Junta de Andalucía
Consejería de Educación y Deporte

UT 4

Objetos

–

Módulo de Programación
1º DAW

Autor: Fran Gómez
2024/2025

Índice de contenidos resumido

¿Qué se va a evaluar?

RA 4 Desarrolla programas organizados en clases analizando y aplicando los principios de la programación orientada a objetos.	10	a) Se ha reconocido la sintaxis, estructura y componentes típicos de una clase.
		b) Se han definido clases.
		c) Se han definido propiedades y métodos.
		d) Se han creado constructores.
		e) Se han desarrollado programas que instancien y utilicen objetos de las clases creadas anteriormente.
		f) Se han utilizado mecanismos para controlar la visibilidad de las clases y de sus miembros.
		g) Se han definido y utilizado clases heredadas.
		h) Se han creado y utilizado métodos estáticos.
		i) Se han definido y utilizado interfaces.
		j) Se han creado y utilizado conjuntos y librerías de clases.

¿Qué se va a evaluar?

RA 7 Desarrolla programas aplicando características avanzadas de los lenguajes orientados a objetos y del entorno de programación.	10	a) Se han identificado los conceptos de herencia, superclase y subclase.
		b) Se han utilizado modificadores para bloquear y forzar la herencia de clases y métodos.
		c) Se ha reconocido la incidencia de los constructores en la herencia.
		d) Se han creado clases heredadas que sobrescriban la implementación de métodos de la superclase.
		e) Se han diseñado y aplicado jerarquías de clases.
		f) Se han probado y depurado las jerarquías de clases.
		g) Se han realizado programas que implementen y utilicen jerarquías de clases.
		h) Se ha comentado y documentado el código.

¿Cómo se va a evaluar?

- Examen → en clase (hay que aprobarlo)
- Práctica → en la empresa

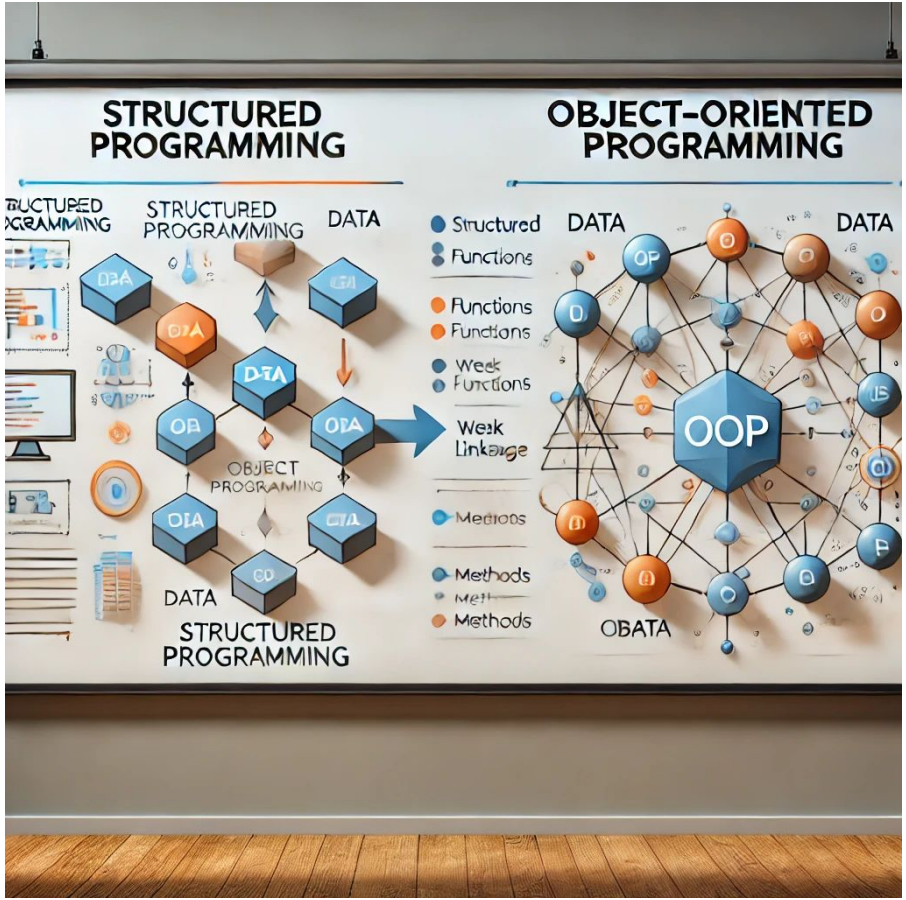
- Calificación = 50% Examen + 50% Práctica

Ejemplo:

Examen = 5, Práctica = 10 → calificación = 7.5

Examen = 4, Práctica = 10 → calificación = 4 (ya que no se ha aprobado el examen)

Introducción



funciones / datos

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

a = 2
b = 3
resultado = 5

La POO es un paradigma de programación que trata de modelar de manera abstracta el mundo real.

Clases: concepto

- Modelar el mundo real
- Plantilla
- Propiedad
  Comportamiento
- Datos + Código
- Atributos y Métodos
- Tipo de dato

Clase Vaca ⇒ Objetos

Clases: sintaxis

[visibilidad] **class** Nombre [**extends** Superclase] [**implements** Interface1, ...] {

    // declaraciones de Atributos…

    // declaraciones de constructores…

    // declaraciones de métodos…

}

```
class NombreClase {
   //definición de la clase
}
```

```
class Persona {
   //definición de Persona
}
```

UT 4 - Objetos

Ejercicios

Ejercicio 1

https://github.com/fgomrom/PROG-UT3-String/blob/main/EjemplosString.java

Ejercicio 2

https://github.com/fgomrom/PROG-UT3-String/blob/main/EjemplosString.java

Convención de nombres

Nombre	Ejemplo
Clase	VacaBrava
Variable	vacaBravaPepa
...	

Atributos


```
class NombreClase {
    tipo atributo1;
    tipo atributo2;
    ...
}
```

```
class Persona {
    String nombre;
    byte edad;
    double estatura;
}
```

Atributos > Inicialización


```
class NombreClase {
    tipo atributo1 = valor;
    ...
}

class Persona {
    String nombre;
    byte edad;
    double estatura;
    final String dni; //una vez asignado no podrá cambiarse
}
```

Clases: Atributos(la información)

- Almacenan los datos o el estado de una clase o sus instancias (objetos)
- Sus nombres suelen ser sustantivos (notación lowerCamelCase)
- Sintaxis: **[visibilidad] [static] [final] tipo Atributo1 [= valorInicial];**
- Se declaran al principio
- No hace falta inicializarlos, pero se puede hacer en:
  - Declaración
  - Método
  - Bloque de inicialización
- Visibilidad mejor a private. Siempre son visibles dentro de la clase.
- Atributos de clase: static (pertenecen a la clase, son compartidos por todos los objetos)
- Atributos miembro: pertenecen a cada objeto

Métodos

Funciones que se implementan dentro de una clase

```
public class NombreClase {
    ... //declaración de atributos

    tipo nombreMétodo (parámetros ) {
        cuerpo del método
    }
}
```

Métodos

```
public class Persona {
    String nombre;
    byte edad;
    double estatura;

    void saludar() {
        System.out.println("Hola. Mi nombre es " + nombre);
        System.out.println("Encantando de conocerte");
    }
```

Los nombres de los atributos suelen ser sustantivos y los de los métodos contener un verbo

Métodos

Ejercicio 4.1

Crea la clase persona correspondiente al modelo dado por el diagrama.

El método cumplirAños debe incrementar en uno la edad

Mientras que crecer debe aumentar la estatura según lo indicado en incremento.

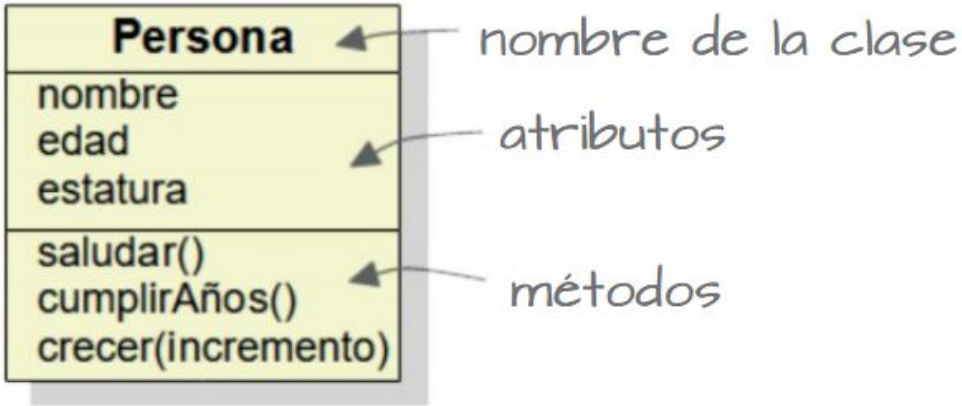


Diagrama de clases

Miembros

A los **atributos** y **métodos** de una clase se les llama **miembros** de la clase.

Cualquier miembro de una clase tiene visibilidad, es decir, puede ser accedido, dentro de la clase.

Ámbito de las variables y atributos


```
class Ambitos {
    int atributo;
    ...
    void metodo() {
        int varLocal;
        ...
        while(...) {
            int varBloque;
            ...
        } //del while
        ...
    } //del método

    ...
} //de la clase
```

ámbito de la clase: podemos utilizar atributo, ~~varLocal~~ y ~~varBloque~~

ámbito del metodo: podemos utilizar atributo, varLocal y ~~varBloque~~

ámbito del while: podemos utilizar atributo, varLocal y varBloque

Ámbito de las variables y atributos

Las variables locales se pueden llamar igual que los atributos

```
public class Ambito {
   int edad; //atributo entero
   void metodo() {
      double edad; //variable local. Oculta al atributo edad (que es entero)
      edad = 8.2; //variable local double, que oculta al atributo de la clase
      ...
   }
}
```

Referencia “this”

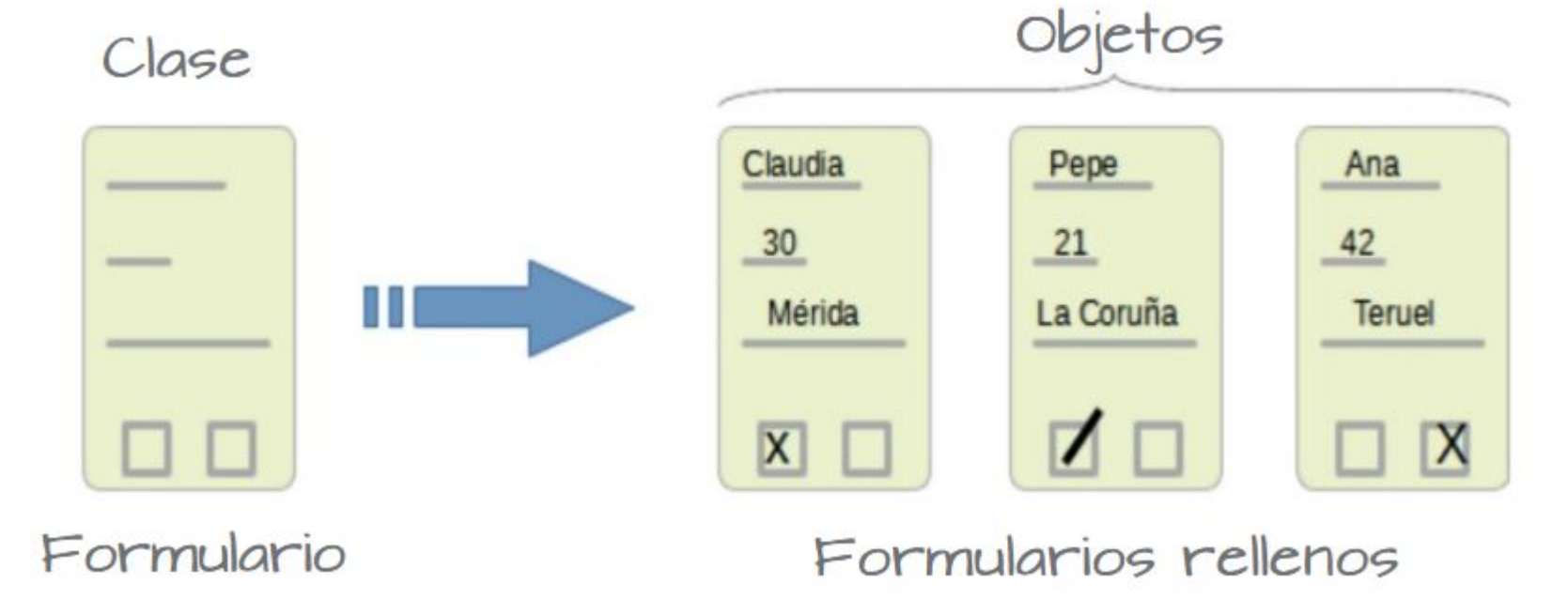
La palabra reservada this hace referencia a la propia clase.

