Curso Programación de dispositivos móviles con Android

# Programación con JAVA

## Generalidades y componentes de Java

* Se puede hacer correr el código en todos los sistemas operativos con la máquina virtual de Java.
* Versiones de JAVA:
  + JAVA Enterprise Edition: proyectos en web
  + JAVA Standard Entreprise: proyectos en aplicaciones móvil
  + JAVA Micro Edition: dispositivos a muy bajo nivel (electrodomésticos)
* JRE (Java Runtime Enviroment): máquina virtual de JAVA que compila los archivos de JAVA en el sistema operativo correspondiente.
* JDK (Java Development Kit): desarrollo de programas

**Requisitos para poder programar en JAVA**

* Editor de código
* Terminal o consola de comandos
* Combinar ambas opciones a través de un IDE (por ejemplo, Eclipse).

**Tipos de datos en JAVA**

* Tipos primitivos
  + Números enteros: byte (-128 a 127: 1 byte), short (-32768 a 32767: 2 bytes), int (rango de datos de -2 billones a 2 billones: 4 bytes), long (rango de datos muy grande: 8 bytes).
  + Números decimales: float (4 bytes), double (8 bytes).
  + Números booleanos
  + Caracteres
* Tipos objetos
  + Arrays (arreglos): almacenamiento de un conjunto de datos del mismo tipo Sintaxis: nombreVariable = new TipoDato[capacidad];  
    Acceso a dato con índice (empieza desde 0):
    - 1 dimensión: nombreVariable[índice] = valor;
    - 2 dimensiones: nombreVariable[índice1] [índice2] = valor;

**Normas de sintaxis**

* Case sensitive (mayúsculas – minúsculas)
* Permite comenzar nombres con $ y – o que éstos siempre estén entre medio
* Técnica camello
  + Clases: todas las palabras empiezan en mayúsculas (**E**jemploDeUpperCamelCase)
  + Métodos, variables y objetos: todas las iniciales en minúsculas y las letras siguientes en mayúsculas (**e**jemploDeLowerCamelCase)

**Casteo de variables** (cambio de un tipo de datos a otro, normalmente para ampliar los bits de una variable que supera el límite) 🡪 Ejemplo: int j = int(d); (la variable d era short).

\*\* Los únicos tipos de datos que no admiten cambios son los booleanos.

**Operadores aritméticos**

* Suma y concatenación de cadenas: +
* Resta: -
* División: /
* Multiplicación: \*
* Módulo: %

**Operadores de asignación:** para ahorrarnos sintaxis

* Suma y concatenación de cadenas: += (a+=b 🡪 a = a + b)
* Resta: -=
* División: /=
* Multiplicación: \*=
* Módulo: %

**Operadores para variables de control:**

* Incremento: ++
* Decremento: --

Pueden ser prefijo (++a) y postfijo (a++): afecta a la hora de realizar operaciones y visualizar los los números, ya que el prefijo muestra el incremento y el postfijo muestra el valor original de la a (se realiza después de la operación o print el incremento del valor de i).

**Operadores de comparación (devuelve booleanos):**

* Igual: ==
* Diferente: =!
* Mayor: >
* Menor: <
* Mayor o igual: >=
* Menor o igual: <=

**Operadores lógicos (devuelve booleanos):**

* AND: &&
* OR: ||
* NOT: !x

**Control de flujo:**

* **If** (condición) { instrucciones }

else { instrucciones }

* **Switch** (variable) {

Case valor1: instrucciones1 break;

Case valor2: instrucciones2 break; }

* While(condición) { instrucciones } 🡪 Ejecuta siempre que la condición se cumpla
* For (tipo\_var\_control var\_control = 0; condición = X) { instrucciones } 🡪 Igual que el while, pero con una variable de control para saber el número de repeticiones
* Foreach: for(TipoDato elemento: colección) { instrucciones }

## Programación orientada a objetos

* Los métodos son los atributos de los objetos.
* Los comportamientos son las acciones que se pueden llevar a cabo con esos objetos.
* Las clases son plantillas para generar objetos del mismo tipo (sólo descriptivas, no almacenan información).

