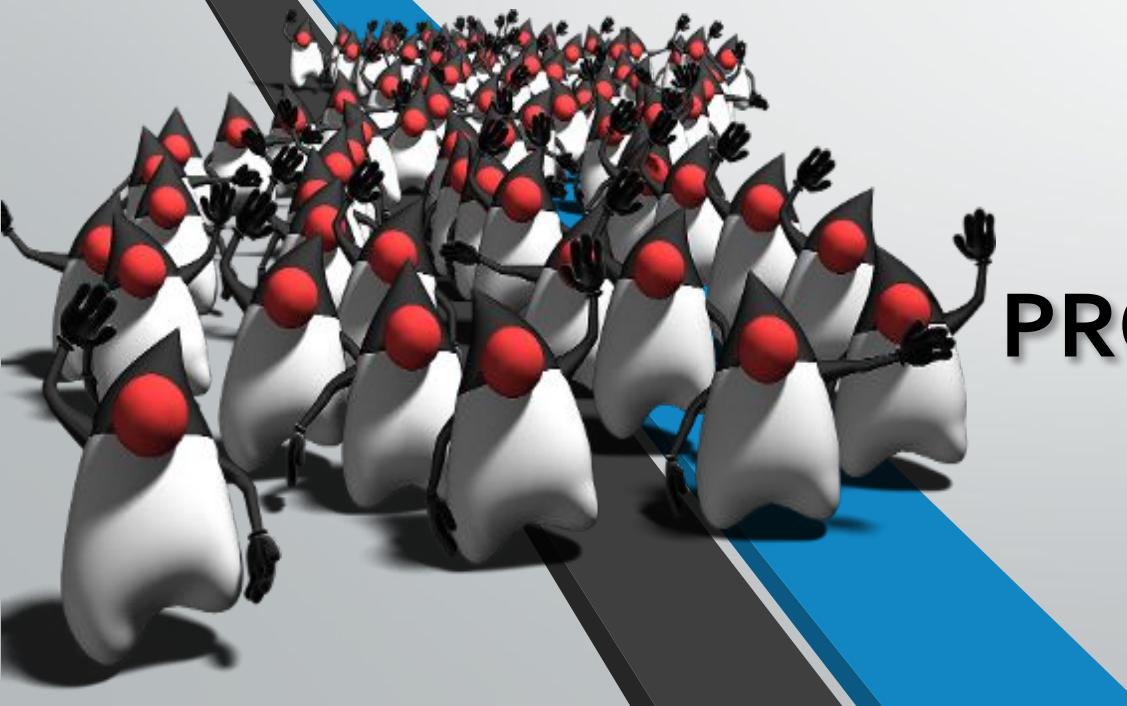


# JAVA

TEMA 05:

PROGRAMACIÓN ORIENTADA A  
OBJETOS

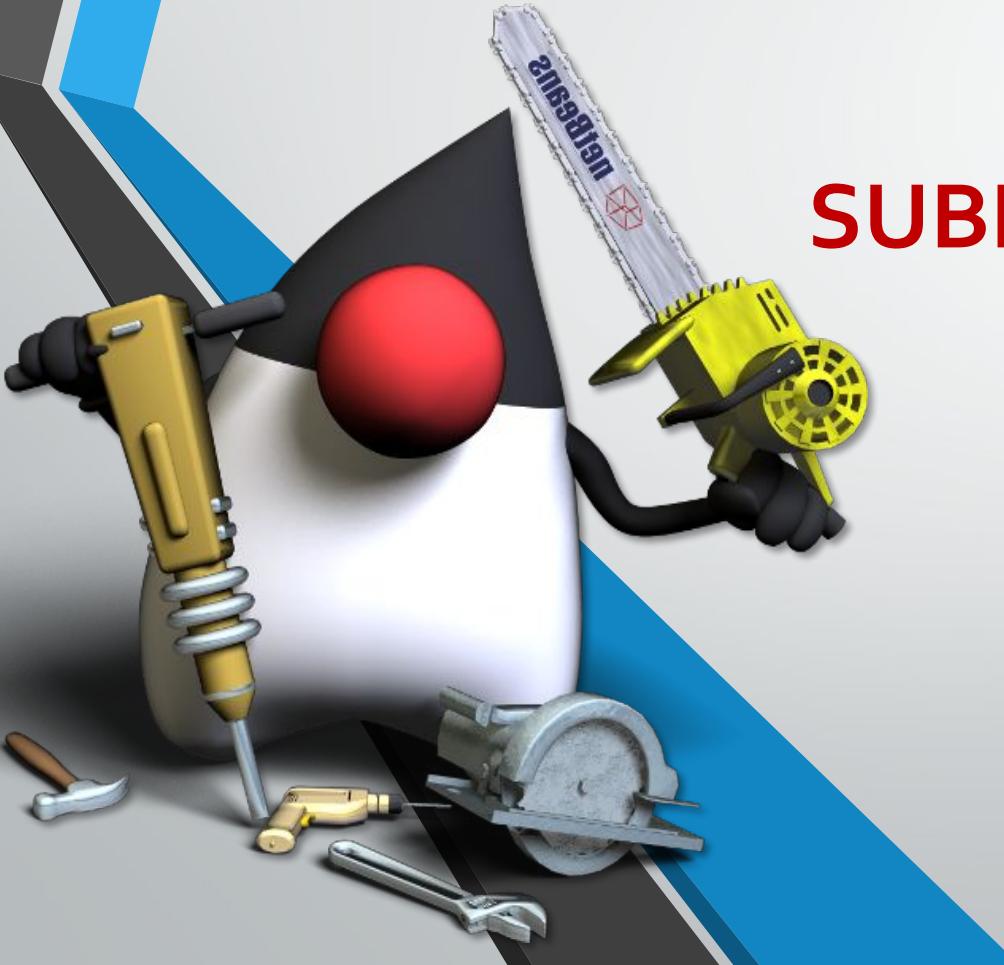


# ÍNDICE

1. Introducción
2. Clases
3. Atributos
4. Métodos
5. Constructores
6. Paso de parámetros a los métodos.
7. Sobrecarga de los métodos
8. Destructores
9. Métodos recursivos
10. La auto-referencia this
11. Herencia
12. Ejercicios de consolidación



# JAVA



## UTILIZACIÓN DE SUBPROGRAMAS Y CLASES DE USO COMÚN

### 1. Introducción

# 1.- Introducción

- Hasta ahora hemos visto un programa como una lista de instrucciones que le indican a la máquina que hacer.
- Con la programación orientada a objetos un programa se convierte en un conjunto de objetos que “dialogan” entre sí para realizar las distintas tareas programadas.
- Por ejemplo si realizamos un programa para una entidad bancaria, esta **entidad** será un objeto que se relacionará con otros objetos tales como los objetos **clientes**, los objetos **cuentas bancarias**, los objetos **sucursales**, los objetos **empleados**...

# 1.- Introducción

- ¿Qué es entonces una clase?
- Pongamos un ejemplo: pensad en un molde con el que hacemos flanes. El molde será la clase y los flanes los objetos. Luego, cada flan, tendrá sus propios atributos como la cantidad de leche, cantidad de azúcar, número de huevos, cantidad de vainilla...
- Para definir un objeto de la clase Flan sería así:

**Flan flanvainilla;** // Declaro el objeto flanvainilla de la clase Flan

**flanvainilla = new Flan();** // Asigno al objeto flanvainilla las características de la clase

- Lo más común es verlo de la siguiente forma:

**Flan flanvainilla = new Flan();**

# 1.- Introducción

- Veamos otro ejemplo: Pensad en un esqueleto para la fabricación de portátiles.
- Este esqueleto sería la clase y los ordenadores portátiles que salgan a partir de él serían los objetos.
- Además, estos objetos tendrán una serie de características (**atributos**) tales como el **color**, **marca**, **procesador**, **tamaño del disco duro**, **cantidad de memoria RAM**, **capacidad de batería**, **peso**...
- Estos objetos también podrán realizar una serie de acciones (**métodos**) tales como **encenderse**, **apagarse**, **cargar el Sistema Operativo**...

# 1.- Introducción

```
package ordenadorportatil;  
/*  
 * @author OLG  
 */  
public class OrdenadorPortatil {  
  
    private String marca;  
    private int peso;  
    private boolean encendido=false; //Mal. Nunca se debe inicializar un atributo. Se debe utilizar un constructor. Lo veremos más adelante.  
  
    public void encenderOrdenador(){  
        if (encendido==true){  
            System.out.println("El ordenador ya estaba encendido");  
        }  
        else{  
            encendido=true;  
        }  
    }  
  
    public void apagarOrdenador(){  
        if (encendido==false){  
            System.out.println("El ordenador ya estaba apagado");  
        }  
        else{  
            encendido=false;  
        }  
    }  
  
    public void establecerMarca (String mar){  
        marca=mar;  
    }  
  
    public void establecerPeso (int pes){  
        peso=pes;  
    }  
  
    public void obtenerEstado(){  
        System.out.println("El estado del ordenador es el siguiente:");  
        System.out.println("Marca: "+marca);  
        System.out.println("Peso: "+peso);  
        if (encendido ==true){  
            System.out.println("El ordenador está encendido");  
        }  
        else{  
            System.out.println("El ordenador está apagado");  
        }  
    }  
}
```

