UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico N° 7

Herencia y Polimorfismo en Java

Comision 16

Marinoni Macarena <*marinonimacarena@gmail.com*>

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional

Programación II

Profesor: Cortez, Alberto

Tutor: Bianchi, Neyén

16/11/2025

ÍNDICE

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.

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**

En esta primera actividad del Trabajo Práctico sobre **Herencia y Polimorfismo**, se implementó un modelo simple de clases para representar vehículos utilizando los principios fundamentales de la Programación Orientada a Objetos.

La consigna solicita crear una **clase base Vehiculo**, con los atributos marca, modelo y año, junto con el método mostrarInfo(). Luego, se debe desarrollar una **subclase Auto**, que incorpora el atributo adicional cantidadPuertas y sobrescribe el método mostrarInfo() para mostrar información más específica.

Finalmente, se instancia un objeto de la clase Auto y se muestran sus datos completos, demostrando:

- Herencia simple (relación **es-un**)
- Especialización mediante atributos adicionales
- Sobrescritura de métodos (@Override)
- Comportamiento polimórfico durante la ejecución

A continuación se adjuntan las capturas de pantalla correspondientes al código implementado y a la ejecución del programa.

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Vehiculo {
    protected String marca;
    protected String modelo;
    protected int anio;

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

    public void mostrarInfo() {
        System.out.println("Vehículo: " + marca + " " + modelo + " (" + anio + ")");
    }
}
```

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Auto extends Vehiculo {
    private int cantidadPuertas;

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

    public int getCantidadPuertas() {
        return cantidadPuertas;
    }

    public void setCantidadPuertas(int cantidadPuertas) {
        this.cantidadPuertas = cantidadPuertas;
    }

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

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Kata1Main {

    public static void main(String[] args) {

        System.out.println("===== KATA 1 - Herencia básica de Vehículo y Auto =====");

        // Creamos un vehículo genérico
        Vehiculo vehiculo = new Vehiculo("Ford", "Transit", 2015);
        vehiculo.mostrarInfo(); // llamada al método de Vehiculo

        System.out.println("----------------------------------------");

        // Creamos un auto específico (subclase)
        Auto auto = new Auto("Toyota", "Corolla", 2022, 4);
        auto.mostrarInfo(); // llamada al método sobrescrito en Auto

        System.out.println("----------------------------------------");

        // Demostración de polimorfismo: referencia de tipo Vehiculo, objeto Auto
        Vehiculo vehiculoPolimorfico = new Auto("Renault", "Sandero", 2020, 5);
        vehiculoPolimorfico.mostrarInfo();
        // Aunque la referencia es Vehiculo, se ejecuta el mostrarInfo() de Auto (polimorfismo)

        System.out.println("===== Fin KATA 1 =====");
    }
}
```

```
run:
===== KATA 1 - Herencia b◆sica de Veh◆culo y Auto =====
Veh◆culo: Ford Transit (2015)
----------------------------------------
Auto: Toyota Corolla (2022), Puertas: 4
----------------------------------------
Auto: Renault Sandero (2020), Puertas: 5
===== Fin KATA 1 =====
BUILD SUCCESSFUL (total time: 0 seconds)
```

2. Figuras geométricas y métodos abstractos

En este ejercicio se implementa una jerarquía de clases utilizando **abstracción** y **polimorfismo**. La **clase abstracta Figura** define la estructura común mediante métodos abstractos, mientras que las **subclases Circulo y Rectangulo** implementan sus propias fórmulas para calcular el área y mostrar la información correspondiente.

Luego, se instancian distintas figuras y se almacenan en un arreglo de tipo Figura, demostrando polimorfismo al invocar métodos sobrescritos en las subclases a través de referencias del tipo padre.

A continuación se muestran las capturas del código y la ejecución.

