TRABAJO PRACTICO N7 – HERENCIA Y POLIMORFISMO

EMILIANO FABIAN JARA

PROGRAMACION 2

ACTIVIDADES:

1) Vehículos y herencia básica

Diseño:

- Vehiculo (base) con marca, modelo y mostrarInfo().
- Auto (subclase) añade cantidadPuertas y **sobrescribe** mostrarInfo() usando super.
  Esto aplica **herencia**, **modificadores de acceso**, y **super en constructor**.

CODIGO

```
// kata1/Vehiculo.java
public class Vehiculo {
  private String marca;
  private String modelo;

  public Vehiculo(String marca, String modelo) { this.marca = marca; this.modelo = modelo; }
  public String mostrarInfo() { return "Vehículo: " + marca + " " + modelo; }
}

// kata1/Auto.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 String mostrarInfo() {
        return super.mostrarInfo() + " | Puertas: " + cantidadPuertas;
    }
}

// kata1/Main.java
public class Main {
    public static void main(String[] args) {
        Auto auto = new Auto("Toyota", "Corolla", 4);
        System.out.println(auto.mostrarInfo());
    }
}
```

2) Figuras geométricas y métodos abstractos

Diseño:

- Figura **abstracta** con calcularArea() y nombre.
- Subclases: Circulo y Rectangulo **implementan** calcularArea().
- Se crea un **array de Figura** y se llama polimórficamente a calcularArea() (late binding).

CODIGO

```
// kata2/Figura.java
public abstract class Figura {
   protected String nombre;
   public Figura(String nombre) { this.nombre = nombre; }
   public abstract double calcularArea();
}

// kata2/Circulo.java
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; }
   @Override public String toString() { return nombre + "(r=" + radio + ")"; }
}

// kata2/Rectangulo.java
public class Rectangulo extends Figura {
   private double ancho, alto;
   public Rectangulo(double ancho, double alto) { super("Rectángulo"); this.ancho
= ancho; this.alto = alto; }
   @Override public double calcularArea() { return ancho * alto; }
   @Override public String toString() { return nombre + "(" + ancho + "x" + alto + ")"; }
}

// kata2/Main.java
public class Main {
   public static void main(String[] args) {
      Figura[] figuras = { new Circulo(3.0), new Rectangulo(4.0, 2.5), new Circulo(1.5)
};
```

```
    for (Figura f : figuras) {
        System.out.printf("%s -> área: %.2f%n", f.toString(), f.calcularArea());
    }
}
```

3) Empleados y polimorfismo (con instanceof)

Diseño:

- Empleado **abstracto** con calcularSueldo().
- EmpleadoPlanta (sueldo base + porcentaje por antigüedad).
- EmpleadoTemporal (horas * valor hora).
- Lista polimórfica de Empleado, uso de instanceof para **clasificar** antes de imprimir.
  Además se muestra **upcasting** (al guardar subclases en la lista de Empleado) y el potencial **downcasting** (no es necesario en este caso).

  CODIGO

```
// kata3/Empleado.java
public abstract class Empleado {
    protected String nombre;
    public Empleado(String nombre) { this.nombre = nombre; }
    public String getNombre() { return nombre; }
    public abstract double calcularSueldo();
}

// kata3/EmpleadoPlanta.java
public class EmpleadoPlanta extends Empleado {
    private double sueldoBase, antiguedadPorcentaje;
    public EmpleadoPlanta(String n, double base, double porc) {
  super(n);sueldoBase = base; antiguedadPorcentaje = porc; }
 @Override public double calcularSueldo() { return sueldoBase * (1 +
antiguedadPorcentaje); }
}

// kata3/EmpleadoTemporal.java
public class EmpleadoTemporal extends Empleado {
    private int horasTrabajadas; private double valorHora;
    public EmpleadoTemporal(String n, int horas, double valor) { super(n);
  horasTrabajadas = horas; valorHora = valor; }
    @Override public double calcularSueldo() { return horasTrabajadas *
  valorHora; }
}
```

```
// kata3/Main.java
import java.util.*;
public class Main {
   public static void main(String[] args) {
      List<Empleado> empleados = new ArrayList<>();
      empleados.add(new EmpleadoPlanta("Ana", 450000, 0.15));
      empleados.add(new EmpleadoTemporal("Luis", 120, 2000));
      empleados.add(new EmpleadoPlanta("Marta", 520000, 0.05));
     empleados.add(new EmpleadoTemporal("José", 80, 1800));

      for (Empleado e : empleados) {
         double sueldo = e.calcularSueldo(); // polimorfismo
         String tipo = (e instanceof EmpleadoPlanta) ? "Planta" :
                  (e instanceof EmpleadoTemporal) ? "Temporal"
:"Desconocido";
         System.out.printf("%s (%s) -> sueldo: $%,.2f%n", e.getNombre(),
tipo,sueldo);
      }
   }
}
```

4) Animales y comportamiento sobrescrito

Diseño:

- Animal con hacerSonido() y describirAnimal().
- Perro, Gato, Vaca **sobrescriben** (@Override) y se demuestra el **polimorfismo** al iterar una lista de Animal.

```
// kata4/Animal.java
public class Animal {
  public String describirAnimal() { return "Soy un animal genérico"; }
  public String hacerSonido() { return "(silencio)"; }
}
```

```
// kata4/Perro.java, Gato.java, Vaca.java
public class Perro extends Animal { @Override public String hacerSonido() { return
"Guau"; } @Override public String describirAnimal(){return "Soy un perro";} }
```

```
public class Gato  extends Animal { @Override public String hacerSonido() { return "Miau"; } @Override public String describirAnimal(){return "Soy un gato";} }

public class Vaca  extends Animal { @Override public String hacerSonido() { return "Muuu"; } @Override public String describirAnimal(){return "Soy una vaca";} }


// kata4/Main.java

import java.util.*;

public class Main {

  public static void main(String[] args) {

    List<Animal> animales = Arrays.asList(new Perro(), new Gato(), new Vaca(), new Animal());

    for (Animal a : animales) {

      System.out.println(a.describirAnimal() + " -> sonido: " + a.hacerSonido());

    }

  }

}
```

Conceptos del marco teórico aplicados

- **Herencia extends y principio *is-a*** (Auto es un Vehículo; Perro/Gato/Vaca son Animales).
- **Modificadores de acceso** (private, protected) y **encapsulación** (getters).
- **Constructores y super** para inicialización de atributos heredados.
- **Clases y métodos abstractos** para definir contratos (Figura.calcularArea(), Empleado.calcularSueldo()).
- **Polimorfismo** por **sobrescritura** (@Override) y **llamada dinámica** en colecciones/arrays del tipo de la superclase.
- **Upcasting** (guardar subclases en referencias de superclase) y **instanceof/downcasting** (cuando hace falta especializar).