DIAGRAMA UML: SISTEMAS DE USUARIO, ZOOLOGICO

Autor:

JAIRO YEISON SANCHEZ LOPEZ

Docente:

CHRISTIAN DAVID JAIMES ACEVEDO

TECNOLOGICO DE ANTIOQUIA, INSTITUCION UNIVERSITARIA

FACULTAD DE INGENIERIA

IGENIERIA DE SOFTWARE

PARADIGMAS ORIENTADOS A OBJETOS

09 de Noviembre de 2025

Tabla de contenido

Tabla de contenido

INTRODUCCIÓN

El desarrollo de los dos trabajos presentados a continuación tiene como finalidad fortalecer las competencias en **Programación Orientada a Objetos (POO)** mediante la aplicación práctica de sus principales conceptos: **herencia, encapsulamiento, polimorfismo y abstracción**.
Cada proyecto aborda un contexto distinto, permitiendo observar cómo los mismos principios pueden aplicarse en escenarios con finalidades diferentes.

El **primer trabajo**, titulado *Sistema de Usuarios*, se enfoca en la creación de una estructura jerárquica que representa distintos tipos de usuarios dentro de una institución educativa. A través de clases como Administrador, Profesor y Estudiante, se modelan roles con características y funciones particulares, demostrando cómo la herencia permite compartir atributos comunes sin perder la individualidad de cada entidad. Este proyecto enfatiza la **organización, modularidad y reutilización del código**, simulando un entorno administrativo digital.

Por otro lado, el **segundo trabajo**, denominado *Sistema del Zoológico*, traslada los mismos fundamentos teóricos a un contexto biológico, donde se modelan especies animales y sus relaciones jerárquicas. En este caso, la clase base Animal da origen a subclases como Mamífero y Reptil, que a su vez derivan en especies específicas como León, Elefante, Cocodrilo e Iguana. Este enfoque permite observar la aplicación del **polimorfismo y la herencia múltiple jerárquica**, representando de manera realista las características y comportamientos de cada especie dentro de un entorno natural simulado.

En conjunto, ambos proyectos demuestran cómo la POO puede adaptarse a distintos tipos de sistemas —uno de carácter administrativo y otro de carácter biológico—, resaltando la **versatilidad y potencia del paradigma orientado a objetos** en el diseño de soluciones informáticas coherentes, escalables y organizadas.

SISTEMAS USUARIOS

Clase Base: Sistemas Usuarios

Ubicación: package sistemas.usuarios;

Descripción

La clase Sistemas Usuarios representa la estructura general que comparten todos los usuarios del sistema.

Define los atributos fundamentales para identificar a cualquier usuario: nombre de usuario, identificación y contraseña.

ATRIBUTOS

```
    private String usuario;
    private int id;
    private String password;
```

- usuario**:** almacena el nombre de usuario o alias del sistema.
- id**:** identifica de forma única a cada usuario.
- password**:** representa la clave de acceso personal.

CONSTRUCTOR:

Inicializa los valores de los atributos principales de la clase.
Cada subclase utilizará este constructor mediante la palabra clave super().

```
public SistemasUsuarios(String usuario, int id, String password) {
    this.usuario = usuario;
    this.id = id;
    this.password = password;
```

MÉTODOS GETTERS Y SETTERS

Permiten acceder y modificar los valores de los atributos de forma controlada:

```
public String getUsuario() { return usuario; }
public void setUsuario(String usuario) { this.usuario = usuario; }

public int getId() { return id; }
public void setId(int id) { this.id = id; }

public String getPassword() { return password; }

public void setPassword(String password) { this.password = password; }
```

PRINCIPIOS APLICADOS

- **Encapsulamiento:** los atributos son private y se manipulan mediante getters/setters.
- **Reutilización:** esta clase es la base para todas las demás, evitando repetir código.

CLASE DERIVADA: **ADMINISTRADOR**

DESCRIPCIÓN:

La clase Administrador extiende de Sistemas Usuarios e incorpora funciones de gestión propias del rol administrativo.
Representa a los usuarios encargados de gestionar permisos, usuarios y notas dentro del sistema.

