UNIVERSIDAD MAYOR DE SAN ANDRÉS
FACULTAD DE CIENCIAS PURAS Y NATURALES

UNIVERSITARIO(S):

- Gonzalo Churqui Quispe
- Jorge Crespo Villamil
- Angel Coaquira Condori
- William Camasita Apaza
- Rodrigo Cuba Ramirez
- Dougles Roman Quispe
- Gabriel Leonardo Copa Cahuaya

PROYECTO: SISTEMA DE GESTIÓN DEL ALBERGUE DE MASCOTAS

CARRERA: INFORMÁTICA

MATERIA: PROGRAMACION II

DOCENTE: ROSALIA LOPEZ M.

LA PAZ - BOLIVIA

Elaboración del Proyecto de un SISTEMA DE GESTIÓN DEL ALBERGUE DE MASCOTAS en el Contexto de la Programación Orientada a Objetos (POO)

Introduccion

1. **Definición del Proyecto**

   1.1 Descripción General

   El proyecto consiste en el diseño e implementación de un sistema de gestión para un albergue de mascotas. Las funcionalidades clave del sistema incluyen:

   - Registro de mascotas (tipo, raza, edad, estado de salud).
   - Registro de adoptantes y personal.
   - Gestión de adopciones y seguimiento de historial de adopciones.
   - Gestión de las reservas y cuidados diarios de las mascotas.

   1.2 Objetivos

   - Aplicar los conceptos fundamentales de la POO para desarrollar un sistema modular y escalable.
   - Mejorar la gestión administrativa del albergue mediante un sistema de fácil uso.
   - Implementar patrones de diseño que optimizan la flexibilidad, reutilización y mantenimiento del código.

2. **Análisis y Diseño**

   2.1 Diagrama UML

   El sistema incluirá los siguientes diagramas UML:

   - Diagrama de clases: Representará las relaciones entre las clases Mascota, Adoptante, Personal y Reserva.

   2.2 Principios de Diseño

   - Herencia: Las clases Perro y Gato heredan de la clase base Mascota.
   - Composición: La clase Reserva incluye un objeto de tipo Mascota y Adoptante.
   - Agregación: Un Adoptante puede estar asociado a varias Reservas.
   - Genericidad: Uso de colecciones genéricas para gestionar listas de Mascotas y Adoptantes.

- Interfaces: Se implementa una interfaz Gestionable para las operaciones CRUD (crear, leer, actualizar, eliminar) relacionadas con Mascotas y Adoptantes.
- Patrones de Diseño: El patrón de diseño Factory se utilizará para la creación de objetos de tipo Mascota (Perro, Gato, etc.) dependiendo del tipo de entrada.

3. Implementación en Java

3.1 Estructura del Proyecto

Paquetes:

modelo: Contendrá las clases del dominio, como Mascota, Adoptante, Reserva, etc.

controlador: Incluirá la lógica de negocio (como el manejo de adopciones, reservas).

Vista: Representará la interfaz de usuario (interacción con el sistema).

3.2 Código Fuente

```
import java.io.Serializable;

import java.util.Scanner;


public class Adoptante extends Persona implements Serializable {

private static final long serialVersionUID = 1L;

    private int id;

   private String desc;


   public Adoptante(String n, String d, int e, String t, String de) {

      super(n, d, e, t);
```

```
            desc = de;
        }
    public Adoptante(int id, String n, String d, int e, String t, String de) {
            super(n, d, e, t);
            desc = de;
            this.id = id;
        }
        @Override
        public String toString() {
            return "Adoptante [nombre=" + nombre + ", direccion=" + direccion + ",
edad=" + edad
                    + ", telefono=" + telefono + ", descripcion=" + desc + "]";
        }public String getNombre() {
            return nombre; // nombre es un atributo heredado de Persona
        }
         public void leer() {
            super.leer();
            desc = leer.nextLine();
        }
    public void mostrar() {
            super.mostrar();
            System.out.println(desc);
        }
public String getDesc() {
```

```
    return desc;
}

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

public void setDesc(String desc) {
    this.desc = desc;
}

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

public void setDireccion(String direccion) {
    this.direccion = direccion;
}

public void setEdad(int edad) {
```

```
    this.edad = edad;
}

public void setTelefono(String telefono) {
    this.telefono = telefono;
}

public void setLeer(Scanner leer) {
    this.leer = leer;
}

public String getDireccion() {
    return direccion;
}

public int getEdad() {
    return edad;
}

public String getTelefono() {
    return telefono;
}
```

```
        public Scanner getLeer() {
            return leer;
        }

    }
```