# 1.- Introducción

The screenshot shows the NetBeans IDE 8.0.2 interface. The title bar reads "OrdenadorPortatil - NetBeans IDE 8.0.2". The menu bar includes File, Edit, View, Navigate, Source, Refactor, Run, Debug, Profile, Team, Tools, Window, Help. The toolbar has various icons for file operations. The left sidebar shows a project tree with a package named "ordenadorportatil" containing a source package "ordenadorportatil" with a class "Test.java". The main editor window displays the following Java code:

```
package ordenadorportatil;
/**
 * @author OLG
 */
public class Test {
    public static void main(String[] args) {
        OrdenadorPortatil miOrdenador = new OrdenadorPortatil();
        OrdenadorPortatil ordenadorInsti = new OrdenadorPortatil();

        miOrdenador.establecerMarca("Asus");
        ordenadorInsti.establecerMarca("TTL");

        miOrdenador.establecerPeso(1600);
        ordenadorInsti.establecerPeso(1900);

        miOrdenador.encenderOrdenador();

        miOrdenador.obtenerEstado();
        ordenadorInsti.obtenerEstado();
    }
}
```

# EJERCICIOS

- **Ejercicio 01.- (OPTATIVO)** Diseña una clase Coche que contenga los siguientes atributos privados:
  - marca (de tipo cadena)
  - modelo (de tipo cadena)
  - color (de tipo cadena)
  - velocidad (de tipo entero)
  - motorEncendido (de tipo booleano inicializado a false)

# EJERCICIOS

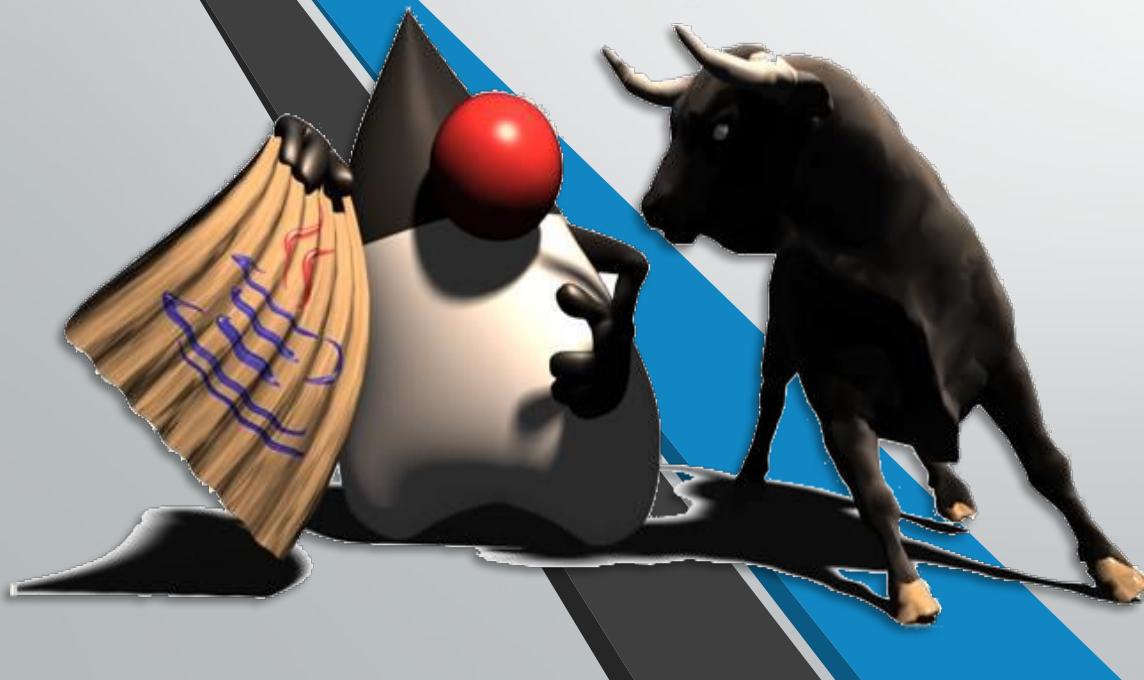
- Además contendrá los siguientes métodos públicos:
  - establecerMarca.
  - establecerModelo.
  - establecerColor.
  - arrancarCoche (pone motorEncendido a true y velocidad a 10)
  - apagarCoche (pone motorEncendido a false y velocidad a 0)
  - acelerarCoche (aumenta en 20 la velocidad)
  - frenarCoche (disminuye 6 la velocidad)
  - obtenerEstado (muestra el valor de los atributos)

# EJERCICIOS

- Para probar el funcionamiento de la clase Coche, crea otra clase llamada Test (dentro del mismo paquete) que contenga el método main donde se creen dos objetos de la clase Coche: *miCoche* y *cochePadre*. Luego realiza las siguientes operaciones:
  - Establece el modelo, marca y color, primero en el objeto miCoche y luego en cochePadre.
  - Arranca miCoche y luego arranca cochePadre.
  - Acelera 5 veces miCoche.
  - Frena 2 veces miCoche.
  - Acelera 3 veces cochePadre.
  - Apaga cochePadre.
  - Muestra el estado de miCoche y luego el estado de cochePadre.

# JAVA

## ESTRUCTURAS DE CONTROL DE FLUJO



2. Clases

## 2.- Clases

- Las clases son las *plantillas* para crear objetos.
- Clase es un tipo definido por el usuario que describe los atributos y los métodos de los objetos que se crearán a partir de la misma.
- La definición de una clase consta de dos partes: el nombre de la clase precedido por la palabra reservada **class**, y el cuerpo de la clase encerrado entre llaves:

```
class NombreClase {  
    ... cuerpo de la clase ...  
}
```

## 2.- Clases

- Veamos en más detalle el cuerpo de una clase:

```
[acceso] class NombreClase {  
    [acceso] [static] [tipo] atributo1;  
    [acceso] [static] [tipo] atributo2;  
    [acceso] [static] [tipo] atributo3;  
    .....  
    [acceso] [static] [tipo] método1(listaDeArgumentos) {  
        ... código del método ...  
    }  
    .....  
}
```

## 2.- Clases

- **[acceso]** va a determinar el alcance de la visibilidad del elemento al que se refieren (clase, atributo o método)
- **[acceso]** podrá ser de tipo **public** (visible para cualquier clase), **protected** (visible para la clase propia y las heredadas), **private** (solo visible para la clase propia). Si no indicamos nada por defecto el acceso será **friendly** (visible para todas las clases del paquete)

## 2.- Clases

zona	private (privado)	sin modificador (friendly)	protected (protegido)	public (público)
Misma clase	X	X	X	X
Subclase en el mismo paquete		X	X	X
Clase (no subclase) en el mismo paquete		X		X
Subclase en otro paquete			X	X
No subclase en otro paquete				X

## 2.- Clases

- **[tipo]** se refiere al tipo de datos, tanto para atributos como para métodos (*char, int, double...*). Los métodos también pueden ser del tipo *void* (vacío) lo que significa que no devuelven ningún valor (el método no contiene la sentencia **return**).
- **[static]** lo utilizaremos cuando queramos que un atributo o un método sea genérico para todos los objetos de la clase. Veamos un ejemplo:

## 2.- Clases

```
package cliente;
/**
 * @author OLG
 */
public class Cliente {

    private static int numeroClientes = 0;

    private String nombre;
    private int numeroCuenta;

    public static void aumentarClientes() {
        numeroClientes++;
    }

    public static int verNumeroClientes() {
        return numeroClientes;
    }

    public void establecerNombreyNumero(String nom, int num) {
        nombre = nom;
        numeroCuenta = num;
        aumentarClientes();
    }

    public String obtenerNombre() {
        return nombre;
    }

    public int obtenerNumeroCuenta() {
        return numeroCuenta;
    }
}
```

The screenshot shows the NetBeans IDE interface with the following details:

- Project Structure:** The left pane shows a project named "Clientes" with a single source package "cliente" containing "Cliente.java" and "Test.java". There are also "Source Packages" and "Libraries" sections.
- Code Editor:** The right pane displays the content of "Test.java". The code initializes two clients, prints their details, and then prints the total number of clients.

```
Clientes - NetBeans IDE 8.0.2
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
C:\Users\OLG\NetBeansProjects\Clientes\src\cliente\Cliente.java
C:\Users\OLG\NetBeansProjects\Clientes\src\cliente\Test.java
Source History ...
Test.java | Cliente.java |
1 package cliente;
2 /**
3 * @author OLG
4 */
5 public class Test {
6
7     public static void main(String[] args) {
8
9         Cliente cliente1 = new Cliente();
10        Cliente cliente2 = new Cliente();
11
12        cliente1.establecerNombreyNumero("Pepe", 10001);
13        cliente2.establecerNombreyNumero("Ana", 10002);
14
15        System.out.println("Vamos a mostrar los datos de los clientes creados: ");
16
17        System.out.println("CLIENTE 1");
18        System.out.println("Nombre del cliente: "+cliente1.obtenerNombre());
19        System.out.println("Número de cuenta: "+cliente1.obtenerNumeroCuenta());
20
21        System.out.println("CLIENTE 2");
22        System.out.println("Nombre del cliente: "+cliente2.obtenerNombre());
23        System.out.println("Número de cuenta: "+cliente2.obtenerNumeroCuenta());
24
25        System.out.println("El numero total de clientes creados es de:"+Cliente.verNumeroClientes());
26    }
27}
```

# EJERCICIOS

- **Ejercicio 02.- (OPTATIVO)** Diseña una clase Curso que contenga los siguientes atributos privados:
  - nombre (de tipo cadena)
  - numeroHoras (de tipo entero)
- Además dispondrá de un atributo estático llamado numeroDeCursos de tipo entero que lo utilizaremos para contar los objetos que vamos creando.
- La clase Curso contendrá los siguientes métodos públicos:
  - establecerNombreyHoras.
  - obtenerNombre.
  - obtenerHoras.