De esta forma podemos acceder a atributos dentro de un método, aunque hayan sido ocultados por variables locales homónimas.

```
public class Ambito {
   int edad; //atributo entero
   void metodo() {
      double edad; //oculta el atributo edad (que es entero)
      edad = 20.0; //variable local, no el atributo
      this.edad = 30; //atributo de la clase
   }
}
```

Objetos

Se llama objetos a las **instancias** de una clase

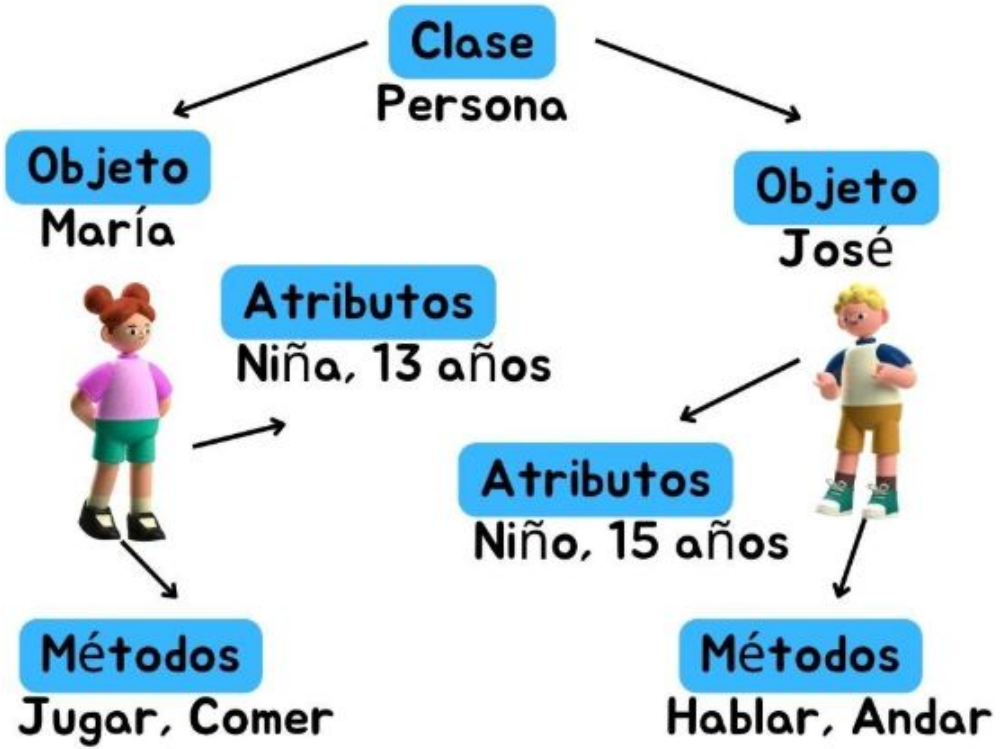


Objetos

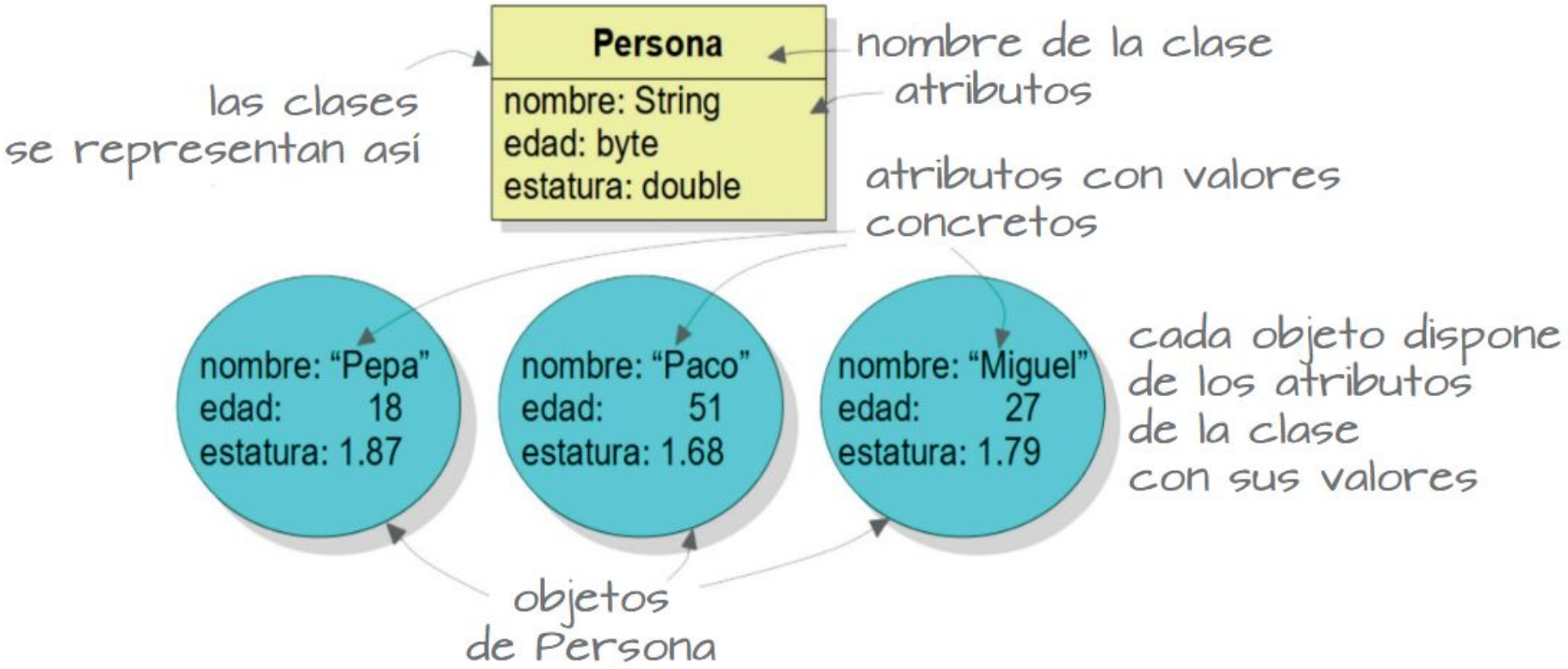
La clase se comporta como un tipo de dato.

Cada objeto es como una variable de ese tipo de dato.

Por tanto, cada objeto tiene sus propios valores para los atributos que han sido definidos mediante su clase.



Objetos


las clases
se representan así
Persona
nombre: String
edad: byte
estatura: double
nombre de la clase
atributos
atributos con valores
concretos
nombre: "Pepa"
edad: 18
estatura: 1.87
nombre: "Paco"
edad: 51
estatura: 1.68
nombre: "Miguel"
edad: 27
estatura: 1.79
cada objeto dispone
de los atributos
de la clase
con sus valores
objetos
de Persona

Referencias

Igual que los arrays, los objetos se acceden mediante referencias a memoria.

De hecho los arrays son objetos también.

Para declarar una variable del tipo de una clase:

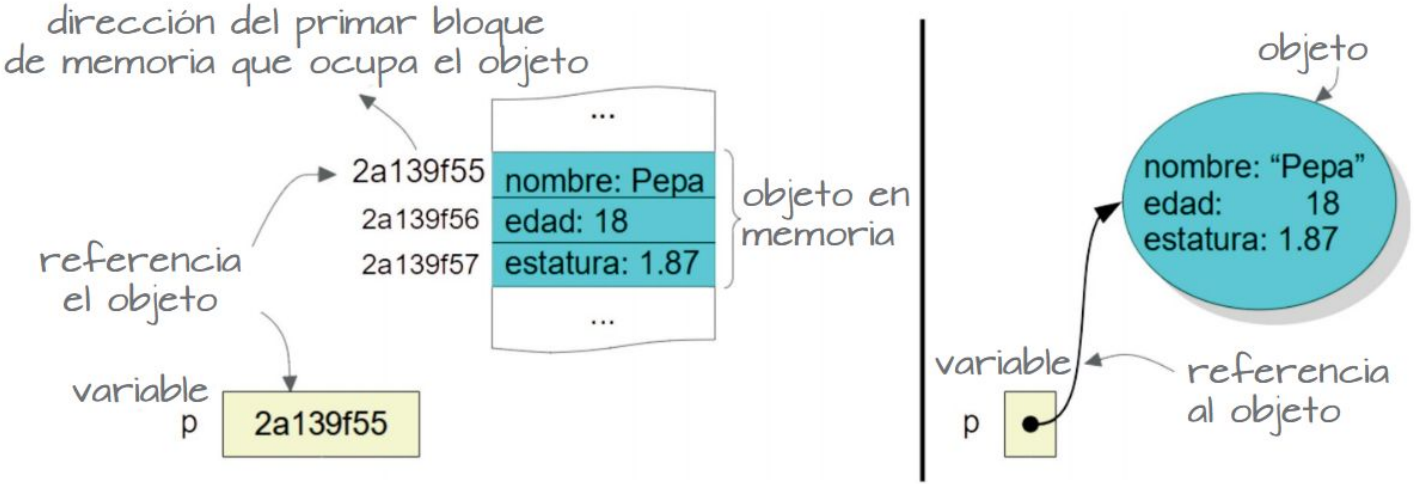
```
Clase nombreVariable;

Persona p; //p es una variable de tipo Persona
```

Creación de objetos

¿Cómo creo un objeto y asigno su referencia a una variable?

Operador "**new**"

```
p = new Persona();
```



Creación de objetos

Ejercicio 4.2

Crea una clase principal de nombre Fiesta con un método main.

Instancia un objeto de tipo Persona e imprime la referencia.

Comprueba como se observa este objeto en el depurador

Operador .

Para acceder a los miembros de un objeto se utiliza el punto “.”

```
p = new Persona();
p.nombre = "Pepa";
p.edad = 18;
p.estatura = 1.87;
```

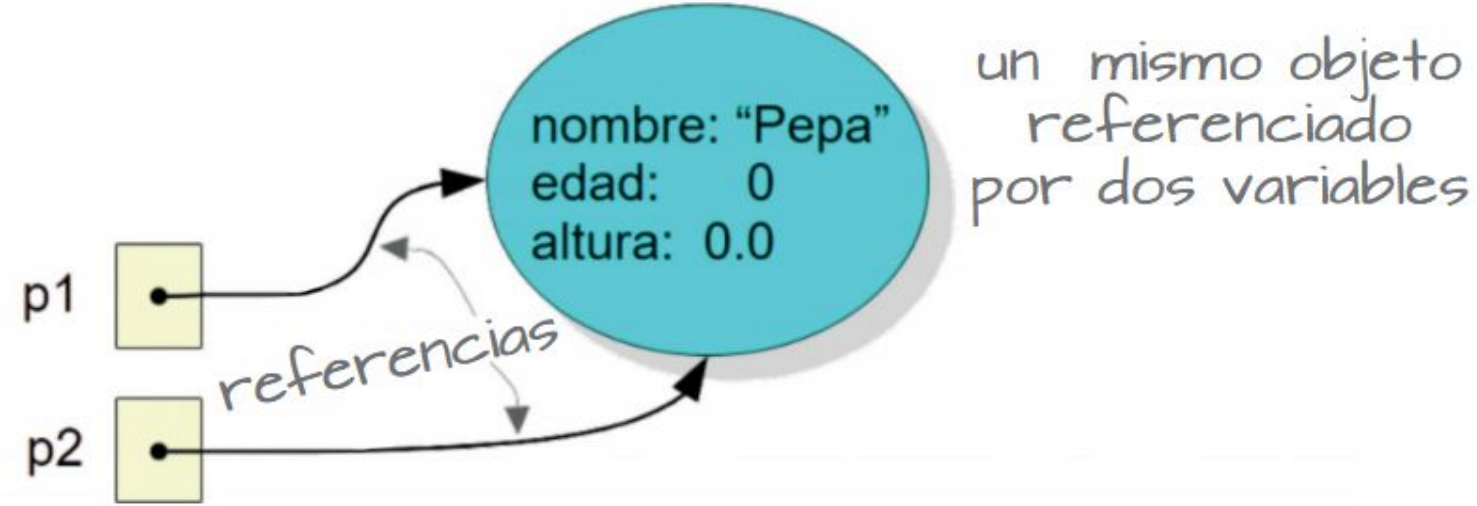
Ejercicio 3.2 (continuación)

Establece valores para los atributos del objeto de tipo Persona que creaste.

