Universidad de La Laguna

Tema 4. Elementos de programación modular. Programación Orientada a Objetos

CHRISTOPHER EXPÓSITO IZQUIERDO
AIRAM EXPÓSITO MÁRQUEZ
ISRAEL LÓPEZ PLATA
MARÍA BELÉN MELIÁN BATISTA
JOSÉ MARCOS MORENO VEGA

La programación modular

- Se considera modularidad como la definición de un programa como un conjunto de subprogramas (**módulos**)

- Cada módulo tiene una función particular, es semi-independiente e incluso puede encontrarse en ficheros de código distintos

- Ventajas:
  - El código resultante es mas limpio y fácil de entender
  - Si los módulos están bien diseñados son sencillos de reutilizar a lo largo del código
  - Permite la división de trabajo
  - Se pueden probar de forma independiente, mejorando así la búsqueda y corrección de errores
  - Facilita la definición de variables, al crear distintos ámbitos

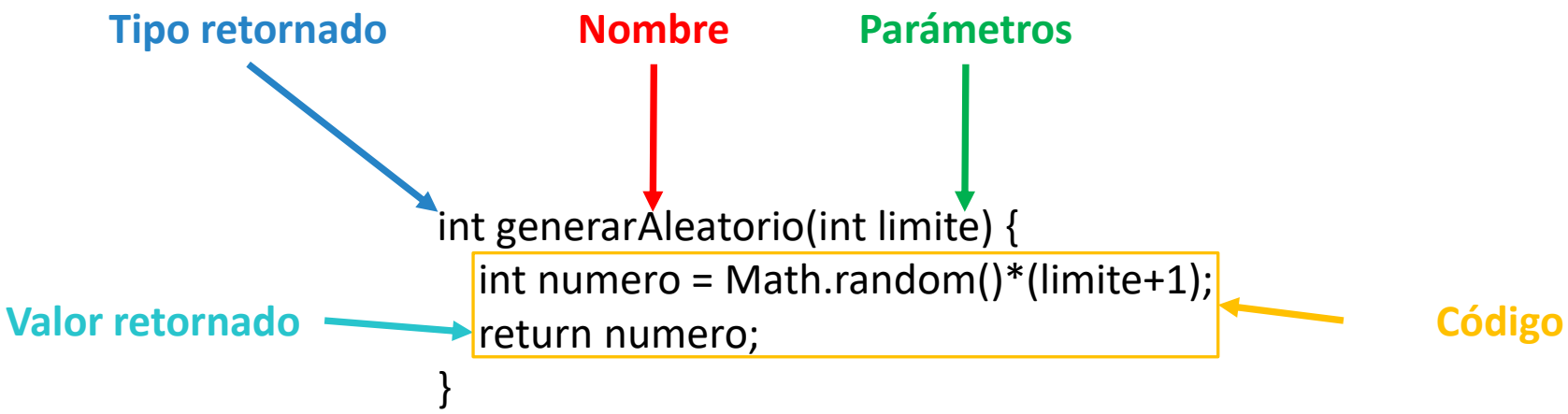
La programación modular

- Dependiendo del nivel de detalle con el que se observan los módulos, podemos distinguir 3 tipos:
  - Librerías
  - Clases
  - Funciones

- Una función es un *trozo* de código que realiza una función concreta y, en ocasiones, devuelve un valor como resultado

- Una función posee una declaración y una o varias llamadas
  - En la declaración se escribe el código de la función
  - El compilador sustituye la llamada a la función por el código de la misma

Estructura de una función



Nombre de una función

- Es el nombre que va a identificar de forma unívoca a la función
- Se utilizará posteriormente para llamar a la función
- Debe seguir una serie de directrices:
  - Debe describir lo que hace la función
  - Debe empezar por minúscula, sin espacios. Se separa cada palabra mediante mayúsculas. Ejemplos:
    - suma
    - imprimirPantalla
    - generarAleatorio

Tipo de una función

- Una función puede retornar un valor como resultado de sus operaciones

- El tipo de ese valor se indica al inicio de la declaración de la función, y puede ser cualquier tipo de datos utilizado en Java

- Si la función no devuelve ningún valor, se debe indicar que es de tipo **void** (vacío)

- A excepción de las tipo void, el resto de las funciones deben indicar el valor que devuelven y este debe ser del tipo indicado. Ejemplos:

```
void imprimirPantalla() {
   System.out.println(“Hola a todos!”);
}
```

```
int generarAleatorio(int limite) {
    int numero = Math.random()*(limite+1);
    return numero;
}
```

