UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico 7

Herencia y Polimorfismo

Programación II

Alumno

Alex Pedro Dauria

Fecha de Entrega

20 de Octubre de 2025

Índice

Link al Repositorio en GitHub

https://github.com/Alex-Dauria/UTN-TUPaD-P2/tree/main/07%20HERENCIA%20Y%20POLIMORFISMO

Kata 1: Vehículos y herencia básica

1) Creé la clase base Vehiculo con atributos marca y modelo (protected para acceso en subclases), un constructor y el método mostrarInfo() que imprime la info básica.

```java
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);
    }
}
```

2) Creé la subclase Auto que extiende Vehiculo usando extends. Agregué atributo cantidadPuertas, constructor que llama super(marca, modelo) y sobrescribí mostrarInfo() para agregar las puertas.

```java
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() {
        super.mostrarInfo();
        System.out.println("Cantidad de puertas: " + cantidadPuertas);
    }
}
```

3) En Main1, instancié un Auto y llamé mostrarInfo() para mostrar todo.

```java
public class Main1 {
    public static void main(String[] args) {
        Auto auto = new Auto("Toyota", "Corolla", 4);
        auto.mostrarInfo();
    }
}
```

Salida del Main

```
run:
Marca: Toyota, Modelo: Corolla
Cantidad de puertas: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kata 2: Figuras geométricas y métodos abstractos

1) Creé la clase abstracta Figura con atributo nombre (protected), constructor, método abstracto calcularArea() (sin implementación) y agregué mostrarArea() para imprimir nombre y área.

```java
public abstract class Figura {
    protected String nombre;

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

    public abstract double calcularArea();

    public String getNombre() {
        return nombre;
    }

    public void mostrarArea() {
        System.out.println(getNombre() + ": " + calcularArea());
    }
}
```

2) Creé subclases Circulo y Rectangulo que extienden Figura. Cada una implementa calcularArea() (para círculo: π * radio²; para rectángulo: ancho * alto).

```java
public class Circulo extends Figura {
    private double radio;

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

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

```java
public class Rectangulo extends Figura {
    private double ancho;
    private double alto;

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

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

3) En Main2, creé un ArrayList de Figura (polimorfismo), agregué un círculo y un rectángulo, y en un loop llamé mostrarArea() para cada uno.

```java
import java.util.ArrayList;

public class Main2 {
    public static void main(String[] args) {
        ArrayList<Figura> figuras = new ArrayList<>();
        figuras.add(new Circulo("Circulo 1", 5));
        figuras.add(new Rectangulo("Rectangulo 1", 4, 6));

        for (Figura figura : figuras) {
            figura.mostrarArea();
        }
    }
}
```

Salida del Main

```
run:
Circulo 1: 78.53981633974483
Rectangulo 1: 24.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kata 3: Empleados y polimorfismo

1) Creé la clase abstracta Empleado con atributo nombre (protected), constructor y método abstracto calcularSueldo().

```java
public abstract class Empleado {
    protected String nombre;

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

    public abstract double calcularSueldo();

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

2) Creé subclases EmpleadoPlanta (sueldo fijo) y EmpleadoTemporal (horas * pago por hora), cada una implementa calcularSueldo().

```java
public class EmpleadoPlanta extends Empleado {
    private double sueldoFijo;

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

    @Override
    public double calcularSueldo() {
        return sueldoFijo;
    }
}
```

```java
public class EmpleadoTemporal extends Empleado {
    private double horasTrabajadas;
    private double pagoPorHora;

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

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

3) En Main3, creé un ArrayList de Empleado, agregué uno de cada, calculé sueldos polimórficamente y usé instanceof para clasificar el tipo.

```java
import java.util.ArrayList;

public class Main3 {
    public static void main(String[] args) {
        ArrayList<Empleado> empleados = new ArrayList<>();
        empleados.add(new EmpleadoPlanta("Juan", 3000));
        empleados.add(new EmpleadoTemporal("Ana", 40, 20));

        for (Empleado empleado : empleados) {
            System.out.println("Sueldo de " + empleado.getNombre() + ": " + empleado.calcularSueldo());
            if (empleado instanceof EmpleadoPlanta) {
                System.out.println("Tipo: Empleado de Planta");
            } else if (empleado instanceof EmpleadoTemporal) {
                System.out.println("Tipo: Empleado Temporal");
            }
        }
    }
}
```

Salida del Main

```
run:
Sueldo de Juan: 3000.0
Tipo: Empleado de Planta
Sueldo de Ana: 800.0
Tipo: Empleado Temporal
BUILD SUCCESSFUL (total time: 0 seconds)
```

Kata 4: Animales y comportamiento sobrescrito

1) Creé la clase Animal con atributo nombre (protected), constructor, método hacerSonido() (genérico) y describirAnimal() que imprime tipo + nombre y llama a hacerSonido().

```java
public class Animal {
    protected String nombre;

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

    public void hacerSonido() {
        System.out.println("Sonido genérico de animal");
    }

    public void describirAnimal() {
        System.out.println("El " + this.getClass().getSimpleName() + " " + nombre + " hace:");
        hacerSonido();
    }
}
```

2) Creé subclases Perro, Gato y Vaca que extienden Animal y sobrescriben hacerSonido() con @Override (Guau, Miau, Muu).

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

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

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

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

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

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

3) En Main4, creé un ArrayList de Animal, agregué uno de cada con nombres específicos (Rex, Misi, Lola) y llamé describirAnimal() para mostrar polimórficamente.

```java
import java.util.ArrayList;

public class Main4 {
    public static void main(String[] args) {
        ArrayList<Animal> animales = new ArrayList<>();
        animales.add(new Perro("Rex"));
        animales.add(new Gato("Misi"));
        animales.add(new Vaca("Lola"));

        for (Animal animal : animales) {
            animal.describirAnimal();
        }
    }
}
```

Salida del Main

```
run:
El Perro Rex hace:
Guau!
El Gato Misi hace:
Miau!
El Vaca Lola hace:
Muu!
BUILD SUCCESSFUL (total time: 0 seconds)
```