public class Personal extends Persona implements Serializable {

```
    private static final long serialVersionUID = 1L;

    private String cargo;
    private Double sueldo;

    public Personal(String n, String d, int e, String t, String c, Double s) {
        super(n, d, e, t);
        cargo = c;
        sueldo = s;
    }

    @Override
    public String toString() {
        return "Personal [nombre=" + nombre + ", direccion=" + direccion + ", edad=" +
edad
                + ", telefono=" + telefono + ", cargo=" + cargo + ", sueldo=" + sueldo + "]";
```

```
}

public void leer() {
    super.leer();
    cargo = leer.nextLine();
    sueldo = leer.nextDouble();
}

public void mostrar() {
    super.mostrar();
    System.out.println("TIPO: PERSONAL");
    System.out.println(cargo + "\t" + sueldo);
}

public String getCargo() {
    return cargo;
}

public Double getSueldo() {
    return sueldo;
}

public String getNombre() {
```

```
        return nombre;
    }

    public String getDireccion() {
        return direccion;
    }

    public int getEdad() {
        return edad;
    }

    public String getTelefono() {
        return telefono;
    }

    public Scanner getLeer() {
        return leer;
    }
}
public class Persona implements Serializable {
    private static final long serialVersionUID = 1L;

    protected String nombre;
```

```
protected String direccion;
protected int edad;
protected String telefono;
transient Scanner leer = new Scanner(System.in);

public Persona(String n, String d, int e, String t) {
    nombre = n;
    direccion = d;
    edad = e;
    telefono = t;
}

public void leer() {
    nombre = leer.nextLine();
    direccion = leer.nextLine();
    edad = leer.nextInt();
    telefono = leer.nextLine();
}

public void mostrar() {
    System.out.println("----------PERSONA----------");
    System.out.println("NOMBRE\tDIRECCION\tEDAD\tTELEFONO");
    System.out.println(nombre + "\t" + direccion + "\t" + edad + "\t" + telefono);
```

```
}

public String getNombre() {
    return nombre;
}

public String getDireccion() {
    return direccion;
}

public int getEdad() {
    return edad;
}

public String getTelefono() {
    return telefono;
}

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

public void setDireccion(String direccion) {
```

```
        this.direccion = direccion;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public void setTelefono(String telefono) {
        this.telefono = telefono;
    }

    public void setLeer(Scanner leer) {
        this.leer = leer;
    }
}
public class Mascota implements Serializable {

    private static final long serialVersionUID = 1L;

    protected int id;
    protected String nombre;
    protected int edad;
    protected String sexo;
```

```
protected String raza;
protected String tipo;

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

public void setEdad(int id) {
    this.id = id;
}

public void setSexo(String sexo) {
    this.sexo = sexo;
}

public void setRaza(String raza) {
    this.raza = raza;
}

public void setTipo(String tipo) {
    this.tipo = tipo;
}
```

```
public void setLeer(Scanner leer) {
    this.leer = leer;
}

transient Scanner leer = new Scanner(System.in);

public Mascota(String n, int e, String s, String r, String t) {
    nombre = n;
    edad = e;
    sexo = s;
    raza = r;
    tipo = t;
}

@Override
public String toString() {
    return "Mascota{"
            + "nombre='" + nombre + '\''
            + ", edad=" + edad
            + ", sexo='" + sexo + '\''
            + ", raza='" + raza + '\''
            + ", tipo='" + tipo + '\''
            + '}';
```

```
}

public void leer() {

    nombre = leer.nextLine();
    edad = leer.nextInt();
    sexo = leer.nextLine();

    raza = leer.nextLine();
    tipo = leer.nextLine();

}

public void mostrar() {
    System.out.println("MASCOTA");
    System.out.println("NOMBRE\tEDAD\tSEXO\tRAZA\tTIPO");
    System.out.println(nombre + "\t" + edad + "\t" + sexo + "\t" + raza + "\t" + tipo);
}

public int getID() {
    return id;
}
public String getNombre() {
```

```
        return nombre;
    }

    public int getEdad() {
        return edad;
    }

    public String getSexo() {
        return sexo;
    }

    public String getRaza() {
        return raza;
    }

    public String getTipo() {
        return tipo;
    }

    int getId() {
        throw new UnsupportedOperationException("Not supported yet."); // Generated
from nbfs://nbhost/SystemFileSystem/Templates/Classes/Code/GeneratedMethodBody
    }
}
```

```
class RegistroAdopcion implements Serializable {

    private static final long serialVersionUID = 1L;

    private Adoptante adoptante;
    private Mascota mascota;
    private String fechaAdopcion;

    public RegistroAdopcion(Adoptante adoptante, Mascota mascota, String
fechaAdopcion) {
        this.adoptante = adoptante;
        this.mascota = mascota;
        this.fechaAdopcion = fechaAdopcion;
    }

    public Adoptante getAdoptante() {
        return adoptante;
    }

    public Mascota getMascota() {
        return mascota;
    }

    public String getFechaAdopcion() {
```

```
        return fechaAdopcion;
    }

    @Override
    public String toString() {
        return "Adoptante: " + adoptante.getNombre()
                + ", Mascota: " + mascota.getNombre()
                + ", Fecha de Adopción: " + fechaAdopcion;
    }
}
public class dbConnection {
    static String url="jdbc:mysql://localhost:3306/Albergue";
    static String user="root";
    static String pass="";

    public static Connection conectar() {
        Connection con= null;
        try {
           con=DriverManager.getConnection(url,user,pass);
            System.out.println("Conexión exitosa");
        } catch (SQLException e) {
            e.printStackTrace();
        }
```

```
        return con;
    }
}

public class Albergue {

    private String nombre;
    private String ubicacion;
    private int capacidad;
    private int np;
    private int nm;
    private int nad;
    private Adoptante[] a = new Adoptante[50];
    private Persona[] p = new Persona[50];
    private Mascota[] m = new Mascota[50];

    public void setAdoptantes(Adoptante[] adoptantes, int nad) {
        this.a = adoptantes;
        this.nad = nad;
    }

    private RegistroAdopcion[] adopciones = new RegistroAdopcion[100];
    private int na = 0;
```

```
public Adoptante[] getAdoptantes() {
    return a;
}

public int getNad() {
    return nad;
}

public Adoptante[] getA() {
    return a;
}

public RegistroAdopcion[] getAdopciones() {
    return adopciones;
}

public int getNa() {
    return na;
}

public String getNombre() {
    return nombre;
```

```
}

public String getUbicacion() {
    return ubicacion;
}

public int getCapacidad() {
    return capacidad;
}

public int getNp() {
    return np;
}

public int getNm() {
    return nm;
}

public Persona[] getP() {
    return p;
}

public Mascota[] getM() {
```

```
    return m;
}

public Albergue(String n, String u, int c) {
    nombre = n;
    ubicacion = u;
    capacidad = c;
    np = 0;
    nm = 0;
    nad = 0;
}

public void adicionarP(Persona px) {