Parámetros de una función

- Conjunto de valores que necesita una función y que proceden de otros módulos del programa
- Sirven para establecer una comunicación entre los diferentes módulos del programa
- En la declaración se indica que parámetros y de que tipo necesita la función, en un orden determinado y separados por comas. En la llamada, se obliga a proporcionarle datos a la función del mismo tipo y orden en el que se han declarado
- Si la función no necesita datos de otros módulos, los parámetros se indican como un paréntesis vacío

```
void imprimirPantalla() {
   System.out.println(“Hola a todos!”);
}
```

Parámetros de una función

- Una función viene definida por su nombre y parámetros

- 2 funciones con el mismo nombre pero con parámetros distintos se consideran distintas

- 2 funciones con el mismo nombre, mismos parámetros pero en distinto orden, se consideran distintas

Distintas

```
void imprimir(String cadena, int numero) {
   System.out.println(cadena + " " + numero);
}
```

```
void imprimir(int numero, String cadena) {
   System.out.println(cadena + " " + numero);
}
```

Iguales

```
int sumar(int a, int b) {
   int suma = a + b;
   return suma;
}
```

```
int sumar(int b, int a) {
   int suma = a + b;
   return suma;
}
```

Tipos de pasos por parámetros

- **Por valor:** En este caso, el parámetro que le llega a la función es **una copia** del valor del parámetro con el que se llama

- Implica que si la función modifica el parámetro, sólo la copia cambia y el valor original permanece intacto

- **Por referencia:** En este caso, el parámetro que le llega a la función es **la dirección de memoria** del parámetro con el que se llama

- Implica que si la función modifica el parámetro, el valor original en el código que llamó a la función cambia

Tipos de pasos por parámetros

- En Java, los parámetros siempre son pasados **por valor**

Ejemplo

```
void sumar(int a, int b) {
   a = a + b;
   System.out.println("Dentro de la función a = " + a);
}

public static void main(String args[]) {
   int a = 4;
   int b = 5;
   System.out.println("Antes de la función a = " + a);
   sumar(a, b);
   System.out.println("Después de la función a = " + a);
}
```

```
Antes de la función a = 4
Dentro de la función a = 9
Después de la función a = 4
```

Posibles parámetros de una función

- Java permite definir parámetros de una función de cualquier tipo básico
  - int
  - double
  - float
  - char
  - String
  - …
- También se permiten estructuras de datos de cualquier tipo
- Un objeto puede ser pasado por parámetro (mas adelante)

Cuerpo de una función

- Conjunto de sentencias que componen la función
- Dentro del cuerpo se tiene acceso a las variables declaradas en el propio cuerpo así como a los parámetros
- En el cuerpo de una función se pueden incluir cualquier sentencia de programación de Java

Valor de retorno

- Toda función, excepto las que retornan **void**, deben devolver algún valor
- La sentencia que indica la devolución de un valor es **return**
- Todo flujo de una función debe terminar en return, y el valor retornado debe ser del tipo de la función
- El valor del return se puede aprovechar fuera de la función

Valor de retorno

```
int sumar(int a, int b) {
   int suma = a + b;
   return suma;
}
```

```
String saludo(boolean caeBien) {
   if (caeBien) {
      return “Hola amigo!!!!”;
   } else {
      return “Eh!”;
   }
}
```

```
double dividir(double dividendo, double divisor) {
   if (divisor != 0) {
      return dividendo / divisor;
   }
}
```

Valor de retorno

- El valor retornado por una función se puede asignar a una variable para ser utilizado fuera de la misma. El tipo de esa variable debe ser igual al tipo de la función.

Ejemplo

```
void sumar(int a, int b) {
   a = a + b;
   System.out.println(“Dentro de la función a = “ + a);
}

public static void main(String args[]) {
   int a = 4;
   int b = 5;
   System.out.println(“Antes de la función a = “ + a);
   int res = sumar(a, b);
   System.out.println(“Después de la función a = “ + a);
   System.out.println(“Resultado = “ + res);
}
```

```
Antes de la función a = 4
Dentro de la función a = 9
Después de la función a = 4
Resultado = 9
```

