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

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

   public String getNombre() {
return nombre;
}

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

```
public class Estudiante extends Persona {
   private String curso;

      public Estudiante(String nombre, int edad, String curso) {
      super(nombre, edad);  // Llamada al constructor de la clase
base
      this.curso = curso;
   }


   public String toString() {
      return super.toString() + ", Curso: " + curso;
   }
}
```

```
public class Principal{
   public static void main(String[] args) {
      Estudiante unEstudiante = new Estudiante("Lucía", 20, "2º DAM");
      System.out.println(unEstudiante);
      System.out.println(unEstudiante.getNombre());
   }
}
```

EJERCICIO 1: Completa la clase Coche y Vehiculo

public class Vehiculo {

```
private String marca;
private int año;

public Vehiculo(String marca , int año) {
    this.marca = marca;
    this.año = año;
}

@Override
public String toString() {
    return "Marca: " + marca + ", Año: " + año;
}


}
```

public class Coche extends Vehiculo{

```
private int puertas;

public Coche(String marca, int año, int puertas {
    super(marca , año);   // Llamada al constructor
de Vehiculo
    this.puertas = puertas;
}

@Override
public String toString() {
    return super.toString() + ", Puertas: " +
puertas;
}


}
```

EJERCICIO 2 : Completa las clases

```
public class Notificacion {

   private String mensaje;

   public Notificacion(String mensaje) {
      this.mensaje = mensaje;
   }

   public String getMensaje() {
      return mensaje;
   }

   // Método que las clases hijas deben especializar
   public void enviar() {
      System.out.println("Enviando notificación
genérica: " + mensaje);
   }
}
```

```
public class EmailNotificacion extends Notificacion {

private String direccionEmail;


public EmailNotificacion(String direccionEmail, String
mensaje) {
    // Llamar al constructor de la clase padre
    super(mensaje);

    this.direccionEmail = direccionEmail;
}

// Sobrescribir el método enviar()
@Override
public void enviar() {
super.enviar();
System.out.println(direccionEmail)
}


}
```

EJERCICIO 3

Debes crear una **clase hija** que herede de `Animal`:

1️⃣ Clase Perro

- Debe tener un atributo extra: `raza`.
- Su constructor debe llamar a `super(nombre, edad)`.
- Debe **sobrescribir** el método `sonido()` para que devuelva `"Guau"`.
- Debe **sobrescribir** `toString()` incluyendo el resultado de `super.toString()` y la raza.

```
public class Animal {

  private String nombre;
  private int edad;

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

  public String getNombre() {
    return nombre;
  }
```

```
Public class Perro extends Animal {
Private String raza;

public Perro(String nombre, int edad, String raza){
super(nombre, edad);
this.raza = raza;
}
public String getRaza(){
return raza;
}
public void setRaza(String raza){
this.raza = raza;
}
```

public int getEdad() {
return edad;
}
 	
public String sonido() {
return "Sonido genérico";
}
 	
@Override
public String toString() {
return "Nombre: " + nombre + ", Edad: " + edad;
}
}
	public String sonido(){
	return super.sonido() + “guau”;
	}
	@Override
	public String toString() {
	return super.toString() + "Raza: " + raza;
	}
	
	}

EJERCICIO 4

Sistema de Vehículos Compartidos

Una empresa de movilidad urbana quiere un programa para gestionar **vehículos compartidos**.

Requisitos:

Todos los vehículos tienen atributos comunes:

- `idVehiculo` (único para cada vehículo, autoincremental)
- `marca`
- `modelo`

- `precioPorHora`

Todos los vehículos deben tener métodos:

- `calcularPrecioAlquiler(int horas)` → devuelve el precio total.
- `descripcion()` → devuelve una cadena con información del vehículo.

2️⃣ Tipos de vehículos

La empresa gestiona distintos tipos de vehículos, cada uno con **atributos específicos** y **comportamientos distintos**:

- **Bicicleta**:
  - Atributo: `tipoFreno` (por ejemplo: "Disco", "V-BRAKE")
  - Método sobrescrito `descripcion()`: incluye tipo de freno.
- **Coche**:
  - Atributos: `numPuertas`, `esElectrico` (boolean)
  - Método sobrescrito `calcularPrecioAlquiler()`: si es eléctrico, aplica un descuento del 10%.
- **Patinete**:
  - Atributo: `autonomiaBateria` (en km)
  - Método sobrescrito `descripcion()`: incluye autonomía.

3️⃣ Programa principal

- Crear al menos **un objeto de cada tipo de vehículo**.Mostrar información y precio de alquiler para un tiempo determinado (ej.: 3 horas).

EJERCICIO 5

Ejemplo en el que la clase hija *necesita* llamar al método del padre (y donde, por tanto, se necesita exponer un método para hacerlo desde `main` o desde fuera).

Imagina que tienes un sistema donde la clase **Padre** valida pagos básicos, y la clase **Hija** añade validaciones extra (por ejemplo, para pagos internacionales).

Pero en ciertos casos necesitas **forzar** la validación básica del padre *sin* aplicar la validación de la hija.

```
class ProcesadorPago {
    public boolean validar(double cantidad) {
        System.out.println("Validación básica del pago");
        return cantidad > 0;
    }
}


class ProcesadorPagoInternacional extends ProcesadorPago {

    @Override
    public boolean validar(double cantidad) {
        System.out.println("Validación internacional más estricta");
        return cantidad > 0 && cantidad < 5000;
    }

    // Método que expone la validación original del padre
    public boolean validarBasico(double cantidad) {
```

```
        return super.validar(cantidad);
    }
}
```

✔ ¿Por qué podría necesitarse?

Imagina que:

- Para pagos normales quieres la validación estricta.
- Pero **para pruebas**, operaciones internas o auditorías necesitas la **validación básica** del padre.

Desde `main`, puedes acceder a ambas:

✔ Main

```
public class Main {
    public static void main(String[] args) {
        ProcesadorPagoInternacional proc = new ProcesadorPagoInternacional();

        System.out.println("Validación estricta:");
        System.out.println(proc.validar(6000));  // Falla por validación de la hija

        System.out.println("\nValidación básica:");
        System.out.println(proc.validarBasico(6000));  // Usa validación del padre (pasa)
    }
}
```

📜 Salida

```
Validación estricta:
Validación internacional más estricta
false

Validación básica:
Validación básica del pago
true
```