    if (np < 50) {
        p[np++] = px;
    } else {
        System.out.println("No se pueden agregar más personas.");
    }
}

public void adicionarAd(Adoptante ad) {
    if (nad < 50) {
        a[nad++] = ad;
```

```
    } else {
        System.out.println("No se pueden agregar más adoptantes.");
    }
}

public void modificarNombreM(String x, String y) {
    for (int i = 0; i < nm; i++) {
        if (m[i].getNombre().equals(x)) {
            m[i].setNombre(y);
        }
    }
}

public void modificarEdadM(String x, int y) {
    for (int i = 0; i < nm; i++) {
        if (m[i].getNombre().equals(x)) {
            m[i].setEdad(y);
        }
    }
}

public void modificarSexoM(String x, String y) {
    for (int i = 0; i < nm; i++) {
```

```
        if (m[i].getNombre().equals(x)) {
            m[i].setSexo(y);
        }
    }
}

public void modificarRazaM(String x, String y) {
    for (int i = 0; i < nm; i++) {
        if (m[i].getNombre().equals(y)) {
            m[i].setRaza(y);
        }
    }
}

public void modificarTipoM(String x, String y) {
    for (int i = 0; i < nm; i++) {
        if (m[i].getNombre().equals(y)) {
            m[i].setTipo(y);
        }
    }
}

public void adicionarM(Mascota mx) {
```

```
    if (nm < 50) {
        m[nm++] = mx;
    } else {
        System.out.println("No se pueden agregar más mascotas.");
    }
}

public void eliminarM(String nombre) {
    boolean encontrado = false;

    for (int i = 0; i < nm; i++) {
        if (m[i].getNombre().equals(nombre)) {
            for (int j = i; j < nm - 1; j++) {
                m[j] = m[j + 1];
            }
            m[nm - 1] = null;
            nm--;
            encontrado = true;
            break;
        }
    }
    if (!encontrado) {
        System.out.println("No se encontró una mascota con el nombre: " + nombre);
```

```
        } else {
            System.out.println("La mascota con el nombre '" + nombre + "' ha sido
eliminada.");
        }
    }

    public void eliminarAdoptante(String nombre) {
        boolean encontrado = false;

        for (int i = 0; i < np; i++) {
            if (p[i] instanceof Adoptante && p[i].getNombre().equals(nombre)) {
                for (int j = i; j < np - 1; j++) {
                    p[j] = p[j + 1];
                }
                p[np - 1] = null;
                np--;
                encontrado = true;
                break;
            }
        }
        if (!encontrado) {
            System.out.println("No se encontró un adoptante con el nombre: " + nombre);
        } else {
```

```
            System.out.println("El adoptante con el nombre '" + nombre + "' ha sido
eliminado.");
        }
    }


    public void eliminarPersonal(String nombre) {
        boolean encontrado = false;


        for (int i = 0; i < np; i++) {
            if (p[i] instanceof Personal && p[i].getNombre().equals(nombre)) {
                for (int j = i; j < np - 1; j++) {
                    p[j] = p[j + 1];
                }
                p[np - 1] = null;
                np--;
                encontrado = true;
                break;
            }
        }
        if (!encontrado) {
            System.out.println("No se encontró a alguien del personal con el nombre: " +
nombre);
        } else {
            System.out.println("El trabajador con el nombre '" + nombre + "' ha sido
eliminado.");
```

```
    }
}

public void mostrarMascota(String x) {
    for (int i = 0; i < nm; i++) {
        if (m[i].getNombre().equals(x)) {
            m[i].mostrar();
        }
    }
}

public void mostrarAdoptante(String x) {
    for (int i = 0; i < np; i++) {
        if (p[i] instanceof Adoptante && p[i].getNombre().equals(x)) {
            p[i].mostrar();
        }
    }
}

public void mostrarP() {
    System.out.println("----------PERSONAS----------");
    System.out.println(nombre + "\t" + ubicacion + "\t" + capacidad);
    for (int i = 0; i < np; i++) {
```

```
        System.out.println("------------Persona " + (i + 1) + "--------------");
        p[i].mostrar();
    }
}

public void mostrarAD() {
    for (int i = 0; i < nad; i++) {
        System.out.println("------------Persona " + (i + 1) + "--------------");
        a[i].mostrar();
    }
}

public void mostrarM() {
    System.out.println("---------------MASCOTAS-----------------");
    System.out.println(nombre + "\t" + ubicacion + "\t" + capacidad);
    for (int j = 0; j < nm; j++) {
        System.out.println("------------Mascota " + (j + 1) + "--------------");
        m[j].mostrar();
    }
}

public void mostrarPersonal(String x) {
    for (int i = 0; i < np; i++) {
```

```
            if (p[i] instanceof Personal && p[i].getNombre().equals(x)) {
                p[i].mostrar();
            }
        }
    }

    public void modificarAdoptante(Adoptante adoptanteModificado) {
        for (int i = 0; i < np; i++) {
            if (p[i] instanceof Adoptante &&
p[i].getNombre().equalsIgnoreCase(adoptanteModificado.getNombre())) {
                p[i] = adoptanteModificado;
                return;
            }
        }
        throw new RuntimeException("Adoptante no encontrado.");
    }

    public void mostrarMascotas() {
        for (int i = 0; i < nm; i++) {
            System.out.println(m[i]);
        }
    }

    public void mostrarAdoptantes() {
```

```
        if (np == 0) {
            System.out.println("No hay adoptantes registrados.");
        } else {
            for (int i = 0; i < np; i++) {
                System.out.println(p[i]);
            }
        }
    }

//====================================PERCISTENCIA==========================
    public void guardarMascotas(String archivo) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(archivo))) {
            oos.writeInt(nm); // Guarda la cantidad de mascotas
            for (int i = 0; i < nm; i++) {
                oos.writeObject(m[i]);
            }
            System.out.println("Mascotas guardadas en el archivo.");
        } catch (IOException e) {
            System.out.println("Error al guardar las mascotas: " + e.getMessage());
        }
    }
```

```
    public void cargarMascotas(String archivo) {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(archivo))) {
            nm = ois.readInt(); // Lee la cantidad de mascotas
            for (int i = 0; i < nm; i++) {
                m[i] = (Mascota) ois.readObject();
            }
            System.out.println("Mascotas cargadas desde el archivo.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error al cargar las mascotas: " + e.getMessage());
        }
    }

    public void guardarAdoptantes(String archivo) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(archivo))) {
            oos.writeInt(np); // Guarda la cantidad de adoptantes
            for (int i = 0; i < np; i++) {
                oos.writeObject(p[i]);
            }
            System.out.println("Adoptantes guardados en el archivo.");
        } catch (IOException e) {
            System.out.println("Error al guardar los adoptantes: " + e.getMessage());
        }
```

```
    }

    public void cargarAdoptantes(String archivo) {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(archivo))) {