Observa en el depurador cómo se actualizan en tiempo real.

Ejemplo mismo objeto, dos variables


```
Persona p1, p2;
p1 = new Persona(); //p1 referencia al objeto creado
p2 = p1; //asignamos a p2 la referencia contenida en p1
p2.nombre = "Pepa" //es equivalente a utilizar p1.nombre
```



Referencia null

Los tipos objeto se inicializan por defecto a null si no se dice nada.

```
Persona p; //se inicializa por defecto a null
p.nombre //¡error!
```
⇐ NullPointerException

Se puede dar el valor null a una variable tipo objeto en cualquier momento

```
Persona p = new Persona(); //p referencia un objeto
...
p = null; //p no referencia nada
```

Constructores

¿Qué valores toman los atributos de un objeto recién creado?

Realiza una tabla en el porfolio con los valores por defecto en función de los tipos de datos

Para dar unos valores iniciales distintos a los de por defecto, podemos hacerlo como acabamos de ver:

```
Persona p = new Persona(); //creamos el objeto
p.nombre = "Claudia"; //asignamos valores
p.edad = 8;
p.estatura = 1.20;
```

Constructores

O bien podemos usar constructores:

Los constructores son métodos especiales dentro de la clase que usaremos para crear objetos

```
public class MiClase {
    public MiClase() {

    }
}
```

Los parámetros que no se inicializan en el constructor, se inicializan a sus valores por defecto según sean de tipos primitivos o no primitivos.

Constructores

```
class Persona {
   ...
   //constructor que asigna valores a todos los atributos
   Persona (String nombre, int edad, double estatura) {
      this.nombre = nombre;
      this.edad = edad;
      this.estatura = estatura;
   }
```

En cualquier parte del código podemos crear objetos usando el operador **new** y el **constructor** que hayamos declarado

```
Persona p = new Persona("Claudia", 8, 1.20); //creamos el objeto
//y lo inicializamos mediante el constructor
```

Tipos de constructores

- Por defecto → Si no declaramos ninguno. En el momento que se declare alguno, el constructor por defecto desaparece.
- Sin parámetros → Persona( )
- Con parámetros (los que queramos)

```
//constructor sobrecargado: solo asigna el nombre
Persona (String nombre) {
   this.nombre = nombre;
   estatura = 1.0; //valor arbitrario para la estatura
   //al no asignar la edad se inicializa por defecto: a 0
}
 Persona b = new Persona("Dolores");
```

Objetos: instancias de una clase

Ejercicio 4.3

En nuestra Fiesta vamos a crear personas de varias maneras.

1. Usando el constructor por defecto crea 2 personas: pepe y paco. Comprueba que sus atributos tienen los valores por defecto.

2. Crea un constructor con todos los parámetros de Persona. ¿Qué ocurre ahora con el constructor por defecto?

3. Crea un constructor sin parámetros para arreglar el error anterior. Este constructor generará una persona estándar de nombre “anónimo”, edad 18 y estatura 1.70.

4. Crea dos personas más usando el constructor con parámetros.

Ejercicio 4.3 (cont)

5. Crea otro constructor con parámetros que incluya sólo el nombre

6. Ejecuta el programa y depura para ver los resultados.

Constructores > this()

this() se usa para invocar a un constructor → reutilizar constructores

```
class Persona {
   ...
   //constructor que asigna valores a todos los atributos
   Persona (String nombre, int edad, double estatura) {
      this.nombre = nombre;
      this.edad = edad;
      this.estatura = estatura;
   }

   //constructor sobrecargado que solo asigna el nombre
   Persona (String nombre) {
      this(nombre, 0, 1.0); //invoca al primer constructor
      //la edad se pone a 0 y la estatura a 1.0
   }
}
```

¡Sólo se puede poner al principio!

Ejercicio 4.3 (cont)

7. Crea otro constructor que reciba junto a todos los parámetros, otro que sea el apellido, y cree una persona invocando a algún constructor que ya teníamos, pero ademas le concatene el apellido al nombre.

Paquetes

¿Cómo organizar los ficheros de código fuente?

Paquete → conjunto de clases e interfaces relacionadas entre sí

Indicar en qué paquete está un fichero → ***package*** *identificadordelPaquete;*

Importar otro paquete en nuestro fichero → ***import*** *identificadordelPaquete;*

El nombre de un paquete puede estar **cualificado** (subpaquetes separados por un punto)

Paquetes

Package	Descripción del contenido
`java.lang`	Contiene las clases e interfaces más empleadas en la mayoría de los programas de Java. Es importado automáticamente por todos los programa Java: no se necesita sentencia `import` para utilizar lo declarado en este paquete.
`java.io`	Contiene clases que permiten las operaciones de entrada y salida de datos de un programa.
`java.util`	Contiene clases e interfaces de utilidades: operaciones con la fecha y la hora, generación de números aleatorios...
`java.applet`	Contiene todas las clases e interfaces necesarias para la construcción de *applets* de Java
`java.net`	Contiene clases que permite a un programa comunicarse a traves de redes (Internet o intranet)
`java.text`	Contiene clases e interfaces que permiten operaciones de números, fechas, caracteres y cadenas.
`java.awt`	Es el paquete *Abstract Windowing Toolkit*. Contiene muchas clases e interfaces necesarias para trabajar con la interfaz de usuario gráfica *clásica*.
`java.beans`	Contiene clases para facilitar a los programadores la generación de componentes de software reutilizables.

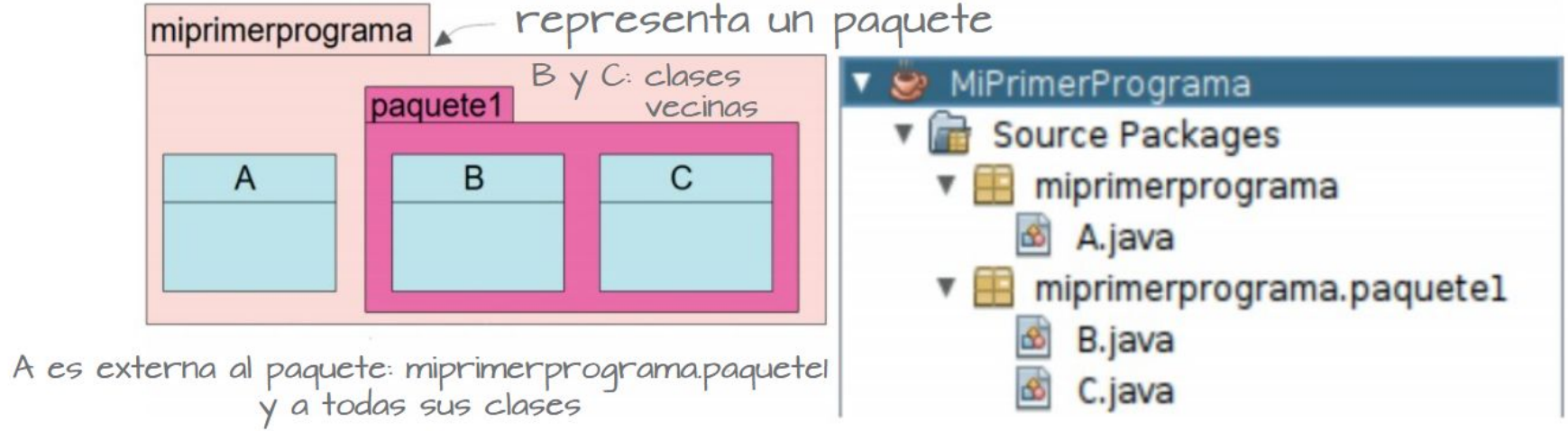
Encapsulación: visibilidad

La visibilidad nos dice si algo podemos usarlo o no:

- Es visible → podemos usarlo
- No es visible → no podemos usarlo

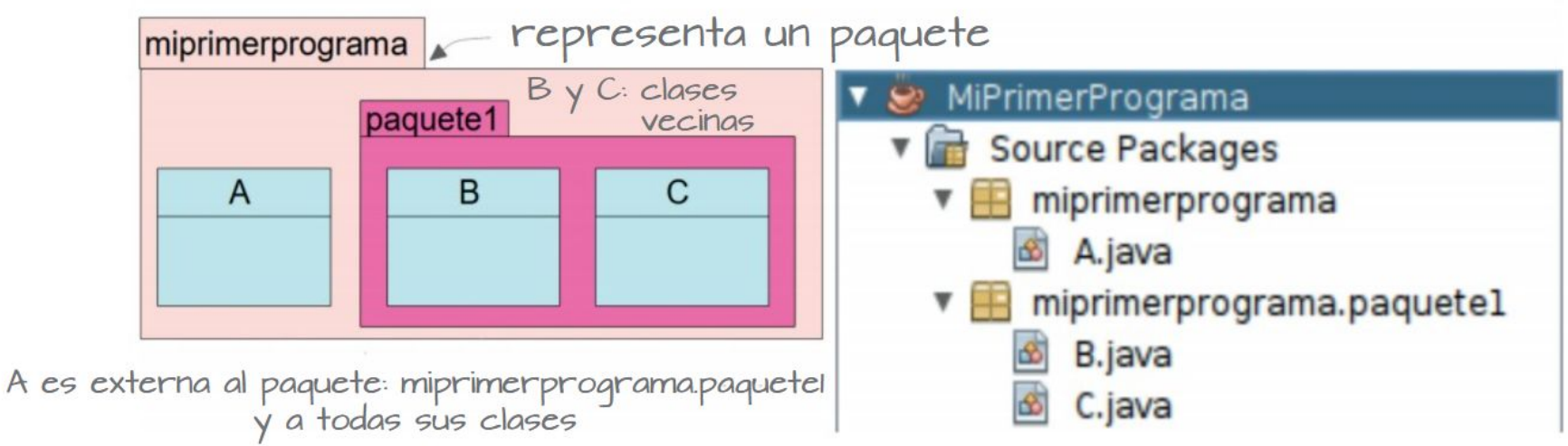
La visibilidad se controla de dos formas:

- Paquetes
- Modificadores de visibilidad



Visibilidad a nivel de clase

Por defecto, si se ubican en el mismo paquete pueden verse



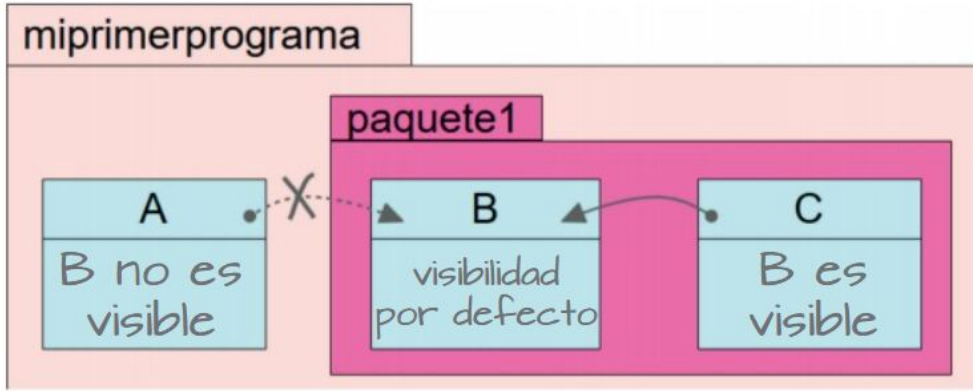
Visibilidad a nivel de clase

Usando modificadores al declarar la clase:

```
[visibilidad] class NombreClase {
    ...
}
```

- **public** → clase visible para todas las clases
- **package** (implícito si no se pone nada) → clase visible para todas las clases del mismo paquete

```
package miprimerprograma.paquete1;
class B { //sin modificador de acceso
    ...
}
```



Buena práctica: visibilidad lo más restrictiva posible

Visibilidad a nivel de clase


```
package miprimerprograma.paquete1;
public class B { //clase marcada como pública
    ...
}

package miprimerprograma;
import miprimerprograma.paquete1.B; //ahora A puede usar la clase B

class A {
    ...
}
```

Buena práctica: visibilidad lo más restrictiva posible

Visibilidad a nivel de clase

Se importan las clases no los paquetes.

