

PORTAFOLIO

de
evidencia





Curso: Programación Orientada a Objetos

Tutor: Romario Salas Cerdas

Tema: Portafolio / Pet Care

Elaborado por:

Trayce Gonzalez Brenes

Fecha de entrega: 07 de Diciembre 2025



Veterinaria

El sistema se desarrolla en una clínica veterinaria llamada PetCare, donde se atienden diferentes tipos de mascotas. En este lugar, los clientes pueden registrar a sus animales, programar citas y dar seguimiento a la atención médica que reciben. La idea es representar una veterinaria moderna, con un ambiente organizado y amigable, que busca aprovechar la tecnología para mejorar la atención.

Elegí este sistema porque es un ejemplo práctico y fácil de entender, donde puedo aplicar todo lo que hemos visto en clase sobre Programación Orientada a Objetos. Permite trabajar con clases, objetos y relaciones como la agregación, composición y dependencia, de una forma clara y realista. Además, me gusta la temática porque combina la parte técnica con algo cotidiano y cercano, como el cuidado de las mascotas.

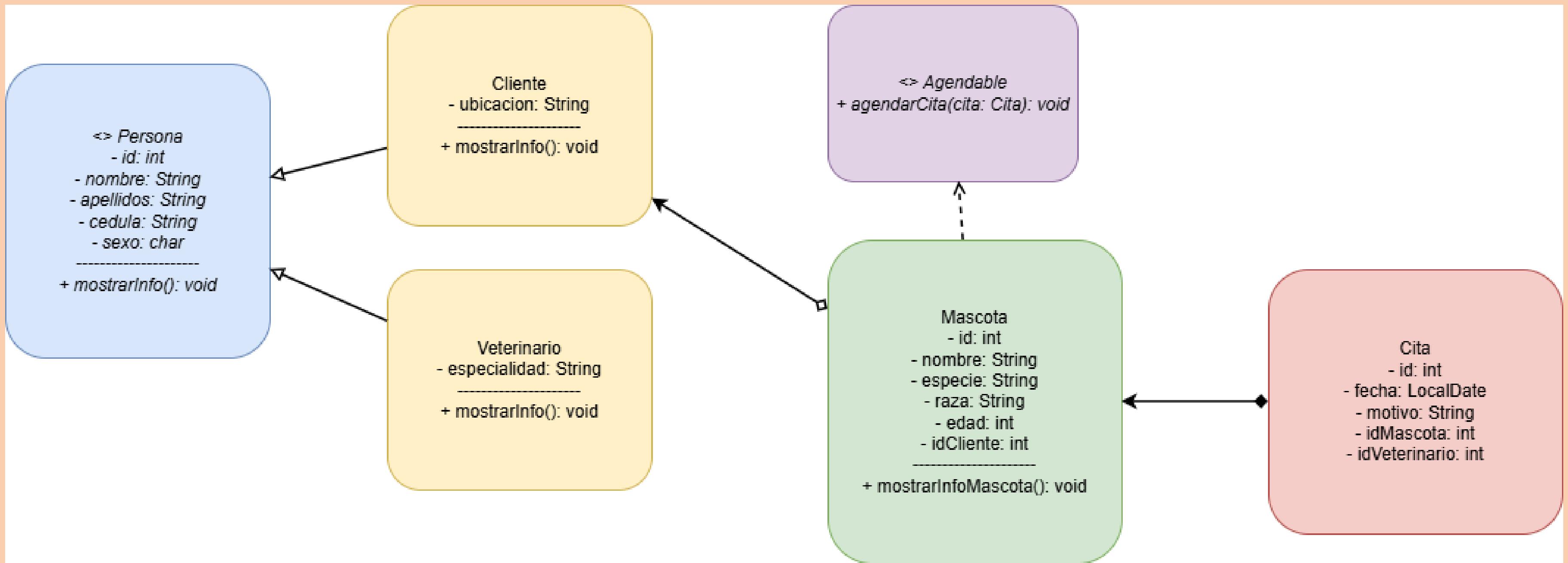
Veterinaria

El sistema “PetCare” surge como respuesta a la necesidad común en las clínicas veterinarias de gestionar de manera eficiente la información de clientes, mascotas, citas y personal veterinario.

En muchos entornos, estos procesos se realizan manualmente, lo cual genera errores, pérdida de información y dificultad para dar seguimiento adecuado a los pacientes.

