Programación II - TP7: Herencia y Polimorfismo

Alumno: *Romero, Abel Tomás (Comisión 5)*

Link del repo de GitHub:

https://github.com/Tomu98/UTN-TUPaD-P2-TPs/tree/main/07%20Herencia%20y%20Polimorfismo

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.
Métodos genéricos y finales	Uso de final para evitar sobrescritura y garantizar estructuras inmutables.

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.

Clase Vehículo:

```java
public class Vehiculo {

    // Atributos
    protected String marca;
    protected String modelo;

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

    // Método
    public void mostrarInfo() {
        System.out.println("Información del vehículo:");
        System.out.println("- Marca: " + marca);
        System.out.println("- Modelo: " + modelo);
    }

}
```

Clase Auto:

```java
public class Auto extends Vehiculo {

    // Atributo
    private int cantidadPuertas;

    // Constructor
    public Auto(int cantidadPuertas, String marca, String modelo) {
        super(marca, modelo);
        this.cantidadPuertas = cantidadPuertas;
    }

    // Método sobrescrito
    @Override
    public void mostrarInfo() {
        System.out.println("Información del Auto:");
        System.out.println("- Marca: " + marca);
        System.out.println("- Modelo: " + modelo);
        System.out.println("- Cantidad de Puertas: " + cantidadPuertas);
    }

}
```

Main:

```java
public class Main {

    public static void main(String[] args) {

        // Instancia de un Auto
        Auto auto1 = new Auto(5, "Mercedes Benz", "GLE");

        // Mostramos su información completa
        System.out.println("================================================");
        auto1.mostrarInfo();
        System.out.println("================================================");

    }

}
```

Output:

```
run:
================================================
Informaci□n del Auto:
- Marca: Mercedes Benz
- Modelo: GLE
- Cantidad de Puertas: 5
================================================
BUILD SUCCESSFUL (total time: 0 seconds)
```

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.

Clase Figura:

```java
public abstract class Figura {

    // Atributo
    protected String nombre;

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

    // Método abstracto
    public abstract double calcularArea();

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

}
```

Clase Circulo:

```java
public class Circulo extends Figura {

    // Atributo
    private double radio;

    // Constructor
    public Circulo(double radio, String nombre) {
        super(nombre);
        this.radio = radio;
    }

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

    // Métodos setter y getter
    public double getRadio() {
        return radio;
    }

    public void setRadio(double radio) {
        if (radio <= 0) {
            System.out.println("ERROR: El radio debe ser positivo.");
        }
        this.radio = radio;
    }

}
```

Clase Rectangulo:

```java
public class Rectangulo extends Figura {

    // Atributos
    private double base;
    private double altura;

    // Constructor
    public Rectangulo(double base, double altura, String nombre) {
        super(nombre);
        this.base = base;
        this.altura = altura;
    }

    // Método sobrescrito
    @Override
    public double calcularArea() {
        return base * altura;
    }

    // Métodos setter y getter
    public double getBase() {
        return base;
    }

    public void setBase(double base) {
        if (base <= 0) {
            System.out.println("ERROR: La base debe ser positiva.");
        }
        this.base = base;
    }

    public double getAltura() {
        return altura;
    }

    public void setAltura(double altura) {
        if (altura <= 0) {
            System.out.println("ERROR: La altura debe ser positiva.");
        }
        this.altura = altura;
    }

}
```