Ejemplo de tienda de telefonía móvil:

* Clase: inventario de teléfonos
* Objeto: teléfono
* Método: id, marca, modelo, precio,..
* Comportamiento: mostrar hojas de características,…

**Sintaxis en JAVA:**

* **CLASE:** public class NombreClase {}
* **COMPORTAMIENTO:** public void nombreComportamiento {}
* **MÉTODO:** public tipo\_var\_ret nombreMetodo (tipo\_arg arg)   
  {instrucciones   
  return var\_ret; }   
  \*\* En caso de que el método no devuelva nada: void
* **OBJETOS:** Se debe de declarar e instanciar dentro del método main.
  + **Declarar el objeto**:   
    *tipoObjeto miObjeto*; 🡪 Ejemplo: *Telefono miTelefono.*
  + **Instanciar el objeto (‘activar’ el objeto, Método constructor):***miObjeto = new Objeto ();* 🡪 Ejemplo: *miTelefono = new Telefono ();*\*\* Si no está declarado en la clase, JAVA nos lo proporciona por defecto
  + **Declarar + Instanciar al mismo tiempo**:  
    *tipoObjeto miObjeto = new tipoObjeto ();   
    Telefono miTelefono = new Telefono ();*

**Utilizando un objeto:**

1. Declararlo 🡪 Telefono miTelefono;
2. Instanciarlo 🡪 miTelefono = new Telefono ();
3. Asignar atributos 🡪 miTelefono.marca = “Motorola”;
4. Realizar acciones de comportamiento 🡪 miTelefono.mostrarDatos();

**Clase MATH:**

Es una clase estática que permite acceder a métodos de forma especial. Tiene definidos métodos estáticos, y accede de la siguiente forma:

* NombreClase.Metodo; 🡪 Atributo PI: Math.PI;
* NombreClase.Comportamiento(); 🡪 Realizar raíz cuadrada: Math.sqrt(25);

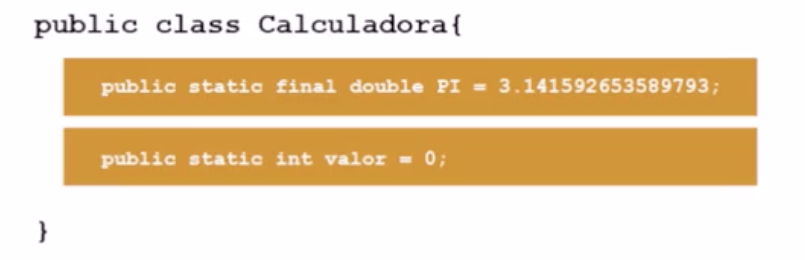
**Métodos y miembros estáticos:**

* Se define con la palabra reservada static.
* Está definido para toda la clase.
* Están disponibles para todo el programa a través del import.

*Import static com.luciagonzalez.operaciones.Calculadora.\**   
(importamos todos los miembros estáticos de esa clase para la clase dónde lo estamos importando).

* Puede ser accedido a partir del nombre de la clase y el del método, sin necesidad de definir un objeto 🡪 nombreClase.nombreMetodo

Definición de la clase y el método:



Acceso:

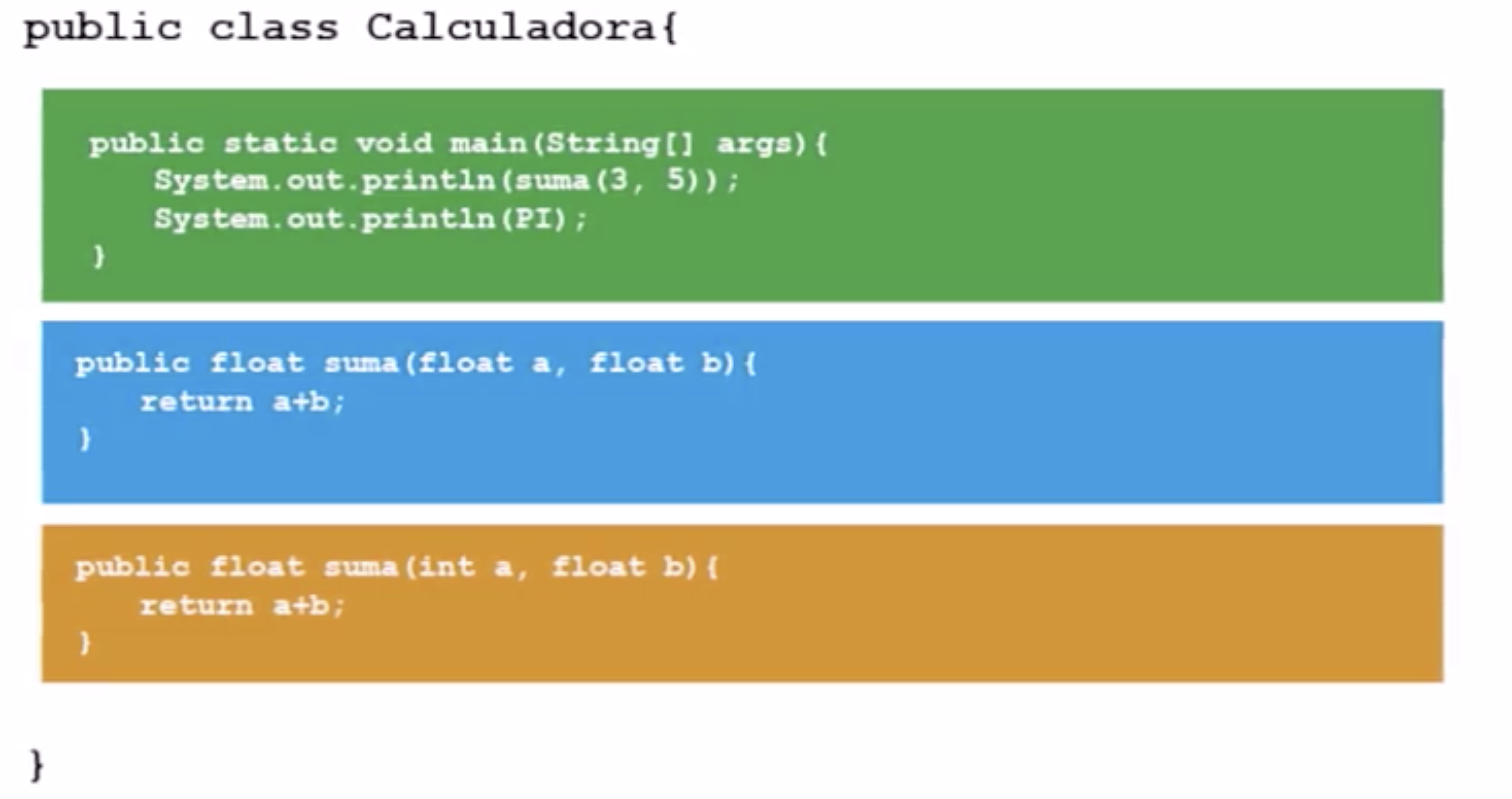
Calculadora.PI;

Calculadora.valor;

