2️⃣

Clases y objetos

▼ Objetos y clases

Aunque parezca una obviedad, la base de la programación orientada a objetos es el objeto. En la vida real todos los objetos tienen una serie de características y un comportamiento. Por ejemplo, una puerta tiene color, forma, dimensiones, material... (goza de una serie de características) y puede abrirse, cerrarse... (posee un comportamiento). En programación orientada a objetos, un objeto es una combinación de unos datos específicos y de las rutinas que pueden operar con esos datos.

De forma que los dos tipos de componentes de un objeto son:

- **Propiedades o atributos:** componentes de un objeto que almacenan datos. También se les denomina variables miembro. Estos datos pueden ser de tipo primitivo (boolean, int, double, char...) o, a su vez, de otro tipo de objeto (lo que se denomina agregación o composición de objetos). La idea es que un atributo representa una propiedad determinada de un objeto.
- **Funciones o métodos:** es un componente de un objeto que lleva a cabo una determinada acción o tarea con los atributos. En principio, todas las variables y rutinas de un programa de Java deben pertenecer a una clase. De hecho, en Java no hay noción de programa principal y las subrutinas no existen como unidades modulares independientes, sino que forman siempre parte de alguna clase.

▼ Clases

Una clase representa al conjunto de objetos que comparten una estructura y un comportamiento comunes. Una clase es una combinación específica de atributos y métodos y puede considerarse un tipo de dato de cualquier tipo no primitivo. Así, una clase es una especie de plantilla o prototipo de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar

con esos objetos. Aunque, por otro lado, una clase también puede estar compuesta por métodos estáticos que no necesitan de objetos (como las clases construidas en los capítulos anteriores que contienen un método estático main). La declaración de una clase sigue la siguiente sintaxis:

```
[modificadores] class IdentificadorClase {
  //Declaraciones de atributos y metodos
}
```

Una instancia es un elemento tangible (ocupa memoria durante la ejecución del programa) generado a partir de una definición de clase. Todos los objetos empleados en un programa han de pertenecer a una clase determinada.

▼ Ejemplo:

El siguiente código muestra la declaración de la clase Precio. La clase Precio consta de un único atributo (euro) y dos métodos: uno que asigna un valor al atributo (set) sin devolver ningún valor y otro que devuelve el valor del atributo (get).

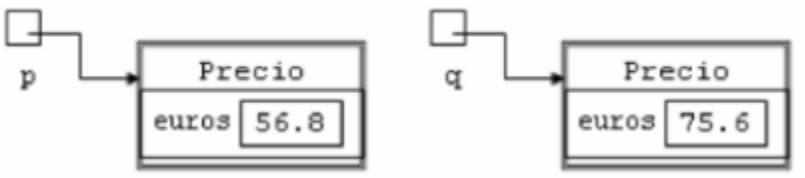
```
public class Precio {
  //Atributo o variable miembro
  public double euros;
  //Metodos
  public double get() {
    return this.euros;
  }
  public double set() {
  }
}
```

La clase Precio no es directamente ejecutable por el intérprete, ya que el código fuente no incluye ningún método principal (main). Para poder probar el código anterior, puede construirse otro archivo con el código fuente que se muestra a continuación:

```
public class PruebaPrecio {
  public static void main(String[] args) {
    Precio p=new Precio(); //Crea e inicializa un objeto de la clase Precio
    p.set(56.8); //Llama al metodo set que asigna 56.8 al atributo euros
    //Llamada al metodo get que devuelve el valor de euros
    System.out.println("Valor="+p.get());
    Precio q=new Precio(); //Crea una referencia y el objeto
    q.euros=75.6; //Asigna 75.6 al atributo euros
    System.out.println("Valor="+q.euros);
```

```
  }
}
```

Representación gráfica del espacio de la memoria utilizado por las referencias e instancias de la clase Precio durante la ejecución del método main de la clase PruebaPrecio.



▼ Constructores

Los constructores permiten inicializar los atributos de los objetos que se creen de una clase, es decir, permiten asignarles un valor inicial determinado. Ahora bien, si en una clase el programador no define ningún constructor, el compilador de Java creará uno por defecto y, cuando se cree un objeto de dicha clase, a sus atributos se les asignará automáticamente valores por defecto.