Main:

```java
public class Main {

    public static void main(String[] args) {

        // Creamos un Array de figuras
        ArrayList<Figura> figuras = new ArrayList<>();

        // Creamos figuras para el array
        Circulo c1 = new Circulo(3.0, "Circulo 1");
        Circulo c2 = new Circulo(8.2, "Circulo 2");
        Circulo c3 = new Circulo(12.3, "Circulo 3");
        Rectangulo r1 = new Rectangulo(5.6, 2.4, "Rectangulo 1");
        Rectangulo r2 = new Rectangulo(1.7, 5.1, "Rectangulo 2");
        Rectangulo r3 = new Rectangulo(3.4, 10.9, "Rectangulo 3");

        // Añadimos las figuras al array
        figuras.add(c1);
        figuras.add(c2);
        figuras.add(c3);
        figuras.add(r1);
        figuras.add(r2);
        figuras.add(r3);

        // Recorremos el array y mostramos el área de cada figura
        System.out.println("================================================");
        for (Figura f : figuras) {
            System.out.printf("El area del %s es: %.2f%n", f.getNombre(), f.calcularArea());
        }
        System.out.println("================================================");

    }

}
```

Output:

```
run:
================================================
El area del Circulo 1 es: 28,27
El area del Circulo 2 es: 211,24
El area del Circulo 3 es: 475,29
El area del Rectangulo 1 es: 13,44
El area del Rectangulo 2 es: 8,67
El area del Rectangulo 3 es: 37,06
================================================
BUILD SUCCESSFUL (total time: 0 seconds)
```

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

Clase Empleado:

```java
public abstract class Empleado {

    // Atributo
    private String nombre;

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

    // Método
    public double calcularSueldo(Empleado e) {
        if (e instanceof EmpleadoPlanta) {
            return 2000;
        }

        if (e instanceof EmpleadoTemporal) {
            return 900;
        }

        return 0;
    }

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

}
```

Clase EmpleadoPlanta:

```java
public class EmpleadoPlanta extends Empleado {

    // Constructor
    public EmpleadoPlanta(String nombre) {
        super(nombre);
    }

}
```

Clase EmpleadoTemporal:

```java
public class EmpleadoTemporal extends Empleado {

    // Constructor
    public EmpleadoTemporal(String nombre) {
        super(nombre);
    }

}
```

Main:

```java
public class Main {

    public static void main(String[] args) {

        // Creamos un array de empleados
        ArrayList<Empleado> empleados = new ArrayList<>();

        // Creamos empleados para el array
        EmpleadoPlanta empl = new EmpleadoPlanta("Fulano");
        EmpleadoPlanta emp2 = new EmpleadoPlanta("Mengano");
        EmpleadoPlanta emp3 = new EmpleadoPlanta("Sultana");
        EmpleadoTemporal emp4 = new EmpleadoTemporal("Pepe");
        EmpleadoTemporal emp5 = new EmpleadoTemporal("Pepa");

        // Añadimos los empleados al array
        empleados.add(empl);
        empleados.add(emp2);
        empleados.add(emp3);
        empleados.add(emp4);
        empleados.add(emp5);

        // Recorremos el array y mostramos el sueldo
        System.out.println("================================================");
        for (Empleado e : empleados) {
            System.out.println("Sueldo del empleado " + e.getNombre() + ": $" + e.calcularSueldo(e));
        }
        System.out.println("================================================");

    }
```

Output:

```
run:
================================================
Sueldo del empleado Fulano: $2000.0
Sueldo del empleado Mengano: $2000.0
Sueldo del empleado Sultana: $2000.0
Sueldo del empleado Pepe: $900.0
Sueldo del empleado Pepa: $900.0
================================================
BUILD SUCCESSFUL (total time: 0 seconds)
```

4. **Animales y comportamiento sobrescrito:**
    - **Clase:** Animal con método `hacerSonido()` y `describirAnimal()`.
    - **Subclases:** Perro, Gato, Vaca sobrescriben `hacerSonido()` con `@Override`.
    - **Tarea:** Crear lista de animales y mostrar sus sonidos con polimorfismo.