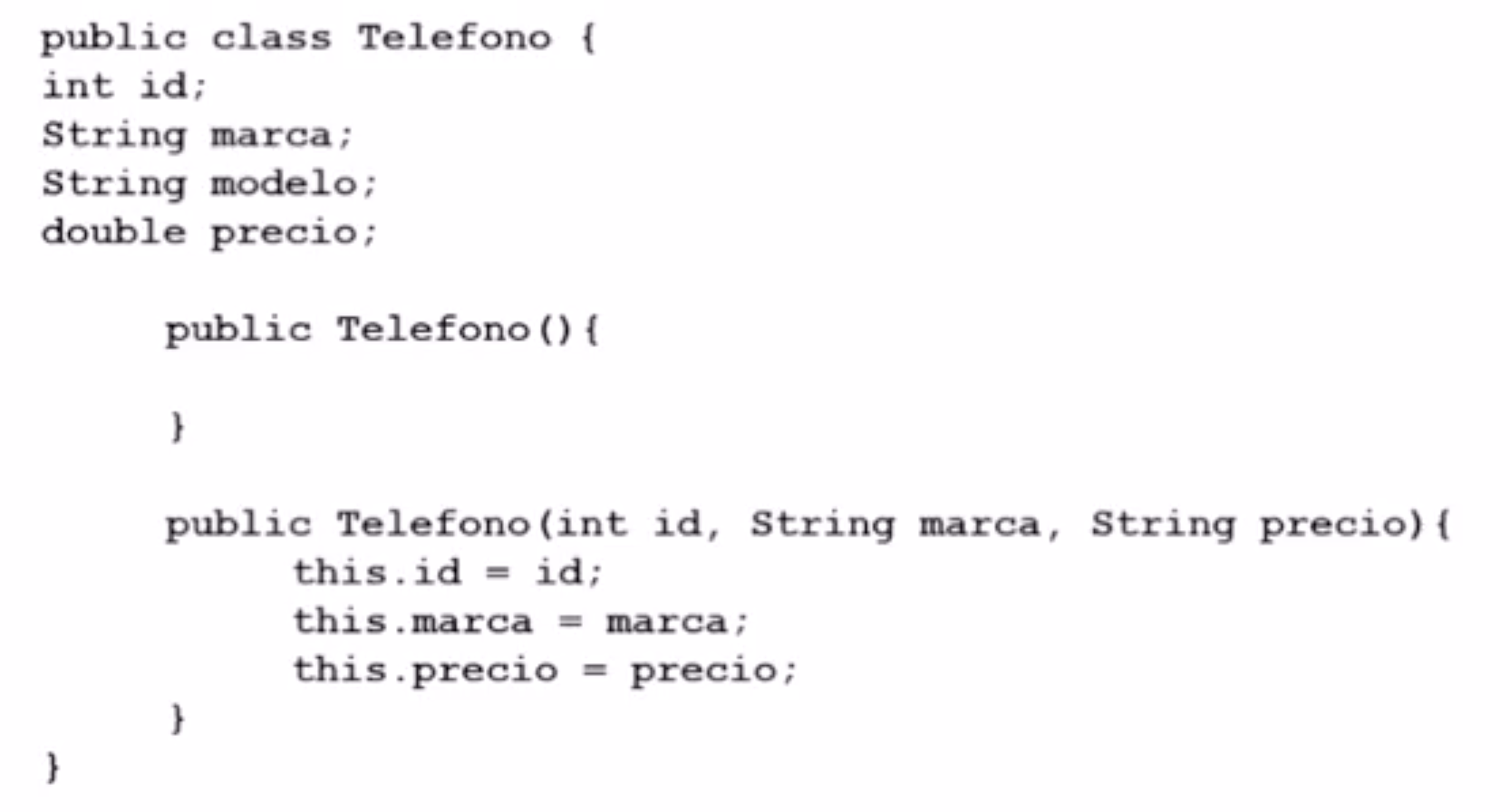
**Sobrecarga de métodos y constructores (overload)**

Surge de la necesidad de que dos métodos tengan el mismo nombre pero diferentes argumentos.

Por ejemplo: necesidad de sumar números enteros y en coma flotante (no son el mismo tipo de datos)



