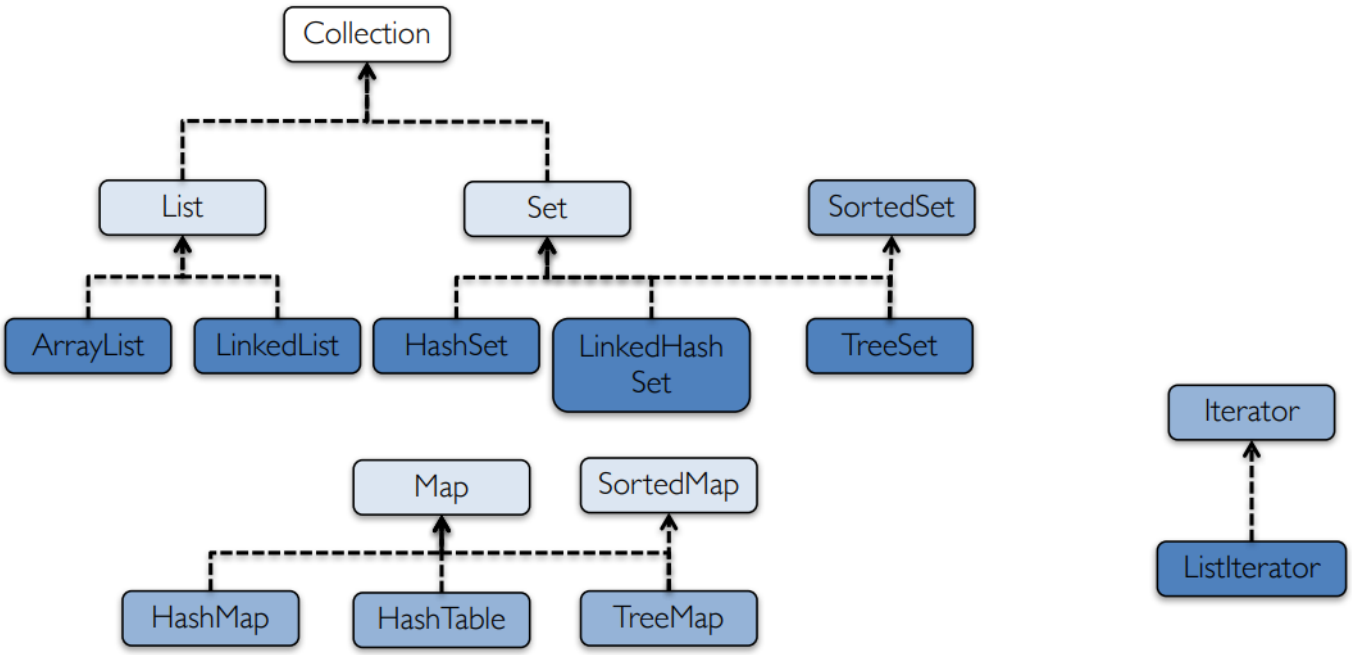
Hacer propiedades private nos ayuda a evitar errores a la hora de editar y acceder a estas.

Así, en vez de igualar (object.variable = XX) llamaremos a un método que editará esta propiedad, normalmente comprobando que la edición puede realizarse de forma segura.

Debemos tener en cuenta que dentro de los métodos las variables declaradas solo son accesibles desde este. Para referirnos a propiedades del objeto

Las **interfaces** son conjuntos de clases que comparten métodos.

VII: Colecciones



Métodos comunes para el manejo:

• add (Object x) • remove (Object x) • contains (Object x)	• size (Object x) • toArray (Object x) • Iterator (Object x)

Lists

- Redimensionable ordered list of elements
- Repatead values are allowed
- ArrayList
- LinkedList:
  - Keep insert order
  - Hampers performance

- add(Object o)
- add(int indice, Object o)
- get(int indice)
- remove(int indice)
- clear()
- indexOf(Object o)
- lastIndexOf(Object o)
- size()
- contains(Object o)

```
import java.util.ArrayList;
[...]

ArrayList<String> cadenas = new ArrayList<String>();
cadenas.add("José");
cadenas.add("Clara");

System.out.println(cadenas.size());
System.out.println(cadenas.get(0));

cadenas.set(0, "Adiós");

for (int i=0; i < cadenas.size(); i++) {
      System.out.println(cadenas.get(i));
}
```

```
for (String s : cadenas) {
      System.out.println(s);
}
```

Sets

- Like an ArrayList, but
  - Only one copy of each object // equals() | hashcode()
  - No array index

- add(Object o)
- remove(Object o)
- clear()
- isEmpty()
- iterator()
- size()

Features:

- Add objects to the set
- Remove objects from the set
- Is an object in the set?

- HashSet
  - Best performance
- LinkedInHashSet
  - Performance: worse than HashSet
  - Keep insertion order
- TreeSet
  - Sorted (lowest to highest) set of Comparable items
  - Worst performance

```
import java.util.TreeSet;
  class TreeSetExample {
   public static void main(String[] arguments) {
   TreeSet<String> cadenas = new TreeSet<String>();
   cadenas.add("Cristian");
   cadenas.add("Andrés");
   cadenas.add("Tania");
   System.out.println(cadenas.first());
   System.out.println(cadenas.last());
   System.out.println(cadenas.size());
   cadenas.remove("Tania");
   for (String s : cadenas) {
      System.out.println(s);
   }
  }
}
```

Maps

Stores a (key, value) pair of objects

- AKA 2 cols table
- Look up the key, get back the value
- No duplicate keys
- One value per key

- clear()
- containsKey(Object o)
- containsValue(Object o)
- get (Object key)
- isEmpty()
- remove(Object key)
- size()

.put()

- HashMap / Hashtable
  - Unordered (pseudo-random)
  - Best performance
  - [Hashtable: no null values]
- LinkedHashMap
  - Keep insert order
  - Performance: worse than HashMap
- TreeMap
  - Sorted (lowest to highest value)
  - Worst performance

```
import java.util.Hashtable;
import java.util.Map.Entry;
  class HashMapExample {
    public static void main(String[] arguments) {
    Hashtable<String, String> cadenas = new Hashtable<String, String>();
    cadenas.put("Álvaro", "alv@urjc.es");
    cadenas.put("Carolina", "caro@urjc.es");
    cadenas.put("Saúl", "saul@urjc.es");
    System.out.println(cadenas.size());
    cadenas.remove("Álvaro");
    System.out.println(cadenas.get("Carolina"));
    for (String s : cadenas.keySet()) {
         System.out.println(s);
    }
    for (String s : cadenas.values()) {
         System.out.println(s); }
    for (Entry<String, String> pairs : cadenas.entrySet()) {
         System.out.println(pairs);
    }
  }
}
```

RECORRER una hasthtable:

- The values in the hashtable are to be traversed by means of an Enumeration object

```
Enumeration<String> miEnumHeroes = heroes.elements();

while (miEnumHeroes.hasMoreElements()){
  System.out.println("Hero: " + miEnumHeroes.nextElement());
}
// prints the four values
```

- elements() :Returns the values in the hashtable
- hasMoreElements(): Yields true if more elements remain in the Enumeration objecT
- nextElement(): Returns the next element in the Enumeration object

- **Hashtable methods (summary)**
  - put(key, value)
    - Adds a new <key,value> pair
  - remove(key)
    - Deletes the given <key,value> pair
  - get(key)
    - Returns the value corresponding to the given key
  - containsKey(key)
    - Yields true if key is in the Hashtable
  - contains(value)
    - Yields true if value is in the Hashtable
  - size()
    - Returns the number of <key,value> pairs in the Hashtable