Clase Animal:

```java
public class Animal {

    // Metodos
    public void hacerSonido() {
        System.out.println("El animal generico hace ruido...");
    }

    public void describirAnimal() {
        System.out.println("El animal generico es...");
    }

}
```

Clase Gato:

```java
public class Gato extends Animal {

    // Método sobrescrito
    @Override
    public void hacerSonido() {
        System.out.println("MIAU");
    }
```

Clase Perro:

```java
public class Perro extends Animal {

    // Método sobrescrito
    @Override
    public void hacerSonido() {
        System.out.println("GUAF");
    }
```

Clase Vaca:

```java
public class Vaca extends Animal {

    // Método sobrescrito
    @Override
    public void hacerSonido() {
        System.out.println("MUUUU");
    }
```

Main:

```java
public class Main {

    public static void main(String[] args) {

        // Creamos un array de animales
        ArrayList<Animal> animales = new ArrayList<>();

        // Creamos animales para el array
        Gato gato1 = new Gato();
        Gato gato2 = new Gato();
        Perro perro1 = new Perro();
        Perro perro2 = new Perro();
        Vaca vaca1 = new Vaca();
        Vaca vaca2 = new Vaca();

        // Añadimos los animales al array
        animales.add(gato1);
        animales.add(gato2);
        animales.add(perro1);
        animales.add(perro2);
        animales.add(vaca1);
        animales.add(vaca2);

        // Recorremos el array y mostramos el sonido que hacen
        System.out.println("================================================");
        for (Animal a : animales) {
            System.out.print("Este animal hace ");
            a.hacerSonido();
        }
        System.out.println("================================================");
```

Output:

```
run:
================================================
Este animal hace MIAU
Este animal hace MIAU
Este animal hace GUAF
Este animal hace GUAF
Este animal hace MUUUU
Este animal hace MUUUU
================================================
BUILD SUCCESSFUL (total time: 0 seconds)
```

5. **Sistema de pagos con polimorfismo y genéricos:**
    - **Interfaz:** Pagable con método `pagar()`.
    - **Clases:** `TarjetaCredito`, `Transferencia`, `Efectivo` implementan `Pagable`.
    - **Método:** `procesarPago(Pagable medio)` genérico para todos los tipos.
    - **Tarea:** Crear distintas formas de pago y procesarlas con una sola función.

Interfaz Pagable:

```java
public interface Pagable {

    // Método
    void pagar();

}
```

Clase TarjetaCredito:

```java
public class TarjetaCredito implements Pagable {

    // Método sobrescrito
    @Override
    public void pagar() {
        System.out.println("Se realizo el pago con tarjeta de credito.");
    }

}
```

Clase Transferencia:

```java
public class Transferencia implements Pagable {

    // Método sobrescrito
    @Override
    public void pagar() {
        System.out.println("Se realizo el pago con transferencia.");
    }

}
```

Clase Efectivo:

```java
public class Efectivo implements Pagable {

    // Método sobrescrito
    @Override
    public void pagar() {
        System.out.println("Se realizo el pago con efectivo.");
    }

}
```

Main:

```java
public class Main {

    public static void main(String[] args) {

        // Creamos un array de formas de pago
        ArrayList<Pagable> formasDePago = new ArrayList<>();

        // Creamos formas de pago para el array
        TarjetaCredito tarjeta1 = new TarjetaCredito();
        Transferencia transferencia1 = new Transferencia();
        Efectivo efectivo1 = new Efectivo();

        // Añadimos las formas de pago al array
        formasDePago.add(tarjeta1);
        formasDePago.add(transferencia1);
        formasDePago.add(efectivo1);

        // Recorremos el array y mostramos el proceso de pago
        System.out.println("================================================");
        for (Pagable p : formasDePago) {
            procesarPago(p);
        }
        System.out.println("================================================");

    }

    public static void procesarPago(Pagable medio) {
        medio.pagar();
    }

}
```

Output:

```
run:
================================================
Se realizo el pago con tarjeta de credito.
Se realizo el pago con transferencia.
Se realizo el pago con efectivo.
================================================
BUILD SUCCESSFUL (total time: 0 seconds)
```