Unidad
Programación Orientada a Objetos

POO

Contenido:

- Introducción a la POO
- Clases y Objetos
- Atributos y métodos
- Constructores
- Paquetes

POO

- El paradigma de POO está ampliamente extendido y la mayoría de los lenguajes de programación soportan la OO.
- Se considera que la POO resuelve los problemas de forma más cercana a como se resuelven los problemas en el mundo real, con la manipulación y comportamiento de los **objetos**.

POO

- Frente a la programación estructurada presenta las siguientes ventajas:
  - Mejora el rendimiento de ejecución
  - Proporciona estructuras más claras de los programas
  - Facilita la reutilización de código
  - La información está más protegida porque es posible controlar el nivel de acceso a los datos de los objetos. (encapsulamiento)

- Los elementos principales cuando trabajamos con POO son:
  - **Clases**
  - **Objetos**

Clases y Objetos

- Un **objeto** en la POO es la representación lógica (informática) de un objeto del mundo real.
- Los objetos de las aplicaciones se comportan como los objetos que representan del mundo real, se definen:
  - Los **datos** que identifican los objetos → **atributos**
  - El **comportamiento** de los objetos → **métodos**
  - Las **relaciones** entre diferentes objetos → **mensajes**

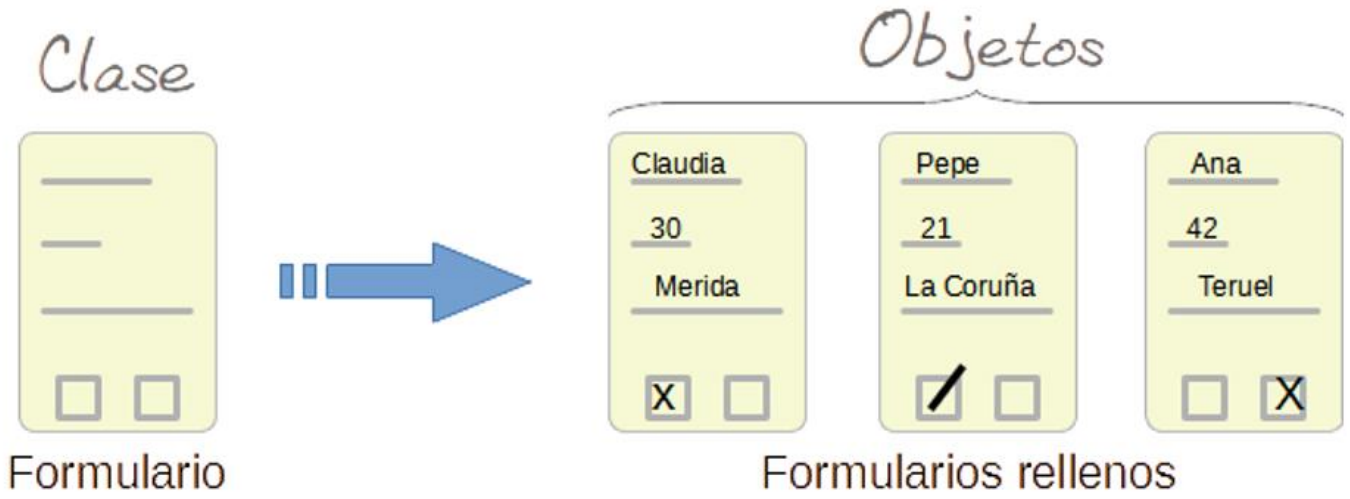
Clases y Objetos

- Una **clase** se puede entender como la **abstracción de objetos que comparten las mismas características**.
  - Ejemplos: Objetos: manzana, ciruela, fresa → Clase: Fruta
    Objetos: BMV, Mercedes, Volvo → Clase: Coche

- ¿Qué es la abstracción?
  - Reducir un concepto a los elementos básicos, sin entrar en detalles 'internos'. ¿Cómo se utiliza? ¿Para qué sirve?
    - Ejemplos: una bombilla, un semáforo, un televisor, un coche…

Clases y Objetos

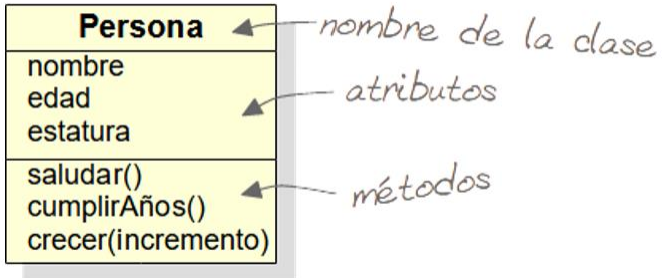
- En definitiva, la clase es la plantilla, molde, a partir de la cual se crean los objetos de dicha clase.



Clases y Objetos

- **Definición de una clase**
  - Predefinidas en el lenguaje de programación (String, Character..)
  - Propias, creadas para proyectos específicos.

```
clase Coche {
        // atributos
        String modelo;
        String marca;
        int anio;
        // métodos
        public void arrancar(){ }
        public void parar(){}

}
```



Representación de una clase en un diagrama de clases

Clases y Objetos

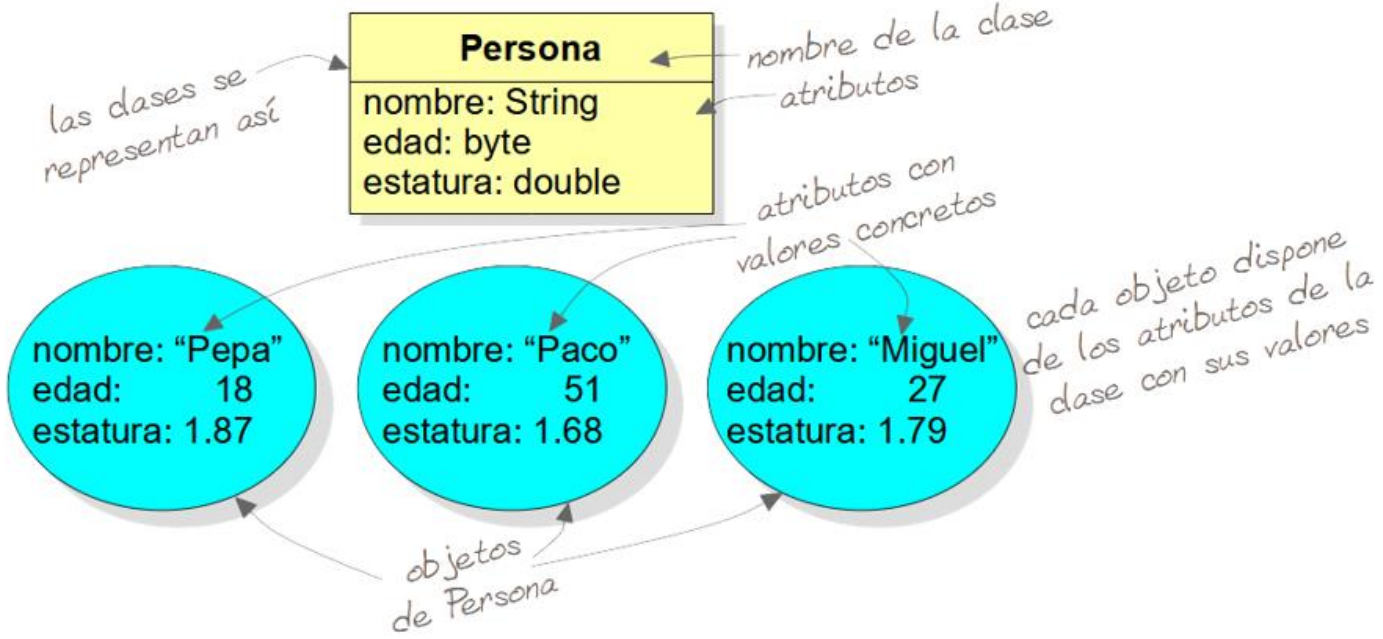
- **Definición de un objeto**
  - Se indica a qué clase pertenece y se le asigna un nombre. De forma similar a la declaración de variables de tipos primitivos.

```
String cadena;
Coche coche01;  // así se define, pero no se crea realmente.
```

  - Una variable de un tipo de clase almacena la r**eferencia al objeto**, y no un valor como sí ocurre con las variables de tipos primitivos.

Clases y Objetos

- **Instanciación**: es la creación/construcción de objetos concretos, con atributos concretos, de una determinada clase.

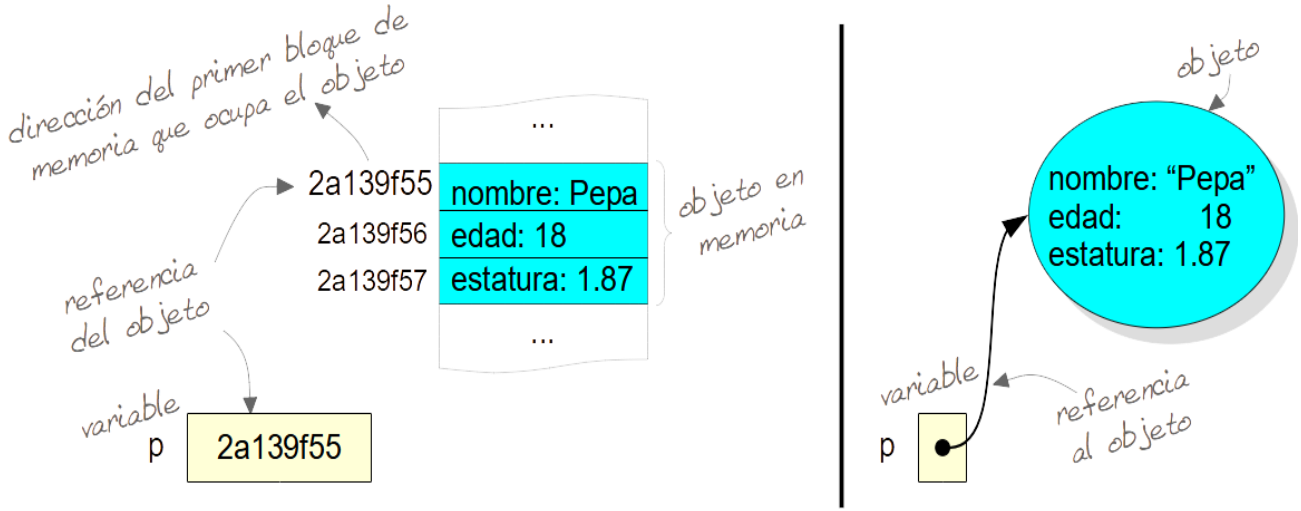


Clases y Objetos

- **Operador new**: operador para la creación de objetos:
  1. Busca hueco en memoria para el objeto
  2. Asigna la referencia al objeto a la variable creada

- **Referencia**: primera dirección de memoria que identifica un objeto (puntero).

p = new Persona();



Clases y Objetos

- **Referencia null**: el valor literal null es una referencia nula, es decir una referencia a ningún bloque.
  - En caso de pretender acceder a un objeto que no existe(referencia null) se obtiene la excepción (error) de '**Null pointer exception**'
  - Casos que pueden provocar esta situación

    1. Crear un objeto (con new) sin asignarlo a una variable

       new(Persona)

    2. Asignar null a la(s) variable(s) que contienen la referencia a un objeto

    3. Sobrescribir la variable con la referencia a otro objeto

       Persona p = new(Persona); → p referencia a un objeto X

       p = new(Persona); → p referencia a un nuevo objeto Y, quedando el X sin referencia.

- **Recolector de basura** (garbage collector, gc): Proceso que se ejecuta periódicamente de forma transparente al desarrollador y que elimina los objetos no referenciados.

Métodos y Atributos

- **Métodos**: funciones que se definen dentro de una clase y que implementan el comportamiento/operaciones que puede realizar un objeto.
- **Referencia a métodos y atributos**:
  - Una vez creado un objeto de una determinada clase, podemos **acceder a los atributos** mediante el nombre de la variable objeto seguido de **punto (.)**

    Ejemplo: asignar valor a los atributos del objeto referenciado en la variable p

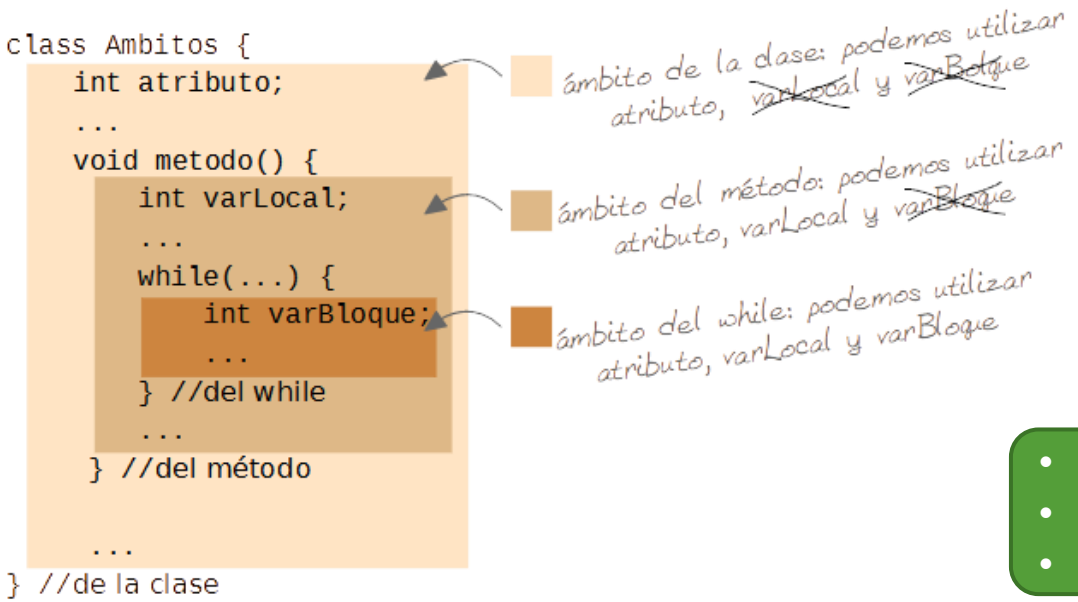
    ```
    p = new Persona();
    p.nombre = "Pepe"; p.edad  = 23 ;
    ```

  - Desde el objeto, también se puede **invocar a los métodos** con el nombre de la variable seguido de **punto (.)**

    ```
    p.saludar();
    ```

Ámbito de las variables y atributos

- El ámbito de una variable/atributo define dónde puede utilizarse, coincide con el bloque en el que se ha declarado.



- Variables de bloque.
- Variables locales (definidas en un método).
- Atributos (variables definidas en una clase).

Ámbito de las variables y atributos

- **Ocultación de atributos**. En general no se puede utilizar el mismo nombre de variable en diferentes ámbitos anidados.
- Sin embargo, existe una **excepción**: una **variable local en un método** puede tener el mismo nombre que un **atributo** de la clase.
  - En este caso, dentro del método, la **variable local prevalece** sobre el atributo. → la variable local oculta al atributo.

- **this**: palabra reservada que se refiere a la clase actual. Las clases utilizan this para refererirse a sí mismas.

- Utilizar this en el ámbito de la clase permite acceder a los atributos aunque estén ocultos.

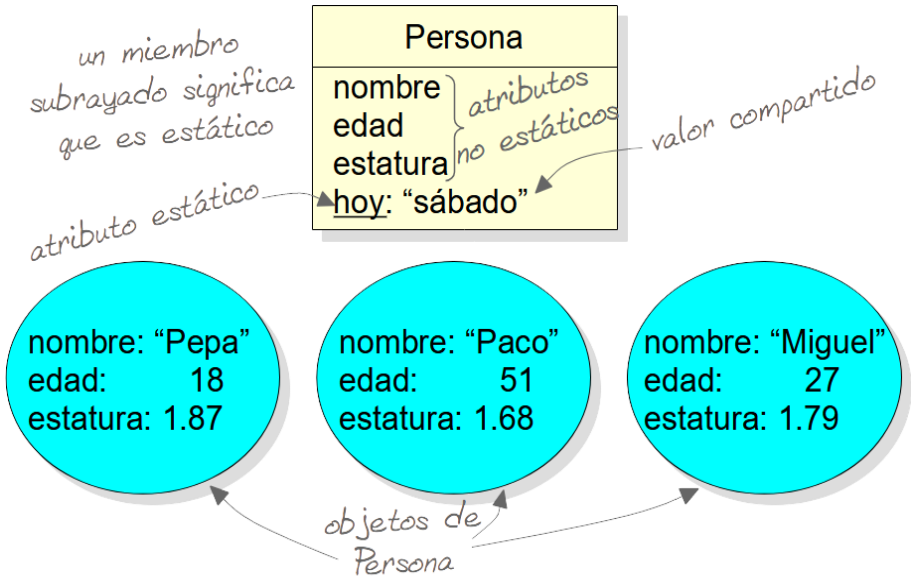
Atributos estáticos

- **Atributo estático**. También llamado atributo de clase, es aquel cuyo valor es compartido por todos los objetos de la clase.
- El atributo estático no pertenece a los objetos, sino a la clase.
- Se utiliza la palabra reservada **static** para declarar un atributo como estático
- Se accede al atributo estático desde la clase:

```
Persona.hoy = "viernes";
```

- El atributo estático se inicializa al cargar la clase en memoria, es decir, cuando se crea el primer objeto

```
class Persona {
    static String hoy= "viernes";
}
```



Métodos estáticos

- **Método estático**. Es aquel que no necesita de ningún objeto para ejecutarse.
- El método estático no puede utilizar atributos que no sean estáticos.
- Desde un método estático solo se pueden invocar métodos y atributos estáticos.

  Por eso hasta ahora todos los métodos implementados eran estáticos : estaban incluidos en el método **main, que siempre es estático*

  - Sin embargo, sí se pueden crear objetos de cualquier clase, incluyendo la del método estático y desde ese objeto invocar métodos no estáticos.

Constructores

- **Los constructores**. Son métodos especiales que se llaman como la propia clase y que nos van a servir para instanciar nuevos objetos. La motivación principal es asignar valor a los atributos
- Se define sin tipo devuelto, ni siquiera se indica void.
- Si en una clase no se define ningún constructor, java genera uno por defecto (sin parámetros). En el momento que se crea un primer constructor, el generado por defecto desaparece.
- Es posible definir **más de un constructor para una clase**, dependiendo de las necesidades u opciones que se quiera ofrecer.
- Los diferentes constructores se distinguen entre sí por el número/tipo de parámetros → **sobrecarga**

Constructores – this()

- Al utilizar la sobrecarga de los constructores es posible reutilizar la funcionalidad de un constructor invocándolo desde el nuevo constructor que se esté implementando.
- this() es un constructor genérico que se utiliza para referenciar a otro constructor
- **this()** debe ser la **primera instrucción** del constructor.

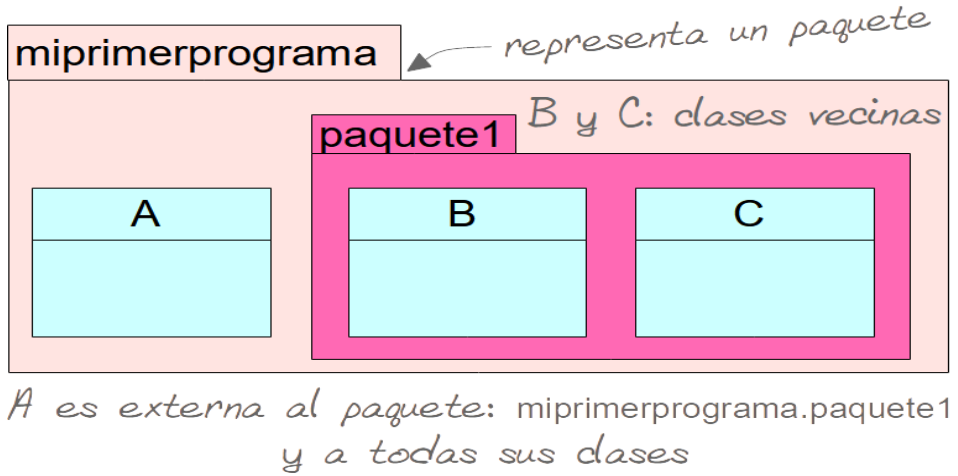
```java
// constructor inicializando todos los atributos
public Persona(String nombre, int edad, double estatura){
    this.nombre = nombre;
    this.edad= edad;
    this.estatura = estatura;
}

// constructor de Persona con solo el nombre, utilizando el constructor anterior
public Persona(String nombre){
    // constructor genérico, referencia el implementado anteriormente,
    // inicializa los atributos edad y estatura con valores fijos.
    this(nombre, edad:22, estatura:1.4);

}
```

Paquetes

- **Los paquetes** permiten controlar la accesibilidad de unas clases desde otras.
- Son contenedores que agrupan diferentes clases. Con la importación se indica qué clases se van a poder acceder desde la clase que se está implementando
- Un archivo fuente .java contiene
  - Directiva **package**
  - Directiva import
  - 1 o más clases de las cuales **solo una** puede ser declarada **pública**. (La recomendación es **solo una por .java**)
- El paquete se creará como estructura de directorios con la clasificación de los archivos fuente que se requiera.

Modificadores de acceso

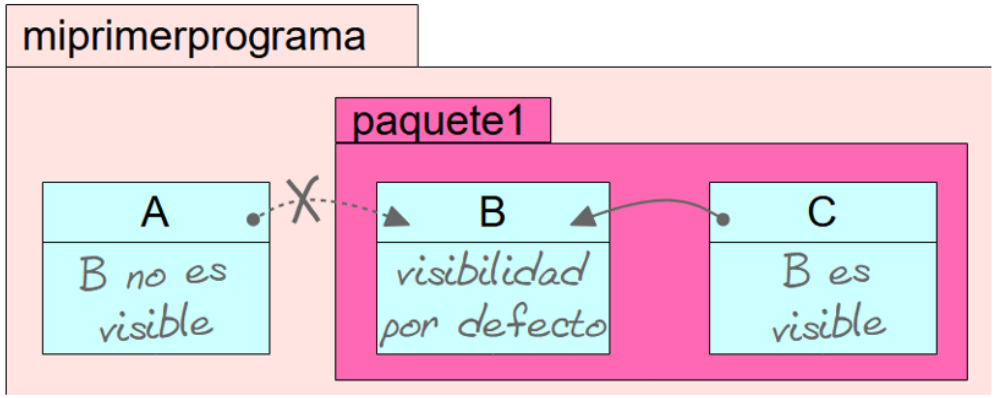
- Una clase será visible por otra dependiendo de si se encuentran o no en el mismo paquete y de los modificadores de acceso que se utilicen.
- También es posible modificar la visibilidad de métodos y atributos.



Visibilidad y Acceso a clases

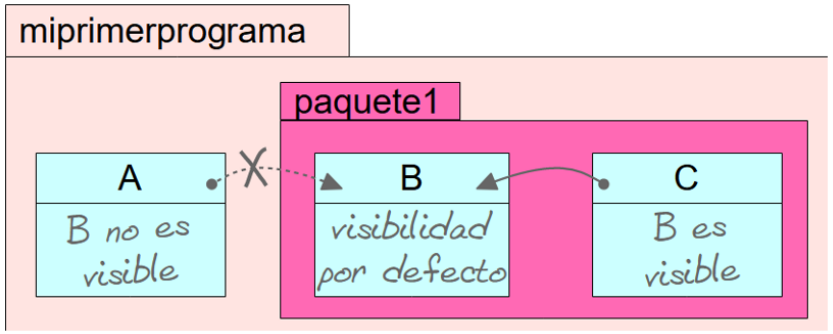
- Relación entre clases:
  - **Vecinas**, si están en el mismo paquete
  - **Externas**, si no están en el mismo paquete

- **Visibilidad por defecto**, sin modificadores de acceso: solo visible por clases vecinas



Visibilidad y Acceso a clases

- Relación entre clases:
  - **Vecinas**, si están en el mismo paquete
  - **Externas**, si no están en el mismo paquete
- **Visibilidad por defecto**, sin modificadores de acceso: solo visible por clases vecinas



- **Visibilidad total**→ modificador **public**. La clase que quiera usar una clase externa pública, deberá **importarla**.

Visibilidad y Acceso a clases

Una clase siempre será visible por sus clases vecinas. Que sea visible —previa importación— por clases externas dependerá de si está declarada como pública

	Visible desde…	
	clases vecinas	clases externas
Sin modificador	✓	
Public	✓	✓

Visibilidad y Acceso a métodos/atributos

- Que un atributo sea visible significa que puede accederse para lectura o modificación
- Que un método sea visible significa que se puede invocar.
- Para que un atributo/método sea visible es necesario que su clase también lo sea.
- Dentro de una clase, todos los miembros son visibles independientemente del modificador.

	Visible desde…		
	la propia clase	clases vecinas	clases externas
private	✓		
sin modificador	✓	✓	
public	✓	✓	✓

- El acceso a miembros privados deberá realizarse desde métodos públicos.

getters / setters

- **getters**: métodos públicos que se invocan para consultar atributos privados.
- **setters**: métodos públicos que se invocan para la actualización de atributos privados. Suelen incluir validación de los datos.
- Se identifican con get/set + nombre atributo:

```
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;
   }
}
```