Funciones. Ámbito de variables

- Las variables en una función tienen un ámbito **local**. Una variable definida en una función no puede ser vista fuera de la función.

```
int mostrarValor(int a) {
   int b = 4;
   System.out.println("A en funcion es = " + a);
   System.out.println("B en funcion es = " + b);
   System.out.println("C en funcion es = " + c);
   return b;
}

public static void main(String args[]) {
   int a = 1;
   int b = 2;
   int c = 5;
   System.out.println("A antes es = " + a);
   System.out.println("B antes es = " + b);
   a = mostrarValor(b);
   System.out.println("A después es = " + a);
   System.out.println("B después es = " + b);
}
```

```
A antes es = 1
B antes es = 2
A en función es = 2
B en función es = 4
C en función es = ERROR
A después es = 4
B después es = 2
```

La función main

- Es la función que todo programa Java debe tener

- Es el punto de entrada a la aplicación

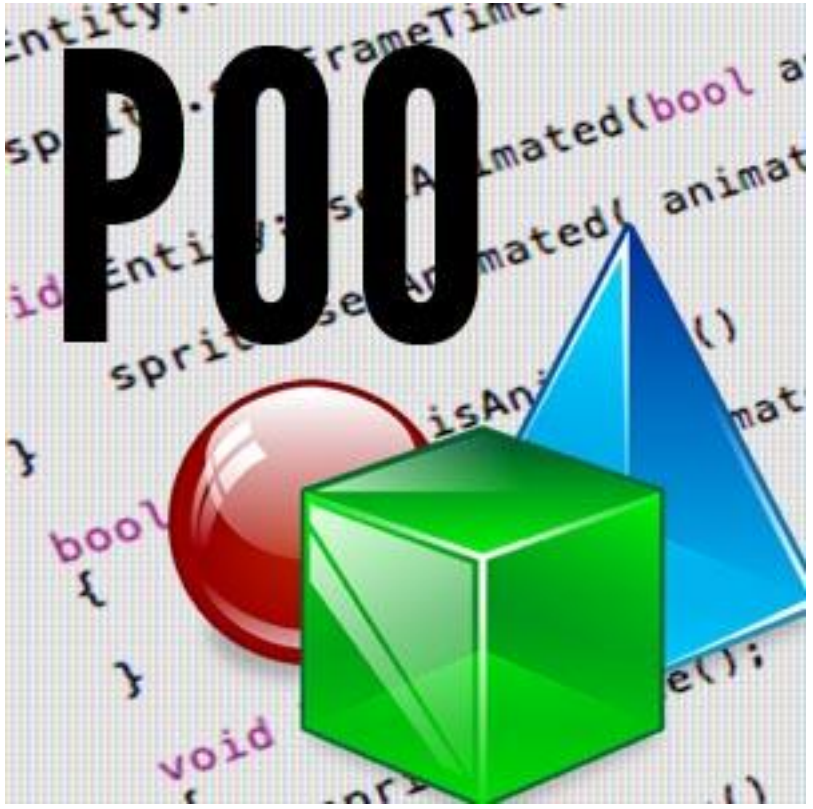
```
public static void main(String args[]) {
```

- Recibe un vector de String, que representa una serie de parámetros que se le pueden pasar a un programa Java

```
java EjemploJava David Charles Young
```

Programación Orientada a Objetos

PROGRAMACIÓN ORIENTADA A OBJETOS



Programación Orientada a Objetos

- La Programación Orientada a Objetos es un **paradigma de programación** en el que todo el código del software creado se organiza en forma de **objetos**

- Supone un nivel de abstracción superior al proporcionado por la programación estructurada clásica

- Incluye todas las características de la programación estructurada, ampliándolas con el concepto de objeto