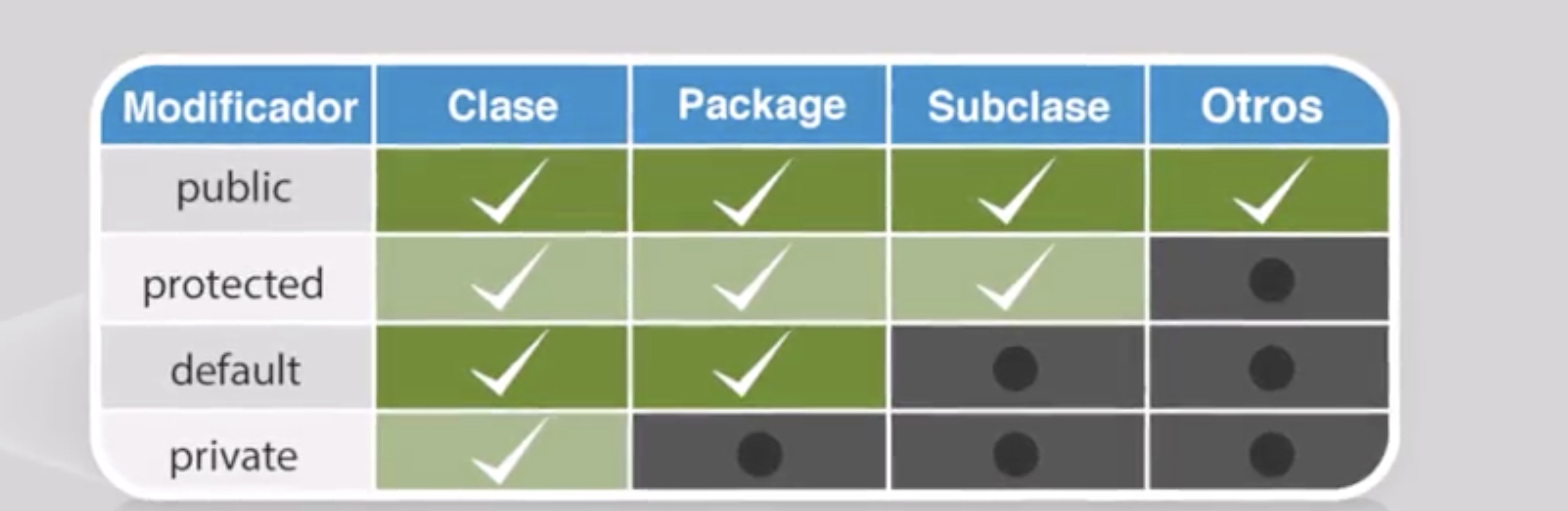
Otro de los usos de esta sobrecarga es definir un objeto según los datos proporcionados, de forma que obliga al usuario a poner X atributos para poder llevar a cabo su creación.



En este caso, si el objeto ya está creado se accede al primer método, pero si no lo está, entonces se define a través del segundo, pero sólo se crea el objeto si tiene como atributos el id, la marca y el precio (no haría falta definir el modelo).

**Modificador de acceso**

Modifica el tipo de acceso a una clase, a un método o a un atributo. En función de la modificación, permitirá acceso a diferentes partes del programa, según la tabla:



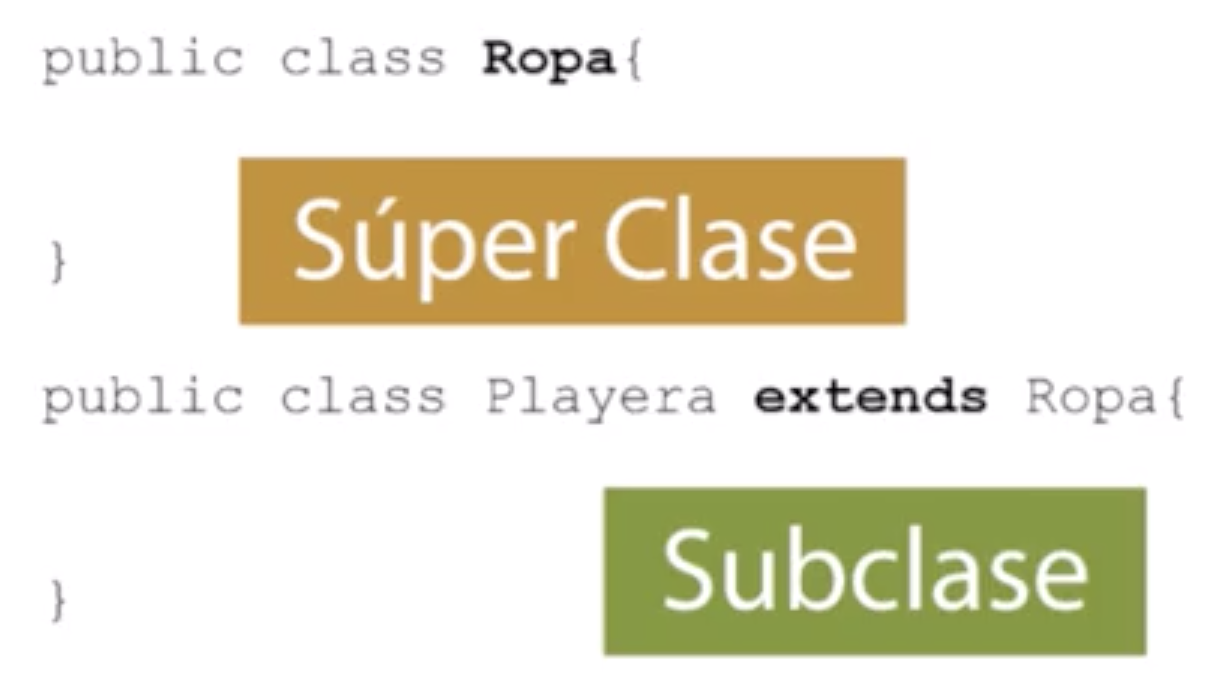
\*\* No es lo mismo que static, no confundir (especifican diferentes cosas).

**Herencia**

* Establece una relación entre dos clases (padre – hijo) de dos objetos diferentes
* La clase padre tendrá definidos atributos, métodos,… que pueden ser pasados o no a la clase hijo.
* Esto permite definir nuevas clases basadas en clases ya existentes.
* A veces es necesario usar el estado y comportamiento de una clase en conjunto con otras.

Ejemplo Tienda de ropa:

* Diferentes clases para cada tipo de producto (calcetines, pantalones, camistas,..)
* Las diferentes clases tienen atributos en común (precio, id, talla, color,…)
* Se puede crear una familia de clases, haciendo que haya atributos y métodos en común a través de una clase padre, que los pase a los hijos 🡪 Superclase (clase padre): Ropa. Subclases (clases hijos): calcetines, pantalones, atributos.
* Nomenclatura: Se debe de añadir la palabra ***‘extends’***



Definición de atributos y métodos: super, this

* Super: indica que una variable o método es de la clase padre.
* This: permite definir que la variable que se está señalando es de la misma clase que la que se está usando.

Sobreescritura de métodos (override):

Cuando una clase hereda de otra, y en la clase hija se redefine un método ya existente en la clase padre, pero con una implementación distinta.

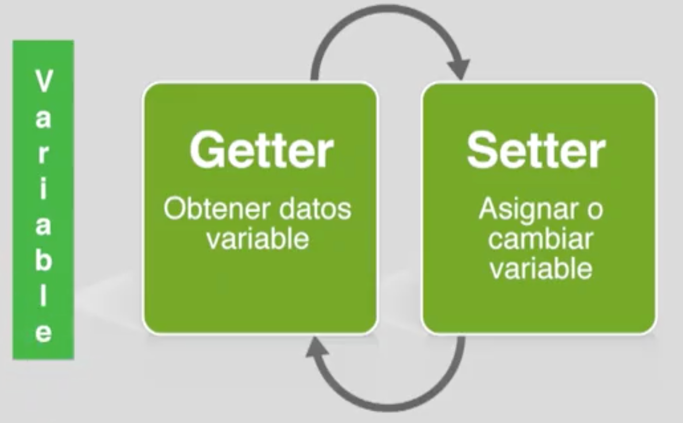
* Para evitar que se puedan sobreescribir los métodos de la clase padre se usa la palabra **final o *static***. En este caso, aunque las clases hijas quieran realizar un *override*, no será posible.
* Los métodos constructores también se pueden sobreescribir: permite la creación de una clase hijo a partir de la clase padre.