Para importar todas las clases de un paquete → *

```
package miprimerprograma;
import miprimerprograma.paquete1.*; //A puede usar cualquier clase pública
                                    //del paquete miprimerprograma.paquete1

class A {
    ...
}
```

	Visible desde…	
	clases vecinas	clases externas
sin modificador	✓	
public	✓	✓

Buena práctica: visibilidad lo más restrictiva posible

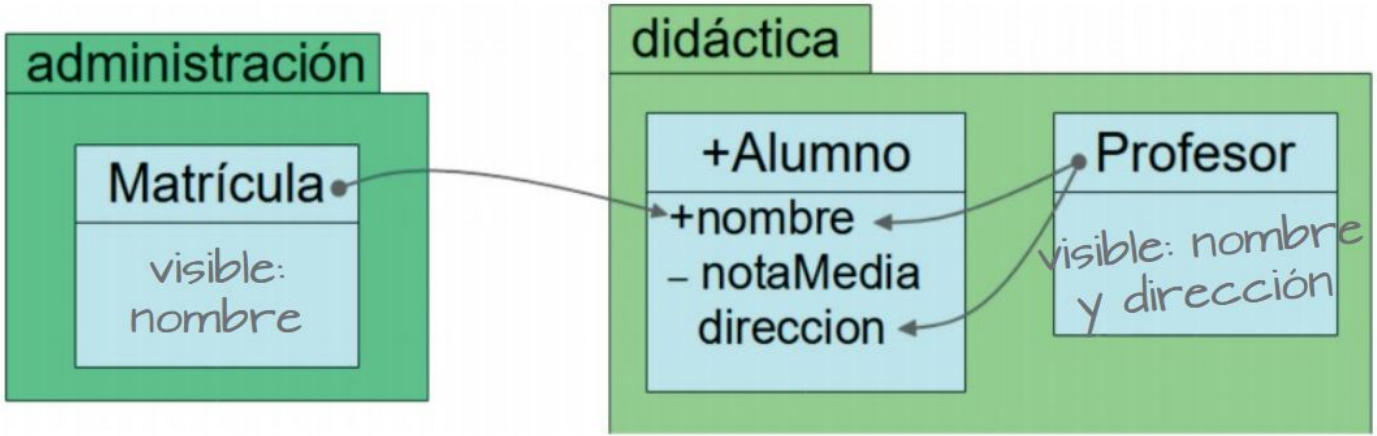
Visibilidad a nivel de miembros

A nivel de los miembros de una clase:

		Mismo paquete		Otro paquete	
		Subclase	Otra	Subclase	Otra
-	**private**	*no*	*no*	*no*	*no*
#	**protected**	*sí*	*sí*	*sí*	*no*
+	**public**	*sí*	*sí*	*sí*	*sí*
~	*package*	*sí*	*sí*	*no*	*no*

Buena práctica: visibilidad lo más restrictiva posible

Visibilidad a nivel de miembros



Ejercicio 4.3 (cont)

8. Establece todos los atributos de la clase persona para que sólo sean visibles desde la misma clase.

9. Intenta modificar el nombre de una persona desde la clase Fiesta.

10. Para permitir lo anterior, permite que sólo el nombre sea visible desde cualquier clase.

Encapsulación: getter y setter

La encapsulación permite ocultar detalles de la implementación y controlar qué atributos pueden usarse libremente o incluso cómo deben usarse.

Para conseguir esto, hay que combinar la visibilidad con métodos que acceden a los atributos para leerlos y escribirlos.

Encapsulación: getter y setter

Estos métodos se identifican con set/get seguido del nombre del atributo. Para el atributo `edad` quedaría:

```
class Persona {
   private int edad;
   ...

   public void setEdad(int edad) {
      if (edad >= 0) //solo los valores positivos tienen sentido
         this.edad = edad;
      } // en caso contrario no se modifica la edad
   }

   public int getEdad() {
      return edad;
   }
}
```

Ejercicio 4.3 (final)

11. Pon todos los atributos con una visibilidad que sólo pueda ser accedidos, fuera de la clase, mediante métodos get y set.

12. Genera los correspondientes getter y setter para todos los atributos

13. Añade varias validaciones mediante los setter

Ejercicio 4.4

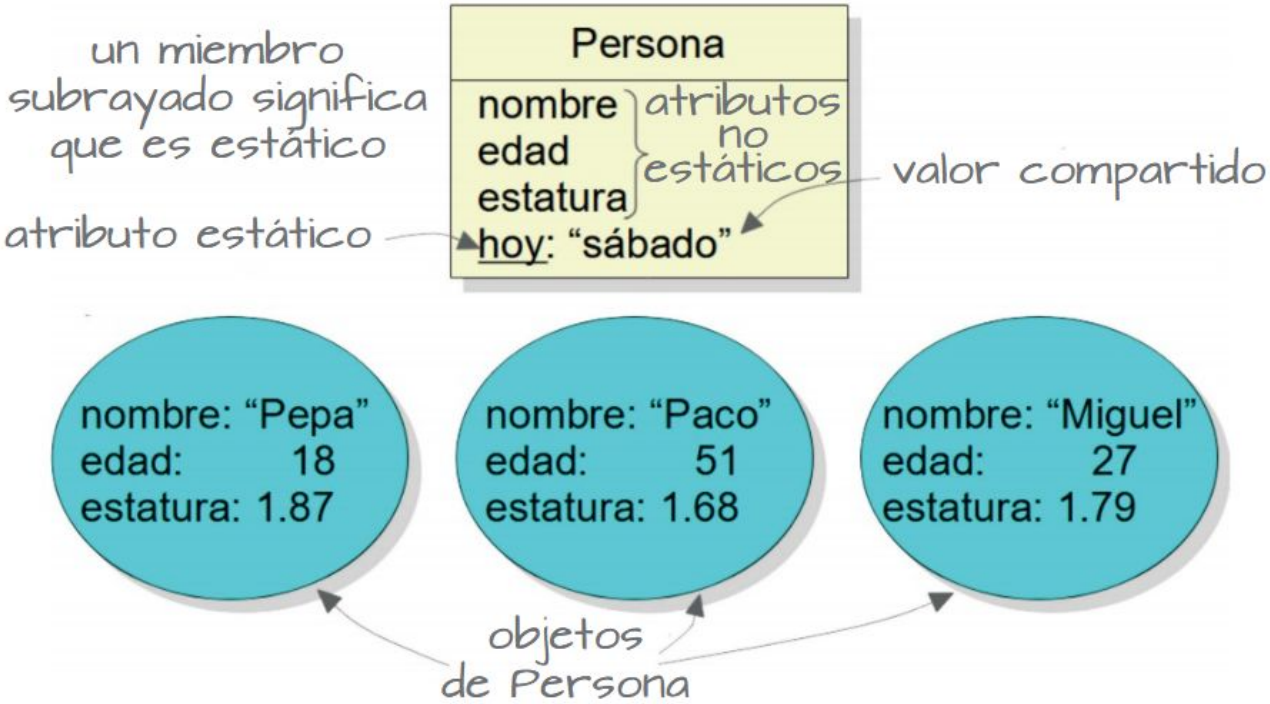
Escribir un programa que lea por teclado una hora cualquiera y un número n que representa una cantidad en segundos. El programa mostrará la hora introducida y las n siguientes, que se diferencian en un segundo.

Para ello hemos de diseñar previamente la clase Hora que dispone de los atributos hora, minuto y segundo.

Los valores de los atributos se controlarán mediante métodos set/get.

Atributos y métodos estáticos

No pertenecen a los objetos sino a la clase y por tanto son compartidos por todos los objetos de la clase



Atributos estáticos

Un atributo estático se declara mediante la palabra reservada `static`.

```
class Persona {
    ...
    static String hoy;
}

Persona.hoy = "domingo";
System.out.println(Persona.hoy);
```

Métodos estáticos

Pertenecen a la clase, por tanto, se ejecutan sin necesidad de crear ningún objeto → por eso desde un método estático sólo se pueden invocar otros métodos estáticos como pasa en el método main.

```
static void hoyEs(int dia) {
    hoy = switch (dia) {
        case 1-> "lunes";
        case 2-> "martes";
        ...
        case 7-> "domingo";
    }
}
```

```
Persona.hoyEs(2);
```

Ejercicio 4.5

Vamos a implementar una clase llamada ContadorObjetos que tenga un atributo estático para llevar la cuenta del número total de objetos creados de esta clase. Además, tendrá un método estático para mostrar cuántos objetos se han creado hasta el momento.

Instrucciones:

1. Crea una clase llamada `ContadorObjetos`.
2. Define un atributo estático llamado `totalObjetos` que sea de tipo entero e inicialízalo en 0.
3. Crea un constructor para la clase que incremente el valor de `totalObjetos` cada vez que se cree un objeto de esta clase.
4. Añade un método estático llamado `mostrarTotalObjetos` que imprima en consola el número total de objetos creados.
5. En el método `main` de otra clase, crea varios objetos de la clase `ContadorObjetos` y utiliza el método estático para mostrar cuántos objetos se han creado.

Enumerados

Son variables limitadas a una serie de valores

```
enum DiaDeLaSemana {
    LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO
}
```

```
Scanner sc = new Scanner(System.in);
String dia = sc.nextLine(); //introducimos LUNES
DiaDeLaSemana ingles = DiaDeLaSemana.valueOf(dia);
```

Ejercicio 4.6

Crea un enumerado llamado Sexo para añadir el atributo en la clase Persona que indique si una persona es Hombre o Mujer.

Ejercicio 4.7

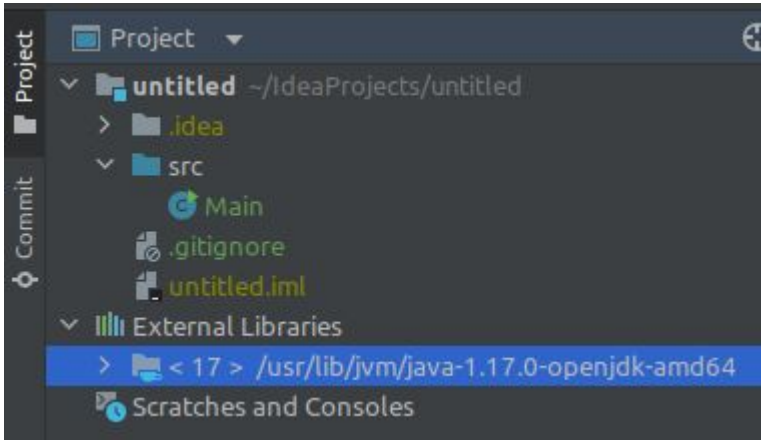
Ver enunciado en Moodle

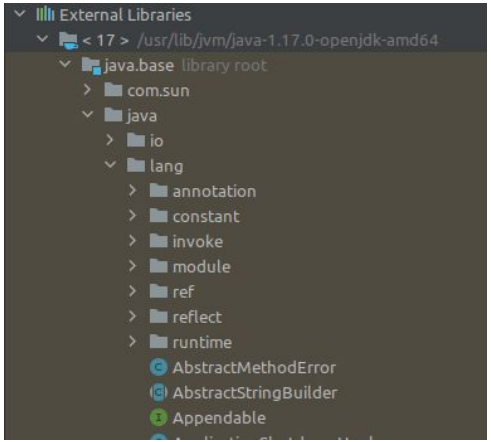
Librería (o biblioteca)

- Es el código ya desarrollado que vamos a utilizar para desarrollar nuevo código.
- En lugar de hacerlo todo desde 0, nos basamos en lo que ya hay hecho. (No reinventamos la rueda)
- De esta forma aceleramos el desarrollo, evitamos errores, facilitamos el mantenimiento, etc… algo parecido a lo que obteníamos organizando el código en funciones, pero a mayor escala.
- Las librerías se distribuyen encapsuladas en uno o varios ficheros. En el caso de java tenemos los **.jar**

Librería (o biblioteca)

- Tanto java.Math como java.util.Ramdon son clases (que están organizadas en diferentes paquetes) de la librería de Java: jdk





Librería (o biblioteca) - Ejercicio 3.4

Busca las siguientes clases dentro de tu librería JDK:

- Object
- String
- Math
- java.util.Random

Librería (o biblioteca)