Cuando se crea un objeto en un programa escrito en Java, si no se especifica otra cosa, a los atributos se les asigna un valor por defecto:

- **0** si son números enteros o reales.
- '**\u0000**' si son caracteres.
- **false** si son datos lógicos.
- **null** si son objetos.

Uso de constructores en Java

- Constructor por defecto
- Constructor sin parámetros
- Constructores con parámetros

▼ Constructor por defecto

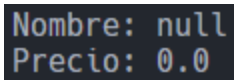
Dada la siguiente clase Articulo, donde no se ha definido ningún constructor:

```
public class Articulo {
  private String nombre;
  private double precio;
  public String getNombre() {
    return nombre;
  }
  public double getPrecio() {
    return precio;
  }
  public void setNombre(String n) {
    nombre=n;
  }
  public void setPrecio(String p) {
    precio=p;
  }
}
```

Al compilar y ejecutar el siguiente código:

```
public class PruebaArticulo {
  public static void main(String[] args) {
    Articulo a1=new Articulo();
    System.out.println("Nombre: "+ai.getNombre());
    System.out.println("Precio: "+ai.getPrecio());
  }
}
```

En la pantalla, se podrá observar que al atributo nombre del objeto creado y referenciado por la variable a1, se le ha asignado el valor null y, al atributo precio, el valor 0.0:



Ejecución del programa PruebaArticulo escrito en Java, donde se pueden ver los valores por defecto asignados a los atributos de un objeto.

Esto es así, ya que, al no definirse ningún constructor en el código fuente de la clase Articulo, Java crea uno por defecto y, por tanto, cuando en el programa se crea el objeto referenciado por la variable a1, a los atributos de dicho objeto (nombre y precio) se les asigna valores por defecto.

No obstante, si a los atributos, nombre y precio del objeto creado, inicialmente, se les quisiera asignar los valores "Vaso" y 3.3, respectivamente. Para ello, se podría definir un constructor sin parámetros o con parámetros.

▼ Constructor sin parámetros

En el ejemplo anterior de la clase Articulo se ha definido un constructor sin parámetros:

```
public class Articulo {
  private String nombre;
  private double precio;
  public Articulo() {
    nombre="Vaso";
    precio=3.3;
  }
}
```

Como se puede apreciar, un constructor tiene que llamarse igual que la clase a la que pertenece (en este caso Articulo) y, además, un constructor no se indica su tipo, ya que no devuelve ningún valor.

Al compilar y ejecutar el siguiente código fuente:

```
public class PruebaArticulo {
  public static void main(String[] args) {
    Articulo a1=new Articulo();
    System.out.println("Nombre: "+ai.getNombre());
    System.out.println("Precio: "+ai.getPrecio());
  }
}
```

```
Nombre: Vaso
Precio: 3.3
```

En este caso, el constructor se ha definido sin parámetros, que es cómo se le ha invocado después de la palabra clave new en la sentencia: `Articulo a1=new Articulo();`

Por otra parte, al igual que con los métodos, en Java se pueden definir constructores con parámetros.

▼ Constructores con parámetros

En Java se pueden definir constructores a los que se les pasen uno o más parámetros (argumentos). Dada la siguiente clase Producto, donde se ha definido un constructor con tres parámetros:

```
public class Producto {
  private String nombre;
  private double precio;
  private int cantidad;
  public Producto(String nombre,double precio,int cantidad) {
    this.nombre=nombre;
    this.precio=precio;
    this.cantidad=cantidad;
  }
  public String getNombre() {
    return nombre;
  }
  public double getPrecio() {
    return precio;
  }
  public double getCantidad() {
    return cantidad;
  }
  public void setNombre(String nombre) {
    this.nombre=nombre;
  }
  public void setPrecio(double precio) {
    this.precio=precio;
  }
  public void setCantidad(int cantidad) {
    this.cantidad=cantidad;
  }
}
```

Se ha utilizado varias veces la palabra clave this, haciendo referencia al objeto actual. De esta forma, por ejemplo el escribir `this.nombre=nombre` en el bloque de código del constructor, se está diferenciando entre el atributo nombre del objeto (this.nombre) y el parámetro formal del constructor llamado igualmente nombre. En el caso de que el identificador de dicho atributo y de dicho parámetro hubiesen sido distintos, no hubiese sido necesario hacer uso de this.