            np = ois.readInt(); // Lee la cantidad de adoptantes
            for (int i = 0; i < np; i++) {
                p[i] = (Adoptante) ois.readObject();
            }
            System.out.println("Adoptantes cargados desde el archivo.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error al cargar los adoptantes: " + e.getMessage());
        }
    }

    public void guardarPersonal(String archivo) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(archivo))) {
            oos.writeInt(np); // Guarda la cantidad de personas
            for (int i = 0; i < np; i++) {
                oos.writeObject(p[i]);
            }
            System.out.println("Personal guardado en el archivo.");
        } catch (IOException e) {
```

```
            System.out.println("Error al guardar el personal: " + e.getMessage());
        }
    }

    public void cargarPersonal(String archivo) {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(archivo))) {
            np = ois.readInt(); // Lee la cantidad de personas
            for (int i = 0; i < np; i++) {
                p[i] = (Persona) ois.readObject();
            }
            System.out.println("Personal cargado desde el archivo.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error al cargar el personal: " + e.getMessage());
        }
    }

    public void guardarAdopciones(String archivo) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
FileOutputStream(archivo))) {
            oos.writeInt(na);
            for (int i = 0; i < na; i++) {
                oos.writeObject(adopciones[i]);
            }
```

```
            System.out.println("Adopciones guardadas en el archivo.");
        } catch (IOException e) {
            System.out.println("Error al guardar las adopciones: " + e.getMessage());
        }
    }

    public void cargarAdopciones(String archivo) {
        try (ObjectInputStream ois = new ObjectInputStream(new
FileInputStream(archivo))) {
            na = ois.readInt();
            for (int i = 0; i < na; i++) {
                adopciones[i] = (RegistroAdopcion) ois.readObject();
            }
            System.out.println("Adopciones cargadas desde el archivo.");
        } catch (IOException | ClassNotFoundException e) {
            System.out.println("Error al cargar las adopciones: " + e.getMessage());
        }
    }

//==========================================================================
==============

    public void adoptarMascota(Adoptante adoptante, Mascota mascota, String
fechaAdopcion) {
        if (adoptante != null && mascota != null) {
            // Registrar la adopción
```

```
            if (na < adopciones.length) {
                adopciones[na++] = new RegistroAdopcion(adoptante, mascota,
fechaAdopcion);
                System.out.println("¡Adopción registrada exitosamente!");
                guardarAdopciones("adopciones.dat"); // Guardar las adopciones
            } else {
                System.out.println("No se pueden registrar más adopciones.");
            }
        } else {
            if (adoptante == null) {
                System.out.println("El adoptante proporcionado es nulo.");
            }
            if (mascota == null) {
                System.out.println("La mascota proporcionada es nula.");
            }
        }
    }

    public void listarAdopciones() {
        System.out.println("------------REGISTRO DE ADOPCIONES------------");
        if (na == 0) {
            System.out.println("No hay adopciones registradas.");
        } else {
            for (int i = 0; i < na; i++) {
```

```
            System.out.println(adopciones[i]);
        }
    }
}

//=====================BASE DE DATOS=====================================
public void guardarAdoptantes(Connection con) {
    String sql = "INSERT INTO adoptantes (nombre, direccion, edad, telefono, descripcion) VALUES (?, ?, ?, ?, ?)";
    try (PreparedStatement stmt = con.prepareStatement(sql)) {
        for (int i = 0; i < nad; i++) {
            stmt.setString(1, a[i].getNombre());
            stmt.setString(2, a[i].getDireccion());
            stmt.setInt(3, a[i].getEdad());
            stmt.setString(4, a[i].getTelefono());
            stmt.setString(5, a[i].getDesc());
            stmt.executeUpdate();
        }
        System.out.println("Adoptantes guardados en la base de datos.");
    } catch (SQLException e) {
        System.out.println("Error al guardar adoptantes: " + e.getMessage());
    }
}
```

```
    public void guardarPersonal(Connection con) {
        String sql = "INSERT INTO personal (nombre, direccion, edad, telefono, cargo,
sueldo) VALUES (?, ?, ?, ?, ?, ?)";
        try (PreparedStatement stmt = con.prepareStatement(sql)) {
            for (int i = 0; i < np; i++) {
                Personal personal = (Personal) p[i];
                stmt.setString(1, personal.getNombre());
                stmt.setString(2, personal.getDireccion());
                stmt.setInt(3, personal.getEdad());
                stmt.setString(4, personal.getTelefono());
                stmt.setString(5, personal.getCargo());
                stmt.setDouble(6, personal.getSueldo());
                stmt.executeUpdate();
            }
            System.out.println("Personal guardado en la base de datos.");
        } catch (SQLException e) {
            System.out.println("Error al guardar personal: " + e.getMessage());
        }
    }

    // Método para cargar mascotas desde la BD
    public void cargarMascotasBD() {
        String query = "SELECT * FROM mascotas";
```

```
        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query); ResultSet rs = stmt.executeQuery()) {


            nm = 0; // Reiniciar el contador

            while (rs.next()) {

                String nombre = rs.getString("nombre");

                int edad = rs.getInt("edad");

                String sexo = rs.getString("sexo");

                String raza = rs.getString("raza");

                String tipo = rs.getString("tipo");


                // Almacenar la mascota en el array

                m[nm] = new Mascota(nombre, edad, sexo, raza, tipo);

                nm++;

            }


        } catch (SQLException e) {

            e.printStackTrace();

        }

    }


// Método para guardar una mascota en la BD

    public void guardarMascotaBD(Mascota mascota) {
```

```
        String query = "INSERT INTO mascotas (nombre, edad, sexo, raza, tipo) VALUES (?, ?, ?, ?, ?)";

        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt = conn.prepareStatement(query)) {


            stmt.setString(1, mascota.getNombre());

            stmt.setInt(2, mascota.getEdad());

            stmt.setString(3, mascota.getSexo());

            stmt.setString(4, mascota.getRaza());

            stmt.setString(5, mascota.getTipo());


            stmt.executeUpdate();


            // Guardar la mascota en el array

            m[nm] = mascota;

            nm++;


        } catch (SQLException e) {

            e.printStackTrace();

        }

    }


// Método para eliminar una mascota de la BD

    public void eliminarMascotaBD(String nombre) {
```

```
        String query = "DELETE FROM mascotas WHERE nombre = ?";

        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {



            stmt.setString(1, nombre);

            stmt.executeUpdate();

            for (int i = 0; i < nm; i++) {

                if (m[i].getNombre().equalsIgnoreCase(nombre)) {

                    for (int j = i; j < nm - 1; j++) {

                        m[j] = m[j + 1];

                    }

                    m[nm - 1] = null;

                    nm--;

                    break;

                }

            }



        } catch (SQLException e) {

            e.printStackTrace();

        }

    }



    public void actualizarMascotaBD(Mascota mascota) {
```

```
        String query = "UPDATE mascotas SET edad = ?, sexo = ?, raza = ?, tipo = ? WHERE nombre = ?";

        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt = conn.prepareStatement(query)) {



            stmt.setInt(1, mascota.getEdad());

            stmt.setString(2, mascota.getSexo());

            stmt.setString(3, mascota.getRaza());

            stmt.setString(4, mascota.getTipo());

            stmt.setString(5, mascota.getNombre());



            int rowsUpdated = stmt.executeUpdate();

            if (rowsUpdated > 0) {

                System.out.println("Mascota actualizada con éxito.");

            }



        } catch (SQLException e) {

            e.printStackTrace();

        }

    }



    public void agregarMascotaBD(Mascota mascota) {

        String query = "INSERT INTO mascotas (nombre, edad, sexo, raza, tipo) VALUES (?, ?, ?, ?, ?)";
```

```
        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {

            stmt.setString(1, mascota.getNombre());

            stmt.setInt(2, mascota.getEdad());

            stmt.setString(3, mascota.getSexo());

            stmt.setString(4, mascota.getRaza());

            stmt.setString(5, mascota.getTipo());


            int rowsInserted = stmt.executeUpdate();


            if (rowsInserted > 0) {

                System.out.println("Mascota agregada con éxito.");

            }


        } catch (SQLException e) {

            e.printStackTrace();

        }

    }


    //====================BASE DE
DATOS====================================

    public void cargarAdoptantesBD() {

        String query = "SELECT * FROM adoptantes";

        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query); ResultSet rs = stmt.executeQuery()) {
```

```
            np = 0; // Reiniciar el contador de adoptantes
            while (rs.next()) {
                String nombre = rs.getString("nombre");
                String direccion = rs.getString("direccion");
                int edad = rs.getInt("edad");
                String telefono = rs.getString("telefono");
                String descripcion = rs.getString("descripcion");

                // Crear el adoptante y agregarlo al array
                p[np] = new Adoptante(nombre, direccion, edad, telefono, descripcion);
                np++;
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void eliminarAdoptanteBD(String nombre) {
        String query = "DELETE FROM adoptantes WHERE nombre = ?";
        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {

            stmt.setString(1, nombre);
```

```
            stmt.executeUpdate();


            for (int i = 0; i < nm; i++) {
                if (p[i].getNombre().equalsIgnoreCase(nombre)) {
                    for (int j = i; j < nm - 1; j++) {
                        p[j] = p[j + 1];
                    }
                    p[nm - 1] = null;
                    nm--;
                    break;
                }
            }


        } catch (SQLException e) {
            e.printStackTrace();
        }
    }


    public void actualizarAdoptanteBD(Adoptante adoptante) {
        String query = "UPDATE adoptantes SET edad = ?, direccion = ?, telefono = ?,
descripcion = ? WHERE nombre = ?";
        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {
```

```
            stmt.setInt(1, adoptante.getEdad());
            stmt.setString(2, adoptante.getDireccion());
            stmt.setString(3, adoptante.getTelefono());
            stmt.setString(4, adoptante.getDesc());
            stmt.setString(5, adoptante.getNombre());

            int rowsUpdated = stmt.executeUpdate();
            if (rowsUpdated > 0) {
                System.out.println("Adoptante actualizado con éxito.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void agregarAdoptanteBD(Adoptante adoptante) {
        String query = "INSERT INTO adoptantes (nombre, edad, direccion, telefono,
descripcion) VALUES (?, ?, ?, ?, ?)";

        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {
            stmt.setString(1, adoptante.getNombre());
            stmt.setInt(2, adoptante.getEdad());
```

```
            stmt.setString(3, adoptante.getDireccion());
            stmt.setString(4, adoptante.getTelefono());
            stmt.setString(5, adoptante.getDesc());

            int rowsInserted = stmt.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("Adoptante agregado con éxito.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //====================BASE DE DATOS====================================
    public void cargarPersonalBD() {
        String query = "SELECT * FROM personal";
        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt = conn.prepareStatement(query); ResultSet rs = stmt.executeQuery()) {

            np = 0; // Reiniciar el contador de personal
            while (rs.next()) {
                String nombre = rs.getString("nombre");
```

```
            String direccion = rs.getString("direccion");
            int edad = rs.getInt("edad");
            String telefono = rs.getString("telefono");
            String cargo = rs.getString("cargo");
            double sueldo = rs.getDouble("sueldo");

            // Crear el personal y agregarlo al array
            p[np] = new Personal(nombre, direccion, edad, telefono, cargo, sueldo);
            np++;
        }
    } catch (SQLException e) {
        e.printStackTrace();
    }
}

public void eliminarPersonalBD(String nombre) {
    String query = "DELETE FROM personal WHERE nombre = ?";
    try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {

        stmt.setString(1, nombre);
        stmt.executeUpdate();

        // Eliminar de la lista local (en el array)
```

```
            for (int i = 0; i < np; i++) {
                if (p[i].getNombre().equalsIgnoreCase(nombre)) {
                    for (int j = i; j < np - 1; j++) {
                        p[j] = p[j + 1];
                    }
                    p[np - 1] = null;
                    np--;
                    break;
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void actualizarPersonalBD(Personal personal) {
        String query = "UPDATE personal SET edad = ?, direccion = ?, telefono = ?, cargo
= ?, sueldo = ? WHERE nombre = ?";
        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {

            stmt.setInt(1, personal.getEdad());
            stmt.setString(2, personal.getDireccion());
```