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public abstract class Figura {

    // Método abstracto: cada figura calcula su área de forma distinta
    public abstract double calcularArea();

    // Método abstracto para mostrar información
    public abstract void mostrarInfo();
}
```

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Circulo extends Figura {

    private double radio;

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

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

    @Override
    public void mostrarInfo() {
        System.out.println("Círculo - Radio: " + radio +
                " | Área: " + calcularArea());
    }
}
```

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Rectangulo extends Figura {

    private double ancho;
    private double alto;

    public Rectangulo(double ancho, double alto) {
        this.ancho = ancho;
        this.alto = alto;
    }

    @Override
    public double calcularArea() {
        return ancho * alto;
    }

    @Override
    public void mostrarInfo() {
        System.out.println("Rectángulo - Ancho: " + ancho +
                ", Alto: " + alto +
                " | Área: " + calcularArea());
    }
}
```

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Kata2Main {

    public static void main(String[] args) {

        System.out.println("===== KATA 2 - Figuras y Métodos Abstractos =====");

        Figura f1 = new Circulo(5);
        Figura f2 = new Rectangulo(4, 7);

        // Polimorfismo: array de tipo Figura que contiene distintos objetos
        Figura[] figuras = { f1, f2 };

        System.out.println("Mostrando información de las figuras:");

        for (Figura fig : figuras) {
            fig.mostrarInfo();   // Llama al método correcto según el objeto real
        }

        System.out.println("===== Fin KATA 2 =====");
    }
}
```

```
run:
===== KATA 2 - Figuras y M◆todos Abstractos =====
Mostrando informaci◆n de las figuras:
C◆rculo - Radio: 5.0 | ◆rea: 78.53981633974483
Rect◆ngulo - Ancho: 4.0, Alto: 7.0 | ◆rea: 28.0
===== Fin KATA 2 =====
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. Empleados y polimorfismo

En este ejercicio se implementa una jerarquía de clases que modela distintos tipos de empleados dentro de una organización, aplicando conceptos clave de herencia, abstracción, sobrescritura y polimorfismo.

La consigna solicita definir una clase abstracta **Empleado**, que declare el método abstracto **calcularSueldo()** y almacene el nombre del empleado. A partir de ella, se deben crear dos subclases con comportamientos diferenciados:

- EmpleadoPlanta, cuyos sueldos se calculan a partir de un salario base más un bono.
- EmpleadoTemporal, cuyo sueldo depende de las horas trabajadas y el valor por hora.

Posteriormente, se instancian diversos empleados de ambos tipos y se almacenan en una colección de tipo Empleado. Esto permite demostrar polimorfismo al invocar métodos sobrescritos mediante referencias de tipo padre. Además, se utiliza el operador instanceof para clasificar dinámicamente los empleados según su tipo concreto.

A continuación se presentan las capturas del código implementado y la ejecución del programa correspondiente.

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public abstract class Empleado {

    protected String nombre;

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

    public String getNombre() {
        return nombre;
    }

    // Método abstracto: cada tipo de empleado calcula su sueldo de forma distinta
    public abstract double calcularSueldo();

    // Método común para mostrar información básica
    public void mostrarInfo() {
        System.out.println("Empleado: " + nombre +
                " | Sueldo: $" + calcularSueldo());
    }
}
```

```java
...4 lines

package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class EmpleadoPlanta extends Empleado {

    private double salarioBase;
    private double bono;

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

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

    @Override
    public void mostrarInfo() {
        System.out.println("Empleado de Planta: " + nombre +
                " | Salario base: $" + salarioBase +
                " | Bono: $" + bono +
                " | Sueldo total: $" + calcularSueldo());
    }
```

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class EmpleadoTemporal extends Empleado {

    private double valorHora;
    private int horasTrabajadas;

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

    @Override
    public double calcularSueldo() {
        return valorHora * horasTrabajadas;
    }

    @Override
    public void mostrarInfo() {
        System.out.println("Empleado Temporal: " + nombre +
                " | Valor hora: $" + valorHora +
                " | Horas trabajadas: " + horasTrabajadas +
                " | Sueldo total: $" + calcularSueldo());
    }
}
```

```java
package Tp7_HerenciaPolimorfismo;

import java.util.ArrayList;
import java.util.List;

/**...4 lines */
public class Kata3Main {

    public static void main(String[] args) {

        System.out.println("===== KATA 3 - Empleados y Polimorfismo =====");

        // Upcasting: referencias de tipo Empleado a objetos concretos
        Empleado e1 = new EmpleadoPlanta("Ana López", 300000, 50000);
        Empleado e2 = new EmpleadoTemporal("Carlos Pérez", 2500, 80);
        Empleado e3 = new EmpleadoPlanta("María Gómez", 280000, 30000);
        Empleado e4 = new EmpleadoTemporal("Juan Díaz", 2000, 60);

        // Lista polimórfica de empleados
        List<Empleado> empleados = new ArrayList<>();
        empleados.add(e1);
        empleados.add(e2);
        empleados.add(e3);
        empleados.add(e4);