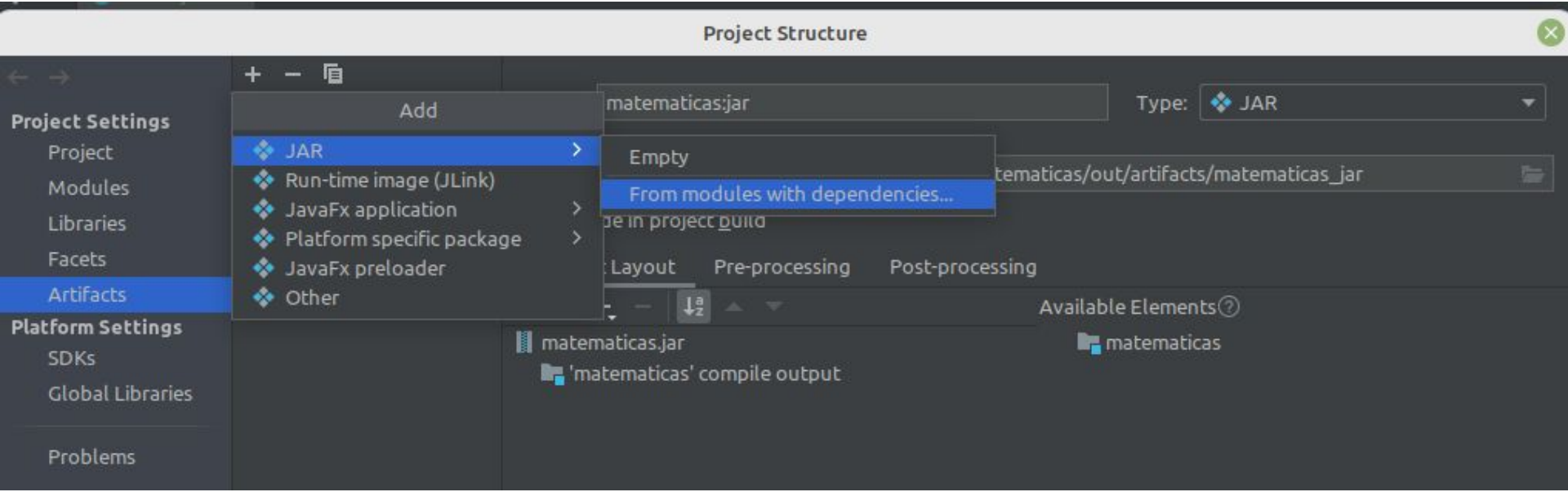
¿Puedo crear mi propia librería?

Podemos hacerlo y compartir nuestros desarrollos con los demás.

Veamos cómo…

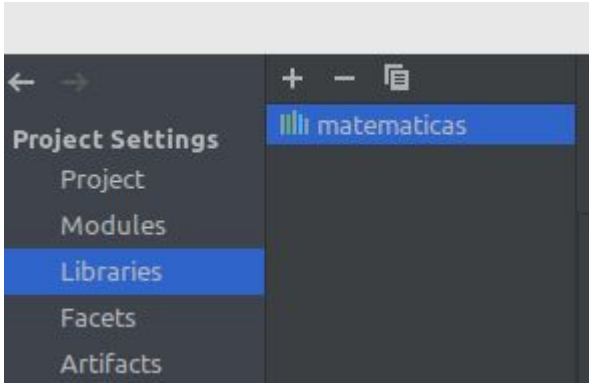
Librería (o biblioteca)

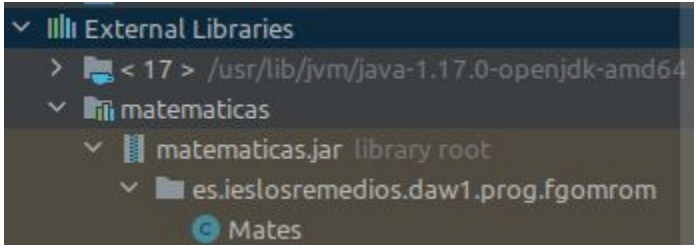
1. Creamos un proyecto nuevo con las clases de nuestra librería
2. Generamos un .jar



3. Añadimos la librería (el .jar generado) a un segundo proyecto en el que queramos hacer uso de ella.

Librería (o biblioteca)





Librerías: Ejercicio 4.8

Vamos a usar nuestra clase CalculadoraBasica dentro de otro proyecto.

Lo primero es convertir sus métodos y atributos en estáticos, ya que realmente es una **clase de utilidades**, como java.lang.Math.

Por otro lado, hay que asegurarse de que los métodos son públicos, pues de otra forma no podremos usarlos desde otros paquetes.

Librerías: Ejercicio 4.8

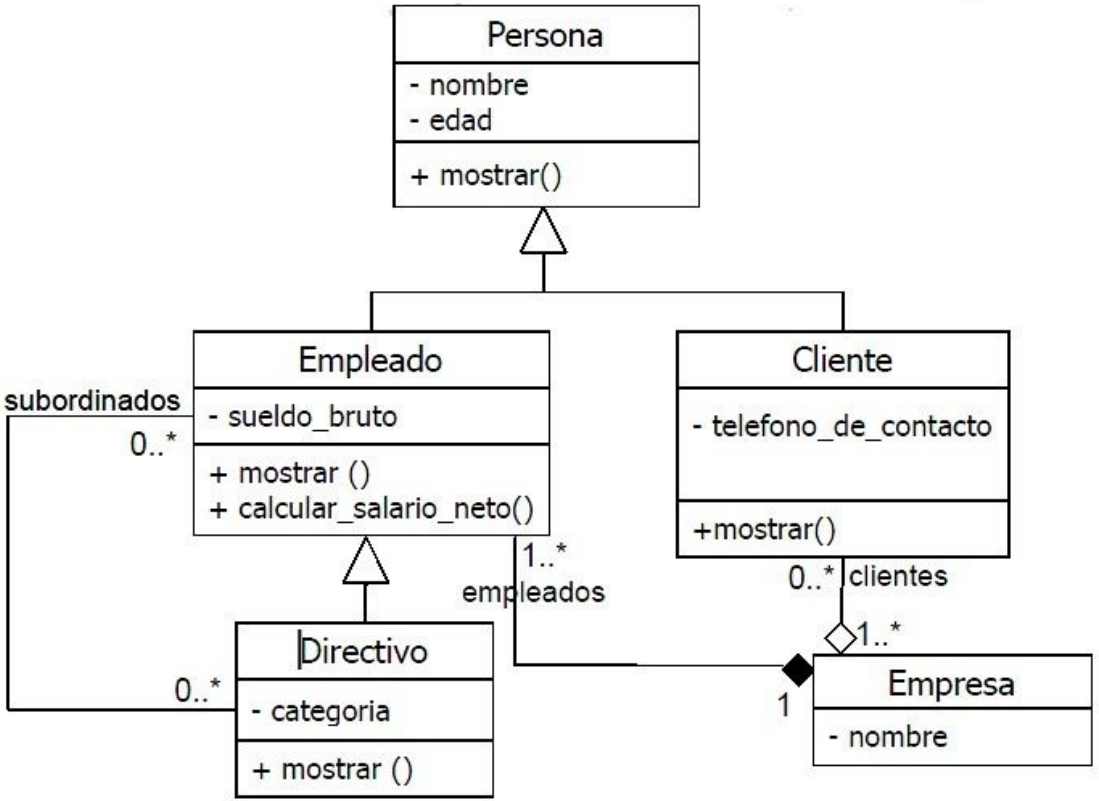
Seguidamente generamos un **jar** para empaquetar y distribuir nuestra clase CalculadoraBasica.

Nota: un jar puede ser ejecutable o no, pues también es la forma de empaquetar una aplicación en Java. En nuestro caso, al ser una librería, no tiene que ser ejecutable.

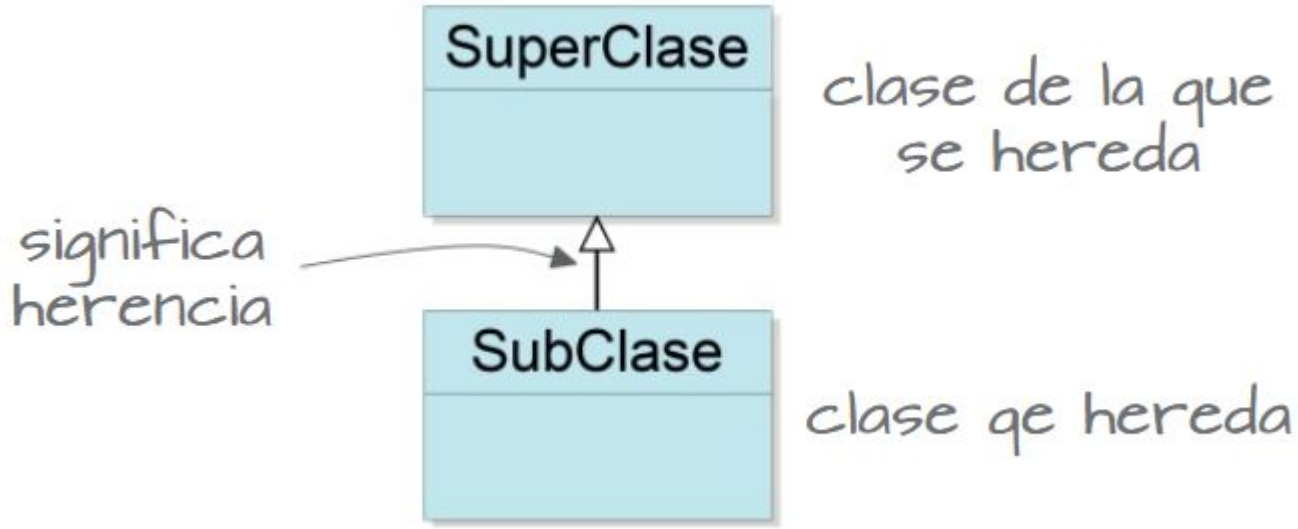
Por último añadimos la librería en un **nuevo proyecto** donde queramos usarla.

Relaciones entre clases

- Asociación
- Agregación
- Composición
- Cardinalidad
- Multiplicidad
- Herencia



Herencia



- Todos los miembros de la superclase son heredados por la subclase

Herencia

- Clase padre (superclase o base) y clase hija (subclase o derivada)
- Todos los miembros del padre pasan a su hija excepto los constructores
- keyword: **extends**
  - Sintaxis: class subclase extends superclase { … }
- No existe herencia múltiple en Java
- Sí existe herencia transitiva
- La primera clase en la jerarquía es Object
- keyword: **super**
  - Es como **this** pero en lugar de hacer referencia a la misma clase, se refiere a la padre.
  - Sirve para acceder a los miembros de la superclase (sólo un nivel por encima)
  - Se puede acceder varios niveles usando un casting explícito.

Herencia: Ejemplo


```
class Persona {
    String nombre;
    byte edad;
    double estatura;
}
class Empleado extends Persona {
    double salario;
    Empleado(String nombre, byte edad, double estatura, double salario) {
        ...
    }
}
```

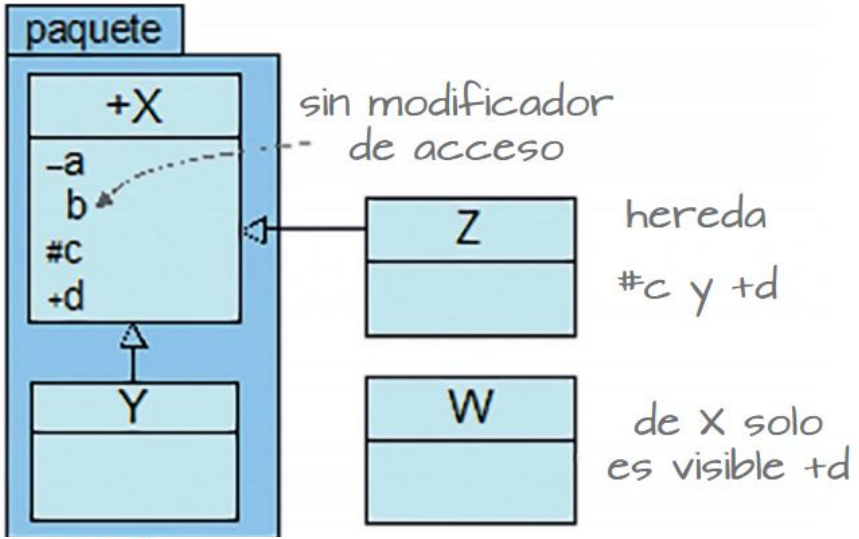
```
Empleado e = new Empleado("Sancho",  25,  1.80,  1725.49);
System.out.println(e.nombre); //muestra un atributo heredado
System.out.println(e.salario); //muestra un atributo propio
```

Herencia: visibilidad

- significa private

significa protected

+ significa public



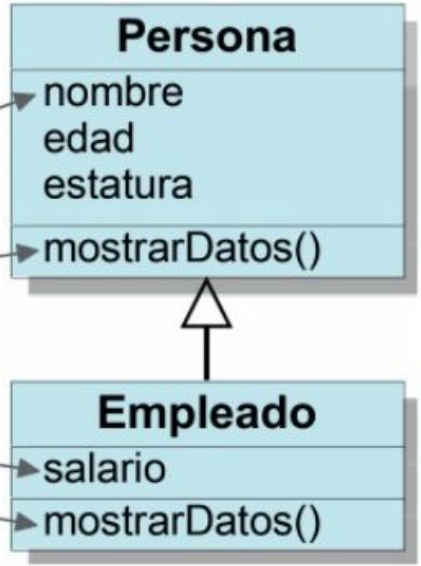

```
public class X {
     private int a; //invisible fuera de la clase
     int b; //visibilidad por defecto: visible en el paquete
     protected int c; //visible en el paquete y para
     //las subclases (aunque sean externas)
     public int d; //visibilidad total
}
```

Herencia: visibilidad

	Visible desde...			
	la propia clase	clases vecinas	subclases	clases externas
`private`	✓			
sin modificador	✓	✓		
`protected`	✓	✓	✓	
`public`	✓	✓	✓	✓

Herencia: Override

```
Persona p = new Persona();
Empleado e = new Empleado();


        e.nombre
p.mostrarDatos()
        e.salario
e.mostrarDatos()
```



Un miembro redefinido en la subclase estará sobreescribiendo el miembro heredado de la superclase

- Subclase declara mismo método que la superclase (mismo nombre, parámetros y tipos)
- Añadimos funcionalidad en la subclase sobre la que ofrecía la superclase
- Anotación @Override

Herencia: Override

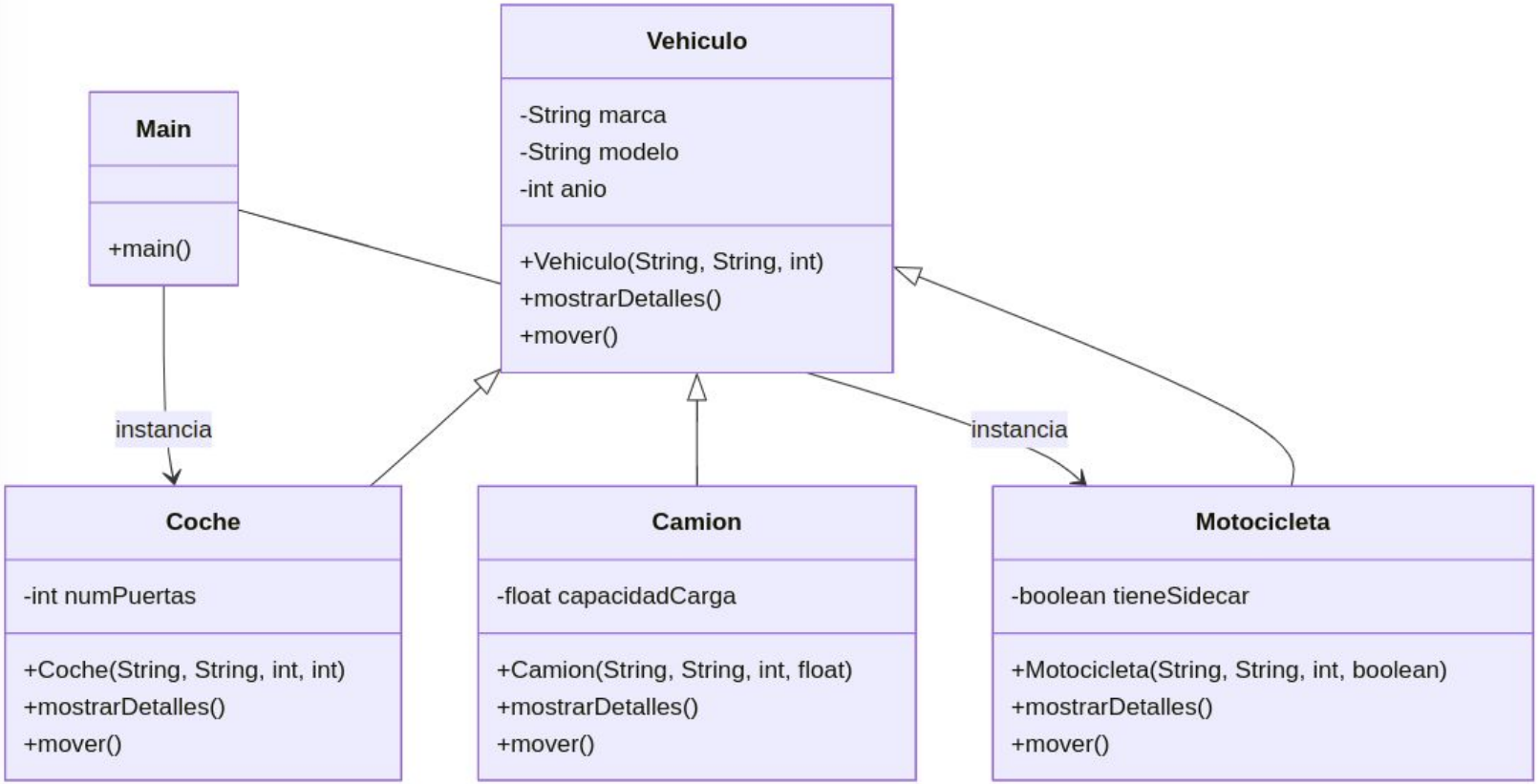
```
class Persona {
     String nombre;
     byte edad;
     double estatura;
     void mostrarDatos() {
            System.out.println(nombre);
            System.out.println(edad);
            System.out.println(estatura);
     }
}
class Empleado extends Persona {
     double salario;
     @Override //significa: sustituye un método de la superclase
     void mostrarDatos() {
            System.out.println(nombre);
            System.out.println(edad);
            System.out.println(estatura);
            System.out.println(salario);
     }
}
```

String estatura; //oculta a: la estatura de tipo byte

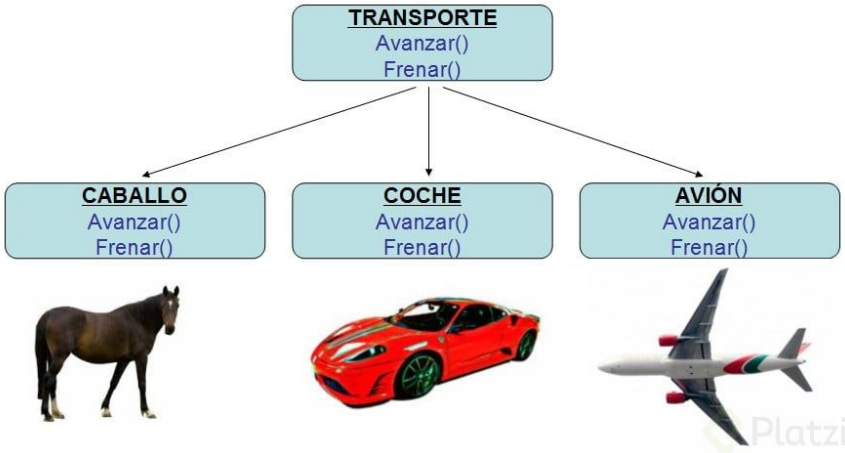
Herencia: super y super()


```
public class Persona {
     String nombre;
     byte edad;
     double estatura;
     ...
     void mostrarDatos() {
            System.out.println(nombre);
            System.out.println(edad);
            System.out.println(estatura);
     }
}
class Empleado extends Persona {
     double salario;
     @Override
     void mostrarDatos() {
            super.mostrarDatos(); /*método de la superclase, muestra los
            atributos definidos en Persona*/
            System.out.println(salario); /*muestra el atributo añadido en
            Empleado*/
     }
}
```

Ejercicio 4.9



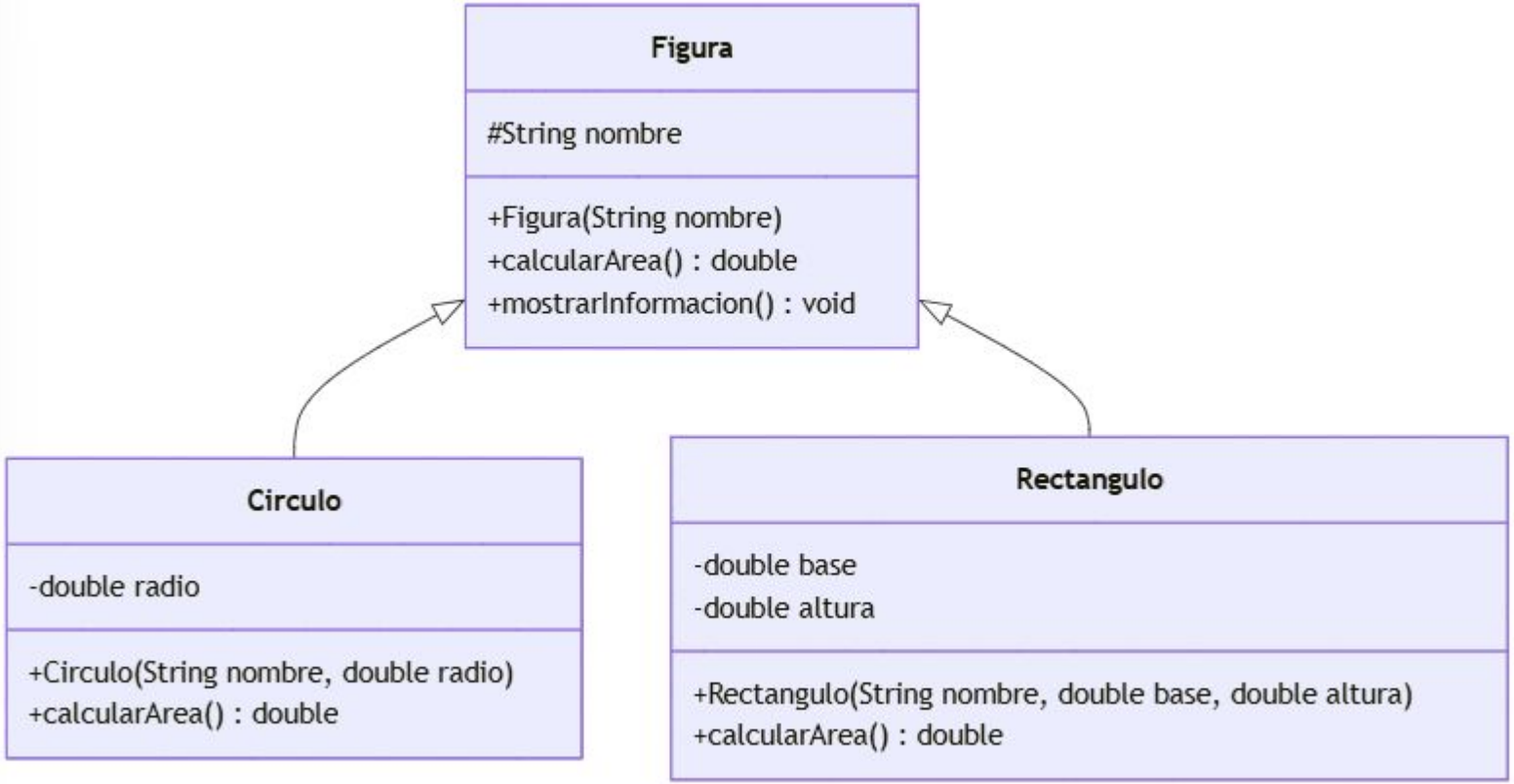
Polimorfismo



Los objetos de la subclase son también objetos de la superclase

```
Empleado e = new Empleado();
Persona p = e;
```

Ejercicio 4.10



https://app.wooclap.com/events/XNIHRR/0

UT4 - 24/25 - Herencia y polimorfismo

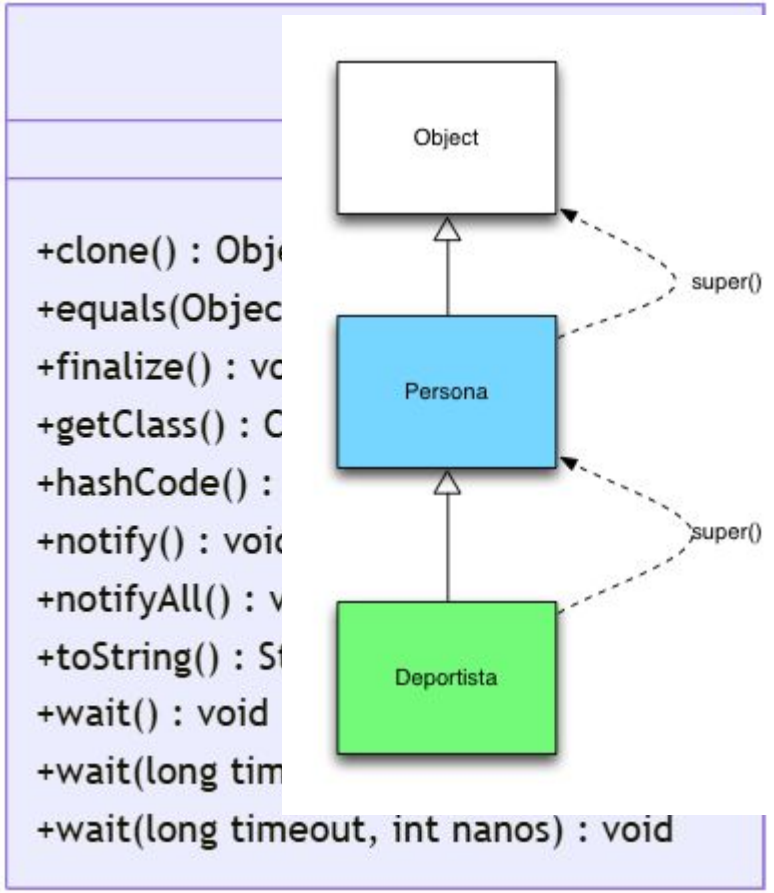
Votes | Messages | Participant pace

How to participate?