- Java está basado en Programación Orientada a Objetos

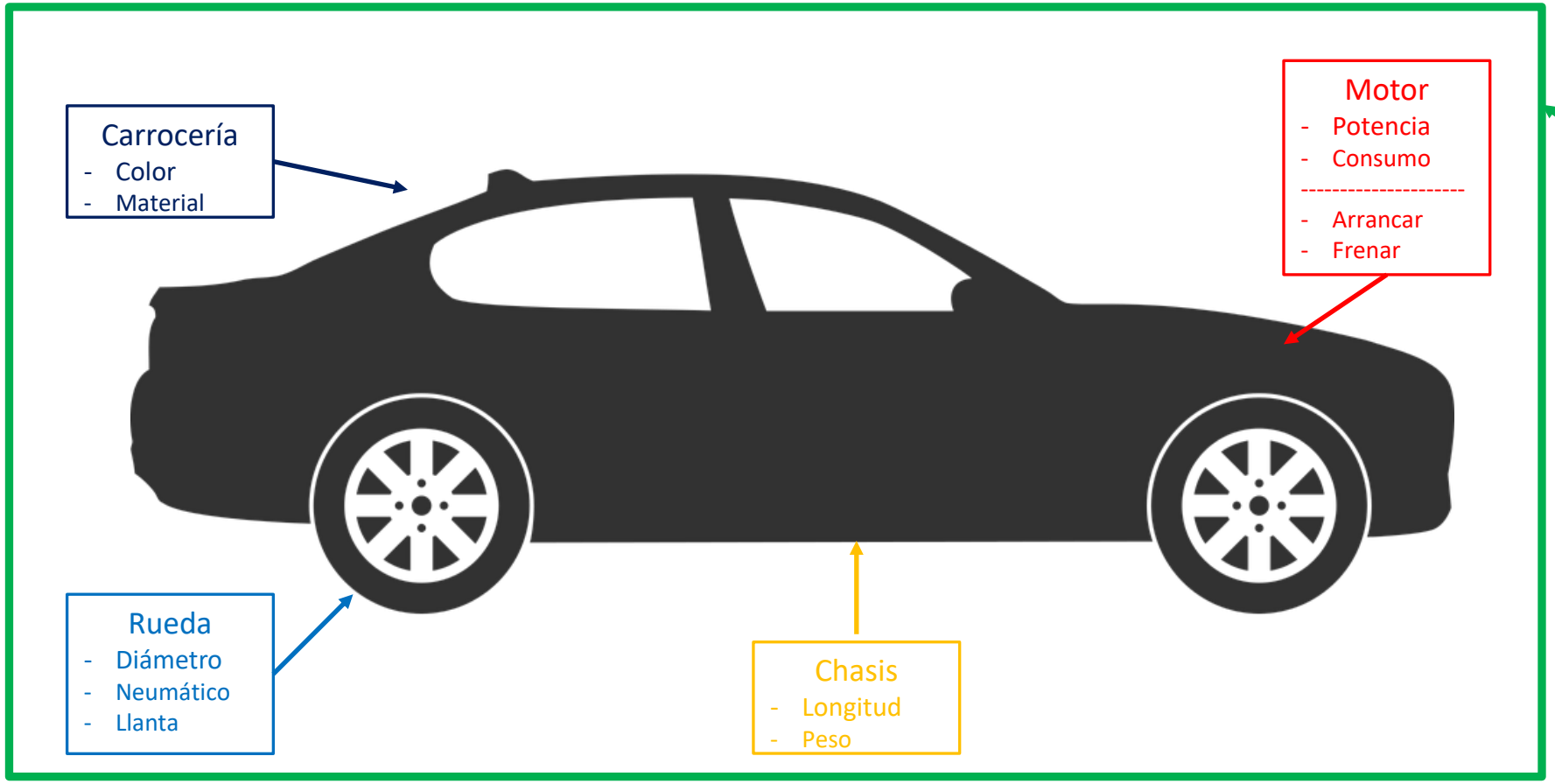
El objeto

- Un objeto es una entidad semi-independiente dentro del software

- Para todo objeto, se definen 3 características:
  - Nombre
  - Características
  - Comportamiento

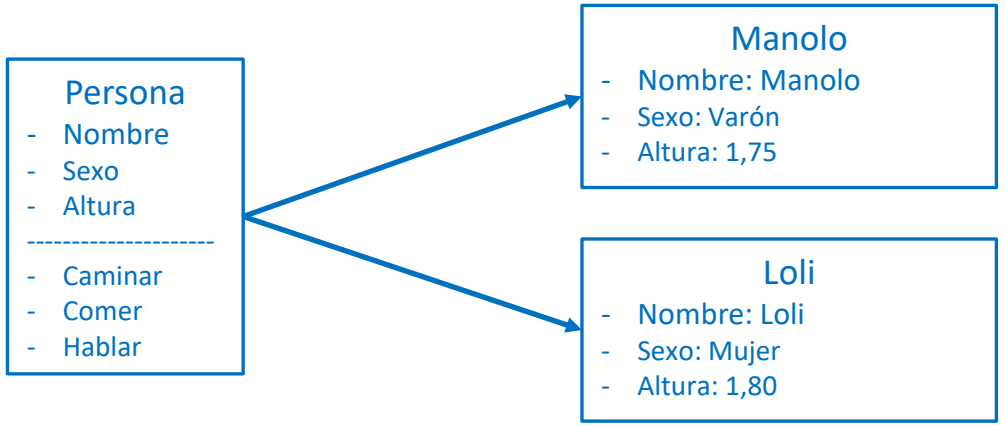
- Ejemplo. Ser humano.
  - Nombre: Paco
  - Características: Altura = 1,80 m, Pelo = Moreno, Ojos = Marrones
  - Comportamiento: Hablar, Caminar, Pensar, Comer