# EJERCICIOS

- Además, la clase `Curso`, dispondrá de dos métodos estáticos (para trabajar con el atributo estático):
  - `sumarCursos`
  - `verNumeroCursos`
- Para probar el funcionamiento de la clase `Curso`, crea otra clase llamada `Test` (dentro del mismo paquete) que contenga el método `main`, donde se creen dos objetos de la clase `Curso`: `curso1` y `curso2`. Luego realiza las siguientes operaciones:
  - Establece el nombre y el número de horas, primero en el objeto `curso1` y luego en `curso2`.
  - Muestra los datos de los 2 cursos creados.
  - Por último, muestra el número de cursos creados almacenados en el atributo estático de clase `numeroDeClientes`.

A stylized graphic on the left side of the slide features the word "JAVA" in large, bold, blue and black letters. Below it is a red circular button with a white outline. To the right of the button is a grey rectangular component with a blue circle containing a white person icon. The background behind the text is a light grey.

# JAVA

## ESTRUCTURAS DE CONTROL DE FLUJO

3. Atributos

## 3.- Atributos

- Los atributos constituyen la estructura interna de los objetos de una clase. Se declaran igual que cualquier variable. Ejemplo:

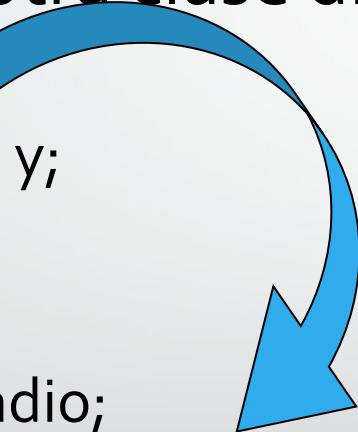
```
class Circulo {  
    private double x, y;  
    private double radio;  
    .... ....  
}
```

- No debes nunca asignarle valores a los atributos a la vez que los declaras. La forma correcta es utilizar un constructor. (Lo veremos más adelante)

## 3.- Atributos

- También puede darse el caso en que un atributo de una clase sea un objeto de otra clase distinta. Veamos un ejemplo:

```
class Coordenada {  
    private double x, y;  
}  
  
class Circulo {  
    private double radio;  
    private Coordenada centro; /* Declaro el objeto centro de la clase  
    Coordenada */  
}
```



# EJERCICIOS

- **Ejercicio 03.- (OPTATIVO)** Diseña una clase Rueda que contenga los siguientes atributos privados:
  - material (de tipo cadena)
  - pulgadas (de tipo entero)
- La clase Rueda contendrá los siguientes métodos públicos:
  - establecerMaterial
  - establecerPulgadas.
  - obtenerMaterial.
  - obtenerPulgadas.

# EJERCICIOS

- **Luego, diseña una clase Coche que contenga los siguientes atributos privados:**
  - marca (de tipo cadena)
  - modelo (de tipo cadena)
  - ruedas (objeto de tipo Rueda)
- La clase Coche contendrá los siguientes métodos públicos:
  - establecerMarca
  - establecerModelo.
  - obtenerMarca.
  - obtenerModelo.
  - establecerRueda.
  - obtenerRueda.

# EJERCICIOS

- Para probar el funcionamiento de las clases creadas, crea otra clase llamada Test (dentro del mismo paquete) que contenga el método main, donde:
  - Se creen 2 objetos de la clase Rueda: *rueda1* y *rueda2*.
  - Establece el material y el tamaño, primero en el objeto *rueda1* y luego en *rueda2*.
  - Muestra los datos (material y tamaño) de las dos ruedas creados.
  - Crea 3 objetos de la clase Coche: *coche1*, *coche2* y *coche3*.
  - Establece la marca y el modelo de los 3 coches.
  - Establece las *ruedas1* para el *coche1* y el *coche2*, y *ruedas2* para el *coche3*.
  - Muestra los datos de los coches creados (marca, modelo, material de su rueda y tamaño de su rueda).

# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA

4. Métodos



## 4.-Métodos

- Como ya vimos anteriormente, esta es la sintaxis de declaración de un método:

```
[acceso] [static] tipo método1(listaDeArgumentos) {  
    ... código del método ...  
}
```

- El tipo indica que tipo de valores devuelve el método. Esto es así porque mediante el comando return puede devolver un determinado valor.
- Si el método no devuelve ningún valor entonces su tipo será void.

## 4.-Métodos

- Las llamadas a los métodos consisten en pasarle el control del programa al método hasta que este finalice o se haga uso del return.
- Al llamar un método, muchas veces será necesario pasarle datos en forma de argumentos separados por comas. Ejemplos:

*miOrdenador.Encender(); //sin argumentos*

*miOrdenador.EstablecerColor ("Rojo"); //Un argumento*

*miOrdenador.EstablecerMarcayPeso ("Acer", 3); //Dos argumentos*

# 4.-Métodos

```
package vehiculo;

public class Vehiculo {
    int ruedas;
    private int velocidad;
    String nombre;

    public void Arrancar() {//void porque no devuelve ningun valor
        velocidad = 0; //Inicializa velocidad a 0
    }
    public void Acelerar(int cantidad) { //void porque no devuelve ningun valor
        //Acelerar recibe como argumento un entero al que se le llama cantidad
        velocidad = velocidad + cantidad;
    }
    public void Frenar(int cantidad) {
        velocidad = velocidad - cantidad;
    }
    public int ObtenerVelocidad(){ //no recibe argumentos
        //ObtenerVelocidad es de tipo int porque devuelve un valor de tipo entero
        return velocidad; // asigna el valor del atributo velocidad al metodo ObtenerVelocidad
    }
    public static void main(String[] args) {
        Vehiculo miCoche = new Vehiculo();
        miCoche.Arrancar(); // Esta llamada no pasa argumentos
        miCoche.Acelerar(12); //Le pasamos 12 como argumento
        miCoche.Frenar(5); //Le pasamos 5 como argumento
        System.out.println(miCoche.ObtenerVelocidad());
    }
}
```

# EJERCICIOS

- **Ejercicio 04.- (OPTATIVO)**  
Transforma el siguiente programa de tal forma que el resultado sea el mismo pero que:
  - El método “muestraPajaros” contenga la sentencia return.
  - El método main se encuentre en una clase distinta.

```
package pajarito;

public class Pajarito {

    static int numpajaros=0;
    String color;
    int edad;

    static void nuevopajaro() {
        numpajaros++;
    }

    public void establecerColoryEdad(String col,int ed){
        color = col;
        edad = ed;
        nuevopajaro();
    }

    static void muestraPajaros() {
        System.out.println("El numero de objetos creados es de: " + numpajaros);
    }

    public static void main(String[] args) {
        // TODO code application logic here
        Pajarito p1 = new Pajarito();
        Pajarito p2 = new Pajarito();
        p1.establecerColoryEdad("rojo", 1);
        p2.establecerColoryEdad("azul", 2);
        muestraPajaros();
    }
}
```

# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA



### 5. Constructores

## 5.- Constructores

- Un constructor es un método que es llamado automáticamente al crear un objeto de una clase, es decir, al usar la instrucción new.
- Hasta ahora no hemos utilizado ninguno, por lo que JAVA genera un constructor por defecto para inicializar los atributos del objeto a null.
- La sintaxis de un constructor no es más que un método que tiene el mismo nombre que la clase, y dentro de él podremos inicializar los atributos del objeto. Veamos un ejemplo:

# 5.- Constructores

```
package pajaro;  
/**  
 * @author OLG  
 */  
public class Pajaro {  
  
    private String color;  
    private int edad;  
  
    Pajaro() { //Constructor por defecto  
    }  
  
    Pajaro(String col, int ed) { //Creo un constructor propio que reciba 2 parametros  
        color = col;  
        edad = ed;  
    }  
  
    public void establecerColor(String col) {  
        color = col;  
    }  
  
    public void establecerEdad(int ed) {  
        edad = ed;  
    }  
  
    public String obtenerColor() {  
        return color;  
    }  
  
    public int obtenerEdad() {  
        return edad;  
    }  
}
```

```
package pajaro;  
/**  
 * @author OLG  
 */  
public class Test {  
  
    public static void main(String[] args) {  
        Pajaro gorrion=new Pajaro(); //Utilizo el constructor por defecto  
        Pajaro verderon=new Pajaro("verde",3); //Utilizo mi constructor  
  
        System.out.println(gorrion.obtenerColor());  
        System.out.println(verderon.obtenerColor());  
    }  
}
```

Output - Pajaro (run) x

run:  
null  
verde  
BUILD SUCCESSFUL (total time: 0 seconds)

## 5.- Constructores

- También es posible editar el constructor por defecto.  
Veamos un ejemplo:

# 5.- Constructores

```
package pajaro;  
/**  
 * @author OLG  
 */  
public class Pajaro {  
  
    private String color;  
    private int edad;  
  
    Pajaro() { //Constructor por defecto editado  
        color = "negro";  
        edad = 0;  
    }  
  
    Pajaro(String col, int ed) { //Creo un constructor propio que reciba 2 parametros  
        color = col;  
        edad = ed;  
    }  
  
    public void establecerColor(String col) {  
        color = col;  
    }  
  
    public void establecerEdad(int ed) {  
        edad = ed;  
    }  
  
    public String obtenerColor() {  
        return color;  
    }  
  
    public int obtenerEdad() {  
        return edad;  
    }  
}
```

```
package pajaro;  
/*  
 * @author OLG  
 */  
public class Test {  
  
    public static void main(String[] args) {  
        Pajaro gorron=new Pajaro(); //Utilizo el constructor por defecto  
        Pajaro verderon=new Pajaro("verde",3); //Utilizo mi constructor  
  
        System.out.println(gorron.obtenerColor());  
        System.out.println(verderon.obtenerColor());  
    }  
}  
run: Pajaro (run) x  
run:  
negro  
verde
```

# EJERCICIOS

- **Ejercicio 05.- (OPTATIVO)** Realiza un programa en JAVA en el que le pidas al usuario las notas de las 6 asignaturas del Ciclo de DAM y te calcule la nota media del curso.
- o Nota 1: Cada una de las asignaturas serán un objeto cuyos atributos serán el nombre y la nota.
- o Nota 2: Crea un constructor con el que puedas asignar directamente el nombre de la asignatura al crear el objeto. En cambio, el atributo nota, será el usuario quien lo introduzca mediante un método al que se le pase la nota como argumento.
- o Nota 3: Crea otro método que reciba las 6 notas como argumentos y devuelva la nota media (return)
- o Ejemplo de ejecución:

*Por favor, introduzca la nota de Programación: 6,5*

*Introduzca la nota de Lenguajes de Marcas: 7,5*

*Introduzca la nota de Bases de Datos: 7,5*

*Introduzca la nota de Entornos de Desarrollo: 8*

*Introduzca la nota de Sistemas Informáticos: 6,5*

*Por último, introduzca la nota de Formación y Orientación Laboral: 6*

*Su nota media del curso es de: 7*



# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA

6. Paso de parámetros a los métodos

## 6.- Paso de parámetros a los métodos

- El paso de argumentos a un método se puede hacer por valor o por referencia.
- Por valor: Los métodos reciben una copia de los datos, es decir la variable que se pasa como argumento no estará afectada por el código. Veamos unos ejemplos:

# 6.- Paso de parámetros a los métodos

```
package ejemplovalor;
/*
 * @author OLG
 */
public class EjemploValor {

    public static void cambiaNumero(int numero) {
        numero = 333;
    }

    public static void main(String[] args) {
        int numero;
        numero = 24;
        System.out.println("Número vale:" + numero);

        cambiaNumero(numero);

        System.out.println("Número vale:" + numero);
    }
}
```

Output - EjemploValor (run) ×

```
run:
Número vale:24
Número vale:24
BUILD SUCCESSFUL (total time: 0 seconds)
```

# 6.- Paso de parámetros a los métodos

```
package ejemplovalor;
/*
 * @author OLG
 */
public class EjemploValor {

    public static void cambiaSaludo(String saludo) {
        saludo = ";Que tal!";
    }

    public static void main(String[] args) {
        String saludo;
        saludo = "hola";

        System.out.println("Saludo: " + saludo);

        cambiaSaludo(saludo);

        System.out.println("Saludo: " + saludo);
    }
}

Output - EjemploValor (run) ×
run:
Saludo: hola
Saludo: hola
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 6.- Paso de parámetros a los métodos

- **Por referencia:** Los métodos reciben la dirección física de memoria del parámetro que se le pasa, por lo que al modificar el valor que se encuentra en la posición de memoria se modifica el valor del parámetro.
- Esto ocurre cuando pasamos un **objeto** o un **array**. Veamos unos ejemplos:

# 6.- Paso de parámetros a los métodos

```
package pajaro;  
/**  
 * @author OLG  
 */  
public class Pajaro {  
  
    private String color;  
    private int edad;  
  
    Pajaro(String col, int ed) { //Creo un con  
        color = col;  
        edad = ed;  
    }  
  
    public void establecerColor(String col) {  
        color = col;  
    }  
  
    public void establecerEdad(int ed) {  
        edad = ed;  
    }  
  
    public String obtenerColor() {  
        return color;  
    }  
  
    public int obtenerEdad() {  
        return edad;  
    }  
}
```

```
package pajaro;  
/**  
 * @author OLG  
 */  
public class Test {  
  
    public static void cambiaColor(Pajaro verderon){  
        verderon.establecerColor("azul");  
    }  
  
    public static void main(String[] args) {  
        Pajaro verderon=new Pajaro("verde",3); //Uti  
  
        System.out.println(verderon.obtenerColor());  
        cambiaColor(verderon);  
        System.out.println(verderon.obtenerColor());  
    }  
}  
Output - Pajaro (run) ×  
run:  
verde  
azul  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# EJERCICIOS

- **Ejercicio 07.- (OPTATIVO)** Implementa un programa en JAVA, orientado a objetos, al que introduciéndole el tamaño de un archivo en MB y la velocidad del ADSL (en megabits) te calcule el tiempo en minutos que tardarías en descargártelo.
- Crearás una clase Descarga que tendrá 3 atributos privados: nombreDescarga, tamañoDescarga y velocidadDescarga, así como los métodos correspondientes.
- Para probar el funcionamiento de la clase Descarga, crea una clase Test en la que crees un objeto con un constructor al que le pases los valores, que elija el usuario, de todos los atributos de la clase. Luego, también le preguntaras al usuario si desea crear otro objeto (si o no), para crear todos los objetos que quiera el usuario. (los objetos se sobreescibirán)
- La clase Test también tendrá un método que recibirá el objeto de la clase Descarga, y mostrará por pantalla el nombre y el tiempo que se tarda en descargar. Utiliza este método para ver el nombre y el tiempo de descarga de los objetos creados.

# EJERCICIOS

• **Ejercicio 07.- (SOLUCIÓN):**

```
package descarga;
/*
 * @author OLG
 */
public class Descarga {
    private String nombreDescarga;
    private double tamaÑoDescarga;
    private double velocidadDescarga;

    Descarga(String nom, double tam, double vel) {
        nombreDescarga=nom;
        tamaÑoDescarga=tam;
        velocidadDescarga=vel;
    }

    public void establecerNombre(String nom) {
        nombreDescarga=nom;
    }
    public void establecerTamaÑo(double tam) {
        tamaÑoDescarga=tam;
    }
    public void establecerVelocidad(double vel) {
        velocidadDescarga=vel;
    }

    public String obtenerNombre() {
        return nombreDescarga;
    }
    public double obtenerTamaÑo() {
        return tamaÑoDescarga;
    }
    public double obtenerVelocidad() {
        return velocidadDescarga;
    }
}
```

```
package descarga;
import java.util.ArrayList;
import java.util.Scanner;
/*
 * @author OLG
 */
public class Test {

    public static String pedirNombre() {
        String nombre;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca el nombre de la descarga");
        nombre = entrada.nextLine();
        return nombre;
    }

    public static double pedirTamaÑo() {
        double tamaÑo;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca el tamaÑo de la descarga");
        tamaÑo = entrada.nextInt();
        return tamaÑo;
    }

    public static double pedirVelocidad() {
        double velocidad;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca la velocidad de la descarga");
        velocidad = entrada.nextInt();
        return velocidad;
    }

    public static void calculoTiempo(Descarga objeto) {
        String nombre = objeto.obtenerNombre();
        double tamaÑo = objeto.obtenerTamaÑo();
        double velocidad = objeto.obtenerVelocidad();
        double tiempo;

        tiempo = ((tamaÑo * 8) / velocidad) / 60;
        System.out.println(nombre + " tardara en descargarse " + tiempo + " minutos");
    }

    public static void main(String[] args) {
        Scanner entrada = new Scanner(System.in);

        String respuesta;
        String nomDes;
        double tamDes;
        double velDes;

        do {
            nomDes = pedirNombre();
            tamDes = pedirTamaÑo();
            velDes = pedirVelocidad();
            Descarga descarga = new Descarga(nomDes, tamDes, velDes);
            calculoTiempo(descarga);
            System.out.println("¿Desea crear otro objeto?");
            respuesta = entrada.nextLine();
        } while (respuesta.equalsIgnoreCase("si"));
    }
}
```

# EJERCICIOS

- **Ejercicio 08.- (OPTATIVO)** Realiza un programa en JAVA, orientado a objetos, en el que crees una clase llamada **Numero** que contendrá un único atributo privado, llamado **valor**, de tipo entero.
- Para probar el funcionamiento de la clase Numero, crea una nueva clase llamada Test donde le pidas al usuario un número por teclado. Luego, crea un objeto, siendo el número introducido por el usuario el valor del atributo **valor**. Posteriormente, le pasarás el objeto a un método que le dirá al usuario si el número introducido es par o impar.

# EJERCICIOS

- **Ejercicio 09.- (OPTATIVO)** Realiza un programa en JAVA, ORIENTADO A OBJETOS, que lea tres números e imprima por pantalla el mayor de ellos.
- Realiza el ejercicio a tu gusto, con los métodos que consideres necesarios (cuantos más mejor).
- Pista: Necesitarás crear 3 objetos de una clase “Número” con un atributo llamado “valor”.

# EJERCICIOS

- **Ejercicio 10.- (OBLIGATORIO)** Escribe un programa en JAVA, orientado a objetos, en el que el usuario introduzca cuatro números enteros y luego el programa los muestre por pantalla ordenados de forma creciente. (de menor a mayor)
- Pista: Necesitarás cuatro objetos de la clase “Numero” con un solo atributo llamado “valor”.