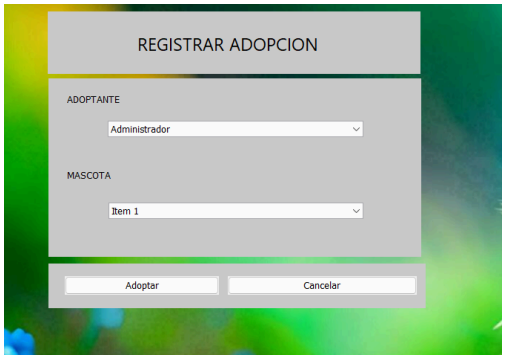
```
            stmt.setString(3, personal.getTelefono());
            stmt.setString(4, personal.getCargo());
            stmt.setDouble(5, personal.getSueldo());
            stmt.setString(6, personal.getNombre());

            int rowsUpdated = stmt.executeUpdate();
            if (rowsUpdated > 0) {
                System.out.println("Personal actualizado con éxito.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    public void agregarPersonalBD(Personal personal) {
        String query = "INSERT INTO personal (nombre, edad, direccion, telefono, cargo,
sueldo) VALUES (?, ?, ?, ?, ?, ?)";

        try (Connection conn = dbConnection.conectar(); PreparedStatement stmt =
conn.prepareStatement(query)) {
            stmt.setString(1, personal.getNombre());
            stmt.setInt(2, personal.getEdad());
            stmt.setString(3, personal.getDireccion());
```

