PROGRAMACIÓN II

Trabajo Práctico 7: Herencia y Polimorfismo en Java

Alumno: Victor Barroeta

DNI: 95805903

Comisión: M2025 – 2

Matricula: 100393

Link de Github: https://github.com/victormbar/UTN_Prog2_VictorBarroeta/tree/main/javaUTN_TPs/src/TPs_JAVA_TP7

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

- 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.

```java
package Vehiculos;

public class Vehiculo {

    protected String marca, modelo;

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

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

```java
package Vehiculos;


public class Auto extends Vehiculo{

    private int cantidadPuertas;

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

    @Override
    public void mostrarInfo(){
        System.out.println("Marca: "+ this.marca + "\nModelo: "+ this.modelo +
                "\nCantidad de puertas: "+ cantidadPuertas);
    }
}
```

```java
package Vehiculos;


public class Main {

    public static void main(String[] args) {
        Auto a = new Auto("Fiat", "2021", 2);

        a.mostrarInfo();
    }

}
```

```
run:
Marca: Fiat
Modelo: 2021
Cantidad de puertas: 2
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.

```java
public class Main {

    public static void main(String[] args) {
        ArrayList<Figura> listaFiguras = new ArrayList<>();

        Circulo c1 = new Circulo("Circulo #1",4);
        Circulo c2 = new Circulo("Circulo #4",8);
        Rectangulo r1 = new Rectangulo("Rectangulo #1",3,5)
        Rectangulo r2 = new Rectangulo("Rectangulo #2",2,8)

        listaFiguras.add(c1);
        listaFiguras.add(c2);
        listaFiguras.add(r1);
        listaFiguras.add(r2);

        for (Figura f : listaFiguras){
            f.calcularArea();
        }

    }
}
```

```java
*/
package FigurasGeometricas;

public abstract class Figura {
    protected String nombre;

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

    public void calcularArea(){};

}
```

```java
package FigurasGeometricas;

import static java.lang.Math.PI;

public class Circulo extends Figura {
    private double radio;
    public Circulo(String nombre, double radio) {
        super(nombre);
        this.radio = radio;
    }

    @Override
    public void calcularArea() {
        System.out.println(nombre.toUpperCase() +
                "\n Area: "+ (radio * PI));
    }
}
```

```java
package FigurasGeometricas;

public class Rectangulo extends Figura{
    private double base, altura;

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

    @Override
    public void calcularArea() {
        System.out.println(nombre.toUpperCase() +
                "\n Area: "+ (base * altura));
    }

}
```

```
run:
CIRCULO #1
 Area: 12.566370614359172
CIRCULO #4
 Area: 25.132741228718345
RECTANGULO #1
 Area: 15.0
RECTANGULO #2
 Area: 16.0
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

```java
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {
        ArrayList<Empleado> listaEmpleados = new ArrayList<>();

        Empleado e1 = new EmpleadoTemporal();
        Empleado e2 = new EmpleadoFijo();
        Empleado e3 = new EmpleadoTemporal();
        Empleado e4 = new EmpleadoFijo();

        listaEmpleados.add(e1);
        listaEmpleados.add(e2);
        listaEmpleados.add(e3);
        listaEmpleados.add(e4);

        for (int i = 0; i < listaEmpleados.size(); i++) {
            Empleado e = listaEmpleados.get(i);
            System.out.println("Empleado: "+ (i+1) +
                    " - Sueldo: "+ e.calcularSueldo(e));
        }
```

```java
package Empleados;

public abstract class Empleado {
    public double calcularSueldo(Empleado e){
        if (e instanceof EmpleadoFijo){
            return 1500000.0;
        } else if (e instanceof EmpleadoTemporal){
            return 1000000.0;
        } else {
            return 0;
        }
    }
}
```

```java
package Empleados;

public class EmpleadoFijo extends Empleado{

}
```

```java
public class EmpleadoTemporal extends Empleado {

}
```

```
Empleado: 1 - Sueldo: 1000000.0
Empleado: 2 - Sueldo: 1500000.0
Empleado: 3 - Sueldo: 1000000.0
Empleado: 4 - Sueldo: 1500000.0
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

```java
import java.util.ArrayList;

public class Main {

    public static void main(String[] args) {
        ArrayList<Animal> listAnimales = new ArrayList<>();

        Animal a1 = new Gato();
        Animal a2 = new Perro();
        Animal a3 = new Vaca();

        listAnimales.add(a1);
        listAnimales.add(a2);
        listAnimales.add(a3);

        for (Animal a : listAnimales){
            a.describirAnimal();
            a.hacerSonido();
            System.out.println("----");
        }

    }
```

```java
package Animales;


public class Animal {
    public void hacerSonido(){};
    public void describirAnimal(){};

}
```

```java
package Animales;


public class Vaca extends Animal{

    @Override
    public void describirAnimal() {
        System.out.println("Soy una Vaca");
    }

    @Override
    public void hacerSonido() {
        System.out.println("MUUUUUUUUU");
    }
}
```

```java
package Animales;


public class Perro extends Animal {

    @Override
    public void describirAnimal() {
        System.out.println("Soy un Perro");
    }

    @Override
    public void hacerSonido() {
        System.out.println("GUAUUU");
    }
}
```

```java
package Animales;

public class Gato extends Animal{

    @Override
    public void describirAnimal() {
        System.out.println("Soy un Gato");
    }

    @Override
    public void hacerSonido() {
        System.out.println("MIAUUU");
    }

}
```

```
run:
Soy un Gato
MIAUUU
----
Soy un Perro
GUAUUU
----
Soy una Vaca
MUUUUUUUUU
----
```

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.