        System.out.println("Listado de empleados y sueldos (polimorfismo):\n");

        int contadorPlanta = 0;
        int contadorTemporales = 0;

        for (Empleado emp : empleados) {
            emp.mostrarInfo(); // llamada polimórfica

            // Clasificación usando instanceof
            if (emp instanceof EmpleadoPlanta) {
                System.out.println("→ Tipo: Empleado de Planta");
                contadorPlanta++;
            } else if (emp instanceof EmpleadoTemporal) {
                System.out.println("→ Tipo: Empleado Temporal");
                contadorTemporales++;
            }
```

```
run:
===== KATA 3 - Empleados y Polimorfismo =====
Listado de empleados y sueldos (polimorfismo):

Empleado de Planta: Ana L�pez | Salario base: $300000.0 | Bono: $50000.0 | Sueldo total: $350000.0
? Tipo: Empleado de Planta
-----------------------------------------
Empleado Temporal: Carlos P�rez | Valor hora: $2500.0 | Horas trabajadas: 80 | Sueldo total: $200000.0
? Tipo: Empleado Temporal
-----------------------------------------
Empleado de Planta: Mar�a G�mez | Salario base: $280000.0 | Bono: $30000.0 | Sueldo total: $310000.0
? Tipo: Empleado de Planta
-----------------------------------------
Empleado Temporal: Juan D�az | Valor hora: $2000.0 | Horas trabajadas: 60 | Sueldo total: $120000.0
? Tipo: Empleado Temporal
-----------------------------------------
Resumen:
Cantidad de empleados de planta: 2
Cantidad de empleados temporales: 2
===== Fin KATA 3 =====
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. Animales y comportamiento sobrescrito

En esta actividad se implementa una jerarquía de clases que modela distintos tipos de animales, aplicando el concepto de **polimorfismo de comportamiento**. Se define una clase base **Animal**, que contiene el atributo común **nombre** y el método **hacerSonido()**, el cual representa el comportamiento general de un animal.

A partir de esta clase se extienden las subclases **Perro**, **Gato** y **Vaca**, cada una sobrescribiendo el método **hacerSonido()** para representar su sonido característico. De esta forma, se demuestra cómo distintas clases pueden compartir una misma interfaz (el método) pero implementar comportamientos diferentes.

Finalmente, se crea una colección de animales y se recorre utilizando referencias del tipo **Animal**. Al invocar **hacerSonido()** sobre cada elemento, se ejecuta automáticamente la implementación correspondiente a la subclase real de cada objeto, evidenciando el funcionamiento del **polimorfismo** durante la ejecución.

A continuación se presentan las capturas del código desarrollado y de la ejecución del programa.

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Animal {

    protected String nombre;

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

    public void hacerSonido() {
        System.out.println(nombre + " hace un sonido genérico.");
    }

    public void mostrarInfo() {
        System.out.println("Animal: " + nombre);
    }
}
```

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Perro extends Animal {

    public Perro(String nombre) {
        super(nombre);
    }

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

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Gato extends Animal {

    public Gato(String nombre) {
        super(nombre);
    }

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

```java
public class Vaca extends Animal {

    public Vaca(String nombre) {
        super(nombre);
    }

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

```java
package Tp7_HerenciaPolimorfismo;

/**...4 lines */
public class Kata4Main {

    public static void main(String[] args) {

        System.out.println("===== KATA 4 - Animales y Polimorfismo =====");

        Animal a1 = new Perro("Firulais");
        Animal a2 = new Gato("Mishito");
        Animal a3 = new Vaca("Lola");

        Animal[] animales = { a1, a2, a3 };

        System.out.println("Los animales hacen sus sonidos:");

        for (Animal a : animales) {
            a.mostrarInfo();
            a.hacerSonido();  // polimorfismo: se ejecuta el método de la subclase real
            System.out.println("--------------------------------");
        }

        System.out.println("===== Fin KATA 4 =====");
    }
}
```

```
run:
===== KATA 4 - Animales y Polimorfismo =====
Los animales hacen sus sonidos:
Animal: Firulais
Firulais: ▯Guau guau!
--------------------------------
Animal: Mishito
Mishito: Miau...
--------------------------------
Animal: Lola
Lola: Muuu...
--------------------------------
===== Fin KATA 4 =====
BUILD SUCCESSFUL (total time: 0 seconds)
```

Link a Repositorio Github

https://github.com/MaquiMarinoni/TUPaD_P2-C22025.git