**Polimorfismo**

Posibilidad de construir varios métodos con el mismo nombre, pero con relación a la clase a la que pertenece cada uno, de forma que tengan comportamientos diferentes según ésta.

**Getters y setters**

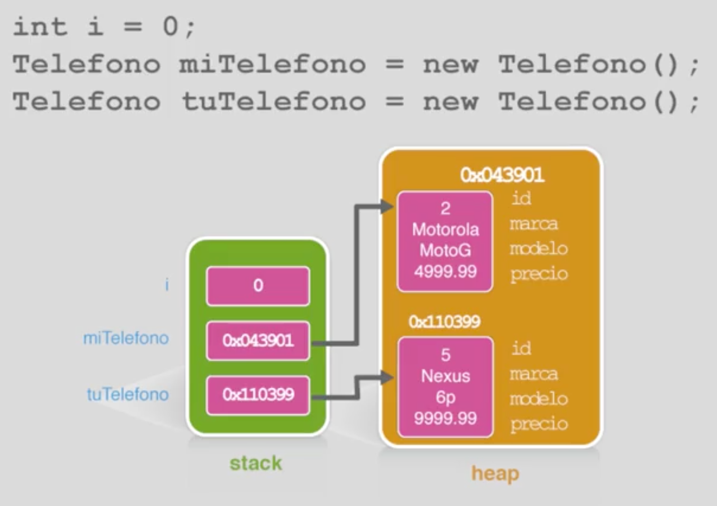
Ayuda a leer y escribir (actualizar) valores de un atributo de una clase.



En eclipse se puede hacer directamente a través de Source > Generate getters and setters.

Crear estos métodos de acceso y actualización de variables es importante, ya que permite establecer normas de actualización como, por ejemplo, no poner valores negativos a una variable.

**Diferencia de almacenamiento entre objetos y variables**



Los objetos ocupan una posición de memoria, mientras que los objetos están instanciados a un lugar de la memoria, que contiene todos los métodos y variables que lo forman.

**Interfaces**

Tipo de referencia similar a una clase, que contiene atributos y métodos, pero no son implementados, sino que estarán definidos para ser implementados posteriormente.

Ejemplo: Tanto una tienda de ropa como una tienda de telefonía debe de tener un método de devolución, motivo por el cual éste puede definirse en una interface.

## Aspectos adicionales de JAVA

**List**

Es una interfaz heredada de Collection, que ayuda a manejar colecciones de objetos (no permite utilizar tipos primitivos).

**ArrayList**

Permite almacenar un array de objetos, que no necesitan un tamaño específico de objetos, sino que crece a medida que se indexan nuevos (son dinámicas).

Se puede almacenar objetos de tipos ya definidos, como puede ser String, o también de clases creadas por nosotros, como puede ser el caso de tipo Teléfono, Camiseta,… Lo que no permite son arrays de tipos primitivos (para eso se usan las vistas anteriormente).

Funciones:

* Definirlo: ArrayList <Tipo> nombreArray = new ArrayList <Tipo> ();
* Añadir nuevo objeto a la lista: nombreArray.add(…);
* Saber el tamaño de la lista: nombreArray.size();
* Acceder a un elemento del array: nombreArray[index];
* Eliminar un elemento: nombreArray.remove(index);