Al compilar y ejecutar el código fuente siguiente:

```java
import java.util.Scanner;

public class ProgProducto
{
    public static void main(String[] args)
    {
        String nombreProducto;
        double precioProducto;
        int cantidadProducto;

        Scanner teclado = new Scanner(System.in);

        System.out.print("Introduzca nombre: ");
        nombreProducto = teclado.nextLine();

        System.out.print("Introduzca precio: ");
        precioProducto = teclado.nextDouble();

        System.out.print("Introduzca cantidad: ");
        cantidadProducto = teclado.nextInt();

        Producto p1 = new Producto(nombreProducto, precioProducto, cantidadProducto);

        System.out.printf("Nombre: %s\n", p1.getNombre());
        System.out.printf("Precio: %f\n", p1.getPrecio());
        System.out.printf("Cantidad: %d\n", p1.getCantidad());
    }
}
```

En la pantalla, se podrá visualizar algo parecido a:

```
Introduzca nombre: Cuchara
Introduzca precio: 1,25
Introduzca cantidad: 40
Nombre: Cuchara
Precio: 1,250000
Cantidad: 40
```

▼ Sobrecarga de métodos y constructores

En una clase, la sobrecarga (overloading) permite definir más de un constructor o método con el mismo nombre, con la condición de que no puede haber dos de ellos con el mismo número y tipo de parámetros.

Dada la siguiente clase Perro,
donde se ha definido el método
cambiar sobrecargado:

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

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

    public String getNombre()
    {
        return nombre;
    }

    public int getEdad()
    {
        return edad;
    }

    public void cambiar(String nombre)
    {
        this.nombre = nombre;
    }

    public void cambiar(int edad)
    {
        this.edad = edad;
    }

    public void cambiar(String nombre, int edad )
    {
        this.nombre = nombre;
        this.edad = edad;
    }
}
```

El método cambiar se ha definido tres veces: con un parámetro de tipo String, con un parámetro de tipo int y con dos parámetros cuyos tipos son String e int.

Al compilar y ejecutar el siguiente código fuente, donde se invoca al método cambiar pasándole uno o dos argumentos:

```java
public class PruebaPerro
{
    public static void main(String[] args)
    {
        Perro perro1 = new Perro("Chispas", 5);
        Perro perro2 = new Perro("Sombra", 3);
        Perro perro3 = new Perro("Zeus", 7);

        System.out.println(perro1.getNombre() + " tiene " + perro1.getEdad() + " años.");
        System.out.println(perro2.getNombre() + " tiene " + perro2.getEdad() + " años.");
        System.out.println(perro3.getNombre() + " tiene " + perro3.getEdad() + " años.");

        perro1.cambiar("Jaque");
        perro2.cambiar(4);
        perro3.cambiar("Goku", 8);

        System.out.println("\nDespués de los cambios:");

        System.out.println(perro1.getNombre() + " tiene " + perro1.getEdad() + " años.");
        System.out.println(perro2.getNombre() + " tiene " + perro2.getEdad() + " años.");
        System.out.println(perro3.getNombre() + " tiene " + perro3.getEdad() + " años.");
    }
```

En la pantalla, se verá:

```
Chispas tiene 5 años.
Sombra tiene 3 años.
Zeus tiene 7 años.

Después de los cambios:
Jaque tiene 5 años.
Sombra tiene 4 años.
Goku tiene 8 años.
```

Dada la siguiente clase Equipo, donde el constructor está sobrecargado:

```java
public class Equipo
{
    private String nombre;
    private String ciudad;
    private int puntos;

    public Equipo(String nombre, String ciudad, int puntos)
    {
        this.nombre = nombre;
        this.ciudad = ciudad;
        this.puntos = puntos;
    }

    public Equipo(String nombre)
    {
        this.nombre = nombre;
        this.ciudad = "ciudad desconocida";
    }

    public String getNombre()
    {
        return nombre;
    }

    public String getCiudad()
    {
        return ciudad;
    }

    public int getPuntos()
    {
        return puntos;
    }
}
```

Al constructor se le puede invocar pasándole uno o tres argumentos. De modo que, al compilar y ejecutar el siguiente código fuente:

```java
public class PruebaEquipo
{
    public static void main(String[] args)
    {
        Equipo equipo1 = new Equipo("Gladiadores", "Valencia", 5);
        Equipo equipo2 = new Equipo("Ases");

        System.out.println("El equipo " + equipo1.getNombre() + " de " + equipo1.getCiudad() + " tiene " + equipo1.getPuntos() + " puntos.");
        System.out.println("El equipo " + equipo2.getNombre() + " de " + equipo2.getCiudad() + " tiene " + equipo2.getPuntos() + " puntos.");
    }
}
```

En la pantalla, se mostrará lo siguiente:

```
El equipo Gladiadores de Valencia tiene 5 puntos.
El equipo Ases de ciudad desconocida tiene 0 puntos.
```

▼ Métodos Getters y Setters

Hay métodos que nos son de utilidad para obtener o almacenar algún valor de un atributo de una clase, comúnmente son llamados getters y setters.

Los getters (de la palabra inglés get - obtener) indica que podemos tomar algún valor de un atributo y los setters (de la palabra inglés set-poner/fijar) podemos guardar algún valor sobre un atributo. Son importantes al momento de crear una clase objeto, ya que de ellos dependen el valor que pueden tomar los atributos o para modificar algún atributo sin necesidad de modificar algún otro atributo.

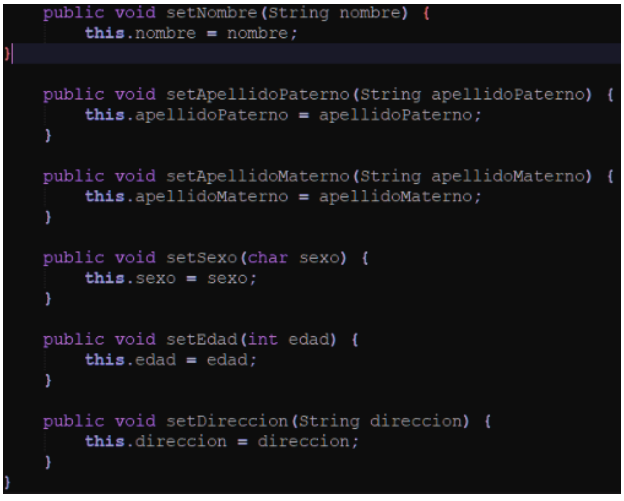
▼ Creación de Getters y Setters

Hay una estructura para crear los getters y setters, en los métodos getters siempre nos retornará el valor del atributo sin necesidad de pasar ningún parámetro, mientras que en los métodos setters siempre nos pedirá algún valor como parámetro para guardarlo al atributo de la clase, y este nunca deberá retornar algún valor.

Nota: El nombre del método depende de cada programador, pero para tomar un buen estilo de programación es recomendable anteponer la palabra get o set y el nombre del atributo utilizando la notación camelCase.

Ejemplo de los métodos get

Ejemplo de los métodos set



```java
public class Persona
{
    public String nombre;
    public String apellidoPaterno;
    public String apellidoMaterno;
    public char sexo;
    public int  edad;
    public String direccion;


    public String getNombre() {
        return nombre;
}

    public String getApellidoPaterno() {
        return apellidoPaterno;
    }

    public String getApellidoMaterno() {
        return apellidoMaterno;
    }

    public char getSexo() {
        return sexo;
    }

    public int getEdad() {
        return edad;
    }

    public String getDireccion() {
        return direccion;
    }
}
```

Para implementar estos métodos nos vamos a la clase Principal que contiene el método main, creamos algunos objetos “Persona” con un constructor por defecto (Que tendrá todos los atributos vacíos) y llamamos a los métodos set para asignarle datos a los atributos, después visualizamos los datos obteniéndolos con los métodos get.

```java
public class Principal {

    public static void main(String[] args) {

            //Guardamos los datos
            Persona persona=new Persona();
            persona.setNombre("Linda");
            persona.setApellidoPaterno("Gómez");
            persona.setApellidoMaterno("Pérez");
            persona.setSexo('M');
            persona.setEdad(28);
            persona.setDireccion("Calle Ariban, Madrid");

            //Obtenemos los datos y los imprimimos
            String nombre=persona.getNombre();
            String app=persona.getApellidoPaterno();
            String apm=persona.getApellidoMaterno();
            char sexo=persona.getSexo();
            int edad=persona.getEdad();
            String direccion=persona.getDireccion();

            System.out.println("Los datos de la persona son: "+nombre+", "+app+", "+apm+", "+sexo+", "+edad+", "+direccion);
    }

}
```

```
Los datos de la persona son: Linda, Gómez, Pérez, M, 28, Calle Ariban, Madrid
```

▼ Tipos de Clases

Clases públicas (public): una clase declarada como pública (public) es accesible desde cualquier otra clase del mismo paquete, también se puede acceder desde otra clase que pertenezca a otro paquete importando la clase [`import NombrePaquete.NombreClase;`]. Pueden ser heredadas.

Sintaxis: `public class cuentaBancaria`

Clases abstractas (abstract): son clases que sirven de base para la herencia. Particularidades:

- No pueden ser instanciadas, es decir, no se pueden crear objetos a partir de ellas.
- Pueden contener métodos con o sin implementación y no pueden definirse como final. También los métodos son definidos como abstractos.
- Generalmente, una clase abstracta es del tipo public a la vez.
- Cuando una clase hereda de una clase abstracta, la primera está obligada a definir o implementar todos los métodos definidos en la clase base.
- Si algún método está implementado en la clase base, al derivar este puede ser sobreescrito (a conveniencia) en la clase que lo hereda.
- Si una clase contiene un método abstracto, dicha clase debe declararse también como abstracta.
- Una clase declarada como abstracta no puede contener métodos privados, ya que si se hereda la clase no sería posible la implementación de los métodos privados.

  Sintaxis: `public abstract class Personas`

Clases Finales (final): son clases que no se pueden heredar. El motivo principal por el que una clase se declara como final obedece a la seguridad, ya que no permite que se modifique las definiciones creadas. También la declaración de las clases como final otorga una mayor eficiencia, puesto que como no son heredables se trabaja solo con las instancias de la propia clase. Una gran cantidad de las librerías de java están declaradas como final.

▼ Modificadores de acceso

Los modificadores de acceso nos introducen al concepto de encapsulamiento. El encapsulamiento busca de alguna forma controlar el acceso a los datos que conforman un objeto o instancia, de este modo podríamos decir que una clase y por ende sus objetos que hacen uso de modificadores de acceso (especialmente privados) son objetos encapsulados.

Los modificadores de acceso permiten dar un nivel de seguridad mayor a nuestras aplicaciones, restringiendo el acceso a diferentes atributos, métodos, constructores, asegurándonos que el usuario deba seguir una "ruta" especificada por nosotros para acceder a la información.

Haciendo uso de los modificadores de acceso podremos asegurarnos de que un valor no será modificado incorrectamente por parte de otro programador o usuario. Generalmente, el acceso a los atributos se consigue por medio de los métodos get y set, pues es estrictamente necesario que los atributos de una clase sean privados.

- Siempre se recomienda que los atributos de una clase sean privados y por tanto, cada atributo debe tener sus propios métodos get y set para obtener y establecer respectivamente el valor del atributo.
- Siempre que se use una clase de otro paquete, se debe importar usando import. Cuando dos clases se encuentran en el mismo paquete no es necesario hacer el import, pero esto no significa que se pueda acceder a sus componentes directamente.

▼ Modificador public

El modificador de acceso public es el más permisivo de todos, básicamente public es lo contrario a private en todos los aspectos (lógicamente), esto quiere decir que si un componente de una clase es public, tendremos acceso a él desde cualquier clase o instancia sin importar el paquete o procedencia de esta.

▼ Modificador private

El modificador private en Java es el más restrictivo de todos, básicamente cualquier elemento de una clase que sea privado puede ser accedido únicamente por la misma clase, nada más. Es decir, si por ejemplo, un atributo es privado, solo puede ser accedido por los métodos o constructores de la misma clase. Ninguna otra clase sin importar la relación que tengan podrá tener acceso a ellos.

▼ Modificador por defecto (default)

Java nos da la opción de no usar un modificador de acceso y al no hacerlo, el elemento tendrá un acceso conocido como default, acceso por defecto que permite que tanto la propia clase como las clases del mismo paquete accedan a dichos componentes (de aquí la importancia de declararle siempre un paquete a nuestras clases).

▼ Modificador protected

El modificador de acceso protected nos permite acceso a los componentes con dicho modificador desde la misma clase, clases del mismo paquete y clases que hereden de ella (incluso en diferentes paquetes).

Modificador	Misma Clase	Mismo paquete	Subclase	Otro paquete
private	Si	No	No	No
default	Si	Si	No	No
protected	Si	Si	Si/No	No
public	Si	Si	Si	Si