# EJERCICIOS

- **Ejercicio 11.- (OPTATIVO)** Realiza un programa en JAVA, ORIENTADO A OBJETOS, en el que le solicites al usuario 2 números y, si el primer número introducido es mayor que 10, se multipliquen, y en caso contrario que se sumen. Se le mostrará al usuario la operación realizada y su resultado.
- Necesitarás una clase “Numero” (con un único atributo llamado “valor”), y una clase Test. En esta última crea dos objetos con el constructor por defecto. Luego, le pides al usuario los valores y se los asignas a los objetos.
- En la clase Test también habrá un método para sumar, otro para multiplicar y otro para pedir los valores al usuario.

# EJERCICIOS

- **Ejercicio 12.- (OBLIGATORIO)** Escribe un programa en JAVA orientado a objetos, en el que se le pida al usuario que introduzca la longitud de los catetos de un triángulo rectángulo y que posteriormente el programa calcule la longitud de la hipotenusa.
- Para ello necesitarás una clase Triangulo con 2 atributos: cateto1, cateto2.
- En una clase Test crearás un objeto de la clase Triangulo donde el tamaño de los catetos se los pedirás al usuario para pasárselos al constructor. En esta clase también tendrás un método que calcule la hipotenusa y otro método se encargará de mostrar los resultados por la pantalla.

# EJERCICIOS

- **Ejercicio 13.- (OPTATIVO)** Crea un programa JAVA orientado a objetos que, utilizando bucles, escriba "Hola, que tal" cinco veces. Para ello tendrás una clase "Saludo" con un atributo "frase" de tipo String.
- Realiza el ejercicio a tu gusto, con los métodos que consideres necesarios (cuantos más mejor). Intenta que el método main sea pequeño.

# EJERCICIOS

- **Ejercicio 14.- (OPTATIVO)** Realiza un programa que, utilizando bucles, imprima la tabla de multiplicar de un número que elija el usuario.
- Utiliza el constructor por defecto. Luego llama a un método que asignará al atributo “valor” de la clase “Numero” el número que elija el usuario.
- Realiza el ejercicio a tu gusto, con los métodos que consideres necesarios (cuantos más mejor). Intenta que el método main sea pequeño.

# EJERCICIOS

- **Ejercicio 15.- (OBLIGATORIO)** Diseña una clase Dirección con los atributos calle, número, piso y ciudad. A continuación, diseña una clase Empleado con los atributos nombre (de tipo String), salario (de tipo int) y dirección (de tipo Dirección).
- Para probar la funcionalidad de las clases creadas crea una clase Test donde crees 3 direcciones y luego crea tres empleados, asignándoles una de las direcciones anteriormente creadas. Por último, crea un método que muestre los datos de cada empleado creado:

*EMPLEADO 1:*

*Nombre:* XXX

*Salario:* XXX

*Dirección:*

*Calle:* XXX

*Número:* XXX

*Puerta:* XXX

*Ciudad:* XXX

*EMPLEADO 2:*

*Nombre:* .....

# EJERCICIOS

- **Ejercicio 16.- (OPTATIVO)** Realiza un programa en JAVA, orientado a objetos, donde el usuario introduzca el nombre y la nota de un alumno (número entero entre 0 y 10) y se escribirá su calificación según el valor de la nota ingresada:
  - 0 a 4 = Suspenso.
  - 5 a 6 = Bien.
  - 7 a 8 = Notable.
  - 9 a 10 = Sobresaliente.
- Nota: Se le avisará al usuario de un error en caso de que la nota que nos introduzca no esté entre 0 y 10.
- Para ello tendrás una clase Alumno con los atributos nombre y nota. En una clase Test crea 3 objetos y los métodos que consideres necesarios (cuantos más mejor). Intenta que el método main sea lo más pequeño posible.

# EJERCICIOS

- **Ejercicio 17.- (OPTATIVO)** Realiza un programa, orientado a objetos, que le haga un examen al usuario, haciéndole 4 preguntas (por ejemplo: ¿cuál es la capital de España?, ¿quién descubrió América?...) Le dirá si ha respondido correctamente o no, cual sería la respuesta correcta, y por último le dirá su nota (Cada pregunta acertada vale 2,5).
- La clase Enunciado tendrá dos atributos de tipo String, *pregunta* y *respuesta*.
- Tendrás una clase Test donde crearás 4 objetos e implementarás los métodos necesarios para preguntar al usuario, y para darle la nota del examen.

# EJERCICIOS

- **Ejercicio 18.- (OPTATIVO)** Realizar un programa orientado a objetos, que pida al usuario dos números por teclado (serán el atributo de dos objetos distintos). Posteriormente el programa mostrará un menú que le permitirá al usuario:
  - 1.- Sumar los números.
  - 2.- Restar los números.
  - 3.- Multiplicar los números.
  - 4.- Dividir los números.
  - 5.- Salir del programa.
- Nota1: Mientras el usuario no pulse 5, el programa no termina y el menú volverá a aparecer pidiendo nuevamente que le introduzcas una opción.
- Nota 2: Controla el caso de división entre 0 mediante la captura de excepciones.
- Nota 3: Crea una clase “Número”, con un solo atributo llamado **valor**, que se inicializará en el constructor con el valor que te introduzca el usuario.

# EJERCICIOS

- **Ejercicio 19.- (OBLIGATORIO)** Desarrolle una aplicación en Java que determine el sueldo bruto para cada uno de los tres empleados de una empresa. La empresa paga la tarifa normal en las primeras 40 horas de trabajo de cada empleado, y paga tarifa y media en todas las horas trabajadas que excedan de 40.
- El programa creará 3 objetos (uno para cada empleado) y se le pedirá al usuario que rellene la información para cada empleado en el constructor.
- Por cada empleado se almacenará su nombre, el número de horas que trabajó, y la tarifa que cobra por una hora de trabajo.
- Para probar la clase Empleado, crea una clase Test con un método que determine y muestre el sueldo bruto de cada empleado.
- Ejemplo:

*Pepe trabajó 42 horas, cobra 20 euros la hora por lo que le corresponde un sueldo de 860 euros.*

# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA



7. Sobrecarga de los métodos

## 7.- Sobrecarga de los métodos

- Es una propiedad de JAVA que nos permite crear distintas variantes de un mismo método con el mismo **nombre, pero diferente número de parámetros y/o tipos** de estos. Además, estos métodos, se encontrarán definidos en una misma clase.
- Veamos un ejemplo:

# 7.- Sobrecarga de los métodos

```
package polimorfismo;
/**
 * @author OLG
 */
public class Polimorfismo {

    public static void Suma (int n1, int n2){
        int resultado=n1+n2;
        System.out.println("Suma: "+(resultado));
    }
    public static void Suma (int n1, int n2, int n3){
        int resultado=n1+n2+n3;
        System.out.println("Suma: "+(resultado));
    }
    public static void Suma (int n1, int n2, double n3){
        int resultado=n1+n2+(int)n3;
        System.out.println("Suma: "+(resultado));
    }
    public static void Suma (int n1, int n2, double n3, double n4){
        int resultado=n1+n2+(int)n3+(int)n4;
        System.out.println("Suma: "+(resultado));
    }
    public static void Suma (int n1, int n2, int n3, double n4, double n5){
        double resultado=n1+n2+n3+n4+n5;
        System.out.println("Suma: "+(resultado));
    }
    public static void Suma (int n1, int n2, double n3, double n4, double n5){
        double resultado=n1+n2+n3+n4+n5;
        System.out.println("Suma: "+(resultado));
    }
}
```

```
public static void main(String[] args) {
    int numero1=1, numero2=2, numero3=3;
    double numero4=4,numero5=5,numero6=6;
    Suma(numero1,numero2);
    Suma(numero3,numero2,numero3);
    Suma(numero1,numero3,numero4);
    Suma(numero2,numero3,numero5,numero6);
    Suma(numero3,numero2,numero3,numero4,numero5);
    Suma(numero1,numero2,numero4,numero5,numero6);
}
```



# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA

### 8. Destructores

## 8.- Destructores

- En **C++** los objetos se crean con la instrucción **new** (igual que en **JAVA**) y se destruyen (liberando la memoria que ocupan) con la instrucción **delete**.
- En **JAVA** no existe la instrucción delete sino que existe un proceso llamado “recolector de basura” (garbage collector) que automáticamente elimina los objetos que han dejado de usarse un determinado tiempo y se ha quedado sin ser referenciados.

## 8.- Destructores

- Antes de que actúe el recolector de basura, JAVA llamará al método **finalize**, que si el usuario no lo ha implementado, JAVA genera uno por defecto. (como ocurría con el constructor)

```
protected void finalize() throws Throwable {  
    System.out.println("El objeto va a ser destruido");  
}
```

## 8.- Destructores

- El ejemplo típico de cuando un objeto deja de ser referenciado es el siguiente:

*String **saludo** = "Hola";*

***saludo** = **saludo** + ", que tal"*

- Las cadenas en JAVA son objetos inmutables, esto significa que una vez que el objeto se ha creado, no puede ser modificado. Por ello, lo que realmente ocurre es que al concatenar "Hola" con ", que tal" se está creando un nuevo objeto "Hola, que tal" que pasa a ser referenciado por el identificador saludo.

- La cadena “Hola”, por tanto, deja de estar referenciado por dicho identificador, con lo que se pierde toda posibilidad de volver a acceder al mismo. A partir de aquí, el recolector de basura de JAVA puede liberar la memoria ocupada por el objeto, si el sistema así lo requiere.