ATRIBUTOS ADICIONALES

```java
package sistemas.usuarios;

public class Administrador extends SistemasUsuarios {
    private int horario_trabajo;
    private String modificar_notas;
    private String eliminar_notas;
    private String asignar_permisos;
    private String agregar_usuarios;
    private String eliminar_usuarios;
    private String ver_lista_usuarios;
```

Estos atributos simulan acciones o permisos que el administrador puede tener.

CONSTRUCTOR:

```java
public Administrador(String usuario, int id, String password, int horario_trabajo,
                     String modificar_notas, String eliminar_notas, String asignar_permisos,
                     String agregar_usuarios, String eliminar_usuarios, String ver_lista_usuarios) {
    super(usuario, id, password);
    this.horario_trabajo = horario_trabajo;
    this.modificar_notas = modificar_notas;
    this.eliminar_notas = eliminar_notas;
    this.asignar_permisos = asignar_permisos;
    this.agregar_usuarios = agregar_usuarios;
    this.eliminar_usuarios = eliminar_usuarios;
    this.ver_lista_usuarios = ver_lista_usuarios;
}
```

Inicializa los valores del administrador y utiliza `super()` para heredar los datos básicos del usuario.

PRINCIPIOS APLICADOS

- **Herencia:** extiende de `SistemasUsuarios`.
- **Especialización:** agrega comportamientos propios del administrador, distintos a los de profesor o estudiante.
- **Encapsulamiento:** cada atributo cuenta con sus respectivos métodos de acceso.

CLASE DERIVADA: PROFESOR

DESCRIPCIÓN

La clase Profesor representa al docente que interactúa con los estudiantes y gestiona sus notas o asignaturas.
Hereda los datos básicos de SistemasUsuarios y agrega funcionalidades específicas.

ATRIBUTOS

```java
package sistemas.usuarios;

public class Profesor extends SistemasUsuarios {
    private String asignar_materia;
    private String horario_clases;
    private String editar_notas;
    private String admin_notas;
    private String crear_notas;
    private String borrar_notas;
```

Estos campos modelan las acciones o permisos del profesor dentro del sistema educativo.

CONSTRUCTOR:

```java
public Profesor(String usuario, int id, String password,
                String asignar_materia, String horario_clases, String editar_notas,
                String admin_notas, String crear_notas, String borrar_notas) {
    super(usuario, id, password);
    this.asignar_materia = asignar_materia;
    this.horario_clases = horario_clases;
    this.editar_notas = editar_notas;
    this.admin_notas = admin_notas;
    this.crear_notas = crear_notas;
    this.borrar_notas = borrar_notas;
}
```

Permite crear instancias de profesores con todos los datos necesarios.

MÉTODOS

Incluye los getters y setters correspondientes para mantener un control adecuado sobre los datos.

```
public String getAsignar_materia() { return asignar_materia; }
public void setAsignar_materia(String asignar_materia) { this.asignar_materia = asignar_materia; }

public String getHorario_clases() { return horario_clases; }
public void setHorario_clases(String horario_clases) { this.horario_clases = horario_clases; }

public String getEditar_notas() { return editar_notas; }
public void setEditar_notas(String editar_notas) { this.editar_notas = editar_notas; }

public String getAdmin_notas() { return admin_notas; }
public void setAdmin_notas(String admin_notas) { this.admin_notas = admin_notas; }

public String getCrear_notas() { return crear_notas; }
public void setCrear_notas(String crear_notas) { this.crear_notas = crear_notas; }

public String getBorrar_notas() { return borrar_notas; }
public void setBorrar_notas(String borrar_notas) { this.borrar_notas = borrar_notas; }
```

PRINCIPIOS APLICADOS

- **Herencia:** recibe atributos y métodos comunes desde SistemasUsuarios.
- **Extensión funcional:** amplía la clase padre con nuevos comportamientos asociados al rol docente.

