	Carátula para entrega de prácticas
Facultad de Ingeniería	Laboratorios de docencia

Laboratorio de Computación Salas A y B

Profesor(a): Ayala Barbosa José Antonio

Asignatura: Programación Orientada a Objetos

Grupo: 02

No de Práctica(s): Práctica 7: Herencia

Integrante(s): Leguizamón Ríos Braulio Favio

Luna Sánchez Edgar Miguel

Sánchez Meza Mario Dirak

No. de lista o brigada:

Semestre: 2026 – I

Fecha de entrega: 16 de octubre del año 2025

Observaciones:

CALIFICACIÓN: _____

Práctica 07 - Herencia en Java.

Previo:

Con base en la clase teórica se obtuvieron clases:

a. Empleado

```
package recursoshumanos;

/**
 * Clase base que representa a un Empleado
 * Contiene los atributos y métodos comunes para todos los empleados
 */
public class Empleado {
    private String nombre;
    private int numEmp;
    private double sueldo;

    // Constructor por defecto
    public Empleado() {
    }

    // Constructor con parámetros
    public Empleado(String nombre, int numEmp, double sueldo) {
        this.nombre = nombre;
        this.numEmp = numEmp;
        this.sueldo = sueldo;
    }

    // Getters y Setters
    public String getNombre() {
        return nombre;
    }

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

    public int getNumEmp() {
        return numEmp;
    }

    public void setNumEmp(int numEmp) {
        this.numEmp = numEmp;
    }

    public double getSueldo() {
        return sueldo;
    }

    public void setSueldo(double sueldo) {
        this.sueldo = sueldo;
```

```
    }

    /**
     * Incrementa el sueldo del empleado en un porcentaje dado
     * @param porcentaje Porcentaje de incremento (ej: 10 para 10%)
     */
    public void incrementarSueldo(int porcentaje) {
        this.sueldo += (this.sueldo * porcentaje) / 100;
    }

    /**
     * Método para mostrar información del empleado
     * @return String con la información del empleado
     */
    public String obtenerInformacion() {
        return "Empleado #" + numEmp + ": " + nombre + " - Sueldo: $" + sueldo;
    }
}
```

