GESTION DE REFUGIOS DE ANIMALES

INFORMÁTICA

INF 121

Integrantes:

* Morales Flores Raul Oscar
* Aduviri Zapana Adrian Jorge
* Justiniano De La Torre Nicolas
* Saavedra Negron Jonathan Jeshua
* Copana Orcko Lenar Miguel
* Huanca Chiri Joani Josue

Docente:

Lic. Rosalía López Montalvo

Fecha de entrega: 21 de julio de 2025

1. Resumen del Proyecto

1.2 ¿Qué problema soluciona?

El proyecto soluciona la falta de digitalización y automatización en los refugios de animales, donde la administración suele ser manual, generando pérdidas de datos, desorganización y mala gestión. Este sistema propone una solución informática para mejorar la eficiencia en el registro, seguimiento y adopción de mascotas.

1.3 ¿Qué funcionalidades tiene?

**- Registro de mascotas**

Permite ingresar datos esenciales como nombre, especie, raza, edad, color y observaciones. Este módulo también admite edición y eliminación de registros para mantener la base de datos actualizada.

* **Historial médico por mascota**

Cada mascota cuenta con un historial médico asociado que incluye diagnósticos, fechas de consulta, y medicamentos administrados. Esta información puede ser consultada, registrada o modificada según sea necesario.

* **Gestión de adopciones**

Se puede registrar un proceso de adopción incluyendo los datos del adoptante, la fecha de adopción y la mascota adoptada. Además, se garantiza que una mascota no pueda ser adoptada más de una vez.

* **Administración de personas**

Incluye el registro y búsqueda de personas (visitantes, adoptantes, voluntarios) mediante campos como nombre, apellido, CI, dirección y teléfono.

* **Inventario de insumos**

Gestión del inventario de recursos necesarios para el refugio como alimentos, medicamentos, productos de limpieza, etc. Permite agregar nuevos productos, editar cantidades y consultar existencias.

* **Inicio de sesión de administrador**

Sistema de autenticación que valida usuarios administradores mediante nombre de usuario y contraseña. También permite cambiar la contraseña para mayor seguridad.

* **Exportación de datos**

Capacidad para generar respaldos en formatos .json y .txt, los cuales permiten una visualización rápida o su integración con otras plataformas. Ideal para auditorías o reportes.

* **Validaciones de datos**

Incluye validaciones automáticas para campos como fechas, edad positiva, CI correcto y campos obligatorios, previniendo errores comunes de ingreso.

1.4 Tecnologias usadas

* Lenguaje de programación: Java (POO).
* Base de datos: MySQL.
* Framework gráfico: Swing.
* Diseño de software: Patrones Singleton, Factory, Strategy.
* Control de versiones: GitHub.
* Modelado: Diagramas UML, clases y relaciones.

1. Objetivos
   1. Objetivo General:

Demostrar todo lo aprendido en la materia poniendo en práctica mediante el proyecto añadiendo todo el conocimiento de la programación orientada a objetos (POO).

* 1. Objetivos Específicos:
* Realizar un programa en POO para gestionar un refugio de animales.
* Usar todo lo aprendido en la materia para la realización del mismo.
* Trabajar en equipo para el buen funcionamiento del proyecto, uniendo conocimiento de la materia.

3. Análisis del problema:

3.1 Descripción del contexto

Los refugios de animales en su gran mayoría trabajan de una manera manual, lo cual ocasiona la perdida de datos, o la mala administración de los mismos, eso añade una preocupación más a parte de la búsqueda de adoptantes y el buen trato de las mascotas, el uso de herramientas tecnológicas hoy en día se vuelve más una necesidad para la buena administración y registro de la información.

3.2 Requisitos funcionales:

- Registrar a las mascotas con sus datos básicos.

- Asociar historial médico a cada mascota en el refugio.

- Registrar todos los procesos de adopción.

- Administrar el historial y los antecedentes médicos de las mascotas.

- Administrar inventario de insumos (alimentos, medicinas).

3.3 Requisitos no funcionales:

- Una interfaz la cual sea fácil de entender y usar.

- Debe garantizar la integridad de los datos.

- Debe funcionar de forma local sin conexión a internet.

