



Cofinanciado por
la Unión Europea



MINISTERIO
DE TRABAJO
Y ECONOMÍA SOCIAL



Fondos Europeos



Junta de Andalucía
Instituto Andaluz del
Deporte y la Juventud
www.juntadeandalucia.es



Ayuda
en Acción
Impulsa Empleo Joven

Programación orientada a Objetos.

Índice:

- 1. Clases y Objetos**
- 2. Modificadores de Acceso en Java**
- 3. Constructores**
- 4. Getters y Setters**
- 5. Programación orientada a objetos en Java (POO)**
 - Encapsulamiento
 - Abstracción
 - Herencia
 - Polimorfismo



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.^a ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



1. Clases y objetos en Java

En Java, **una clase** es una plantilla o molde que define las propiedades (**atributos**) y comportamientos (**métodos**) de un objeto.

Un objeto es una instancia concreta de una clase, con valores específicos en sus atributos y la capacidad de ejecutar sus métodos.

💡 Concepto Básico

- **Clase:** Define las características y comportamientos comunes.
- **Objeto:** Es una instancia de la clase, con valores concretos.

Creación de una clase Coche

```
// Definición de la clase Coche
class Coche {
    // Atributos (propiedades)
    String marca;
    int velocidad;

    // Método (comportamiento)
    void acelerar() {
        velocidad += 10;
        System.out.println("Velocidad actual: " + velocidad + " km/h");
    }
}
```

El nombre de la clase indicamos la primera letra en mayúsculas.





Creando y Usando un Objeto

Ahora creamos un objeto de la clase Coche en el método main y lo usamos:

```
public class Main {
    public static void main(String[] args) {
        // Crear un objeto de la clase Coche
        Coche miCoche = new Coche();

        // Asignar valores a los atributos
        miCoche.marca = "Toyota";
        miCoche.velocidad = 0;

        // Usar el método acelerar
        System.out.println("Marca: " + miCoche.marca);
        miCoche.acelerar(); // Aumenta la velocidad en 10 km/h
    }
}
```



Explicación:

- Clase Coche:** Define atributos (marca, velocidad) y un método (acelerar).
 - Objeto miCoche:** Se crea en main usando *new Coche()*.
 - Uso del Objeto:** Se asignan valores a los atributos y se llama al método *acelerar()*.
 - Para acceder a los atributos y métodos de un objeto se usa la notación punto, "objeto.atributo, objeto.metodo"**
- ◇ Con esto ya puedes empezar a trabajar con clases y objetos en Java.





2. Modificadores de Acceso en Java

Al definir una clase en Java, los **modificadores de acceso** determinan qué tan accesibles son sus propiedades y métodos desde otras partes del código.

💡 Tipos de Modificadores de Acceso en Java

Existen cuatro modificadores de acceso para propiedades (**variables o atributos**), métodos y clases:

Modificador	Accesible dentro de la misma clase	Accesible en el mismo paquete	Accesible en subclases (herencia)	Accesible desde cualquier parte
private	<input checked="" type="checkbox"/> Sí	<input type="checkbox"/> No	<input type="checkbox"/> No	<input type="checkbox"/> No
(Sin modificador) (default)	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí	<input type="checkbox"/> No	<input type="checkbox"/> No
protected	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí	<input checked="" type="checkbox"/> Sí	<input type="checkbox"/> No
public	<input checked="" type="checkbox"/> Sí			

1. private (Privado)

Un atributo o método **solo puede ser accedido dentro de la misma clase**. Es el nivel de acceso más restrictivo.

💡 Ejemplo de private:

```
class Persona {
    private String nombre; // Solo accesible dentro de esta clase

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

    public String getNombre() { // Método público para acceder a `nombre`
        return nombre;
    }
}

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona("Juan");
        // System.out.println(p.nombre); ✗ ERROR: `nombre` es privado
        System.out.println(p.getNombre()); // ✅ Acceso correcto mediante un método público
    }
}
```

Usamos un método `getNombre()` para acceder a `nombre`, ya que es un atributo privado.





2. (Sin modificador) - "default" (Por defecto)

Si no especificamos un modificador, el acceso está **limitado al mismo paquete**. Otras clases dentro del mismo paquete pueden acceder, pero **no las que estén fuera del paquete**.

Ejemplo de acceso default (sin modificador):

```
class Persona { // Clase sin modificador → Solo accesible en el mismo paquete
    String nombre; // Propiedad sin modificador → Solo accesible en el mismo paquete

    public Persona(String nombre) {
        this.nombre = nombre;
    }
}
```

Se puede acceder desde otra clase en el mismo paquete, pero no desde otro paquete.

3. protected (Protegido)

- ✓ Permite acceso **dentro del mismo paquete**.
- ✓ Permite acceso desde **clases hijas (subclases)**, incluso si están en otro paquete.

Ejemplo de protected:

```
class Persona {
    protected String nombre; // Accesible en el mismo paquete y en subclases

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

class Empleado extends Persona {
    public Empleado(String nombre) {
        super(nombre);
    }

    public void mostrarNombre() {
        System.out.println("Nombre del empleado: " + nombre); // ✓ Acceso permitido
    }
}
```

Empleado puede acceder a nombre porque es protected y está heredando de Persona.





4. public (Público)

- ✓ **Accesible desde cualquier parte del programa.**
- ✓ Puede ser accedido desde otras clases y paquetes sin restricciones.

Ejemplo de public:

```
class Persona {
    public String nombre; // Accesible desde cualquier parte

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

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona("Ana");
        System.out.println(p.nombre); // ✓ Acceso permitido
    }
}
```

nombre es public, así que puede ser accedido directamente desde cualquier parte.

Resumen

Modificador	Nivel de acceso
private	Solo dentro de la misma clase
(sin modificador) (default)	Solo dentro del mismo paquete
protected	Dentro del mismo paquete y subclases
public	Accesible desde cualquier parte

¿Cuál usar?

1. **private** → Para ocultar datos sensibles ( Mejor práctica).
2. **Default (sin modificador)** → Cuando solo quieres acceso dentro del mismo paquete.
3. **protected** → Cuando necesitas acceso desde clases hijas.
4. **public** → Para métodos o atributos que deben estar disponibles en todo el código.

En el siguiente ejemplo vemos cada uno de los diferentes Modificadores.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.ª ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



Cofinanciado por
la Unión Europea



MINISTERIO
DE TRABAJO
Y ECONOMÍA SOCIAL



Fondos Europeos



```

class Persona {
    private String nombre; // Atributo privado

    // Método privado: solo accesible dentro de esta clase
    private void metodoPrivado() {
        System.out.println("Este método es privado");
    }

    // Método default (sin modificador): accesible en el mismo paquete
    void metodoDefault() {
        System.out.println("Este método es default");
    }

    // Método protegido: accesible en el mismo paquete y en subclases
    protected void metodoProtegido() {
        System.out.println("Este método es protegido");
    }

    // Método público: accesible desde cualquier parte
    public void metodoPublico() {
        System.out.println("Este método es público");
    }
}

```

```

// Método para demostrar el uso de "private"
public void probarMetodos() {
    metodoPrivado(); // ✅ Se puede llamar dentro de la misma clase
}

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona();

        // p.metodoPrivado(); ❌ Error: No se puede acceder a un método privado
        // p.metodoDefault(); ❌ Error: No se puede acceder si está en otro paquete
        // p.metodoProtegido(); ❌ Error: No se puede acceder si está en otro paquete y no es
        // el mismo que el que lo define

        p.metodoPublico(); // ✅ Se puede acceder desde cualquier parte
        p.probarMetodos(); // ✅ Método público que llama a un método privado internamente
    }
}

```



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.ª ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



En algunos de los ejemplos vemos el uso de **this**:

Cuando un atributo de la clase tiene el mismo nombre que un parámetro del constructor o método, usamos **this** para referirnos al atributo de la clase.

Modificadores adicionales para métodos

Además de los **modificadores de acceso**, los métodos pueden tener otros modificadores que afectan su comportamiento:

Modificador	Descripción
static	Método de clase (no necesita una instancia para ser llamado)
final	El método no puede ser sobreescrito en subclases
abstract	Método sin implementación, que debe ser implementado por las subclases
synchronized	Permite que solo un hilo ejecute el método a la vez (programación concurrente)

Ejemplo de static

Los métodos static pertenecen a la **clase**, no a una instancia específica

```
class Utilidades {
    public static void mostrarMensaje() {
        System.out.println("Mensaje desde un método estático");
    }
}

public class Main {
    public static void main(String[] args) {
        Utilidades.mostrarMensaje(); // ✅ Se puede llamar sin crear un objeto
    }
}
```

No es necesario instanciar Utilidades para usar mostrarMensaje().





Ejemplo de final

Un método final **no puede ser sobrescrito** en una subclase.

```
class Animal {
    public final void hacerSonido() {
        System.out.println("Este sonido no puede ser cambiado");
    }
}

class Perro extends Animal {
    // ❌ Esto daría error:
    // public void hacerSonido() {
    //     System.out.println("Guau guau");
    // }
}
```

Ejemplo de abstract

Los métodos abstract **no tienen implementación** y deben ser implementados por las subclases.

```
abstract class Animal {
    abstract void hacerSonido(); // Método sin implementación
}

class Perro extends Animal {
    @Override
    void hacerSonido() {
        System.out.println("Guau guau");
    }
}

public class Main {
    public static void main(String[] args) {
        Perro p = new Perro();
        p.hacerSonido(); // ✅ Implementación obligatoria en La subclase
    }
}
```

Se usa cuando queremos que las subclases definan su propio comportamiento.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.^a ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



¿Cuál usar?

- 1** **private** → Si el método solo debe usarse dentro de la clase.
- 2** **public** → Si el método debe ser accesible desde cualquier parte.
- 3** **protected** → Si necesitas que subclases accedan al método.
- 4** **static** → Si el método pertenece a la clase y no a una instancia.
- 5** **final** → Si quieres evitar que las subclases modifiquen el método.
- 6** **abstract** → Si quieres que las subclases definan su propia implementación.

3. Constructores

No, **no siempre es necesario** definir un constructor en una clase en Java, pero **siempre existe uno**. Si no escribimos un constructor, **Java crea un constructor por defecto de manera automática**.

¿Qué es un constructor?

Un **constructor** es un método especial que se ejecuta cuando se crea un objeto de la clase. Se usa para **inicializar** los atributos del objeto.

◊ Ejemplo de clase con constructor explícito

```
class Persona {
    String nombre;

    // Constructor de la clase
    public Persona(String nombre) {
        this.nombre = nombre;
    }

    public void mostrarNombre() {
        System.out.println("Nombre: " + nombre);
    }
}

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona("Juan"); // Se llama al constructor
        p.mostrarNombre(); // Imprime: Nombre: Juan
    }
}
```

el constructor inicializa el atributo nombre cuando se crea el objeto.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.ª ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



¿Qué pasa si no definimos un constructor?

Si no escribimos un constructor, **Java crea un constructor por defecto.**

Este constructor:

- ✓ **No recibe parámetros.**
- ✓ **No realiza ninguna acción especial**, solo permite crear objetos.
- ◊ **Ejemplo sin constructor (se usa el constructor por defecto)**

```
class Persona {
    String nombre; // No hay constructor explícito
}

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona(); // Java usa un constructor vacío
        System.out.println(p.nombre); // Imprime: null (porque no se inicializó)
    }
}
```

Java nos permite crear objetos, aunque no definamos un constructor.





4.Getters y Setters

Los **getters** y **setters** son métodos que permiten acceder y modificar **atributos privados** de una clase.

- **Setters (set)** → Modifican el valor de un atributo.
- **Getters (get)** → Obtienen el valor de un atributo.

```
class Coche {
    private String color; // Atributo privado

    // Setter: Permite modificar el color del coche
    public void setColor(String color) {
        this.color = color;
    }

    // Getter: Permite obtener el color del coche
    public String getColor() {
        return color;
    }
}

public class Main {
    public static void main(String[] args) {
        Coche miCoche = new Coche();
        miCoche.setColor("Rojo"); // Asigna un color
        System.out.println("El color del coche es: " + miCoche.getColor());
    }
}
```





5. Programación Orientada a Objetos (POO) en Java

Principios fundamentales:

- 1 Encapsulamiento**
- 2 Abstracción**
- 3 Herencia**
- 4 Polimorfismo**

Encapsulamiento

El **encapsulamiento** protege los datos de una clase evitando accesos directos desde fuera de la misma.

Para esto:

- ✓ Usamos **private** en los atributos.
- ✓ Usamos métodos **get** y **set** para acceder/modificar los atributos.

```
class Persona {
    private String nombre; // Atributo privado

    // Constructor
    public Persona(String nombre) {
        this.nombre = nombre;
    }

    // Getter (para obtener el nombre)
    public String getNombre() {
        return nombre;
    }

    // Setter (para modificar el nombre)
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}

public class Main {
    public static void main(String[] args) {
        Persona p = new Persona("Juan");
        System.out.println(p.getNombre()); // Accede de forma controlada

        p.setNombre("Carlos"); // Modifica el nombre de forma segura
        System.out.println(p.getNombre());
    }
}
```



Ventaja: Protege los datos y permite control sobre su acceso/modificación.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.ª ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



Abstracción

La **abstracción** oculta los detalles internos y solo expone lo esencial de un objeto.