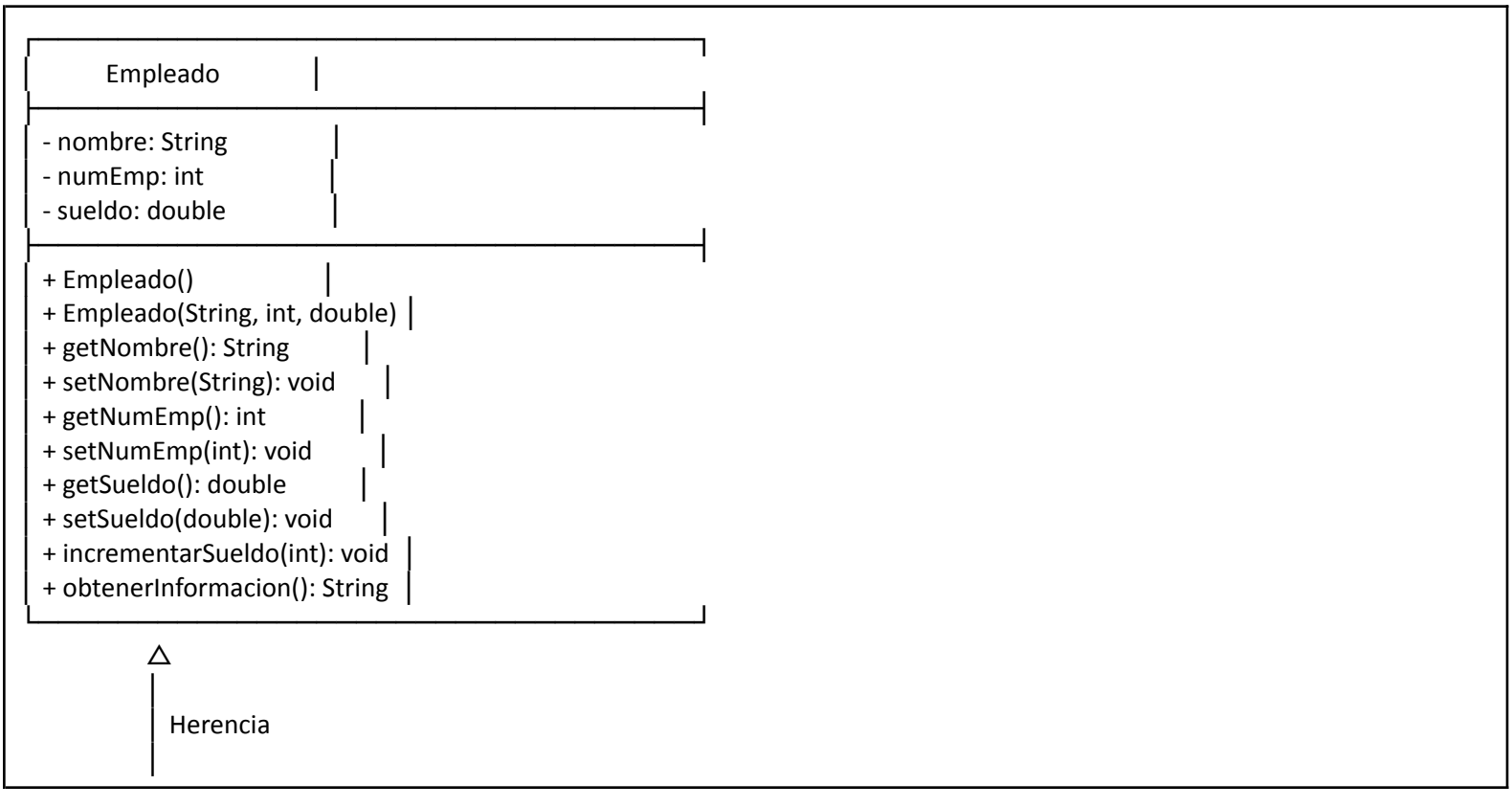
b. Gerente

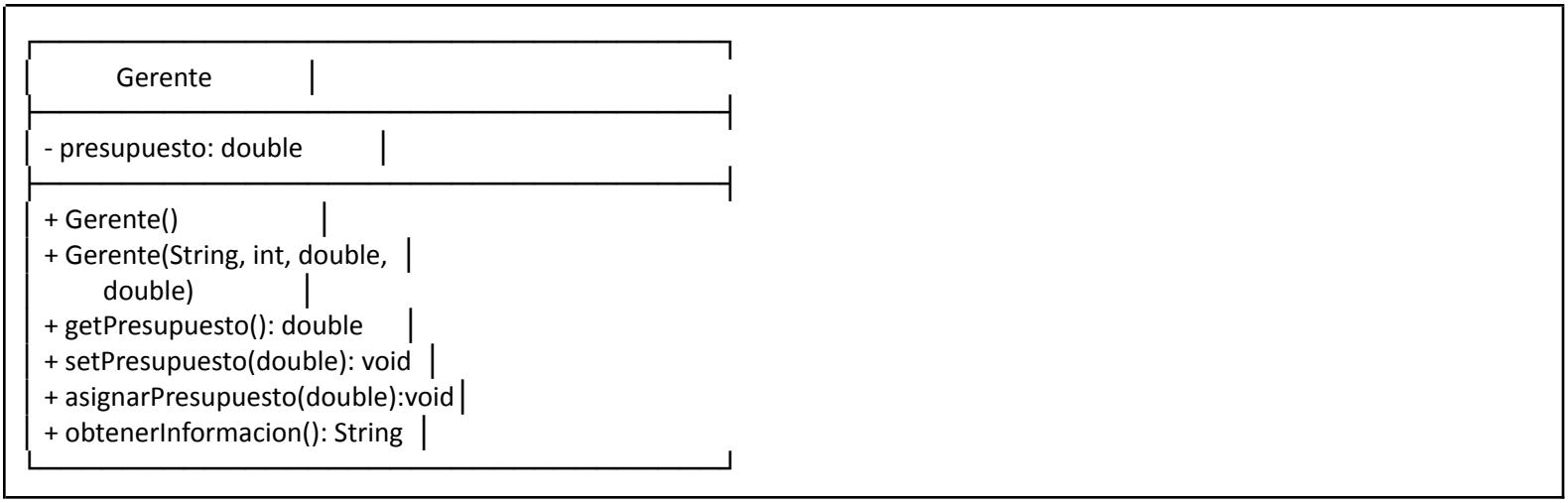
```
package recursoshumanos;

/**
 * Clase Gerente que hereda de Empleado
 * Representa un empleado con responsabilidades de gestión y presupuesto
 */
public class Gerente extends Empleado {
    private double presupuesto;

    // Constructor por defecto
    public Gerente() {
        super(); // Llama al constructor de la clase padre
    }

    // Constructor con parámetros
    public Gerente(String nombre, int numEmp, double sueldo, double presupuesto) {
        super(nombre, numEmp, sueldo); // Llama al constructor del padre
        this.presupuesto = presupuesto;
    }

    // Getters y Setters
    public double getPresupuesto() {
        return presupuesto;
    }

    public void setPresupuesto(double presupuesto) {
        this.presupuesto = presupuesto;
```

```
    }

    /**
     * Asigna un nuevo presupuesto al gerente
     * @param presupuesto Nuevo monto del presupuesto
     */
    public void asignarPresupuesto(double presupuesto) {
        this.presupuesto = presupuesto;
    }

    /**
     * Método sobreescrito para mostrar información del gerente
     * Incluye información del empleado más el presupuesto
     * @return String con la información completa del gerente
     */
    @Override
    public String obtenerInformacion() {
        return "Gerente #" + getNumEmp() + ": " + getNombre() +
               " - Sueldo: $" + getSueldo() +
               " - Presupuesto: $" + presupuesto;
    }
}
```

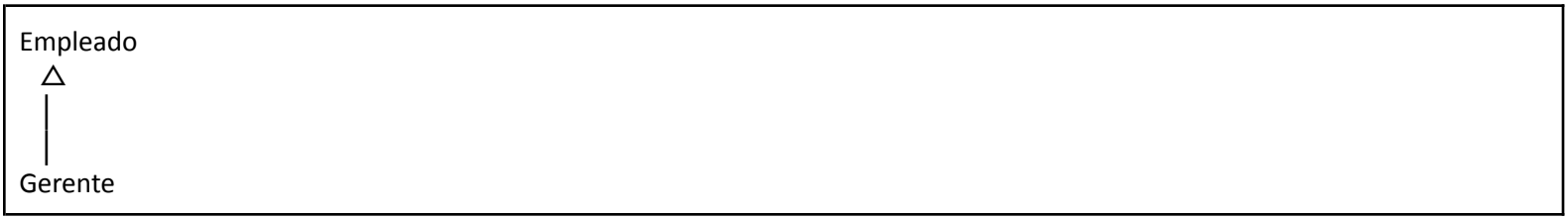
De las cuales podemos obtener los siguientes diagramas requeridos para el previo de la práctica;

1. Diagrama de clases creadas en teoría:

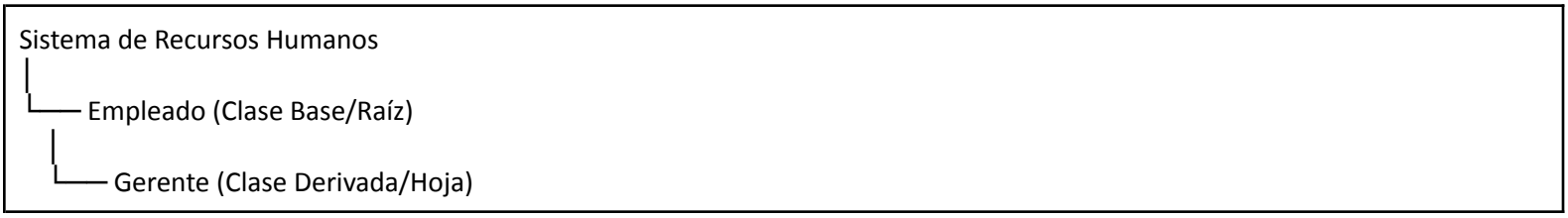




2. Diagrama normalizado en árbol de jerarquías:



O también:



Así, obtenemos la siguiente información respecto a las clases y los diagramas:

a. Características de la Jerarquía
   - Clase Empleado (Padre/Superclase)

· Atributos base: nombre, numEmp, sueldo

· Métodos comunes: acceso, cálculo de sueldo

· Rol: Clase generalizada

   - Clase Gerente (Hija/Subclase)

· Hereda de: Empleado

· Atributos adicionales: presupuesto

· Métodos adicionales: gestión de presupuesto

· Sobrescribe: obtenerInformacion()

· Rol: Clase especializada

b. Explicación de los Diagramas

I. Diagrama de Clases:

· Muestra estructura completa de cada clase

· Incluye atributos, métodos y modificadores de acceso

· Detalla parámetros de constructores y métodos

· Muestra relaciones entre clases (herencia)

II. Diagrama de Jerarquías:

· Enfoque en la relación de herencia

· Estructura árbol genealógico de clases

· Simple y claro para entender la jerarquía

· Normalizado = organizado de forma estándar

Ventajas de esta Jerarquía

Herencia: Empleado ← Gerente

├── Reutilización de código

├── Extensibilidad

├── Polimorfismo

└── Mantenibilidad

Esta estructura permite que Gerente herede toda la funcionalidad de Empleado mientras añade características específicas de su rol, demostrando perfectamente el principio de herencia en POO.

Práctica 07: Herencia.

I. Introducción

La herencia es uno de los pilares fundamentales de la Programación Orientada a Objetos (POO) que permite crear nuevas clases basadas en clases existentes, facilitando la reutilización de código y estableciendo relaciones jerárquicas entre clases. En esta práctica se implementa el concepto de herencia mediante un sistema de gestión de animales organizado en una jerarquía que demuestra cómo las clases especializadas heredan atributos y comportamientos de clases más generales.

El objetivo principal es comprender cómo la herencia permite extender funcionalidades, sobrescribir métodos y crear estructuras organizadas que representen relaciones del mundo real, optimizando el desarrollo mediante la reutilización de código y el polimorfismo.

II. Objetivo

Implementar los conceptos de herencia en un lenguaje de programación orientado a objetos.

III. Actividades

Realizar clase Círculo con atributos privados, métodos y constructores.

Realizar clase Principal (POOP5) para crear objetos y mostrar resultados de área y perímetro.

Realizar clase Coche como ejemplo extra de abstracción y encapsulamiento aplicado a un contexto cotidiano.

IV. Desarrollo de Actividades

1. Análisis del Diagrama de Clases

Se diseñó una jerarquía de herencia con Animal como clase base, de la cual derivan tres subclases principales:

AnimalAcuatico

AnimalAereo

AnimalTerrestre

Posteriormente, se crearon clases más especializadas:

Ballena (hereda de AnimalAcuatico)

Pajaro (hereda de AnimalAereo)

Perro (hereda de AnimalTerrestre)

2. Implementación de la Herencia

Cada clase hija utiliza la palabra clave extends para heredar de su clase padre, demostrando los diferentes niveles de especialización:

Reutilización de código: Las clases hijas heredan automáticamente los atributos y métodos de las clases padres.

Extensibilidad: Cada clase hija añade nuevos atributos y comportamientos específicos.

Sobrescritura de métodos: Se redefine el método comer() en cada clase para demostrar comportamientos específicos.

Constructores con super(): Se utiliza para inicializar correctamente los atributos heredados.

3. Demostración de Funcionalidad

En la clase principal se instanciaron objetos de diferentes niveles de la jerarquía, demostrando:

Acceso a métodos heredados.

Comportamientos específicos de cada clase.

Polimorfismo mediante la sobrescritura de métodos.

La relación "ES-UN" entre clases.

Composición del Archivo y Códigos Fuente;

Estructura del Proyecto:

```
poop7/
├── POOP7.java (Clase principal)
├── Empleado.java
├── Gerente.java
└── animales/
    ├── Animal.java
    ├── AnimalAcuatico.java
    ├── AnimalAereo.java
    ├── AnimalTerrestre.java
    ├── Ballena.java
    ├── Pajaro.java
    └── Perro.java
```

V. Código fuente

Código *POOP7.java*

```
package poop7;
import animales.Ballena;

/**
 * @author Reid_
 */
public class POOP7 {
    public static void main(String[] args) {

        Empleado emp1 = new Empleado("Juan",18,30000);
        System.out.println(emp1.getNombre());
        System.out.println(emp1.toString());

        Gerente emp2 = new Gerente(200000,"Luis",20,40000);
        System.out.println(emp2.toString());

        Ballena ballena1 = new Ballena(20, 2, "moby", "atlantico", "gris");
        ballena1.pelearconPinocho();
        System.out.println(ballena1.toString());
        ballena1.sonido();
    }
}
```

Código *Animal.java*

```
package animales;

/**
 * @author Reid_
 */
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;
    }

    // Getters y Setters
    public String getNombre() { return nombre; }
    public void setNombre(String nombre) { this.nombre = nombre; }
    public String getLugarOrigen() { return lugarOrigen; }
    public void setLugarOrigen(String lugarOrigen) { this.lugarOrigen =
lugarOrigen; }
    public String getColor() { return color; }
    public void setColor(String color) { this.color = color; }

    public void sonido(){
        System.out.println("auuuuu");
    }

    public void comer(){

        System.out.println("comiendo");
    }

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

CÓDIGO *AnimalAcuatico.java*

```
package animales;

/**
 * @author Reid_
 */
public class AnimalAcuatico extends Animal{
    private int numeroAletas;

    public AnimalAcuatico() {
    }

    public AnimalAcuatico(int numeroAletas, String nombre, String
lugarOrigen, String color) {
        super(nombre, lugarOrigen, color);
        this.numeroAletas = numeroAletas;
    }

    public int getNumeroAletas() { return numeroAletas; }
    public void setNumeroAletas(int numeroAletas) { this.numeroAletas =
numeroAletas; }

    public void nadar(){
        System.out.println("animal nadando");
    }

    @Override
    public void comer(){
        System.out.println("comiendo peces");
    }

    @Override
    public String toString() {
        return super.toString() + " AnimalAcuatico{" + "numeroAletas=" +
numeroAletas + '}';
    }
}
```

CÓDIGO *AnimalAereo.java*

```
package animales;
```

```
/**
 * @author Reid_
 */
public class AnimalAereo extends Animal {
    private int numAlas;

    public AnimalAereo() {
    }

    public AnimalAereo(int numAlas, String nombre, String lugarOrigen,
String color) {
        super(nombre, lugarOrigen, color);
        this.numAlas = numAlas;
    }

    public int getNumAlas() { return numAlas; }
    public void setNumAlas(int numAlas) { this.numAlas = numAlas; }

    public void volar(){
        System.out.println("volandoo");
    }

    @Override
    public void comer(){
        System.out.println("wakala");
    }

    @Override
    public String toString() {
        return super.toString() + " AnimalAereo{" + "numAlas=" + numAlas +
'}';
    }
}
```

CÓDIGO *AnimalTerrestre.java*

```
package animales;

/**
 * @author Reid_
 */
public class AnimalTerrestre extends Animal {
    private int numPatas;

    public AnimalTerrestre() {
    }

    public AnimalTerrestre(int numPatas, String nombre, String lugarOrigen,
String color) {
        super(nombre, lugarOrigen, color);
        this.numPatas = numPatas;
    }

    public int getNumPatas() { return numPatas; }
    public void setNumPatas(int numPatas) { this.numPatas = numPatas; }

    public void correr(){
        System.out.println("fium fium");
    }

    @Override
    public void comer(){
        System.out.println("comiendo pasto");
    }

    @Override
    public String toString() {
        return super.toString() + " AnimalTerrestre{" + "numPatas=" +
numPatas + '}';
    }
}
```

CÓDIGO *Ballena.java*

```
package animales;
```

```
/**
 * @author Reid_
 */
public class Ballena extends AnimalAcuatico {
    private int largo;

    public Ballena() {
    }

    public Ballena(int largo, int numeroAletas, String nombre, String
lugarOrigen, String color) {
        super(numeroAletas, nombre, lugarOrigen, color);
        this.largo = largo;
    }

    public int getLargo() { return largo; }
    public void setLargo(int largo) { this.largo = largo; }

    public void pelearconPinocho(){
        System.out.println("muere pinocho ");
    }

    @Override
    public String toString() {
        return super.toString() + " Ballena{" + "largo=" + largo + '}';
    }
}
```

CÓDIGO *Pajaro.java*

```
package animales;

/**
 * @author Reid_
 */
public class Pajaro extends AnimalAereo {
    private String tipoPico;

    public Pajaro() {
    }

    public Pajaro(String tipoPico, int numAlas, String nombre, String
lugarOrigen, String color) {
        super(numAlas, nombre, lugarOrigen, color);
        this.tipoPico = tipoPico;
    }

    @Override
    public String toString() {
        return super.toString() + " Pajaro{" + "tipoPico=" + tipoPico + '}';
    }
}
```

CÓDIGO *Perro.java*

```
package animales;

/**
 * @author Reid_
 */
public class Perro extends AnimalTerrestre{
    private String colorCollar;

    public Perro() {
    }

    public Perro(String colorCollar, int numPatas, String nombre, String
lugarOrigen, String color) {
        super(numPatas, nombre, lugarOrigen, color);
        this.colorCollar = colorCollar;
    }

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

    public void hacerTrucos(){
        System.out.println("miren mi truco");
    }

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

CÓDIGO *Empleado.java*

```
package poop7;

/**
 *
 * @author Reid_
 */
public class Empleado {
    private String nombre;
    private int numEmpleado;
    private double sueldo;

    //Override
    public Empleado() {
    }

    public Empleado(String nombre, int numEmpleado, double sueldo) {
        this.nombre = nombre;
        this.numEmpleado = numEmpleado;
        this.sueldo = sueldo;
    }

    public String getNombre() {
        return nombre;
    }

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

    public int getNumEmpleado() {
        return numEmpleado;
    }

    public void setNumEmpleado(int numEmpleado) {
        this.numEmpleado = numEmpleado;
    }

    public double getSueldo() {
        return sueldo;
    }

    public void setSueldo(double sueldo) {
        this.sueldo = sueldo;
    }

    public void aumentarSueldo(int porcentaje){
        // sueldo = sueldo + sueldo*porcentaje/100
        sueldo+= sueldo*porcentaje/100;

    }

    //Sobreescritura
    @Override
    public String toString(){
        return "nombre=" +nombre+ " numEmpleado=" +numEmpleado+ "
sueldo=" +sueldo+'}';
    }
}
```

CÓDIGO *Gerente.java*

```
package poop7;

/**
 *
```

```
 * @author Reid_
 */
public class Gerente extends Empleado{
    private double presupuesto;

    public Gerente() {
    }

    public Gerente(double presupuesto, String nombre, int numEmpleado,
double sueldo) {
        super(nombre, numEmpleado, sueldo);
        this.presupuesto = presupuesto;
    }

    public double getPresupuesto() {
        return presupuesto;
    }

    public void setPresupuesto(double presupuesto) {
        this.presupuesto = presupuesto;
    }

    //Azúcar Sintáctica
    public void asignarPresupuesto(double presupuesto){
        setPresupuesto(presupuesto);
    }

    @Override
    public String toString() {
        return super.toString()+"Gerente{" + "presupuesto=" + presupuesto + '}';
    }

}
```

VI. URL

GitHub: https://github.com/EdLuna237/POOP07
GitHub Pages: https://edluna237.github.io/POOP07/

VII. Conclusiones

A través del desarrollo de esta práctica sobre herencia en Java, se logró comprender y aplicar de manera efectiva los principios fundamentales de la Programación Orientada a Objetos. La implementación del sistema de gestión de animales permitió evidenciar las ventajas significativas que ofrece la herencia en el desarrollo de software.

En primer lugar, se pudo constatar cómo la herencia facilita la reutilización de código mediante la creación de una jerarquía bien estructurada. La clase Animal funcionó como una base sólida que encapsuló los atributos y comportamientos comunes, mientras que las subclases especializadas como AnimalAcuatico, AnimalAereo y AnimalTerrestre extendieron estas funcionalidades sin necesidad de duplicar código. Este enfoque demostró ser eficiente y mantenible.

En segundo término, la práctica permitió experimentar con el polimorfismo a través de la sobrescritura de métodos. El método comer(), redefinido en cada subclase, mostró cómo diferentes tipos de animales pueden responder de manera específica al mismo mensaje, reflejando comportamientos particulares según su naturaleza. Este concepto resultó fundamental para diseñar sistemas flexibles y extensibles.

Adicionalmente, se comprobó la importancia del uso correcto de constructores con super() para garantizar la inicialización adecuada de los atributos heredados. La cadena de construcción desde las clases más especializadas hasta la clase base aseguró que todos los objetos se crearan en un estado consistente y válido.

En cuanto a la organización del proyecto, la estructura de paquetes implementada demostró ser una estrategia efectiva para mantener un código ordenado y modular. La separación clara entre las diferentes categorías de clases mejoró la legibilidad y facilitó el mantenimiento del sistema.

Finalmente, esta práctica consolidó la comprensión de que la herencia no solo es un mecanismo técnico, sino una herramienta conceptual que permite modelar relaciones del mundo real en el software. La relación "ES-UN" entre clases, como en el caso de "Ballena ES-UN AnimalAcuatico", proporcionó una forma natural e intuitiva de representar dominios complejos.

En conclusión, el dominio de la herencia resulta indispensable para cualquier desarrollador que busque crear sistemas robustos, escalables y mantenibles, constituyendo una competencia fundamental en la formación de un ingeniero de software.