Práctica 7 - Herencia en Java

Osiris Gabriel Hernández Callado
Correo: gabrielhernadezescanor@gmail.com
Repositorio GitHub: https://github.com/Gabo21092005/POOP7.git

Joshua Salvador Vargas Arreaga
Correo: joshy76familia@gmail.com

***Resumen*—El objetivo de esta práctica es implementar los conceptos de herencia en el lenguaje de programación Java, generando una jerarquía de clases, aplicando el principio de reutilización de código y demostrando el comportamiento de los objetos mediante instanciación y pruebas en un método principal (`main`). Adicionalmente, se aprovechan herramientas del entorno NetBeans (*Insert Code / Alt + Insert*) para acelerar la generación de código repetitivo como constructores, getters, setters y `toString()`.**

I. Objetivo

Implementar los conceptos de herencia en un lenguaje de programación orientado a objetos mediante la creación de una estructura jerárquica de clases y su comprobación mediante instanciación y ejecución de métodos.

II. Actividades Realizadas

- Creación de la jerarquía de clases con herencia: clase base y subclases concretas.
- Encapsulamiento con atributos privados y generación de getters/setters.
- Implementación de `toString()` para una salida legible en consola.
- Organización por paquetes (p. ej., `animales` y `poop7`).
- Desarrollo de un `main` que instancia objetos y prueba métodos heredados y propios.
- Uso de *Insert Code (Alt + Insert)* en NetBeans para agilizar la escritura de código.

III. Desarrollo

Se partió de una clase base con atributos encapsulados y se derivaron subclases que amplían el comportamiento mediante atributos y métodos propios. Se sobrescribió `toString()` en las clases para presentar el estado de los objetos de forma clara durante las pruebas en consola.

Para reducir tiempos y errores humanos en la escritura de código repetitivo, se empleó la función *Insert Code (Alt + Insert)* de NetBeans, que permite generar automáticamente constructores, getters, setters y el propio `toString()`. Esto favorece la estandarización del código y la productividad durante la práctica.

La jerarquía se organizó en paquetes: las clases del dominio en `animales` y la clase con `main` en `poop7`. En el `main` se crearon instancias de las clases concretas y se invocaron métodos heredados (demostrando herencia) y métodos específicos (demostrando especialización). Además, se utilizó un método `sonido(String)` para mostrar polimorfismo simple en tiempo de ejecución.

IV. Código Fuente (Partes Clave)

IV-A. *Clase base con encapsulación y `toString()`*

```
// Paquete: animales
public class Animal {
    private String nombre;
    private String lugarOrigen;
    private String color;

    public Animal() {}

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

    public String getNombre() { return nombre; }
    public void setNombre(String n) { this.nombre =
        n; }

    public String getLugarOrigen() { return
        lugarOrigen; }
    public void setLugarOrigen(String l) { this.
        lugarOrigen = l; }

    public String getColor() { return color; }
    public void setColor(String c) { this.color = c;
         }

    public void sonido(String sonido){
        System.out.println(nombre + " hace: " +
            sonido);
    }

    @Override
    public String toString() {
        return "Animal{nombre=" + nombre
               + ", lugarOrigen=" + lugarOrigen
               + ", color=" + color + "}";
    }
}
```

IV-B. *Subclase concreta generada con Insert Code*

```
// Paquete: animales
public class Perro extends AnimalTerrestre {
    private String colorCollar;

    public Perro() { super(); }

    // Constructor y getters/setters generados con
        Alt+Insert
    public Perro(String colorCollar, int numeroPatas
        ,
                 String nombre, String lugarOrigen,
                     String color) {
```

```
        super(numeroPatas, nombre, lugarOrigen,
            color);
        this.colorCollar = colorCollar;
    }

    public String getColorCollar() { return
        colorCollar; }
    public void setColorCollar(String cc) { this.
        colorCollar = cc; }

    public void hacerTrucos() {
        System.out.println(getNombre() + " est
            haciendo trucos.");
    }

    @Override
    public String toString() {
        return super.toString() + " | Perro{
            colorCollar=" + colorCollar + "}";
    }
}
```

IV-C. `main` de prueba (instanciación y métodos)

```
// Paquete: poop7
import animales.*;

public class POOP7 {
    public static void main(String[] args) {
        Perro perro = new Perro("Negro", 4, "Junior"
            , "Alemania", "Caf ");
        System.out.println(perro);
        perro.sonido("Guau guau");
        perro.hacerTrucos();
    }
}
```

V. Conclusión

Se implementó una jerarquía de clases en Java utilizando herencia y encapsulación, reforzada con la sobrescritura de `toString()` para facilitar la inspección de objetos. El uso de *Insert Code (Alt + Insert)* en NetBeans permitió generar rápidamente constructores, getters, setters y `toString()`, mejorando la productividad y la consistencia del código. Las pruebas en el `main` evidenciaron la correcta interacción entre clases base y subclases, así como la extensión de comportamiento en las clases concretas.