Por ello, se propone una solución digital que permita organizar los datos de forma estructurada y confiable, aplicando buenas prácticas de software. Este sistema fue elegido por su relevancia, aplicabilidad real y riqueza conceptual para evidenciar los principios de la POO.

Diagrama UML



El diagrama UML ilustra la estructura del sistema y las relaciones entre clases.

La clase abstracta Persona se posiciona como padre de Cliente y Veterinario.

La clase Mascota implementa la interfaz Agendable, indicando capacidad de registrar citas.

La relación de agregación entre Cliente y Mascota se representa con un rombo vacío, mientras que la composición entre Mascota y Cita utiliza un rombo sólido, señalando dependencia total.

Las flechas abiertas punteadas representan implementación de interfaz y las flechas triangulares sólidas indican herencia.

Principios de POO aplicados al proyecto

Abstracción

Se utiliza la clase abstracta Persona como modelo genérico del cual se derivan las clases Cliente y Veterinario. Esta clase define los atributos comunes sin necesidad de especificar su implementación concreta.

Encapsulamiento

Todos los atributos son declarados como privados y se accede a ellos mediante getters y setters. Esto protege la integridad de los datos y evita manipulaciones directas desde el exterior.

Herencia

Las clases Cliente y Veterinario heredan de Persona, permitiendo reutilizar código y especializar funcionalidades. Esto reduce duplicación y promueve un diseño limpio.

Interfaces

La interfaz Agendable define el comportamiento necesario para los objetos capaces de gestionar citas. La clase Mascota implementa esta interfaz, indicando que puede recibir y manejar citas asociadas.

Agregación

Un Cliente puede tener múltiples Mascota. La vida de la mascota no depende completamente del cliente a nivel de modelado, lo cual refleja una relación de agregación.

Composición

Una Mascota puede tener asociada una o varias Cita; estas no tienen sentido sin la existencia de la mascota. Esta relación refleja la composición, donde la vida del objeto depende de otro.

Dependencias

La capa ui depende de bl para ejecutar reglas de negocio, y bl depende de dl para acceder a la base de datos. Esto permite un diseño desacoplado y modular.

Arquitectura del sistema

Capa ui (User Interface)

Contiene el menú e interacción con el usuario.

No implementa lógica de negocio.

Capa bl (Business Logic)

Gestiona operaciones, valida datos y coordina llamadas al DAO.

Capa dl (Data Layer / DAO)

Se encarga de las operaciones CRUD hacia MySQL mediante JDBC.

Capa tl (Transfer Layer)

Representa las entidades que dan forma al modelo del dominio: Cliente, Mascota, Veterinario, Cita, etc.

Capa utils

Contiene validadores y funciones auxiliares reutilizables.

Patrón MVC + DAO

Modelo: entidades + DAOs

Vista: menú en consola

Controlador: GestorVeterinaria

La separación clara brinda mantenibilidad, modularidad y escalabilidad.

Aplicación de los conceptos en el proyecto

PetCare

Abstracción

La abstracción consiste en identificar los elementos importantes de la realidad y representarlos en el código.

En este proyecto, se abstraen los objetos del mundo real que forman parte de una clínica veterinaria: Cliente, Mascota y Cita.

Cada clase define únicamente los atributos y comportamientos esenciales, sin mostrar detalles innecesarios.

```
public class Mascota { 14 usages  ↗ Trey0594 *  
  
    // Aquí aplicamos ABSTRACCIÓN, ya que solo incluimos lo necesario  
    // (nombre, especie, edad, raza y dueño) sin detalles irrelevantes  
  
    // Atributos  
    private String nombre; 5 usages  
    private String especie; 4 usages  
    private String raza; 3 usages  
    private int edad; 3 usages  
    private Cliente duenio; 5 usages  
    private Cita cita; 3 usages
```

Encapsulamiento

El encapsulamiento protege los datos de las clases para evitar modificaciones directas desde fuera.

En este proyecto, todos los atributos son privados, y se accede a ellos mediante métodos públicos getters y setters.

```
// ENCAPSULAMIENTO: los atributos son privados y se acceden mediante métodos públicos

public class Cliente { 16 usages  ↗ Trey0594

    // Atributos
    private String nombre; 5 usages
    private String apellidos;| 5 usages
    private String cedula; 4 usages
    private char sexo; 3 usages
    private String ubicacion; 5 usages

    // Constructores
    public Cliente(String nombre, String apellidos, String cedula, char sexo, String ubicacion) {
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.cedula = cedula;
        this.sexo = sexo;
        this.ubicacion = ubicacion;
    }

    public Cliente(String nombre, String apellidos, String ubicacion) { no usages  ↗ Trey0594
        this.nombre = nombre;
        this.apellidos = apellidos;
        this.ubicacion = ubicacion;
    }
```

Dependencia

La dependencia se da cuando una clase usa otra temporalmente, sin tenerla como atributo fijo.