## 8.- Destructores

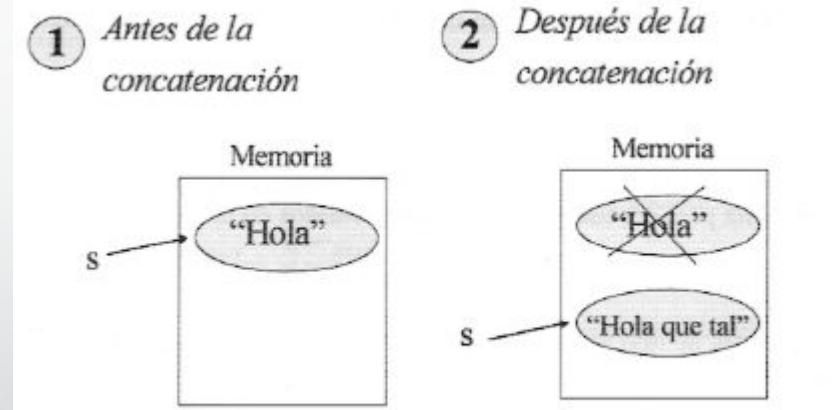


Fig. 57. Situación de la memoria antes y después de concatenar dos textos



# **JAVA**

## **ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA**

**9. Métodos recursivos**

## 9.- Métodos recursivos

- Son aquellos métodos que se llaman a sí mismos.
- El ejemplo típico de recursividad es el cálculo del factorial de un número:

$$\text{factorial}(n) = n * \text{factorial}(n-1)$$

$$5! = 5 * 4 * 3 * 2 * 1$$

- Veamos como quedaría programado en JAVA:

# 9.- Métodos recursivos

```
package ejemplo;
import java.util.Scanner;
/**
 * @author Oscar Laguna García
 */
public class Ejemplo {    // Calculo del factorial de un número
    public static int factorial (int numero) {
        if (numero==0){
            return 1;
        }
        else {
            return numero * factorial (numero-1);
        }
    }
    public static void main(String[] args) {
        int numerosuario;
        int fac;
        Scanner entrada = new Scanner (System.in );
        System.out.println("Introduzca un numero para calcular su factorial:");
        numerosuario = entrada.nextInt();
        fac=factorial(numerosuario);
        System.out.println("El factorial de "+ numerosuario + " es: " +fac);
    }
}
```

# **JAVA**

## **ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA**



**10. La autoreferencia this**

## 10.- La autoreferencia this

- Para referirse a los atributos del objeto desde un método del mismo, se puede hacer directamente con su nombre o utilizando **this**.
- La palabra reservada **this** se utiliza, sobretodo, cuando existe ambigüedad entre nombres de parámetros de un método y atributos del objeto. (otros usos se explicarán más adelante).
- Veamos un ejemplo:

# 10.- La autoreferencia this

- **this** funciona como una referencia especial, de forma que *this.nombre* se refiere al atributo **nombre** declarado en la clase, para diferenciarlo de la variable **nombre** declarada como parámetro del método.

```
package alumno;  
/**  
 * @author OLG  
 */  
public class Alumno {  
  
    private String nombre;  
    private int edad;  
  
    Alumno(String nombre, int ed){  
        this.nombre=nombre;  
        edad=ed;  
    }  
  
    public void establecerNombre(String nom){  
        nombre=nom;  
    }  
  
    public void establecerEdad(int edad){  
        this.edad=edad;  
    }  
}
```

# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA



11. Herencia

## 11.- Herencia

- La herencia es una de las características más potentes en la Programación Orientada a Objetos, ya que nos permite que una clase herede los atributos y métodos de otra clase (*¡ojo!*: los constructores no se heredan)
- Esta característica garantiza la reutilización de código.

## 11.- Herencia

- Con la herencia se establece un sistema de jerarquía entre las clases, donde habrá clases que tengan una clase superior (llamada **superclase** o clase base) que tendrán **subclases** que heredan de ella.

*Ejemplo: En el gráfico Vehículo es la **superclase** y Coche, Camión y Moto son las **subclases** que heredan de la superior.*



## 11.- Herencia

- El termino heredar significa que las subclases disponen de todos los métodos y atributos de la superclase, extendiendo su funcionalidad. Además estas subclases podrán añadir atributos y métodos propios.
- A la hora de definir una clase que va a heredar de otra clase, se utiliza la palabra reservada **extends**, seguida del nombre de la superclase:

```
public class Subclase extends Superclase{  
    //código de la subclase  
}
```

## 11.- Herencia

- Como norma universal, cada vez que en JAVA se crea un objeto de una subclase, antes de ejecutarse el constructor de dicha subclase se ejecutará primero el de su superclass. Ejemplo sencillo:

```
package herencia;
/**
 * @author OLG
 */
public class Vehiculo {

    Vehiculo(){
        System.out.println("Constructor de la superclass");
    }
}
```

```
package herencia;
/**
 * @author OLG
 */
public class Coche extends Vehiculo {

    Coche(){
        System.out.println("Constructor de la subclase");
    }
}
```

```
package herencia;
/**
 * @author OLG
 */
public class Herencia {

    public static void main(String[] args) {
        Coche cocheVecino=new Coche();
    }
}

Output - Herencia (run) x
run:
Constructor de la superclass
Constructor de la subclase
BUILD SUCCESSFUL (total time: 0 seconds)
```

## 11.- Herencia

- La explicación es que JAVA añade, si no lo ha hecho el usuario, como primera línea de código la siguiente instrucción:

***super();*** //Esta línea provoca una llamada al constructor sin parámetros de la superclase

- Si en vez de llamar al constructor sin parámetros quisiésemos llamar a otro deberemos hacerlo explícitamente:

***super(argumento1, argumento2...);***//Llama al constructor correspondiente

## 11.- Herencia

- Vamos a hacer un ejemplo que contenga 4 clases: Vehículo, Coche (hereda de Vehículo), Camión (hereda de Vehículo) y Herencia (test para probar las clases y que contiene los métodos estáticos)

# 11.- Herencia

```
package herencia;
/**
 * @author OLG
 */
public class Vehiculo {

    private String marca;
    private String modelo;
    private int velocidad;
    private boolean motorEncendido;

    public Vehiculo(String marca, String modelo) {
        this.marca = marca;
        this.modelo = modelo;
        this.velocidad = 0;
        this.motorEncendido = false;
    }

    public void arrancarVehiculo() {
        this.motorEncendido = true;
        this.velocidad = 10;
    }

    public void apagarVehiculo() {
        this.motorEncendido = false;
        this.velocidad = 0;
    }

    public void acelerarVehiculo(int incremento) {
        if (this.motorEncendido == true) {
            this.velocidad = this.velocidad + incremento;
        }
    }

    public void frenarVehiculo(int decremento) {
        if (this.motorEncendido == true) {
            this.velocidad = this.velocidad - decremento;
        }
    }
}
```

```
public String obtenerMarca() {
    return marca;
}

public void establecerMarca(String marca) {
    this.marca = marca;
}

public String obtenerModelo() {
    return modelo;
}

public void establecerModelo(String modelo) {
    this.modelo = modelo;
}

public int obtenerVelocidad() {
    return velocidad;
}

public void establecerVelocidad(int velocidad) {
    this.velocidad = velocidad;
}

public boolean obtenerMotorEncendido() {
    return motorEncendido;
}

public void establecerMotorEncendido(boolean motorEncendido) {
    this.motorEncendido = motorEncendido;
}

public void mostrarEstado(){
    System.out.println("Marca: "+this.marca);
    System.out.println("Modelo: "+this.modelo);
    if (this.motorEncendido == true) {
        System.out.println("El motor esta encendido");
    } else{
        System.out.println("El motor esta apagado");
    }
    System.out.println("Velocidad: "+this.velocidad);
}
}
```

# 11.- Herencia

```
package herencia;  
/**  
 * @author OLG  
 */  
public class Coche extends Vehiculo {  
    private int numeroPuertas;  
  
    Coche(String marca, String modelo, int numeroPuertas){  
        super(marca, modelo);  
        this.numeroPuertas=numeroPuertas;  
    }  
  
    public int obtenerNumeroPuertas() {  
        return numeroPuertas;  
    }  
  
    public void establecerNumeroPuertas(int numeroPuertas) {  
        this.numeroPuertas = numeroPuertas;  
    }  
    public void mostrarEstado(){  
        super.mostrarEstado();  
        System.out.println("Número de puertas: "+this.numeroPuertas);  
    }  
}
```

```
package herencia;  
/**  
 * @author OLG  
 */  
public class Camion extends Vehiculo {  
    private int tonelaje;  
  
    Camion(String marca, String modelo, int tonelaje){  
        super(marca, modelo);  
        this.tonelaje=tonelaje;  
    }  
  
    public int obtenerTonelaje() {  
        return tonelaje;  
    }  
  
    public void establecerNumeroPuertas(int tonelaje) {  
        this.tonelaje = tonelaje;  
    }  
    public void mostrarEstado(){  
        super.mostrarEstado();  
        System.out.println("Capacidad de carga en toneladas: "+this.tonelaje);  
    }  
}
```

# 11.- Herencia

```
package herencia;
/*
 * @author OLG
 */
public class Herencia {

    public static void main(String[] args) {
        Coche cocheVecino=new Coche("Audi","R8",3);
        Camion camionPadre=new Camion("Mercedes","RailTrail",2000);

        cocheVecino.arrancarVehiculo();
        cocheVecino.acelerarVehiculo(50);
        cocheVecino.frenarVehiculo(10);
        cocheVecino.apagarVehiculo();

        camionPadre.acelerarVehiculo(40);
        camionPadre.arrancarVehiculo();
        camionPadre.acelerarVehiculo(90);
        camionPadre.frenarVehiculo(10);

        System.out.println(" --- Estado del coche del vecino: ");
        cocheVecino.mostrarEstado();
        System.out.println("\n --- Estado del camión de mi padre: ");
        camionPadre.mostrarEstado();
    }
}
```

Output - Herencia (run) x

```
--- Estado del coche del vecino:  
Marca: Audi  
Modelo: R8  
El motor esta apagado  
Velocidad: 0  
Número de puertas: 3  
  
--- Estado del camión de mi padre:  
Marca: Mercedes  
Modelo: RailTrail  
El motor esta encendido  
Velocidad: 90  
Capacidad de carga en toneladas: 2000
```

# JAVA

## ESTRUCTURA Y SINTAXIS DE UN PROGRAMA EN JAVA

12. Ejercicios de Consolidación



# EJERCICIOS

- **Ejercicio 20.- (OBLIGATORIO)** Se pretende desarrollar una aplicación que simule el funcionamiento de un cajero automático. Primeramente, se debe crear una clase llamada Cuenta, que gestione las operaciones sobre la cuenta. Además de los constructores y campos que se estimen necesarios, la clase contará con los métodos:

`void ingresar (float c) //Agrega al saldo de la cuenta la cantidad recibida.`

`void extraer(float c) //Descuenta del saldo la cantidad recibida. Tras la llamada a este método, el saldo podrá quedar en negativo.`

`float getSaldo() //Devuelve el saldo actual`

- Por otro lado, existirá una clase con el método main encargada de la captura y presentación de datos, y de la gestión de la cuenta. Al iniciarse la aplicación se mostrará el siguiente menú:

# EJERCICIOS

***1.- Crear cuenta vacía.***

***2.- Crear cuenta con saldo inicial.***

***3.- Ingresar dinero.***

***4.- Sacar dinero.***

***5.- Ver saldo.***

***6.- Salir.***

- La opción 1 crea un objeto Cuenta con saldo 0, la opción 2 solicita una cantidad y crea un objeto Cuenta con este saldo inicial. En la opción 3 se solicita una cantidad y la ingresa en el objeto creado en las opciones 1 o 2 (debe haber pasado antes por estas opciones), mientras que en la opción 4 se solicita una cantidad y la extrae del objeto creado en las opciones 1 o 2 (también debe haber pasado antes por estas opciones). Finalmente, la opción 5 muestra el saldo, mientras que la 6 finaliza el programa.
- El menú vuelve a presentarse en pantalla mientras no se elija la opción de salir.

# EJERCICIOS

- **Ejercicio 21.- (OBLIGATORIO)** Añade al ejercicio anterior una nueva clase llamada CuentaClave. Esta clase será una subclase de Cuenta y tendrá las siguientes características:
  - Incluirá un nuevo dato miembro llamado clave.
  - Sobrescribirá el método extraer(), de modo que sólo permita la extracción si hay saldo suficiente, sino no hará nada.
- En cuanto al funcionamiento del programa será igual que en el caso anterior, sólo que al elegir las opciones 1 y 2 para la creación de la cuenta, se pedirá también al usuario la clave que se le quiere asociar.
- No se enviará ningún tipo de aviso al usuario si se intenta sacar más dinero del que se dispone.

# EJERCICIOS

- **Ejercicio 22.- (OBLIGATORIO)** Desarrolle a un programa en JAVA en el que se creen 3 objetos. Cada objeto contiene como atributos el nombre de un producto de una tienda de mascotas, su precio y su stock.
- Utiliza un constructor para establecer el nombre de cada producto, su precio y su stock. Establécelos tú al crear el objeto, no hace falta que se los pidas al usuario.
- Realiza un método en el que le muestre al usuario un menú para que elija que producto comprar y luego le pregunte cuantas unidades desea de él. Luego se le preguntará si desea comprar otro producto o salir. Por último se le mostrará el importe total de la compra.
- Actualiza el valor del stock de un producto cuando el usuario lo compre. En caso de que el usuario pida más unidades de las que quedan se le avisará por pantalla del error, se le comunicarán las unidades restantes y le volverá a mostrar el menú para que elija un producto. Para ello, necesitarás un método para comprobar el stock, que devolverá true o false en función de si hay o no hay stock suficiente.

# EJERCICIOS

-- Bienvenido a mi Tienda de Mascotas --

Pulse 1 para comprar Peces cuyo precio es de 10 euros y el stock es de 40 unidades.  
Pulse 2 para comprar Tortugas cuyo precio es de 4 euros y el stock es de 8 unidades.  
Pulse 3 para comprar Canarios cuyo precio es de 15 euros y el stock es de 4 unidades.

1

Ha elegido comprar Peces.  
¿Cuántas unidades desea?

6

Venta Realizada con éxito.  
¿Desea comprar otro producto?  
si

Pulse 1 para comprar Peces cuyo precio es de 10 euros y el stock es de 34 unidades.  
Pulse 2 para comprar Tortugas cuyo precio es de 4 euros y el stock es de 8 unidades.  
Pulse 3 para comprar Canarios cuyo precio es de 15 euros y el stock es de 4 unidades.

2

Ha elegido comprar Tortugas.  
¿Cuántas unidades desea?

50

Lo sentimos, solo tenemos disponibles 8 unidades.  
¿Desea comprar otro producto?  
si

Pulse 1 para comprar Peces cuyo precio es de 10 euros y el stock es de 34 unidades.  
Pulse 2 para comprar Tortugas cuyo precio es de 4 euros y el stock es de 8 unidades.  
Pulse 3 para comprar Canarios cuyo precio es de 15 euros y el stock es de 4 unidades.

3

Ha elegido comprar Tortugas.  
¿Cuántas unidades desea?

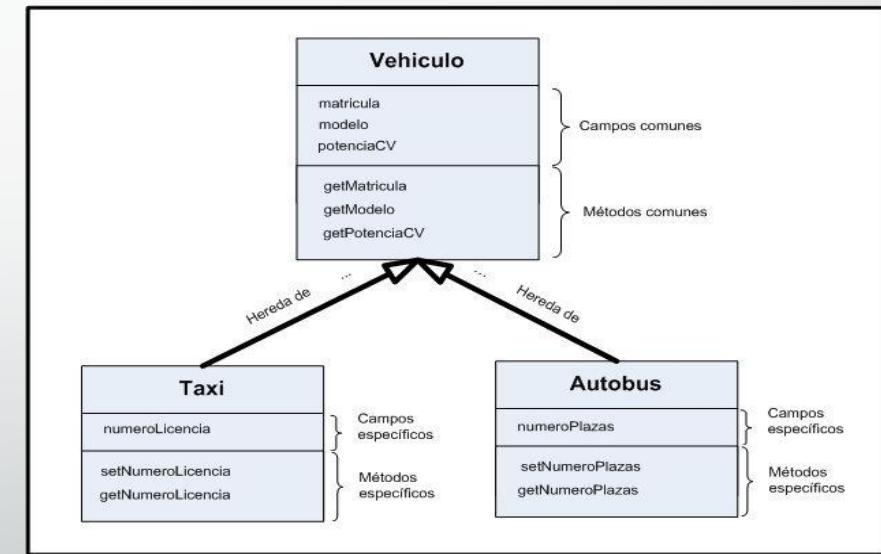
8

Venta Realizada con éxito.  
¿Desea comprar otro producto?  
no

El total de su compra asciende a **92** Euros.  
Muchas gracias. Vuelva cuando quiera.

# EJERCICIOS

- **Ejercicio 23.- (OPTATIVO)** Fíjate en el siguiente esquema, y basándote en él, crea un programa en Java de tal forma que cada clase debe disponer de constructor y permitir establecer (set) y recuperar (get) el valor de sus atributos. También crea un método que permita mostrar la información del objeto cuando sea procedente.
- Por otro lado, crea una clase test que contenga el método main. En ella crea un objeto taxi y un objeto autobús utilizando un constructor que reciba como parámetros los valores de los atributos.
- Crea otros 2 objetos, uno por cada clase, utilizando el constructor por defecto y, luego, utiliza los métodos creados para asignarle valores a sus atributos.
- Por último muestra la información de cada objeto por pantalla.



# EJERCICIOS

- **Ejercicio 24.- (OBLIGATORIO)** Se plantea desarrollar un programa Java que permita la gestión de una empresa agroalimentaria que trabaja con tres tipos de productos: productos frescos, productos refrigerados y productos congelados.
- Todos los productos llevan esta información común: fecha de caducidad y número de lote.
- A su vez, cada tipo de producto lleva alguna información específica:
  - i. Los productos frescos deben llevar la fecha de envasado y el país de origen.
  - ii. Los productos refrigerados deben llevar el código del organismo de supervisión alimentaria.
  - iii. Los productos congelados deben llevar la temperatura de congelación recomendada.
- Crear el código de las clases Java implementando una relación de herencia desde la superclase Producto hasta las subclases ProductoFresco, ProductoRefrigerado y ProductoCongelado.
- Cada clase debe disponer de constructor y permitir establecer (set) y recuperar (get) el valor de sus atributos y tener un método que permita mostrar la información del objeto.
- Crear una clase Test con el método main donde se cree un objeto de cada tipo y se muestren los datos de cada uno de los objetos creados.