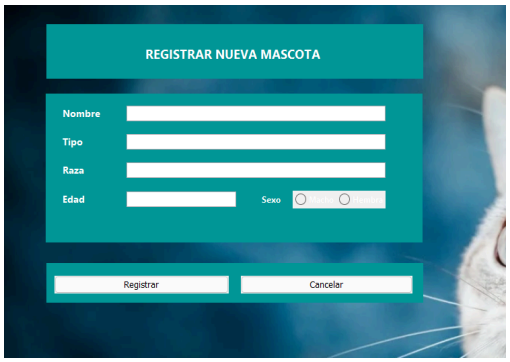
```
            stmt.setString(4, personal.getTelefono());
            stmt.setString(5, personal.getCargo());
            stmt.setDouble(6, personal.getSueldo());

            int rowsInserted = stmt.executeUpdate();
            if (rowsInserted > 0) {
                System.out.println("Personal agregado con éxito.");
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }

    //====================BASE DE
DATOS====================================
}
```

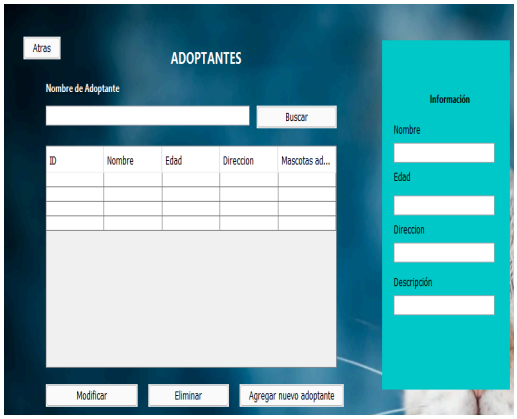
```
public class Main {


    public static void main(String[] args) {


        dbConnection dbc = new dbConnection();

        Connection con = dbc.conectar();


        if (con != null) {

            // Crear objetos y poblar el albergue

            Albergue albergue = new Albergue("Patas", "Prado", 100);


            // Agregar datos al albergue

            albergue.adicionarAd(new Adoptante("Carlos", "El Alto", 20, "77733341",
"Adopta gatos"));

            albergue.adicionarP(new Personal("Reina", "El Alto", 20, "77733341",
"Cuidador", 200.0));

            albergue.adicionarM(new Mascota("Manchas", 3, "Macho", "Calico", "Gato"));


            // Guardar los datos en la base de datos

            albergue.guardarAdoptantes(con);

            albergue.guardarPersonal(con);

        } else {

            System.out.println("No se pudo establecer conexión con la base de datos.");

        }
```