En este caso, la clase Cita depende del veterinario (solo utiliza su nombre para registrar la cita, pero no crea un objeto “Veterinario”).

También la clase Main depende de las tres clases, ya que las instancia y las usa para ejecutar el programa.

```
// DEPENDENCIA: la clase Cita usa un dato de tipo String para el veterinario.  
// No existe una clase Veterinario, solo se usa su información temporalmente.  
  
public class Cita { 16 usages  ↗ Trey0594  
  
    // Atributos  
    private String fecha; 4 usages  
    private String motivo; 4 usages  
    private String veterinario; 4 usages
```

Agregación

a agregación representa una relación “tiene un”, donde un objeto contiene otro, pero ambos pueden existir por separado.

Aquí, un Cliente puede tener una o más Mascotas, y si el cliente se elimina, las mascotas pueden seguir existiendo.

```
// AGREGACIÓN: una mascota pertenece a un cliente, pero puede existir por sí sola.  
private Cliente dueño; // Mascota tiene una referencia a Cliente 5 usages  
  
// Atributos  
private String nombre; 5 usages  
private String especie; 4 usages  
private String raza; 3 usages  
private int edad; 3 usages  
private Cita cita; 3 usages
```

Composición

La composición es una relación más fuerte que la agregación.

Si un objeto se elimina, los objetos que dependen de él también desaparecen.

En este proyecto, una Mascota tiene una Cita, y si la mascota se elimina la cita deja de tener sentido.

```
// COMPOSICIÓN: la cita existe dentro de la mascota.  
// Si la mascota se elimina, su cita también desaparece.  
  
public class Cita { 16 usages 👤 Trey0594  
    // información  
    public void mostrarInfoCita() { 1 usage 👤 Trey0594  
        System.out.println("Fecha: " + fecha);  
        System.out.println("Motivo: " + motivo);  
        System.out.println("Veterinario: " + veterinario);  
    }  
}
```

Persistencia de datos

El sistema utiliza MySQL como gestor de base de datos, aplicando JDBC y el patrón DAO para interactuar con las tablas.

El archivo `veterinaria_db.sql` contiene la estructura completa de la base de datos, incluyendo llaves primarias y foráneas, cumpliendo los requisitos de persistencia del proyecto.



```
C:\Users\trei0\.jdks\openjdk-24.0.2+12-54\bin\java.exe "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.2.1\lib\idea_rt.jar" -Dfile.encoding=UTF-8
✓ Conexión exitosa a veterinaria_db con usuario trey
Process finished with exit code 0
```

Reflexión personal

El desarrollo de este proyecto representó un proceso de aprendizaje significativo en términos de diseño, programación y arquitectura de software.

Fue necesario comprender a profundidad conceptos como abstracción, interfaces, composición y patrones de diseño, aplicándolos de manera práctica en un sistema real.

Asimismo, trabajé con persistencia en MySQL y gestioné el proyecto mediante un repositorio GitHub, fortaleciendo habilidades esenciales para el desarrollo profesional.

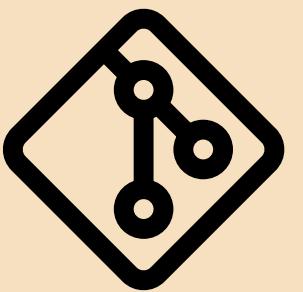
La experiencia permitió visualizar la importancia de la POO para construir sistemas modulares, mantenibles y escalables.

El portafolio refleja la comprensión integral de los principios de POO y su aplicación en el ciclo completo de desarrollo: análisis, diseño, implementación, pruebas, documentación y despliegue. El sistema PetCare constituye una solución robusta y extensible que ejemplifica buenas prácticas de arquitectura y programación.

Si desea ver el código completo del proyecto, puede hacer clic en el ícono de GitHub ➔ para acceder directamente a mi repositorio.

Ha sido una experiencia muy valiosa desarrollar este proyecto y aplicar todo lo aprendido sobre Programación Orientada a Objetos.

¡Gracias por su guía y por todo lo enseñado! 🐾



MUCHAS

gracias