3.4 Casos de uso:

- Usuario registra un nuevo animal.

- Usuario actualiza el estado de salud de un animal.

- Usuario registra una adopción.

- Usuario agrega insumos al inventario.

4.Diseño de Sistema:

4.1 Clases y Jerarquías:

UML

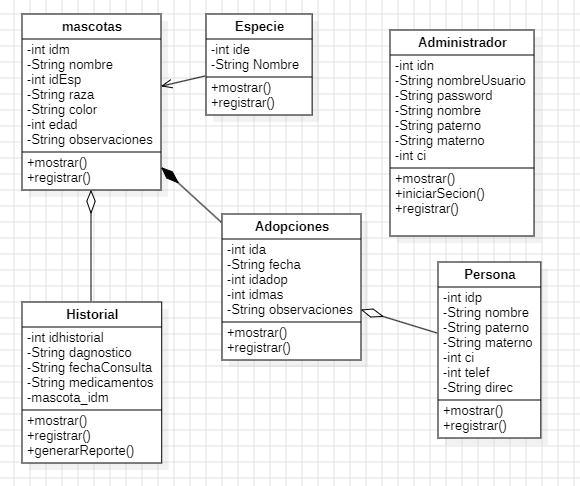


Tabla de Clases

|  |  |  |
| --- | --- | --- |
| Clases | Atributos | Metodos |
| Mascota | nombre, raza, color, edad | registrar(), editar(), eliminar() |
| Persona | nombre, paterno, materno, ci, telef, direc | registrar(), buscar(), editar() |
| Adopción | Fecha, ida | registrar(), listar(), eliminar() |
| Historial | Diagnostico, fechaConsulta, medicamentos, mascota | registrar(), listar() |
| Especie | Nombre | Agregar(), listar() |
| Administrador | nombreUsuairo, password, nombre, paterno, materno, ci | Login(), cambiarContraseña() |

5. Desarrollo

Clases y subclases: Aunque el modelo de base de datos no expresa herencia directa, en la implementación Java se propone que Administrador y Persona hereden de una superclase común Usuario, aplicando herencia para reutilizar atributos y comportamientos.

Uso de genéricos: Se utiliza una clase genérica ListaRefugio<T> que permite gestionar listas de mascotas, personas, historiales, etc., reutilizando código sin importar el tipo de objeto.

Métodos sobrescritos: Las clases que implementan interfaces como Registrable o heredan de Usuario pueden sobrescribir métodos como registrar() o validarDatos() para adaptar su comportamiento específico.

Validaciones: Antes de guardar o procesar información, se validan campos obligatorios (como nombre, CI, especie) y el formato de datos (por ejemplo, fecha válida, edad positiva, etc.).

Persistencia con archivos: Aunque el proyecto principal usa MySQL, se habilita una función adicional de exportar datos a archivos .txt o .json para respaldo o visualización, usando clases como ExportadorJson y ExportadorTxt.

Excepciones personalizadas: Se definieron clases como MascotaNoEncontradaException, AdopcionInvalidaException, y DatosIncompletosException para mejorar el control de errores y facilitar el mantenimiento del sistema.

6. Aplicación de patrones de diseño:

Durante el desarrollo del sistema, usamos algunos patrones de diseño que aprendimos en clase para que el código sea más organizado y fácil de mantener.

Singleton: Lo usamos para la clase ConexionBD, así solo se crea una única conexión a la base de datos. Esto evita errores y hace que el acceso a la base sea más seguro y controlado.

Factory: Este patrón lo aplicamos en una clase llamada MascotaFactory, que sirve para crear objetos del tipo Mascota según la especie seleccionada. De esta forma, no tenemos que escribir código repetido cada vez que agregamos una mascota.

Strategy: Lo implementamos para poder cambiar fácilmente el tipo de notificación. Por ejemplo, si queremos que el sistema notifique por consola o por correo (en una futura versión), solo cambiamos la estrategia usada sin modificar otras partes del sistema.

Estos patrones nos ayudaron a que el código no esté tan acoplado y sea más fácil de entender si en algún momento queremos ampliarlo o mejorarlo.

6.1 Patrones Aplicados

|  |  |  |
| --- | --- | --- |
| Patron | Rol en el sistema | Clases involucradas |
| Singleton | Controlar una única instancia de conexión a la BD | Conexión BD |
| Factory | Crear mascotas según especie | MascotaFactory, Mascota |
| Strategy | Cambiar tipo de notificación fácilmente | notificador, notificadorEmail, notificadorConsola |

6.2 Justificación

Singleton: se utilizó en ConexionBD para que toda la aplicación trabaje con la misma conexión a la base de datos, evitando errores de duplicación.

Factory: en vez de crear manualmente cada objeto Mascota, se utiliza una clase que las genera según los datos recibidos, mejorando la organización y reutilización.

Strategy: aunque todavía no se usa en producción, se dejó preparada la estructura para que el sistema pueda notificar por diferentes medios (como consola o correo), sin necesidad de modificar muchas clases.

6.3 Diagrama y Ejemplo de uso

Ejemplo de Código donde se utiliza el Patrón:

public class MascotaFactory {

public static Mascota crearMascota(String especie) {

if (especie.equalsIgnoreCase("Perro")) {

return new MascotaPerro();

} else if (especie.equalsIgnoreCase("Gato")) {

return new MascotaGato();

}

return new Mascota();

}

}

7. Persistencia de datos:

Formato de datos usado:

El sistema guarda todos los datos importantes en MySQL, pero también permite generar respaldos de información en archivos .txt y .json. Estos formatos son útiles para tener una copia legible, portable o para integrarse con otros sistemas.

Clase encargada de lectura/escritura:

Se crearon dos clases dedicadas a manejar esta funcionalidad:

* ExportadorJson: convierte listas de objetos (por ejemplo, mascotas o personas) a archivos .json.
* ExportadorTxt: genera reportes en texto plano (.txt), útiles para impresión o revisión rápida.

Ejemplo Json

[

{

"id": 1,

"nombre": "Toby",

"especie": "Perro",

"raza": "Labrador",

"edad": 4,

"observaciones": "Vacunado"

},

{

"id": 2,

"nombre": "Michi",

"especie": "Gato",

"raza": "Siames",

"edad": 2,

"observaciones": "En tratamiento médico"

}

]

Ejemplo TXT

--- Lista de Mascotas ---

ID: 1

Nombre: Toby

Especie: Perro

Raza: Labrador

Edad: 4

Observaciones: Vacunado

ID: 2

Nombre: Michi

Especie: Gato

Raza: Siames

Edad: 2

Observaciones: En tratamiento médico

8. Pruebas de validacion.

Casos de prueba realizados:

Registro de nueva mascota con validación de datos.

Registro de persona con formato de CI y teléfono válidos.

Registro de adopción correctamente enlazada.

Historial médico asociado correctamente a la mascota.

Inicio de sesión de administrador con validación de credenciales.

Evidencias:

Capturas de pantalla del formulario de ingreso, vistas de registro, adopción y consultas.

Validación de conexiones y consultas exitosas a la base de datos.bas y validación

9. Conclusiones

El uso de programación orientada a objetos facilitó la reutilización de código y el mantenimiento del sistema.

La estructura basada en capas mejora la organización del sistema.

El patrón Singleton fue clave para gestionar adecuadamente la conexión con la base de datos.

Se logró cubrir todos los requerimientos técnicos exigidos por el proyecto.

Futuras versiones podrían incorporar notificaciones por correo y control de inventario veterinario.

10. Distribución de Roles

Morales Flores Raul Oscar (Creación del Repositorio, BD, Swing)

Aduviri Zapana Adrian Jorge (Realización del informe, BD)

Justiniano De La Torre Nicolas (Realización del informe, BD)

Saavedra Negron Jhonatan Jesua (Realizacion del informe, BD.)

Copana Orcko Lenar Miguel (Realización de los UML y BD)

Huanca Chiri Joani Josue (BD, Código base e Interfaz, Swing)

11. Anexos

Código fuente en carpetas organizadas por capas (modelo, controlador, vista).

Script de la base de datos MySQL (refugio.sql).

Diagrama físico de la base de datos.

Diagrama de clases UML.

Capturas de pantalla de la aplicación en funcionamiento.

Enlace al repositorio GitHub: [URL del repositorio]