Composición de un objeto



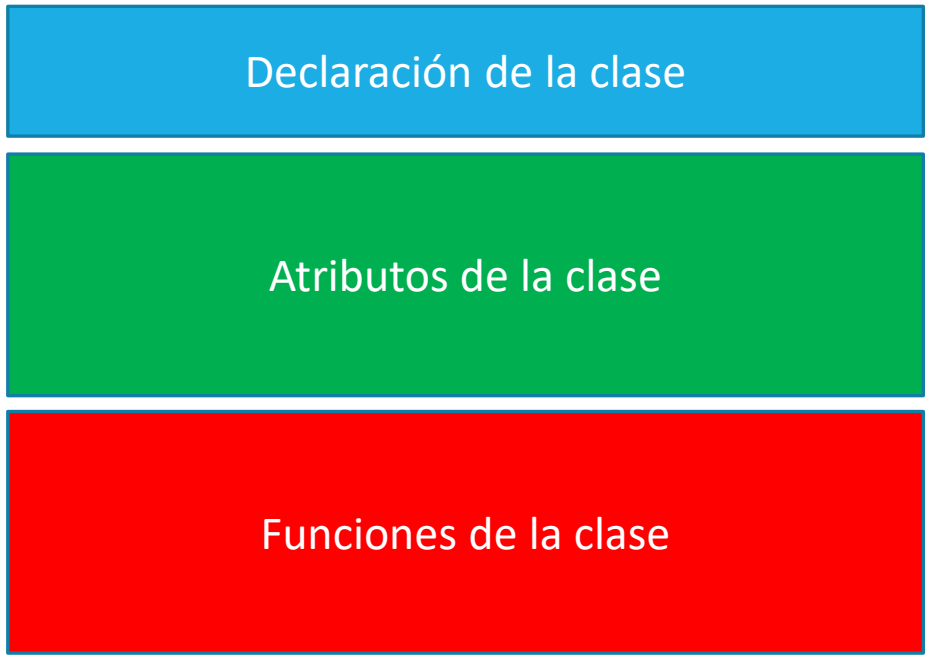
La clase

- La definición de un objeto se hace en las llamadas **clases**
- En una clase se debe definir:
  - Nombre
  - Atributos (Características)
  - Funciones (Comportamiento)
- Un objeto es un caso particular de una clase
- Para crear un nuevo objeto, se hace uso de la sentencia **new NombreClase();**, la cual llama al constructor de la clase (se verá mas adelante)
- Ejemplo:



Definición de clase

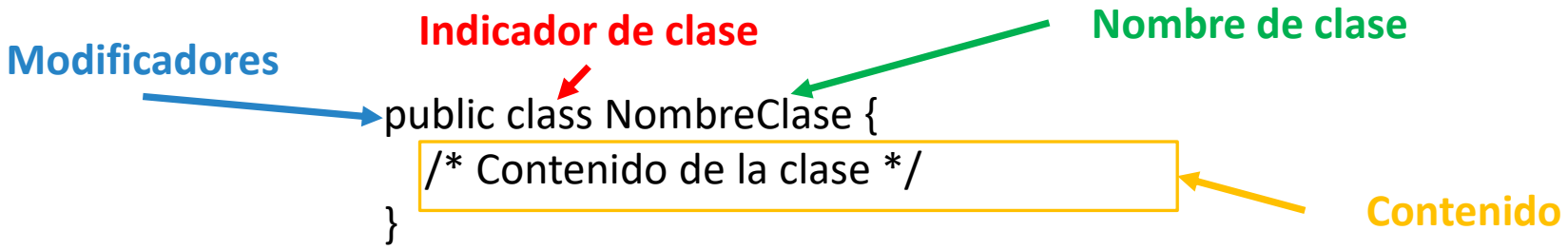
- La definición de una clase sigue este esquema:



- En Java, se crea un fichero por cada clase que se quiera definir, con el mismo nombre de la clase