1. ¿Qué es la herencia en Java?
2. ¿Cómo se define una subclase en Java?
3. ¿Qué es el polimorfismo en Java?
4. ¿Qué es la sobrecarga de métodos en Java?
5. ¿Qué es la sobreescritura de métodos en Java?
6. ¿Qué significa el modificador "protected" en Java?
7. ¿Qué hace la palabra clave "super" en Java?

La clase Object

- La clase `Object` es la superclase base de todas las clases en Java.
- Todas las clases en Java heredan implícitamente de `Object`.
- Permite que cualquier variable de tipo `Object` pueda referirse a un objeto de cualquier clase.
- Proporciona funcionalidad esencial y común a todas las clases.
- Métodos como `equals()`, `toString()`, y `hashCode()` son frecuentemente sobrescritos para personalizar el comportamiento.



Métodos de Object

Método	Descripción
Object clone()	Crea un nuevo objeto, igual al que se duplica
boolean equals (Object objeto)	Determina si un objeto es igual a otro
void finalize()	Llama antes de que un objeto no utilizado sea reciclado.
Class getClass()	Obtiene la clase de un objeto en tiempo de ejecución.
int hashCode()	Devuelve el código asociado al objeto que realiza la llamada.
void notify()	Reiniciar la ejecución de un hilo en espera en el objeto que realiza la llamada.
void notifyAll()	Reiniciar la ejecución de todos los hilos en espera en el objeto que realiza la llamada.
String toString()	Devuelve una cadena que describe el objeto
void wait() void wait (long milisegons) void wait (long milisegons,int nanosegons)	Espera en otro subproceso de ejecución

Los métodos ***getClass***, ***notify***, ***notifyAll*** y ***wait*** están declarados como ***final***, el resto pueden sobrescribir.

toString

¿Qué hace?

- Convierte un objeto en una representación de cadena

¿Para qué se usa?

- Proporciona una representación legible del objeto para poder imprimirlo

Ejemplo de salida

- `Persona@1a2b3c4d`

¿Cómo podríamos utilizarlo para imprimir nuestros objetos personalizados?

```
public class Persona {
    private String nombre;
    private int edad;

    @Override
    public String toString() {
        return "Persona: " + nombre + ", Edad: " + edad;
    }
}
```

¿Cómo lo usamos?

Uso explícito

- `Persona persona = new Persona("Fran", 30);`
  `persona.toString();` → **`Persona: Fran, Edad: 30`**

Uso implícito

- `System.out.println(persona)`
- `String.valueOf(persona)`
- Concatenación de cadenas: `"Info: " + persona`

Buenas prácticas

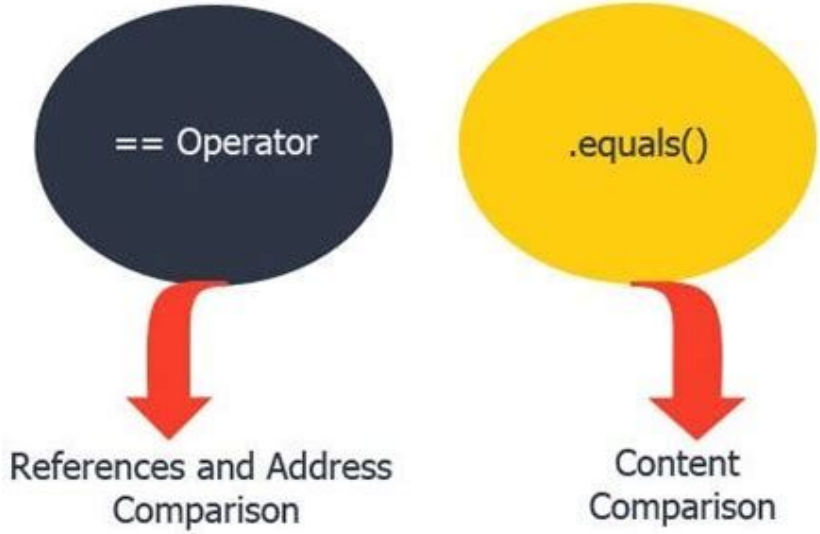
- Sobrescribir en clases propias
- Incluir atributos relevantes
- Mantener un formato consistente

equals()

Sirve para comparar si dos objetos son **equivalentes en términos de contenido**, no solo si son el mismo objeto en memoria.

Implementación por defecto:

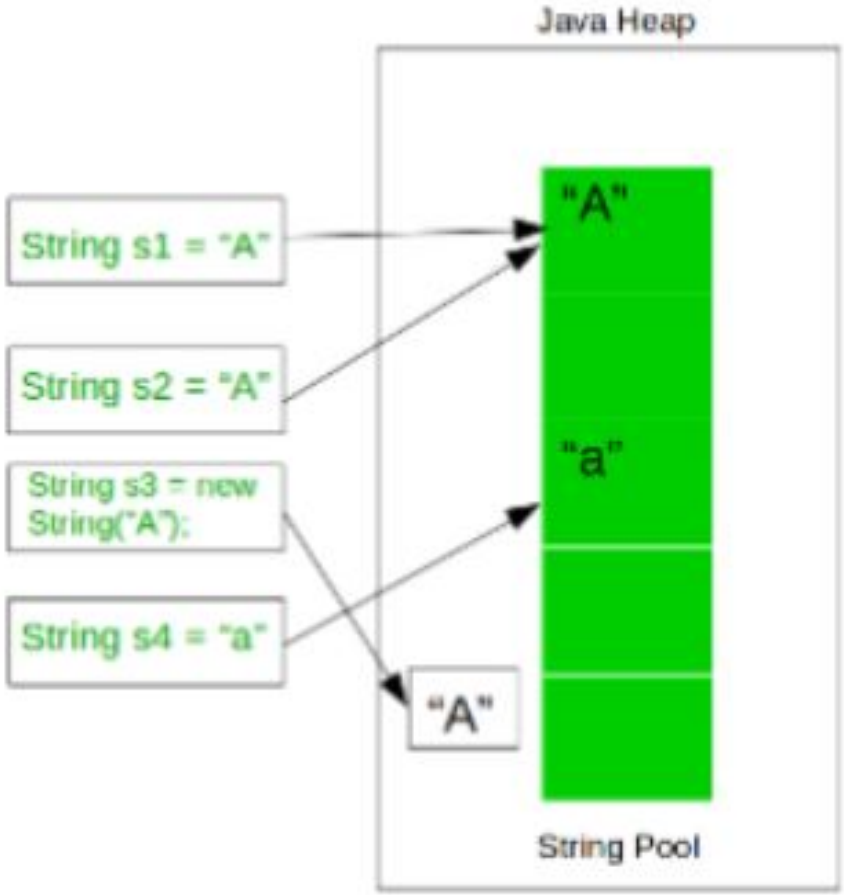
```
public boolean equals(Object obj){
    return (this == obj);
}
```



Sobrescritura del método equals()


```java
public class Persona {
    private String nombre;
    private int edad;

    public Persona(String nombre, int edad) {
        this.nombre = nombre;
        this.edad = edad;
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj) return true; // Mismo objeto
        Persona persona = (Persona) obj;
        // Comparar atributos
        return edad == persona.edad && nombre.equals(persona.nombre);
    }
}
```

Comparar cadenas en Java



s1.equals(s2) → true
s1.equals(s3) → true
s1.equals(s4) → false
s1 == s2 → true
s1 == s3 → false
s1 == s4 → false

Buenas prácticas al sobrescribir equals()

1. Consistencia con hashCode(): Siempre que sobrescribas `equals()`, también sobrescribe `hashCode()` para garantizar un comportamiento consistente.
2. Null Safety: Asegúrate de manejar valores nulos para evitar excepciones.
3. Simetría: Si `x.equals(y)` es `true`, entonces `y.equals(x)` también debe serlo.
4. Transitividad: Si `x.equals(y)` y `y.equals(z)` son `true`, entonces `x.equals(z)` también debe serlo.

getClass()

Devuelve el **tipo del objeto en tiempo de ejecución**

```
Persona persona = new Persona();
Object personaObjeto = new Persona();

persona.getClass().getName()          → Persona
personaObjeto.getClass().getName()    → Persona

System.out.println(personaObjeto instanceof Persona); → true
System.out.println(personaObjeto instanceof Object);  → true
```

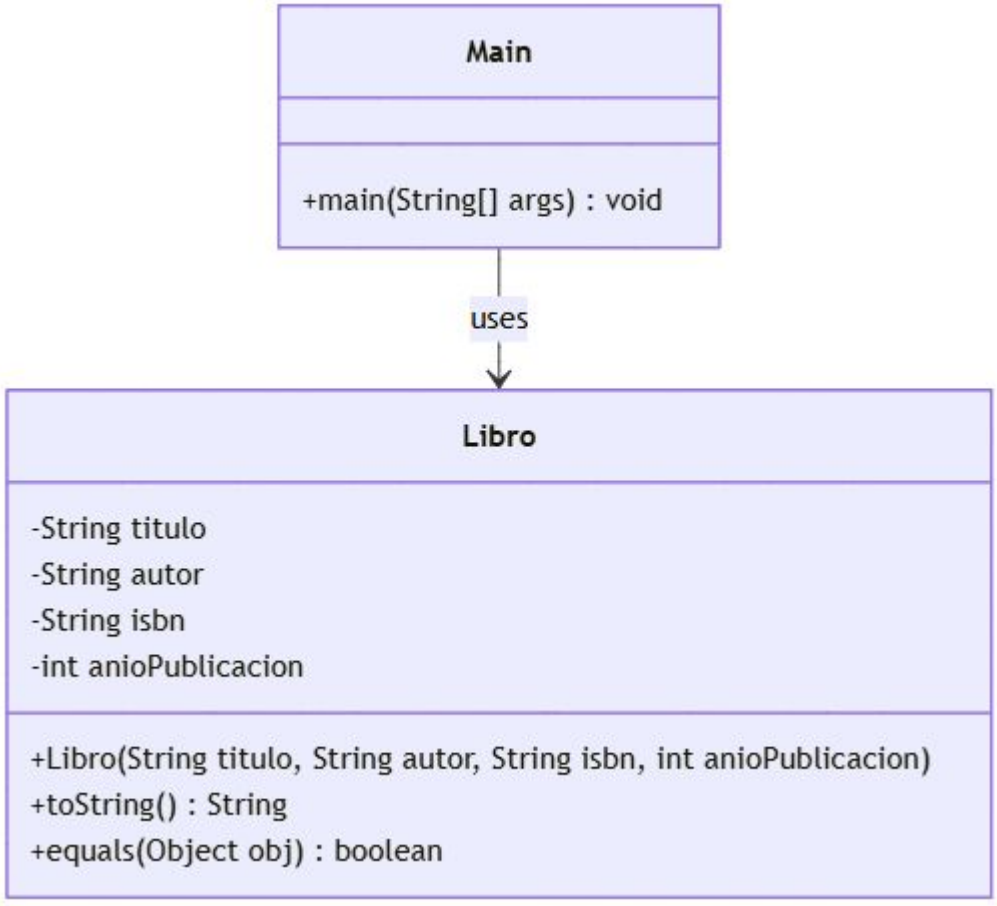
Ejercicio 11

Crea una clase `Libro` con

- `titulo` (String)
- `autor` (String)
- `isbn` (String)
- `anioPublicacion` (

Tareas:

1. Implementa el con
2. Sobrescribe el mét
   "Título: [titulo], Aut
3. Sobrescribe el mét
4. En el método `main`
   - Imprime cad
   - Compara dc
   - Compara dc



con el siguiente formato:

ISBN.

Responde...

How to participate?

1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
RGDLUO

Enable answers by SMS

Métodos abstractos

Son métodos pensados solamente para ser sobreescritos

- Declarados sin implementación
- Utilizan la palabra clave `abstract`
- No tienen cuerpo de método, terminan con punto y coma
- Ejemplo: `public abstract void hacerSonido();`
- Los métodos abstractos deben ser implementados por las subclases concretas

Métodos abstractos - Ejemplo


```java
abstract class Figura {
    // Método abstracto
    public abstract double calcularArea();

    // Método concreto
    public void mostrarTipo() {
        System.out.println("Soy una figura geométrica.");
    }
}

class Circulo extends Figura {
    private double radio;

    @Override
    public double calcularArea() {
        return Math.PI * radio * radio;
    }
}
```

Clases abstractas

Una clase abstracta es una clase que no puede ser instanciada directamente

- Se declara usando la palabra clave `abstract`
- Una clase que contiene al menos un método abstracto debe ser declarada como abstracta
- Las clases abstractas pueden contener métodos abstractos y concretos (con implementación)
- Las subclases deben implementar todos los métodos abstractos

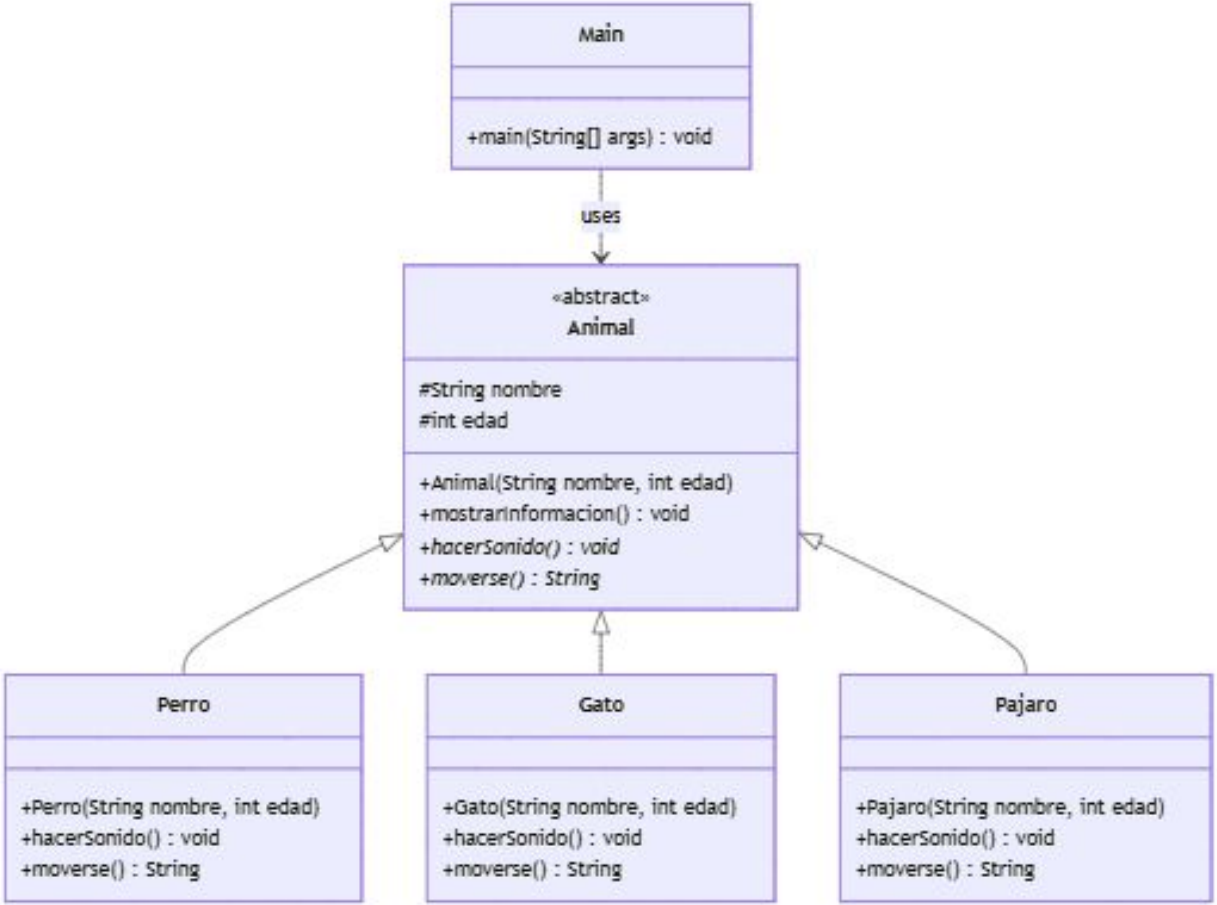
¿Para qué sirven?

- Sirven como base para otras clases
- Declaran el “contrato” que las subclases deben cumplir
- Facilita el polimorfismo y la abstracción

Clases abstractas - Ejemplo

```
abstract class Animal {
    protected String nombre;

    public abstract void hacerSonido();

    public void dormir() {
        System.out.println(nombre + " está durmiendo.");
    }
}

class Perro extends Animal {
    public void hacerSonido() {
        System.out.println("Guau!");
    }
}
```

Ejercicio 12



Un, dos, tres, responde otra vez

How to participate?

1 Go to wooclap.com

2 Enter the event code in the top banner

Event code
RGDLUO

Enable answers by SMS

Atributos y métodos finales

Atributos Finales:

- Son constantes; su valor no puede cambiarse después de la inicialización.
- Deben inicializarse al declararse o en el constructor.
- Ejemplo: `final int MAX_VALUE = 100;`

Métodos Finales:

- No pueden ser sobrescritos por subclases.
- Garantizan que la implementación permanezca inalterada.
- Ejemplo:

```
public final void display() {
    System.out.println("Este es un método final.");
}
```

Clases finales

- Una clase declarada como `final` no puede ser heredada por otras clases.
- Se utiliza la palabra clave `final` antes de la declaración de la clase.
- Todos los métodos de una clase final son implícitamente finales.
- Ejemplo:

```
public final class MiClaseFinal {

    // Contenido de la clase

}
```

Usos comunes:

- Clases de utilidad (como `java.lang.Math`)
- Clases inmutables (como `java.lang.String`)
- Clases que no deben ser extendidas por razones de seguridad o diseño →evitamos que nadie la modifique.

Ejercicio 4.13

Creación de una Clase de Utilidades Matemáticas

Variables finales de tipo objeto


```
final Persona p = new Persona("Paco");
```

¿Puedo hacer cambios en el objeto p?

```
p.nombre = "Fran";
```

¿Puedo cambiar la referencia de p?

```
p = new Persona("Fran");
```

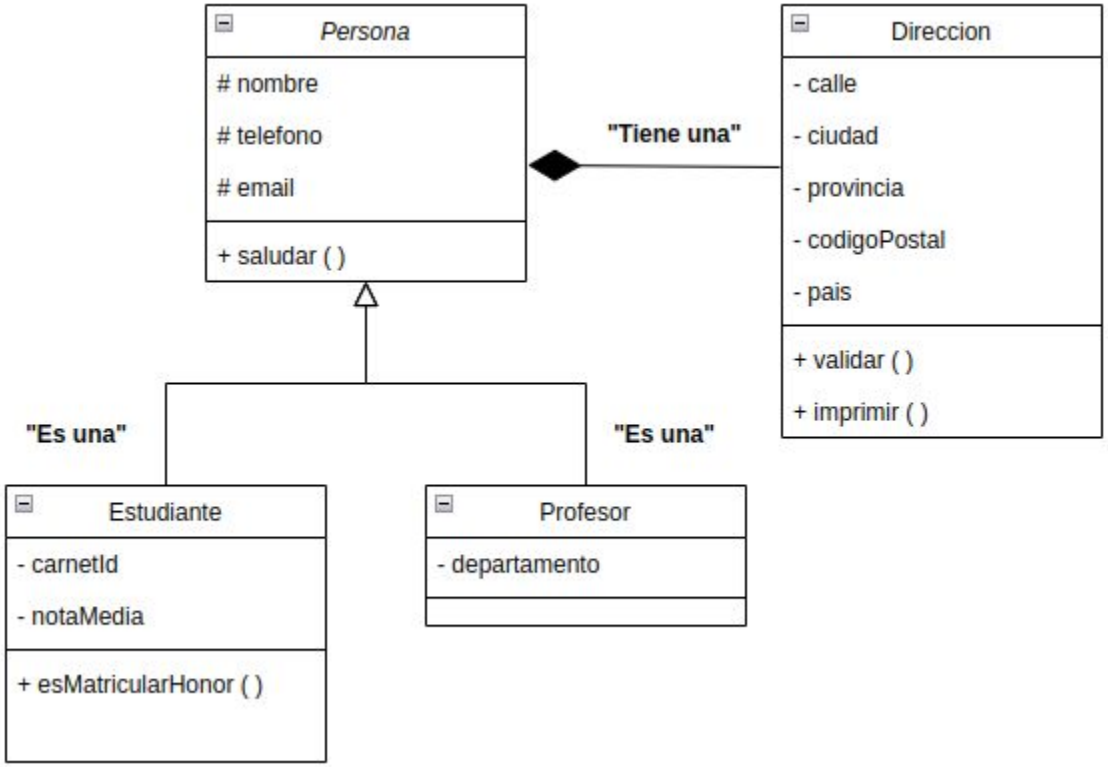
Your wooclap poll will be displayed here

Install the **Chrome** or
Firefox extension

Make sure you are in
presentation mode

wooclap

Ejercicio 4.14: COMPOSICIÓN Y HERENCIA




```java
public class Persona {
    private String nombre;
    private String telefono;
    private String email;

    private Direccion direccion;
```

```java
public class Estudiante extends Persona{
    private Integer carnet;
    private Float notaMedia;
```

Paso por valor y por referencia


```
Persona p1 = new Persona();
p1.setNombre("Fran");
```

```
public void setNombre(String nombre) {   nombre: "Fran"
    this.nombre = nombre;   nombre: "Fran"   nombre: null
}
```

```
Direccion d1 = new Direccion();   d1: Direccion@697
d1.setCalle("13 Rue del Percebe");
```

```
p1.setDireccion(d1);   p1: Persona@692   d1: Direccion@697
```

```
direccion = {Direccion@697}
> calle = "13 Rue del Percebe"
  ciudad = null
  provincia = null
  codigoPostal = null
  pais = null
```

```
public void setDireccion(Direccion direccion) {   direccion: Direccion@697
    this.direccion = direccion;   direccion: Direccion@697   direccion: null
}
```

Paso por valor y por referencia

- Los argumentos se pueden pasar de dos formas:
  - Por valor → se pasa una copia del original (cambios en el parámetro **no afectan** al original)
  - Por referencia → se pasa la referencia (dirección de memoria) del original (cambios en el parámetro **sí afectan** al original)

Paso por valor y por referencia

- En Java siempre se pasan los argumentos por valor, pero…
  - Argumentos de tipo primitivo → Se pasa la copia del dato
  - Argumentos de tipo objeto → Se pasa la copia de la referencia

Debido a esta gestión de la memoria, los argumentos tipo objeto **¡sí modifican su valor original!**

Ejercicio 4.15

Implementa y ejecuta la siguiente clase y comprueba los resultados obtenidos con los que esperabas.

```
public class ComparacionParametros {

    public static void modificarPrimitivo(int numero) {
        numero = numero * 2;
        System.out.println("Dentro del método, numero = " + numero);
    }

    public static void modificarObjeto(StringBuilder texto) {
        texto.append(" modificado");
        System.out.println("Dentro del método, texto = " + texto);
    }

    public static void main(String[] args) {
        int valorPrimitivo = 5;
        StringBuilder valorObjeto = new StringBuilder("Hola");

        System.out.println("Antes de llamar al método, valorPrimitivo = " + valorPrimitivo);
        modificarPrimitivo(valorPrimitivo);
        System.out.println("Después de llamar al método, valorPrimitivo = " + valorPrimitivo);

        System.out.println("\nAntes de llamar al método, valorObjeto = " + valorObjeto);
        modificarObjeto(valorObjeto);
        System.out.println("Después de llamar al método, valorObjeto = " + valorObjeto);
    }
}
```