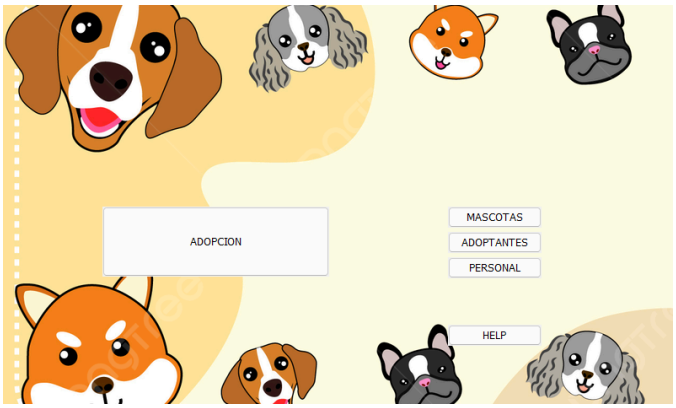
}

} 3.3 Diseño de interfaces

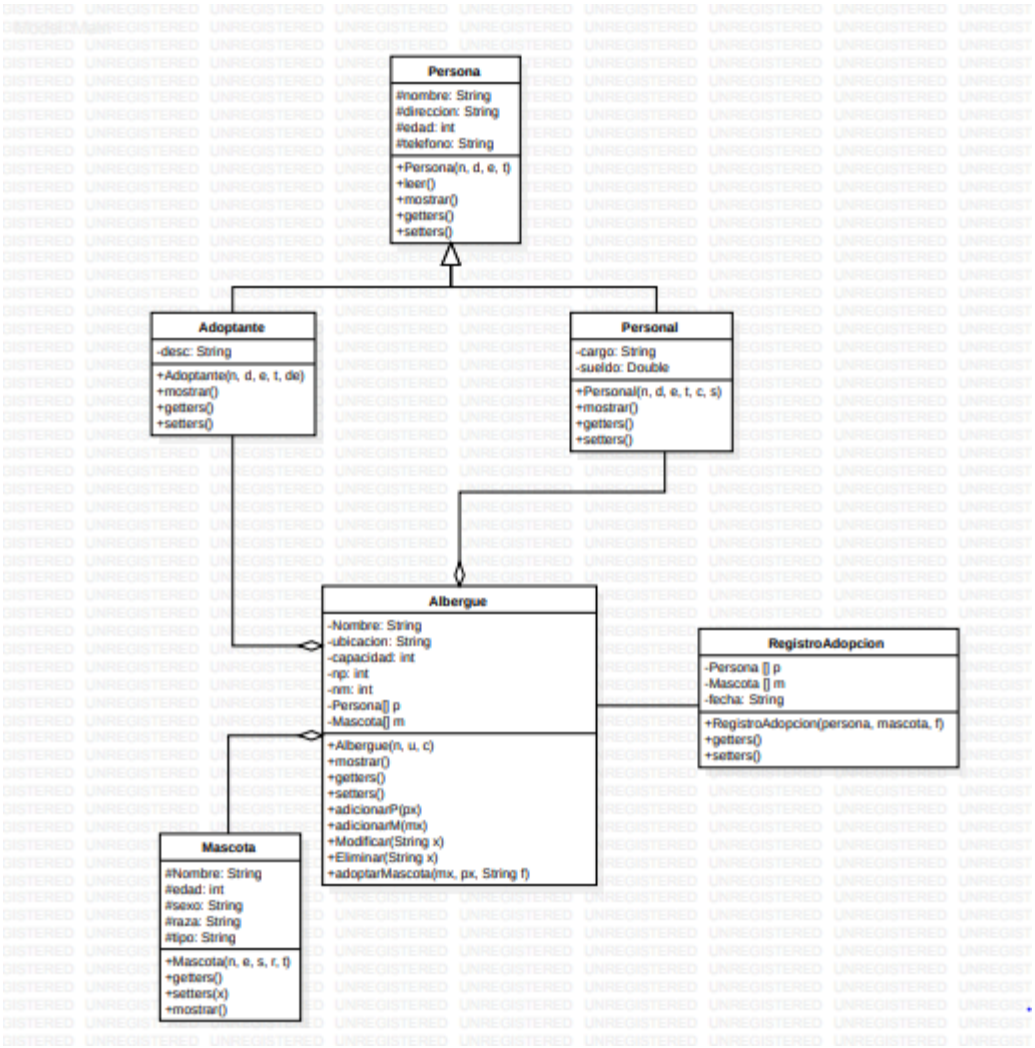








3.3.1 Diagrama de clases



3.4 Manejo de Archivos

El sistema implementará la persistencia de datos mediante el uso de una base de datos relacional, como MySQL, para almacenar los datos de mascotas, adoptantes y reservas.

3.5 Uso de Patrones de Diseño

El patrón Factory será utilizado para crear objetos de las diferentes razas de mascotas de manera flexible.

4. Pruebas del Sistema

Se llevarán a cabo pruebas unitarias y de integración para verificar la funcionalidad de los métodos de gestión de adopciones, reservas y

mantenimiento de registros de las mascotas. Además, se realizarán pruebas de rendimiento para asegurar la escalabilidad del sistema.

5. Conclusión

El sistema desarrollado para la gestión del albergue de mascotas incorpora los principios clave de la Programación Orientada a Objetos, proporcionando una solución eficiente, modular y escalable. La implementación de patrones de diseño y la estructura del sistema aseguran su mantenimiento a largo plazo, así como la facilidad de adaptarlo a futuras necesidades del albergue.