Declaración de una clase

- Indica las características principales de una clase



- Los elementos en la declaración de una clase son los siguientes:
  - **Modificadores:** Indican el tipo de la clase. En Java básico, usar siempre public
  - **Indicador de clase:** Palabra fija *class*. Indica que lo que definimos es una clase
  - **Nombre de clase:** Nombre único que se le da a la clase. En Java, el nombre **siempre** empieza por mayúscula y describe lo que hace la clase

Atributos de una clase

- Son los datos del objeto que se quiere crear

- En una clase se pueden definir tantos atributos como se quiera y del tipo que se quiera, ya sean básicos, estructuras de datos u otros objetos

- Para acceder a un atributo de una clase, se hace mediante **nombreObjeto.nombreAtributo**

```
public class ClaseEjemplo {
   public int numero;
   public String cadena;
   public int[] vector;
}
```

```
ClaseEjemplo objeto = new ClaseEjemplo();
objeto.numero = 3;
String nuevaCadena = objeto.cadena;
objeto.vector[0] = 2;
```

Funciones de una clase

- Son un conjunto de funciones que puede realizar el objeto que define la clase
- Las funciones se definen tal y como hemos visto en apartados anteriores, añadiendo un modificador (private, public, etc.) al inicio
- Se puede acceder a los atributos del objeto desde las funciones sin pasarlos por parámetros
- Al igual que los atributos, se llaman a las funciones con **nombreObjeto.nombreFuncion**

```
public class ClaseEjemplo {
   public int numero;
   public String cadena;
   public int[] vector;

   public int sumarANumero(int otro) {
      return numero + otro;
   }

   public String concatenar(String nuevaCadena) {
      return cadena + " " + nuevaCadena;
   }
}
```

```
ClaseEjemplo objeto = new ClaseEjemplo();
objeto.numero = 4;
objeto.cadena = "Hola"
System.out.println(objeto.sumarANumero(3));
System.out.println(objeto.concatenar("Clase"));
```

```
7
Hola Clase
```

Constructor

- Función especial que debe tener toda clase en Java. Pueden existir uno o varios constructores, cambiando los parámetros que recibe
- Tiene como finalidad crear un nuevo objeto de la clase
- En un constructor se obliga a inicializar todos los atributos de la clase
- Tiene exactamente el mismo nombre de la clase

```
public class ClaseEjemplo {
   public int numero;
   public String cadena;
   public int[] vector;

   public ClaseEjemplo() {
      numero = 0;
      cadena = “”;
      vector = new int[10];
   }

public ClaseEjemplo(int num, int cad, int[] vec) {
      numero = num;
      cadena = cad;
      vector = vec;
   }
}
```

```
ClaseEjemplo objeto = new ClaseEjemplo();
objeto.numero; //Vale 0
objeto.cadena; //Vale “”
objeto.vector; //Vale un vector de 10 elementos


int[] vector = new int[] {1, 2, 3};
ClaseEjemplo objeto = new ClaseEjemplo(2, “Hola”, vector);
objeto.numero; //Vale 2
objeto.cadena; //Vale “Hola”
objeto.vector; //Vale un vector de 3 elementos = [1, 2, 3]
```

Modificador

- Tanto la clase, como los atributos o las funciones tienen un modificador que indica la **visibilidad**
- Este modificador puede ser **public, private** o **protected**
- Un elemento con atributo **public** significa que es accesible desde fuera de la clase
- Un elemento con atributo **private** significa que sólo es accesible desde dentro de la clase

```
public class ClaseEjemplo {
   public int numero;
   private String cadena;
   public int[] vector;

   public ClaseEjemplo() {
      numero = 0;
      cadena = “”;
      vector = new int[10];
   }
}
```

```
ClaseEjemplo objeto = new ClaseEjemplo();
objeto.numero; //Vale 0
objeto.cadena; //ERROR, no accesible
objeto.vector; //Vale un vector de 10 elementos
```

Modificador

- Por norma general, todos los atributos se definen como private
- Si se quiere acceder a un atributo privado, se suelen crear funciones para acceder a ellos:
  - Setter: Sirve para darle un valor a un atributo privado
  - Getter: Sirve para obtener el valor de un atributo privado

```
public class ClaseEjemplo {
   private int numero;

   public int getNumero() {
      return numero;
   }

   public void setNumero(int nuevoNumero) {
      numero = nuevoNumero;
   }

}
```

```
ClaseEjemplo objeto = new ClaseEjemplo();
objeto.setNumero(3);
objeto.getNumero(); //Devuelve un 3
objeto.numero; //Devuelve ERROR
```