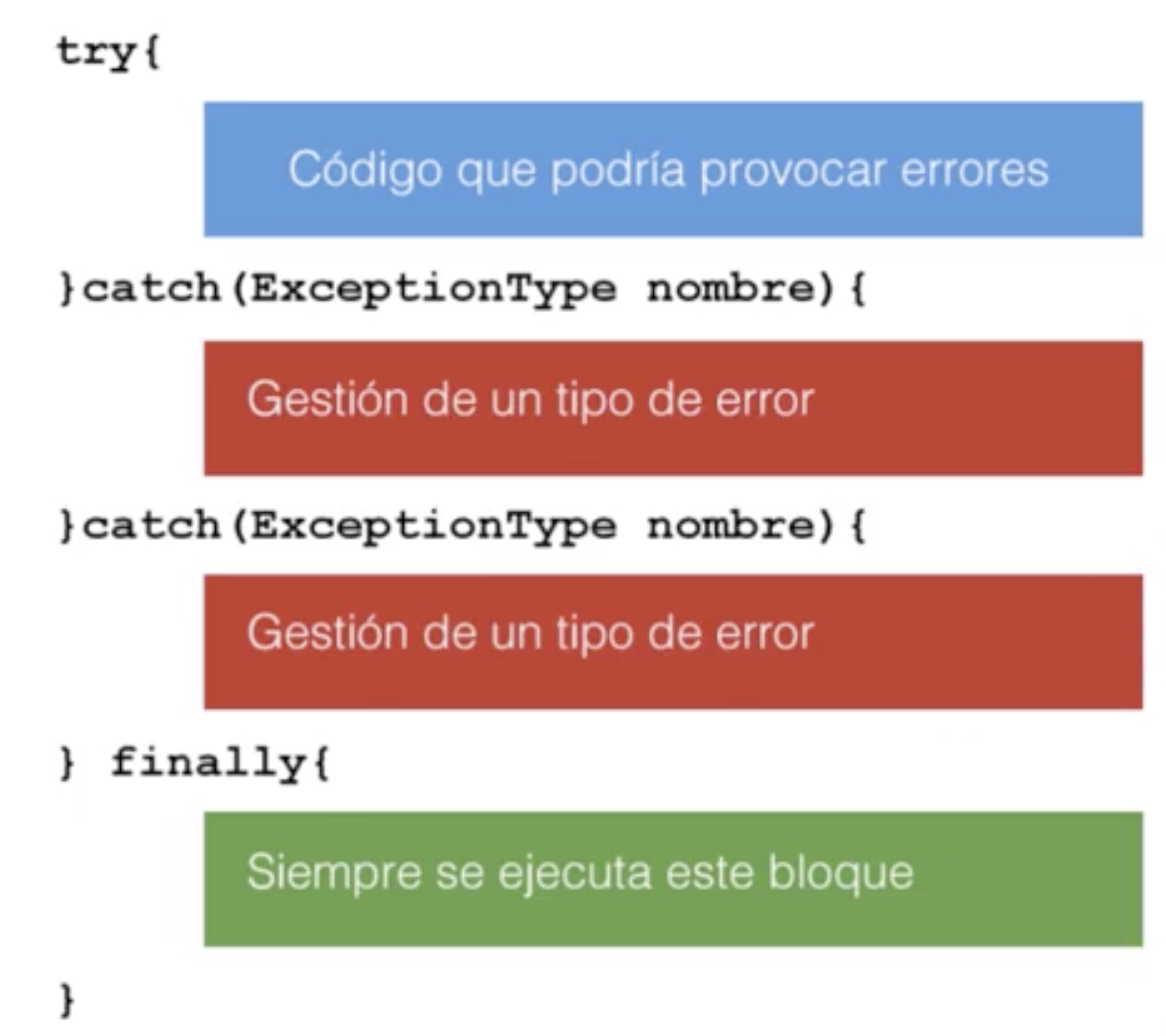
**Vectors**

El vector y el ArrayList funcionan de forma similar, incluso en las funciones que implementan cada uno. La diferencia es la forma de construcción. Los vectores utilizan hilos (threads) constantemente, algo que tiene mucho más coste computacional 🡪 Mejor usar ArrayList.

**Excepciones**

Sirve para probar hacer un trozo de código sensible a errores y, en caso de que ocurra un error inesperado, entonces poder tratar estos errores por separado, haciendo una gestión de cada una de ellas.

Esquema de excepciones:



El bloque FINALLY es la continuación del código sensible. Es decir, la forma en la que continuará el código tengamos o no alguno de estos errores.

# Bibliografia

* Deitel, P. Y Deitel H. (2012). *Cómo programar en Java Novena Edición*. Pearson