CLASE DERIVADA: ESTUDIANTE

DESCRIPCIÓN:

La clase Estudiante modela a los usuarios que representan a los alumnos del sistema. Su rol es más limitado, enfocado principalmente en el seguimiento académico y la entrega de trabajos.

ATRIBUTOS:

```
package sistemas.usuarios;

public class Estudiante extends SistemasUsuarios {
    private String horario_clases;
    private String grupo;
    private String subir_trabajos;
```

CONSTRUCTOR:

```
public Estudiante(String usuario, int id, String password,
                  String horario_clases, String grupo, String subir_trabajos) {
    super(usuario, id, password);
    this.horario_clases = horario_clases;
    this.grupo = grupo;
    this.subir_trabajos = subir_trabajos;
}
```

Asigna tanto los datos heredados como los nuevos campos propios del estudiante.

MÉTODOS

Los métodos getter y setter permiten acceder y modificar los atributos de manera controlada.

```
public String getHorario_clases() { return horario_clases; }

public void setHorario_clases(String horario_clases) { this.horario_clases = horario_clases; }

public String getGrupo() { return grupo; }
public void setGrupo(String grupo) { this.grupo = grupo; }

public String getSubir_trabajos() { return subir_trabajos; }
public void setSubir_trabajos(String subir_trabajos) { this.subir_trabajos = subir_trabajos; }
```

PRINCIPIOS APLICADOS

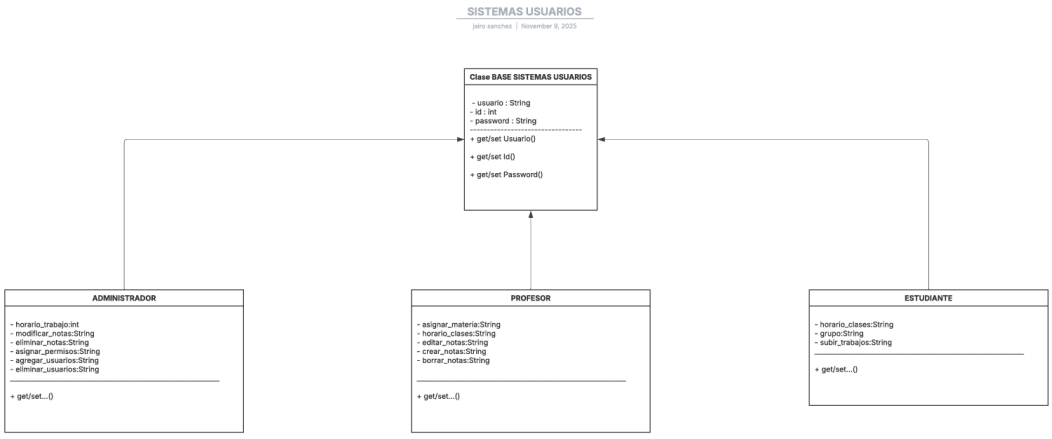
- **Herencia:** comparte los atributos comunes con las otras clases.
- **Especialización:** añade las particularidades del estudiante (grupo, horario, trabajos).

PRINCIPIOS DE POO APLICADOS

- **Herencia:**
  Todas las clases (Administrador, Profesor, Estudiante) derivan de la clase SistemasUsuarios.
  Esto evita la duplicación de código y mejora la organización jerárquica del sistema.
- **Encapsulamiento:**
  Los atributos son privados (private) y solo accesibles mediante métodos públicos (get y set), garantizando la seguridad de los datos.

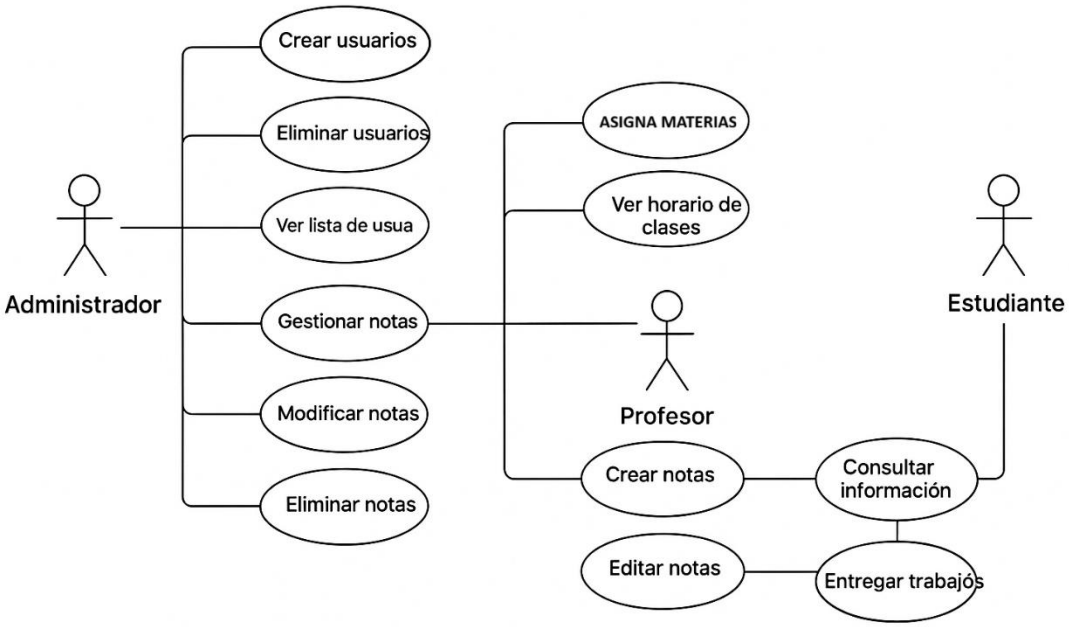
- **Abstracción:**
  Cada clase representa un concepto del mundo real (usuario, administrador, profesor, estudiante) con sus características relevantes.
- **Polimorfismo (potencial):**
  Si en el futuro se manejan objetos de tipo SistemasUsuarios, podrán comportarse como Administrador, Profesor o Estudiante según la instancia creada.

DIAGRAMA UML



- **SistemasUsuarios**
  Clase padre **que guarda los datos generales de cualquier usuario del sistema.**
- Administrador
  Puede gestionar usuarios y permisos **dentro del sistema.**
- Profesor
  Puede crear, editar o borrar notas**, además de** asignar materias**.**
- Estudiante
  Solo puede subir trabajos **y** consultar su horario y grupo**.**

CASO DE USO


Crear usuarios
Eliminar usuarios
Ver lista de usua
Gestionar notas
Modificar notas
Eliminar notas
Administrador
ASIGNA MATERIAS
Ver horario de clases
Profesor
Crear notas
Editar notas
Consultar información
Entregar trabajos
Estudiante

ZOOLOGICO

CLASES DE ANIMALES

ANIMAL (CLASE BASE)

- **Atributos:** nombre, habitat, edad.
- **Métodos:**
  - mostrarFicha(): Muestra los datos generales del animal.
  - actividadDiaria(): Acción diaria del animal (se redefine en subclases).
  - limpiarHabitat(): Limpieza del espacio del animal (también redefinido).

```java
package ZOOLOGICO;

public class Animal {
    protected String nombre;
    protected String habitat;
    protected int edad;

    public Animal(String nombre, String habitat, int edad) {
        this.nombre = nombre;
        this.habitat = habitat;
        this.edad = edad;
    }

    public void mostrarFicha() {
        System.out.println("Animal: " + nombre);
        System.out.println("Hábitat: " + habitat);
        System.out.println("Edad: " + edad + " años");
    }

    public void actividadDiaria() {
        System.out.println(nombre + " realiza su actividad diaria en el zoológico.");
    }

    public void limpiarHabitat() {
        System.out.println("El hábitat de " + nombre + " está siendo limpiado de forma general.");
    }
}
```

Esta clase sirve como plantilla para todas las demás especies y permite aplicar **herencia y polimorfismo**.

MAMÍFEROS Y REPTILES (SUBCLASES DE ANIMAL)

- **Mamifero:** Añade método cuidarCrias() y redefine actividadDiaria().
- **Reptil:** Añade método cambiarPiel() y redefine actividadDiaria().

```java
package ZOOLOGICO;

public class Mamifero extends Animal {

    public Mamifero(String nombre, String habitat, int edad) {
        super(nombre, habitat, edad);
    }

    @Override
    public void actividadDiaria() {
        System.out.println(nombre + " corre y socializa con otros mamíferos.");
    }

    public void cuidarCrias() {
        System.out.println(nombre + " está cuidando a sus crías.");
    }
}
```

```java
package ZOOLOGICO;

public class Reptil extends Animal {

    public Reptil(String nombre, String habitat, int edad) {
        super(nombre, habitat, edad);
    }

    @Override
    public void actividadDiaria() {
        System.out.println(nombre + " toma el sol para regular su temperatura corporal.");
    }

    public void cambiarPiel() {
        System.out.println(nombre + " está mudando su piel.");
    }
}
```

Estas clases especializadas muestran **comportamientos propios de cada tipo de animal** manteniendo la estructura común de Animal.

ANIMALES ESPECÍFICOS

- **León, Elefante, Tigre:** Heredan de Mamifero y personalizan su actividadDiaria() y limpiarHabitat().
- **Cocodrilo, Iguana, Serpiente:** Heredan de Reptil y personalizan sus métodos.

```java
package ZOOLOGICO;

public class Leon extends Mamifero {

    public Leon(String nombre, String habitat, int edad) {
        super(nombre, habitat, edad);
    }

    @Override
    public void actividadDiaria() {
        System.out.println(nombre + " patrulla su territorio y descansa al sol.");
    }

    @Override
    public void limpiarHabitat() {
        System.out.println("Se retiran los restos y huesos del hábitat del león " + nombre + ".");
    }
}
```

```java
package ZOOLOGICO;

public class Cocodrilo extends Reptil {

    public Cocodrilo(String nombre, String habitat, int edad) {
        super(nombre, habitat, edad);
    }

    @Override
    public void actividadDiaria() {
        System.out.println(nombre + " nada sigilosamente esperando a su presa.");
    }

    @Override
    public void limpiarHabitat() {
        System.out.println("Se drena y limpia el estanque del cocodrilo " + nombre + ".");
    }
}
```

Aquí se evidencia el **polimorfismo**, ya que cada animal realiza actividades diferentes usando los mismos métodos.

CLASES DE USUARIOS

USUARIO (ABSTRACTA)

- **Atributos:** nombre, idUsuario, rol.
- **Métodos:**
  - verAnimales(): Permite revisar la lista de animales.
  - registrarActividad(): Guarda la actividad realizada por un animal.
  - mostrarFunciones(): Método abstracto, implementado en subclases.

```java
package ZOOLOGICO;

public abstract class Usuario {
    protected String nombre;
    protected int idUsuario;
    protected String rol;

    public Usuario(String nombre, int idUsuario, String rol) {
        this.nombre = nombre;
        this.idUsuario = idUsuario;
        this.rol = rol;
    }

    public void verAnimales(Animal[] animales) {
        System.out.println(nombre + " esta revisando la lista de animales:");
        for (Animal a : animales) {
            System.out.println(" - " + a.nombre + " (" + a.getClass().getSimpleName() + ")");
        }
    }

    public void registrarActividad(Animal animal) {
        System.out.println(nombre + " registro la actividad diaria de " + animal.nombre + ".");
    }

    public abstract void mostrarFunciones();
}
```

Esta clase asegura **abstracción**, definiendo la estructura general de los usuarios.

ADMINISTRADOR Y TRABAJADOR

- **Administrador:** Agrega/elimina animales, ve reportes y supervisa.
- **Trabajador:** Alimenta animales, limpia habitats y registra actividades.

```java
package ZOOLOGICO;

public class Administrador extends Usuario {

    public Administrador(String nombre, int idUsuario) {
        super(nombre, idUsuario, "Administrador");
    }

    public void agregarAnimal(Animal animal) {
        System.out.println("El administrador " + nombre + " agrego al animal: " + animal.nombre);
    }

    public void eliminarAnimal(String nombreAnimal) {
        System.out.println("El administrador " + nombre + " elimino al animal: " + nombreAnimal);
    }

    public void verReportes() {
        System.out.println("El administrador " + nombre + " esta revisando los reportes de salud y alimentacion.");
    }

    @Override
    public void mostrarFunciones() {
        System.out.println("Funciones del administrador:");
        System.out.println(" - Agregar y eliminar animales");
        System.out.println(" - Ver reportes");
        System.out.println(" - Supervisar actividades del zoologico");
    }
}
```

```java
package ZOOLOGICO;

public class Trabajador extends Usuario {

    public Trabajador(String nombre, int idUsuario) {
        super(nombre, idUsuario, "Trabajador");
    }

    public void alimentarAnimal(Animal animal) {
        System.out.println(nombre + " alimento al animal " + animal.nombre + ".");
    }

    public void limpiarHabitat(Animal animal) {
        System.out.println(nombre + " limpio el habitat de " + animal.nombre + ".");
    }

    @Override
    public void mostrarFunciones() {
        System.out.println("Funciones del trabajador:");
        System.out.println(" - Alimentar animales");
        System.out.println(" - Limpiar habitats");
        System.out.println(" - Registrar actividades diarias");
    }
}
```

Estas subclases muestran **herencia y especialización** según el rol del usuario.

INTERACCIONES Y SIMULACIÓN (ZOOLOGICO.JAVA**)**

- Se crea un **arreglo de animales** y se inicializan usuarios.
- Los usuarios realizan acciones:
    - Revisar animales.
    - Alimentar y limpiar habitats.
    - Registrar actividades.
    - Agregar nuevos animales.

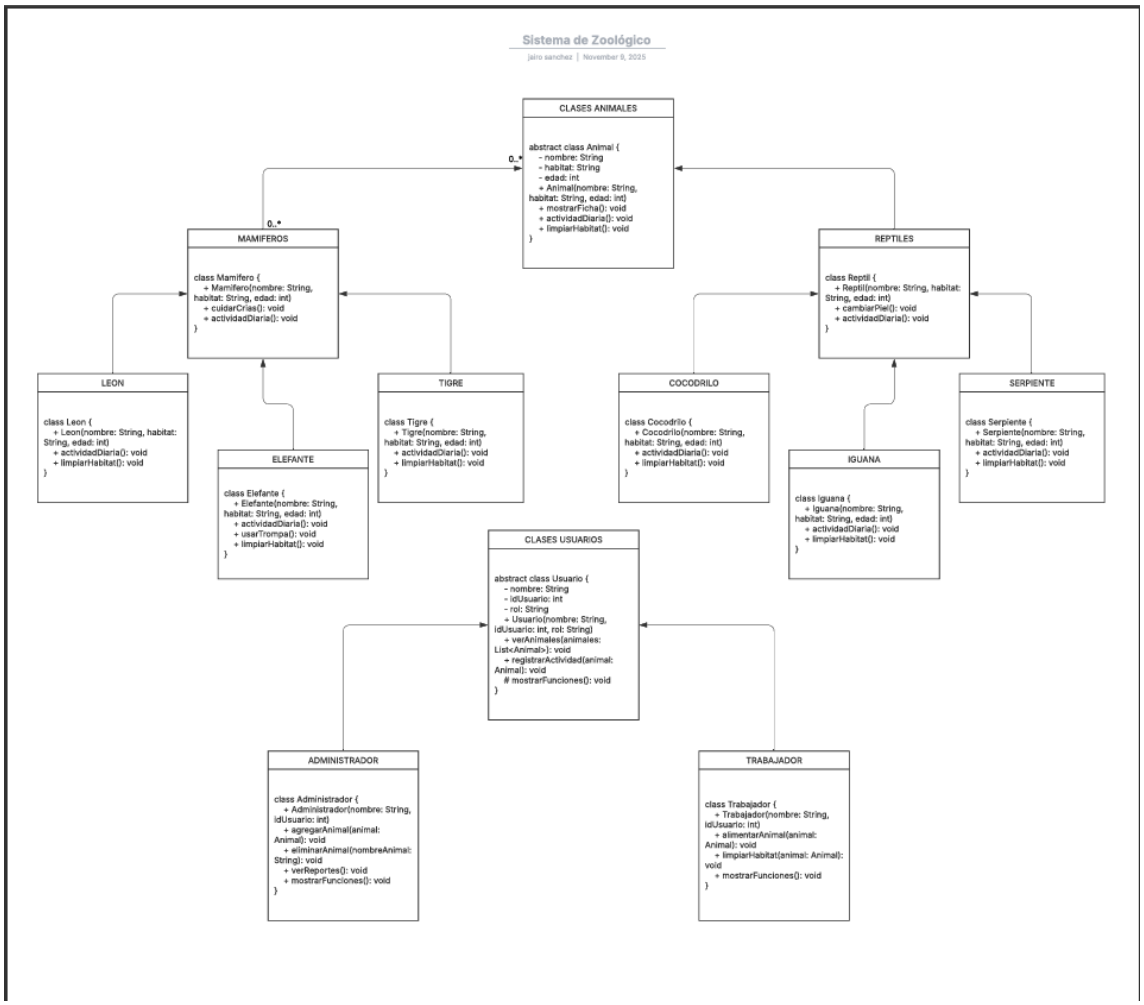
```java
public static void main(String[] args) {

    Animal[] animales = {
        new Leon("Simba", "Sabana Africana", 6),
        new Elefante("Dumbo", "Pradera húmeda", 10),
        new Tigre("Shere Khan", "Selva Asiática", 7),
        new Cocodrilo("Rex", "Pantano tropical", 12),
        new Iguana("Iggy", "Terrario de selva", 4),
        new Serpiente("Kaa", "Terrario desértico", 5)
    };


    Administrador admin = new Administrador("Laura", 1001);
    Trabajador cuidador = new Trabajador("Jairo", 2001);

    System.out.println("\n=== USUARIOS DEL SISTEMA ===");
    admin.mostrarFunciones();
    cuidador.mostrarFunciones();

    System.out.println("\n=== INTERACCION DE USUARIOS ===");
    admin.verAnimales(animales);
    cuidador.verAnimales(animales);

    admin.agregarAnimal(new Leon("Nala", "Sabana Africana", 4));
    admin.verReportes();

    cuidador.alimentarAnimal(animales[1]);
    cuidador.limpiarHabitat(animales[3]);

    admin.registrarActividad(animales[0]);
    cuidador.registrarActividad(animales[2]);
}
```

Esto permite ver **cómo todas las clases interactúan** en un escenario real, mostrando el flujo completo del sistema.

PRINCIPIOS DE PROGRAMACIÓN ORIENTADA A OBJETOS APLICADOS

- **Herencia:** Clases específicas de animales heredan de Animal, y usuarios heredan de Usuario.
- **Polimorfismo:** Métodos como actividadDiaria() y limpiarHabitat() se redefinen en cada subclase.
- **Abstracción:** Usuario obliga a implementar mostrarFunciones().
- **Encapsulamiento:** Atributos protegidos para controlar el acceso a la información.

DIAGRAMA UML



CASO DE USO

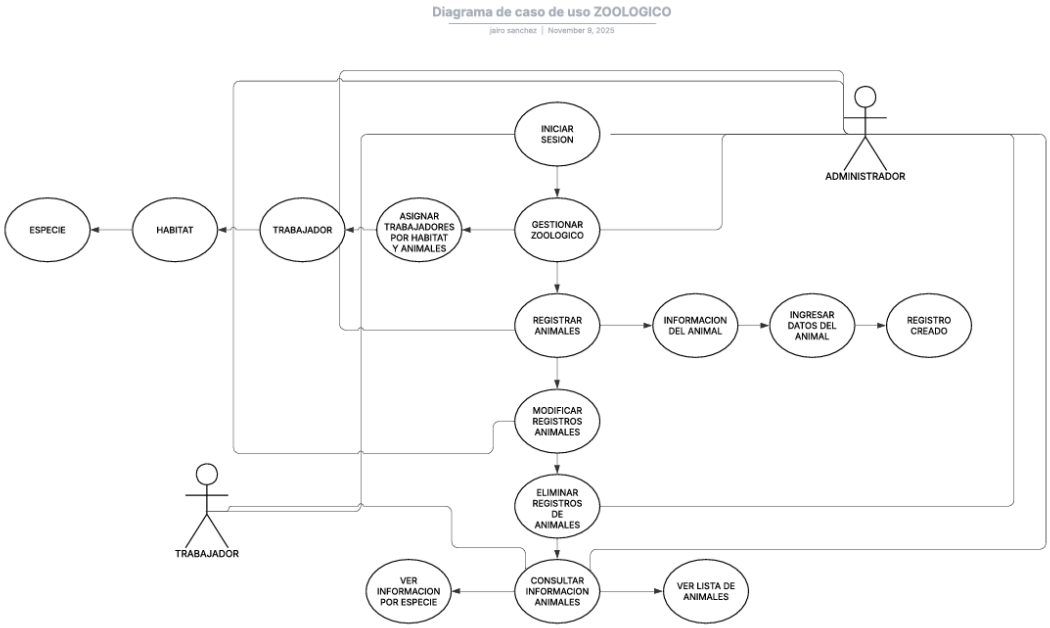


Diagrama de caso de uso ZOOLOGICO
jairo sanchez | November 9, 2025
INICIAR SESION
ADMINISTRADOR
ESPECIE
HABITAT
TRABAJADOR
ASIGNAR TRABAJADORES POR HABITAT Y ANIMALES
GESTIONAR ZOOLOGICO
REGISTRAR ANIMALES
INFORMACION DEL ANIMAL
INGRESAR DATOS DEL ANIMAL
REGISTRO CREADO
MODIFICAR REGISTROS ANIMALES
TRABAJADOR
ELIMINAR REGISTROS DE ANIMALES
VER INFORMACION POR ESPECIE
CONSULTAR INFORMACION ANIMALES
VER LISTA DE ANIMALES

CONCLUSIONES

La realización de ambos proyectos permitió evidenciar las **diferencias de aplicación** de la Programación Orientada a Objetos en contextos diversos, así como la **coherencia estructural** que ofrece este paradigma en cualquier tipo de sistema.

En el **Sistema de Usuarios**, se consolidó el uso de la herencia para modelar jerarquías humanas, destacando la importancia del **encapsulamiento y la seguridad de datos** en entornos administrativos. Este trabajo resalta el valor de la organización y la claridad en el manejo de información sensible, características esenciales en sistemas educativos o institucionales.

Por su parte, el **Sistema del Zoológico** permitió profundizar en la representación de entidades naturales y en la implementación del **polimorfismo**, al mostrar cómo diferentes clases hijas pueden redefinir comportamientos heredados. El enfoque biológico brindó una comprensión más visual y dinámica del funcionamiento de las jerarquías, reforzando la idea de que las clases pueden adaptarse a múltiples contextos sin perder coherencia lógica.

En conclusión, la comparación entre ambos proyectos evidencia que, aunque los entornos sean distintos —uno humano y otro animal—, los principios de la POO permanecen constantes. Ambos sistemas demuestran que el uso correcto de la **herencia, la abstracción y el polimorfismo** permite construir programas estructurados, flexibles y de fácil mantenimiento, reflejando la capacidad del programador para aplicar conceptos teóricos en situaciones prácticas y variadas.