- ✓ Se usa con **clases abstractas o interfaces**.
- ✓ Nos permite definir métodos sin implementarlos aún (abstract).

```
abstract class Figura {
    abstract double calcularArea(); // Método sin implementación
}

class Circulo extends Figura {
    private double radio;

    public Circulo(double radio) {
        this.radio = radio;
    }

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

public class Main {
    public static void main(String[] args) {
        Figura f = new Circulo(5);
        System.out.println("Área del círculo: " + f.calcularArea());
    }
}
```

ventaja: Permite definir un comportamiento común sin preocuparnos por los detalles.

@override se usa en Java para indicar que un método está **sobrescribiendo** un método de una clase padre o interfaz, asegurando que la firma del método es correcta.

Si volvemos a definir un método de la clase padre sin indicar **@override** estamos **Sobreponiendo**.



REGISTRO NACIONAL DE ASOCIACIONES N°611922
DECLARADA ENTIDAD DE UTILIDAD PÚBLICA ESTATAL
AGENCIA DE COLOCACIÓN: ID 0100000017

CENTRO EN MÁLAGA
C/DOS ACERAS 23, 29012
MÁLAGA | (+34) 952 300 500
ARRABAL@ARRABALEMPLEO.ORG

CENTRO EN CÁDIZ
TR.^a ALAMEDA DE SOLANO, 32, 11130
CHICLANA DE LA FRONTERA | (+34) 956 900 312
CHICLANA@ARRABALEMPLEO.ORG



Herencia

La **herencia** permite que una clase herede atributos y métodos de otra.

- ✓ Se usa con `extends`.
- ✓ Evita repetir código.

```
class Animal {
    String nombre;

    public void hacerSonido() {
        System.out.println("Hace un sonido...");
    }
}

class Perro extends Animal { // Perro hereda de Animal
    public void ladrar() {
        System.out.println(nombre + " está ladrando 🐶");
    }
}

public class Main {
    public static void main(String[] args) {
        Perro miPerro = new Perro();
        miPerro.nombre = "Firulais";
        miPerro.hacerSonido(); // Método heredado
        miPerro.ladrar(); // Método propio de Perro
    }
}
```

Ventaja: Reutiliza código y permite modelar relaciones entre objetos.



Polimorfismo

El **polimorfismo** permite que un mismo método tenga diferentes comportamientos según el objeto que lo use.

- ✓ Se da con **métodos sobreescritos (@Override)** o **sobrecargados**.
- ✓ Permite usar una misma referencia para múltiples tipos.

```
class Animal {
    public void hacerSonido() {
        System.out.println("Hace un sonido...");
    }
}

class Gato extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("Miau 🐱");
    }
}

class Perro extends Animal {
    @Override
    public void hacerSonido() {
        System.out.println("Guau 🐶");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal miAnimal1 = new Gato(); // Polimorfismo
        Animal miAnimal2 = new Perro();

        miAnimal1.hacerSonido(); // Llama a la versión de Gato
        miAnimal2.hacerSonido(); // Llama a la versión de Perro
    }
}
```





```

class Animal {
    public void hacerSonido() {
        System.out.println("El animal hace un sonido genérico");
    }
}

class Lobo extends Animal {
    @Override
    public void hacerSonido() {
        super.hacerSonido(); // Llama al método de la superclase
        System.out.println("El lobo aúlla: ¡Auuuu!");
    }
}

public class Main {
    public static void main(String[] args) {
        Lobo miLobo = new Lobo();
        miLobo.hacerSonido();
    }
}

```

El animal hace un sonido genérico
El lobo aúlla: ¡Auuuu!

Si queremos llamar al método original de la superclase dentro de la sobrescritura, usamos **super**.

Sobrescritura: Cambiar el comportamiento de un método heredado.

Sobrecarga: Crear múltiples métodos con el mismo nombre, pero con diferentes parámetros.

Principio	Descripción	Ejemplo
Encapsulamiento	Restringe acceso a datos usando private y métodos get/set	Ocultamos el atributo nombre en Persona
Abstracción	Oculta detalles internos y solo expone lo necesario	Clase Figura con método calcularArea() abstracto
Herencia	Permite que una clase herede atributos y métodos de otra	Perro hereda de Animal
Polimorfismo	Un método puede comportarse de diferentes maneras según el objeto	hacerSonido() se comporta distinto en Perro y Gato