El valor null

- En Java existe un valor que pueden obtener todos los objetos que es **null**
- Este valor representa que la variable apunta a un objeto que no existe de momento
- Permite tener variables de objetos de diferentes tipos sin haber aún ejecutado el constructor
- Si se intenta acceder a un atributo/función de una variable cuyo objeto es null, el programa dará un error conocido como **NullPointerException**

Ejemplo de una clase Java

```
public class Rectangulo {

   private int ancho;
   private int largo;

   public Rectangulo() {
       ancho = 0;
       alto = 0;
   }

   public Rectangulo(int otroAncho, int otroAlto) {
       ancho = otroAncho;
       alto = otroAlto;
   }

   public int getAncho() {
      return ancho;
   }

   public int getAlto() {
      return alto;
   }

   public int perimetro() {
      return ancho*2 + alto*2;
   }

   public int area() {
      return ancho*alto;
   }

}
```

Ejemplo de una clase Java

```
public class Coche {

   private String marca;
   private String modelo;
   private double precio;

   private Carroceria carrocería;
   private Chasis chasis;
   private Motor motor;
   private Rueda[] ruedas;

   public Coche() {
       marca = “Genérico”;
       modelo = “General”;
       carrocería = new Carroceria();
       chasis = new Chasis();
       motor = new Motor();
       ruedas = new Rueda[4];
   }

   public void abrir() {
      //Función de apertura de la puerta
   }

   public void arranchar() {
      //Función de arrancar el motor
   }

}
```

Librerías

- El definir todo como clases, facilita la creación de librerías
- Una librería es un conjunto de clases destinadas a hacer una funcionalidad completa
- Permite una modularización muy alta del código, así como una forma de compartir código entre distintos programadores
- Java posee un gran número de librerías, ya sea internas como desarrolladas por programadores externos

Librerías

- Para importar una librería, se debe incluir dentro del proyecto de Java

- Para utilizar una clase de una librería, se utiliza la sentencia **import**

- Ejemplo:

```
import java.io.PrintWriter;
import java.util.Arrays;
```

Librerías

Estructuras de datos:

- Estructuras de datos explicadas en el tema 3 y muchas mas
  - **Vector dinámico:** ArrayList
  - **Mapa:** HashMap
  - **Conjunto:** HashSet

Modificación de ficheros:

- Librería File
- La veremos en el siguiente tema

Librerías

Pruebas unitarias (JUnit):

