

<https://github.com/Coman-Che/Programacion-II/tree/main/Unidad%207>

# PROGRAMACIÓN II

## Trabajo Práctico 7: Herencia y Polimorfismo en Java

### OBJETIVO GENERAL

Comprender y aplicar los conceptos de herencia y polimorfismo en la Programación Orientada a Objetos, reconociendo su importancia para la reutilización de código, la creación de jerarquías de clases y el diseño flexible de soluciones en Java.

### MARCO TEÓRICO

Concepto	Aplicación en el proyecto
Herencia	Uso de `extends` para crear jerarquías entre clases, aprovechando el principio is-a.
Modificadores de acceso	Uso de private, protected y public para controlar visibilidad.
Constructores y super	Invocación al constructor de la superclase con super(...) para inicializar atributos.
Upcasting	Generalización de objetos al tipo de la superclase.
Instanceof	Comprobación del tipo real de los objetos antes de hacer conversiones seguras.
Downcasting	Especialización de objetos desde una clase general a una más específica.
Clases abstractas	Uso de abstract para definir estructuras base que deben ser completadas por subclases.

Métodos abstractos	Declaración de comportamientos que deben implementarse en las clases derivadas.
Polimorfismo	Uso de la sobrescritura de métodos (@Override) y llamada dinámica de métodos.
Herencia	Uso de `extends` para crear jerarquías entre clases, aprovechando el principio is-a.

1

## Caso Práctico

Desarrollar las siguientes Katas en Java aplicando herencia y polimorfismo. Se recomienda repetir cada kata para afianzar el concepto.

### 1. Vehículos y herencia básica

- Clase base: Vehículo con atributos marca, modelo y método **mostrarInfo()**
- Subclase: Auto con atributo adicional **cantidadPuertas**, sobrescribe **mostrarInfo()**
- Tarea: Instanciar un auto y mostrar su información completa.

Rta:

```
package Ejercicio_1;
```

```
public class Vehiculo {
```

```
    protected String marca;
    protected String modelo;
```

```
    public Vehiculo(String marca, String modelo) {
        this.marca = marca;
        this.modelo = modelo;
    }
```

```
    public void mostrarInfo() {
        System.out.println("Marca: " + marca + ", Modelo: " + modelo);
    }
```

```
}
```

---

```
package Ejercicio_1;
```

```
public class Auto extends Vehiculo {
```

```

private int cantidadPuertas;

public Auto(String marca, String modelo, int cantidadPuertas) {
    super(marca, modelo); // Invoca al constructor de Vehiculo
    this.cantidadPuertas = cantidadPuertas;
}

@Override
public void mostrarInfo() {
    System.out.println("Auto - Marca: " + marca + ", Modelo: " + modelo
        + ", Puertas: " + cantidadPuertas);
}
}

```

---

```

package Ejercicio_1;

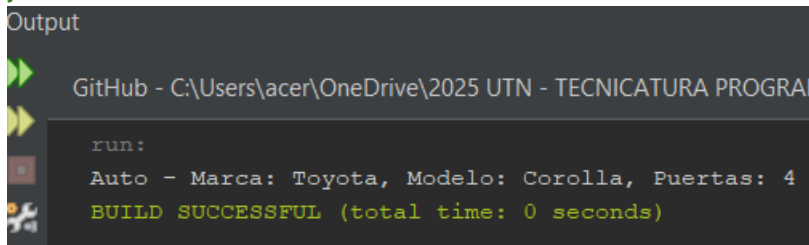
public class Ejercicio1_Main {

    public static void main(String[] args) {

        Auto miAuto = new Auto("Toyota", "Corolla", 4);
        miAuto.mostrarInfo();

    }
}

```



## 2. Figuras geométricas y métodos abstractos

- Clase abstracta: Figura con método **calcularArea()** y atributo nombre
- Subclases: **Círculo y Rectángulo** implementan el cálculo del área
- Tarea: Crear un array de figuras y mostrar el área de cada una usando polimorfismo.

Rta:

```

package Ejercicio_2;

public abstract class Figura {

    protected String nombre;

```

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

// Método abstracto
public abstract double calcularArea();
}
```

---

```
package Ejercicio_2;

public class Circulo extends Figura {

    private double radio;

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

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

---

```
package Ejercicio_2;

public class Rectangulo extends Figura {

    private double base;
    private double altura;

    public Rectangulo(double base, double altura) {
        super("Rectángulo");
        this.base = base;
        this.altura = altura;
    }

    @Override
    public double calcularArea() {
        return base * altura;
    }
}
```

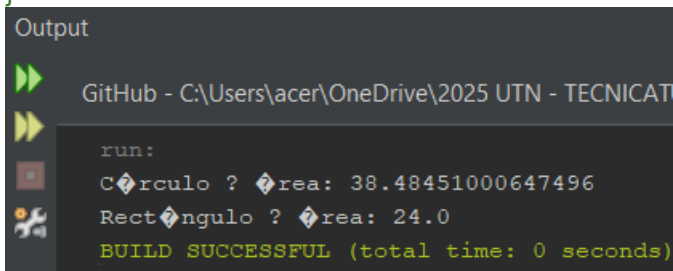
---

```
package Ejercicio_2;

public class Ejercicio2_Main {

    public static void main(String[] args) {
```

```
Figura[] figuras = {  
    new Circulo(3.5),  
    new Rectangulo(4, 6)  
};  
  
for (Figura f : figuras) {  
    System.out.println(f.nombre + " → Área: " + f.calcularArea());  
}  
  
}
```



```
Output  
GitHub - C:\Users\acer\OneDrive\2025 UTN - TECNICA  
run:  
Circulo ? Área: 38.48451000647496  
Rectangulo ? Área: 24.0  
BUILD SUCCESSFUL (total time: 0 seconds)
```

### 3. Empleados y polimorfismo

- Clase abstracta: Empleado con método **calcularSueldo()**
- Subclases: **EmpleadoPlanta**, **EmpleadoTemporal**
- Tarea: Crear lista de empleados, invocar **calcularSueldo()** polimórficamente, usar instanceof para clasificar

Rta:

```
package Ejercicio_3;
```

```
public abstract class Empleado {  
  
    protected String nombre;  
  
    public Empleado(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public abstract double calcularSueldo();  
  
}
```

---

```
package Ejercicio_3;
```

```
public class EmpleadoPlanta extends Empleado {
```

```

private double salarioBase;
private double bonificacion;

public EmpleadoPlanta(String nombre, double salarioBase, double bonificacion) {
    super(nombre);
    this.salarioBase = salarioBase;
    this.bonificacion = bonificacion;
}

@Override
public double calcularSueldo() {
    return salarioBase + bonificacion;
}
}

package Ejercicio_3;

public class EmpleadoTemporal extends Empleado {

    private int diasTrabajados;
    private double pagoPorDia;

    public EmpleadoTemporal(String nombre, int diasTrabajados, double pagoPorDia) {
        super(nombre);
        this.diasTrabajados = diasTrabajados;
        this.pagoPorDia = pagoPorDia;
    }

    @Override
    public double calcularSueldo() {
        return diasTrabajados * pagoPorDia;
    }
}

package Ejercicio_3;

import java.util.ArrayList;

public class Ejercicio3_Main {

    public static void main(String[] args) {

        ArrayList<Empleado> empleados = new ArrayList<>();
        empleados.add(new EmpleadoPlanta("Laura", 80000, 10000));
        empleados.add(new EmpleadoTemporal("Carlos", 20, 2500));

        for (Empleado e : empleados) {

```

```
Output
GitHub - C:\Users\acer\OneDrive\2025 UTN - TECNICA T
run:
Laura ? Sueldo: $90000.0
Tipo: Empleado de Planta

Carlos ? Sueldo: $50000.0
Tipo: Empleado Temporal

BUILD SUCCESSFUL (total time: 0 seconds)
```

```
public void describirAnimal() {
    System.out.println("Este es un animal llamado " + nombre + ".");
}
```

```

    }
}

package Ejercicio_4;

public class Gato extends Animal {
    public Gato(String nombre) {
        super(nombre);
    }

    @Override
    public void hacerSonido() {
        System.out.println(nombre + " dice: ¡Miau!");
    }
}

package Ejercicio_4;

public class Perro extends Animal {
    public Perro(String nombre) {
        super(nombre);
    }

    @Override
    public void hacerSonido() {
        System.out.println(nombre + " dice: ¡Guau guau!");
    }
}

package Ejercicio_4;

public class Vaca extends Animal {
    public Vaca(String nombre) {
        super(nombre);
    }

    @Override
    public void hacerSonido() {
        System.out.println(nombre + " dice: ¡Muuu!");
    }
}

package Ejercicio_4;

public class Ejercicio4_Main {

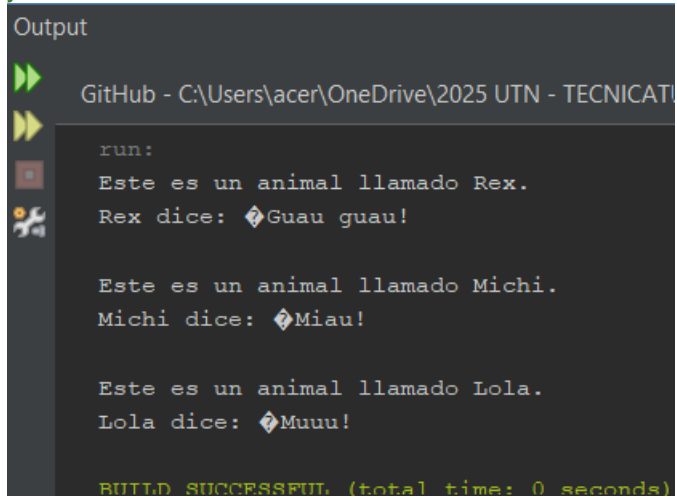
    public static void main(String[] args) {

        Animal[] animales = {
            new Perro("Rex"),
            new Gato("Michi"),
            new Vaca("Lola")
        };
    }
}

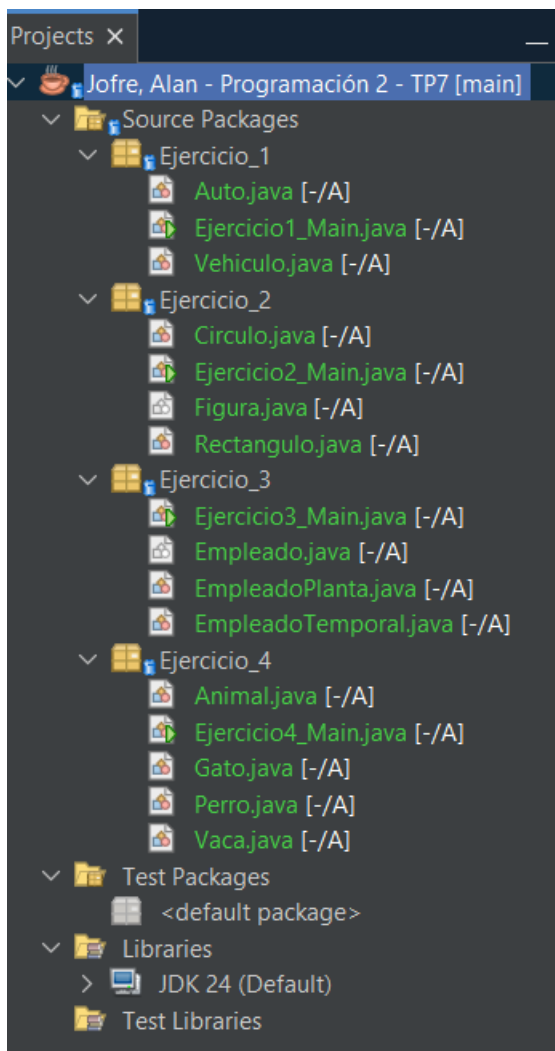
```



```
        for (Animal a : animales) {  
            a.describirAnimal();  
            a.hacerSonido();  
            System.out.println();  
        }  
    }  
}
```



```
Output  
GitHub - C:\Users\acer\OneDrive\2025 UTN - TECNICAT...  
run:  
Este es un animal llamado Rex.  
Rex dice: 🐾Guau guau!  
  
Este es un animal llamado Michi.  
Michi dice: 🐾Miau!  
  
Este es un animal llamado Lola.  
Lola dice: 🐾Muuu!  
  
BUILD SUCCESSFUL (total time: 0 seconds)
```



## CONCLUSIONES ESPERADAS

- Comprender el mecanismo de herencia y sus beneficios para la reutilización de código.
- Aplicar polimorfismo para lograr flexibilidad en el diseño de programas.
- Inicializar objetos correctamente usando **super** en constructores.
- Controlar el acceso a atributos y métodos con modificadores adecuados.
- Identificar y aplicar **upcasting**, **downcasting** y **instanceof** correctamente.
- Utilizar clases y métodos abstractos como base de jerarquías lógicas.
- Aplicar principios de diseño orientado a objetos en la implementación en Java.