# EJERCICIOS

- **Ejercicio 25.- (OBLIGATORIO)** Realiza un programa en JAVA que constará de las siguientes clases: Ordenador, Servidor (hereda de Ordenador), Portátil (hereda de Ordenador) y Test contiene el metodo main y otros métodos estáticos.
- La superclase Ordenador contiene los siguientes atributos: cantidad de memoria RAM, capacidad del disco duro, modelo de procesador, modelo de tarjeta gráfica y precio. A la hora de que el usuario cree un objeto, en la clase Test, deberemos controlar que la capacidad del disco duro sea múltiplo de 5, el modelo de procesador lo permitirá elegir entre una lista, y el precio siempre sea mayor que 0.
- La subclase Servidor contiene los siguientes atributos: tamaño del monitor, modelo de teclado y modelo de ratón. A la hora de que el usuario cree un objeto, en la clase Test, deberemos controlar que el tamaño del monitor sea mayor que 14. (en caso de que el usuario no lo cumpla deberemos volvérselo a pedir, tantas veces como sea necesario.)

# EJERCICIOS

- La subclase Portátil contiene los siguientes atributos: marca, tamaño de pantalla y peso.
- La clase Test contendrá los métodos estáticos necesarios para controlar que el usuario introduzca los datos correctos. Además, contiene el método main donde se crearán 2 objetos de la clase Servidor y 2 de la clase Portátil mediante un constructor que reciba los parámetros. También se crearán 1 objetos de la clase Servidor y 1 de la clase Portátil mediante el constructor por defecto, por lo que necesitarás llamar posteriormente a los métodos “establecer...” para asignar los valores que te indique el usuario a los atributos del objeto.
- Por último, muestra los datos de los 6 objetos creados.

# EJERCICIOS

- **Ejercicio 26.- (OPTATIVO)** El restaurante mejicano *Mr.Fajitas* cuya especialidad son las papas con chocos, nos pide diseñar un método con el que se pueda saber cuántos clientes pueden atender con la materia prima que tienen en el almacén. El método recibe la cantidad de papas y chocos en kilos y devuelve el número de clientes que puede atender el restaurante teniendo en cuenta que por cada tres personas, *Mr.Fajitas* utiliza un kilo de papas y medio de chocos.

# EJERCICIOS

- **Ejercicio 27.- (OPTATIVO)** Modifica el programa anterior orientándolo a objetos, creando una clase Almacen que permita almacenar los kilos de papas y chocos del restaurante. Implementa los siguientes métodos:
  - public void addChocos(int x): Añade x kilos de chocos a los ya existentes.
  - public void addPapas(int x): Añade x kilos de papas a los ya existentes.
  - public int getComensales(): Devuelve el número de clientes que puede atender el restaurante (este es el método anterior)
  - public void showChocos(): Muestra por pantalla los kilos de chocos que hay en el almacén.
  - public void showPapas(): Muestra por pantalla los kilos de papas que hay en el almacén.
  - Implementa una clase Test para crear un objeto y probar el funcionamiento del programa.

# EJERCICIOS

- **Ejercicio 28.- (OPTATIVO)** Implementa una clase Consumo, la cual forma parte de la centralita electrónica de un coche y tiene las siguientes características:
  - Atributos:
    - kms. Kilometros recorridos por el coche.
    - litros. Litros de combustible consumido.
    - vmed. Velocidad media.
    - pgas. Precio de la gasolina
  - Métodos:
    - getTiempo. Indicará el tiempo empleado en realizar el viaje.
    - consumoMedio. Consumo medio del vehículo (en litros cada 100 kilómetros)
    - consumoEuros. Consumo medio del vehículo (en euros cada 100 kilómetros)
    - Los métodos establecerXXX y obtenerXXX para cada atributo.
  - No olvides crear un constructor para la clase que reciba el valor de los atributos y se los asigne.
  - Por último crea una clase Test en la que crearás un objeto con los datos del viaje que te introduzca el usuario, y le mostrarás el tiempo empleado en realizar el viaje, el consumo medio en litros y el consumo medio en euros.

# EJERCICIOS

- **Ejercicio 29.- (OPTATIVO)** Realice una clase, de nombre Examen, para guardar información sobre los exámenes de un centro educativo. La información que se guarda de un examen es: el nombre de la asignatura, el aula, la fecha y la hora. La fecha y la hora serán dos Strings con el formato dd/mm/aaaa, y hh:mm respectivamente.
- Por otro lado tendrás una clase Test que contendrá el método main donde crearás 3 objetos de la clase Examen, pidiéndole datos al usuario y pasándoselos a un constructor por parámetros. Además, esta clase contendrá diversos métodos para pedir los datos al usuario y comprobar su veracidad: que día esté entre 1 y 31, mes entre 1 y 12, año entre 1900 y 2016, hora entre 0 y 23, y minuto entre 0 y 60. En caso de que el usuario se equivoque al introducir un dato, se lo volveremos a pedir las veces que hagan falta. Por último, mostrarás la información de los objetos creados.

# EJERCICIOS

- **Ejercicio 30.- (OPTATIVO)** Realice una clase, de nombre Examen, para guardar información sobre los exámenes de un centro educativo. La información que se guarda de un examen es: el nombre de la asignatura, el aula, la fecha y la hora. Para guardar la fecha y la hora hay que realizar dos clases, Fecha y Hora.
- La clase Fecha guarda día, mes y año. Todos los valores se reciben en el constructor por parámetros. Además, esta clase debe tener un método que devuelva cada uno de los atributos, y un método mostrarConFormato que devuelva la información de una fecha en un String con el formato dd/mm/aaaa.
- La clase Hora guarda la hora y el minuto. También recibe los valores para los atributos por parámetro en el constructor, tiene métodos que devuelven cada uno de los atributos, y un método mostrarConFormato que devuelva la información de una fecha en un String con el formato hh:mm.
- Por otro lado, tendrás una clase Test. Esta contendrá el método main donde crearás 3 objetos de la clase Examen, pidiéndole datos al usuario y pasándoselos a un constructor por parámetros. Además, esta clase contendrá diversos métodos para pedir los datos al usuario y comprobar su veracidad: que día esté entre 1 y 31, mes entre 1 y 12, año entre 1900 y 2016, hora entre 0 y 23, y minuto entre 0 y 60. En caso de que el usuario se equivoque al introducir un dato, se lo volveremos a pedir las veces que hagan falta. Por último, mostrarás la información de los objetos creados utilizando los métodos de mostrarConFormato para la fecha y la hora.

# EJERCICIOS

- **Ejercicio 31.- (OBLIGATORIO)** Realice un método recursivo en JAVA que escriba “Hola” 5 veces por pantalla.

# EJERCICIOS

- **Ejercicio 32.- (OPTATIVO)** Realice un método recursivo que sume 2 números.

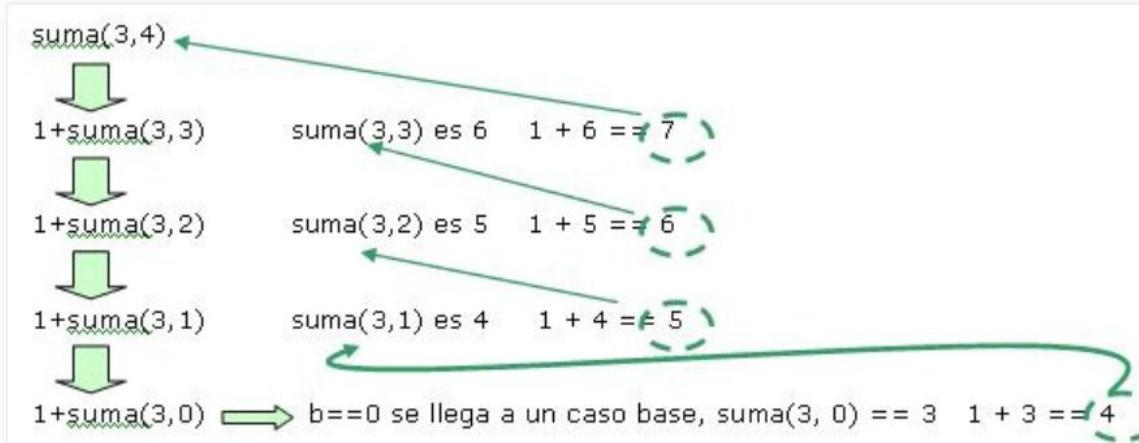
Por ejemplo:

Dados los números  $a=3$  y  $b = 4$

la suma de  $3 + 4$  es igual que sumar  $1 + (3 + 3)$

A su vez, sumar  $3 + 3$  es igual que  $1 + (3 + 2)$

Si repetimos el proceso hasta que  $b$  sea 0 obtendremos la suma de forma recursiva:



# EJERCICIOS

- **Ejercicio 33.- (OPTATIVO)** Realice un método recursivo que realice la suma de los primeros N enteros.
- Ejemplo:

*Introduzca un entero para calcular la suma desde 1 hasta él:*

5

*La suma es 15.  $(5+4+3+2+1)$*

# EJERCICIOS

- **Ejercicio 34.- (OPTATIVO)** Programar un método recursivo que permita invertir un número.
- Ejemplo:

*Introduzca un entero para invertirlo:*

**125**

*El número invertido es el 521.*

# EJERCICIOS

- **Ejercicio 35.- (OPTATIVO)** Realice un método recursivo que permita sumar los dígitos de un número.
- Ejemplo:

*Introduzca un entero para calcular la suma:*

**538**

*La suma es 16.*