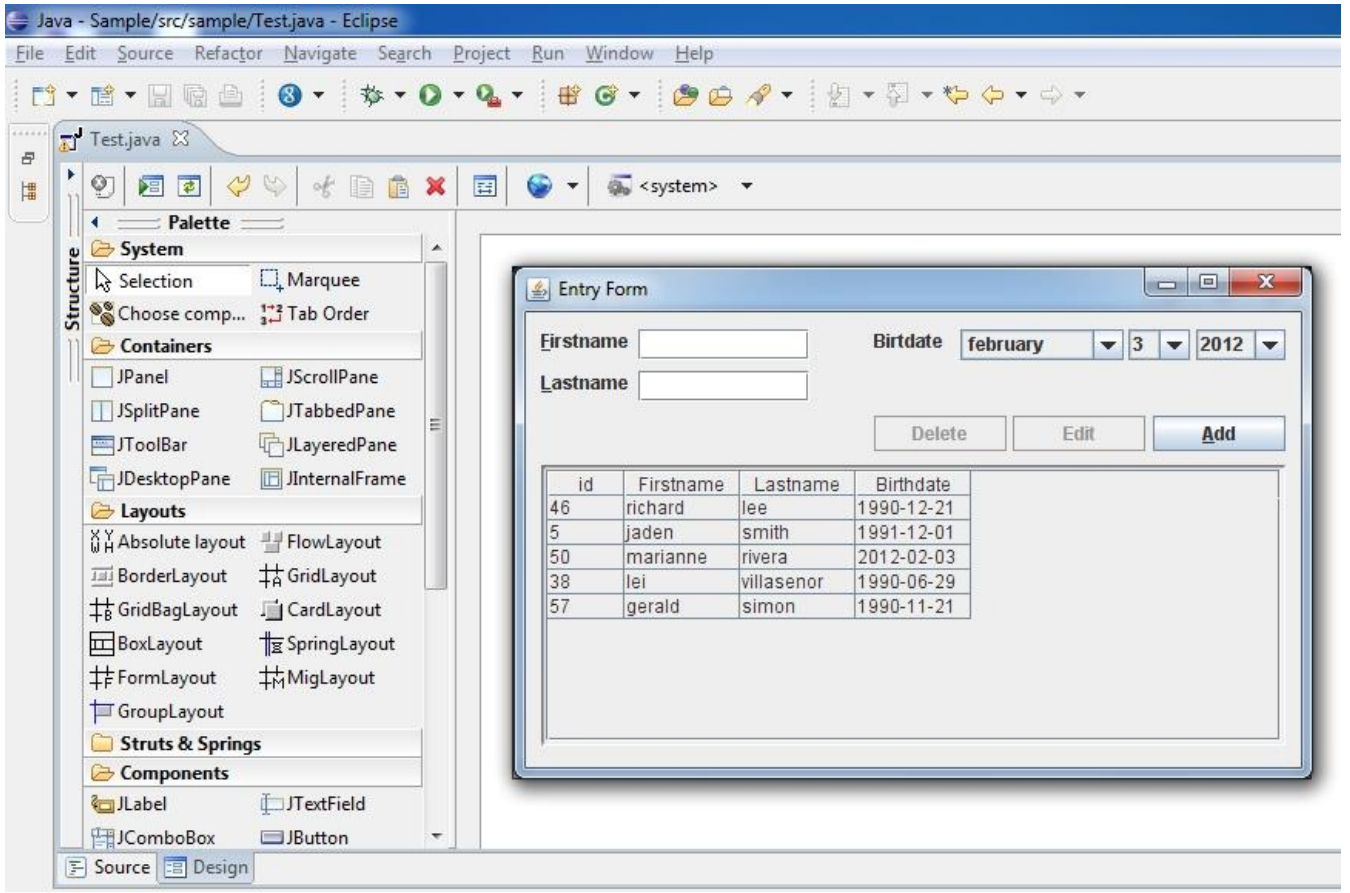
Coverage Report - All Packages

Package	# Classes	Line Coverage		Branch Coverage		Complexity
All Packages	221	84%	2970/3513	81%	859/1060	1.727
junit.extensions	6	82%	52/63	87%	7/8	1.25
junit.framework	17	76%	399/525	90%	139/154	1.605
junit.runner	3	49%	77/155	41%	23/56	2.225
junit.textui	2	76%	99/130	76%	23/30	1.686
org.junit	14	85%	196/230	75%	68/90	1.655
org.junit.experimental	2	91%	21/23	83%	5/6	1.5
org.junit.experimental.categories	5	100%	67/67	100%	44/44	3.357
org.junit.experimental.max	8	85%	92/108	86%	26/30	1.969
org.junit.experimental.results	6	92%	37/40	87%	7/8	1.222
org.junit.experimental.runners	1	100%	2/2	N/A	N/A	1
org.junit.experimental.theories	14	96%	119/123	88%	37/42	1.674
org.junit.experimental.theories.internal	5	88%	98/111	92%	39/42	2.29
org.junit.experimental.theories.suppliers	2	100%	7/7	100%	2/2	2
org.junit.internal	11	94%	149/157	94%	53/56	1.947
org.junit.internal.builders	8	98%	57/58	92%	13/14	2
org.junit.internal.matchers	4	75%	40/53	0%	0/18	1.391
org.junit.internal.requests	3	96%	27/28	100%	2/2	1.429
org.junit.internal.runners	18	73%	306/415	63%	82/130	2.155
org.junit.internal.runners.model	3	100%	26/26	100%	4/4	1.5
org.junit.internal.runners.rules	1	100%	35/35	100%	20/20	2.111
org.junit.internal.runners.statements	7	97%	92/94	100%	14/14	2
org.junit.matchers	1	9%	1/11	N/A	N/A	1
org.junit.rules	20	89%	203/226	96%	31/32	1.444
org.junit.runner	12	93%	150/161	88%	30/34	1.378
org.junit.runner.manipulation	9	85%	36/42	77%	14/18	1.632
org.junit.runner.notification	12	100%	98/98	100%	8/8	1.162
org.junit.runners	16	98%	321/327	96%	95/98	1.737
org.junit.runners.model	11	82%	163/198	73%	73/100	1.918

Report generated by Cobertura 1.9.4.1 on 12/22/12 2:25 PM.

Librerías

Interfaz de usuario (Swing):



Librerías

Desarrollo para móviles:

Librerías

Y mucho mas: