PROGRAMACIÓN II

TRABAJO PRÁCTICO 7: HERENCIA Y POLIMORFISMO EN JAVA

RESUMEN

Aplicación de los conceptos de herencia y polimorfismo en la Programación Orientada a Objetos: reutilización de código, creación de jerarquías de clases y diseño flexible de soluciones en Java.

Nicolás Olima

nicolima200@gmail.com

- REPO GITHUB

PROGRAMACIÓN II

Trabajo Práctico 7: Herencia y Polimorfismo en Java

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
public class Vehiculo {
    private String marca;
    private String modelo;

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

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

    public String getMarca() {
        return marca;
    }

    public String getModelo() {
        return modelo;
    }
}
```

```java
public class Auto extends Vehiculo{
    private int cantidadPuertas;

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

    @Override
    public void mostrarInfo(){
        System.out.println("Marca: "+super.getMarca()+" - "+"Modelo: "+
                super.getModelo()+" - "+"Cant. puertas: "+cantidadPuertas);
    }
}
```

```java
public static void main(String[] args) {
    ////////////// 1. Vehículos y herencia básica
    Auto auto= new Auto(5, "Fiat", "Diavlo");

    auto.mostrarInfo();
```

SALIDA POR CONSOLA:

```
run:
Marca: Fiat - Modelo: Diavlo - Cant. puertas: 5
```

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 abstract class Figura {
    private String nombre;

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

    public String getNombre() {
        return nombre;
    }

    public abstract double calcularArea();
}
```

```java
public class Rectangulo extends Figura {
    private double ladoA;
    private double ladoB;

    public Rectangulo(String nombre,double ladoA, double ladoB) {
        super(nombre);
        this.ladoA = ladoA;
        this.ladoB = ladoB;
    }

    @Override
    public double calcularArea() {
        return ladoA*ladoB;
    }
}
```

```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*Math.pow(radio, 2);
    }
}
```

```java
public static void main(String[] args) {

    ////////////// 2. Figuras geométricas y métodos abstractos
    ArrayList<Figura> figuras= new ArrayList<>();
    figuras.add(new Circulo("circulo 1", 8));
    figuras.add(new Circulo("circulo 2", 5.4));
    figuras.add(new Rectangulo("rectangulo 1", 5.4,6.8));
    figuras.add(new Circulo("circulo 3", 81));
    figuras.add(new Rectangulo("rectangulo 2", 10, 2.15));
    figuras.add(new Circulo("circulo 4", 3.5));
```

SALIDA POR CONSOLA:

```
run:
Area circulo 1: 201.06192982974676
Area circulo 2: 91.60884177867838
Area rectangulo 1: 36.72
Area circulo 3: 20611.989400202634
Area rectangulo 2: 21.5
Area circulo 4: 38.48451000647496
```

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
public abstract class Empleado {
    public double calcularSueldo(Empleado e){
        if (e instanceof EmpleadoPlanta){
            return 800;
        }else if(e instanceof EmpleadoTemporal){
            return 500;
        }else{
            return 0;
        }
    }
}
```

```java
public class EmpleadoPlanta extends Empleado {
}
```

```java
public class EmpleadoTemporal extends Empleado{
}
```

```java
public static void main(String[] args) {
    /////////////// 3. Empleados y polimorfismo
    ArrayList<Empleado> empleados = new ArrayList<>();

    EmpleadoPlanta emp1 = new EmpleadoPlanta();
    EmpleadoPlanta emp2 = new EmpleadoPlanta();
    EmpleadoPlanta emp3 = new EmpleadoPlanta();
    EmpleadoTemporal emp4 = new EmpleadoTemporal();
    EmpleadoTemporal emp5 = new EmpleadoTemporal();

    empleados.add(emp1);
    empleados.add(emp2);
    empleados.add(emp3);
    empleados.add(emp4);
    empleados.add(emp5);

    int i=0;
    for (Empleado e : empleados) {
        System.out.println("El "+e.getClass().getSimpleName()+" nro "+i+
                " tiene un sueldo de "+e.calcularSueldo(e));
        i++;
    }
```

SALIDA POR CONSOLA:

```
run:
El EmpleadoPlanta nro 0 tiene un sueldo de 800.0
El EmpleadoPlanta nro 1 tiene un sueldo de 800.0
El EmpleadoPlanta nro 2 tiene un sueldo de 800.0
El EmpleadoTemporal nro 3 tiene un sueldo de 500.0
El EmpleadoTemporal nro 4 tiene un sueldo de 500.0
```

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
public class Animal {
    public void hacerSonido(){
        System.out.println("¡Sonido de animal!");
    }

    public void describirAnimal(){
    }
}
```

```java
public class Perro extends Animal{
    @Override
    public void hacerSonido(){
        System.out.println("¡GUAU GUAU!");
    }
}
```

```java
public class Gato extends Animal{
    @Override
    public void hacerSonido(){
        System.out.println("¡MIAU!");
    }
}
```

```java
public class Vaca extends Animal{
    @Override
    public void hacerSonido(){
        System.out.println("¡MUUUUUU!");
    }
}
```

```java
public static void main(String[] args) {
    /////////////// 4. Animales y comportamiento sobrescrito
    ArrayList<Animal> animales = new ArrayList<>();

    Perro p1 = new Perro();
    Gato g1 = new Gato();
    Vaca v1 = new Vaca();

    animales.add(p1);
    animales.add(g1);
    animales.add(v1);

    for (Animal a : animales) {
        a.hacerSonido();
    }
```

SALIDA POR CONSOLA:

```
run:
GUAU GUAU!
MIAU!
MUUUUUU!
```

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