Portafolio de Programación Orientada a Objetos

Primer Portafolio

Integrantes:

Amanda Padilla Leiva

Institución Educativa:

Universidad Cenfotec

Curso:

Programación Orientada a objetos

Profesor: Álvaro Cordero

Fecha de entrega: 10 de Octubre de 2025

Tabla de Contenidos

Portada..... 1

Tabla de Contenidos..... 2

Justificación..... 3

Conceptos Aplicados en el Sistema..... 4

Estructura del Código..... 9

Principios de Programación Orientada a Objetos..... 10

Diagrama de Flujo..... 14

Resultado del Código..... 15

Reflexión Personal..... 16

Justificación

El problema seleccionado es el diseño e implementación de un sistema tipo biblioteca de películas o “Tienda de alquiler de películas”, pensado para la gestión, organización y registro de películas. La razón de esta elección surge de la necesidad común de administrar colecciones de películas de forma organizada, la cual permite que se puedan clasificar por género, director, actores, duración, si son apropiadas o no para todo público. Este tipo de sistema ha estado presente en varios contextos a lo largo de la historia. En el pasado , se empleaba de manera extensa en establecimientos que alquilaban películas en VHS, DVDs o Blue-Ray, donde la información de cada una debía registrarse en una computadora, o incluso, en registros físicos dentro de cada local. En la actualidad, ese concepto se mantiene tanto en plataformas de streaming que tienen catálogos de series y películas organizados como en las plataformas sociales como Letterbox, donde los usuarios pueden calificar, clasificar y hacer reseñas sobre las películas según sus gustos.

Como este sistema incluye una variedad de entidades interconectadas que pueden ser modeladas de forma natural como clases y objetos, el contexto es perfecto para implementar la Programación Orientada a Objetos (POO). La POO permite una representación clara y ordenada de elementos como *Película, Director, Género, Sinopsis* y *Biblioteca,* así como las relaciones que existen entre ellos. Asimismo, posibilita una solución escalable y adaptable, que en el futuro, podría incorporar nuevas funcionalidades, como recomendaciones personalizadas o hasta la combinación con una base de datos.

Conceptos aplicados en el sistema.

En esta sección voy a dar una breve descripción y explicación de conceptos aprendidos en clase y que aprendí durante el proceso de hacer el portafolio. Además de métodos, que no conocía o comprendía del todo al inicio y fui aprendiendo conforme se diseñaba el sistema.

Abstracción: Es la capacidad de un lenguaje de programación para representar y manejar conceptos esenciales, ignorando los detalles innecesarios. En POO se logra mediante clases y herencia, permitiendo que una clase general (por ejemplo, Película) sea extendida por otras más específicas (Actor, Director), enfocándose solo en las características relevantes.

Encapsulamiento: Consiste en agrupar atributos y métodos dentro de una misma clase, protegiendo los datos del acceso externo no autorizado. Se controla mediante modificadores de acceso como public, private y protected, que determinan qué partes del código pueden acceder o modificar los atributos y métodos.

Modularidad: Es el principio de dividir un sistema o programa en módulos más pequeños y manejables, donde cada módulo cumple una función específica e independiente dentro del conjunto. Esto facilita la comprensión, el mantenimiento, la reutilización del código y la detección de errores, ya que cada parte del sistema puede desarrollarse y probarse de forma separada.

Asociación: Representa una relación general entre dos clases u objetos, donde ninguno depende del otro para existir. Ejemplo: un Profesor puede estar asociado con varios Estudiantes, pero ambos pueden existir por separado.

Agregación: Es un tipo especial de asociación en la que una clase contiene o agrupa a otra, pero ambas pueden existir de forma independiente. Ejemplo: una Biblioteca contiene Libros, pero los Libros pueden existir sin la Biblioteca.

Composición: Es una forma más fuerte de agregación en la que un objeto depende completamente de otro. Si el objeto contenedor se destruye, también lo hacen sus componentes. Ejemplo: una Casa está compuesta por Habitaciones; sin la Casa, las Habitaciones no existen.

Dependencia: Ocurre cuando una clase utiliza temporalmente a otra para realizar una tarea específica. Es una relación débil, donde un cambio en la clase utilizada puede afectar a la que depende. Ejemplo: una clase Factura depende de Cliente para obtener los datos al generarse.

Métodos de Comportamiento: En Programación Orientada a Objetos (POO), los métodos de comportamiento son básicamente las funciones o acciones que un objeto puede realizar. Cada clase define atributos (lo que el objeto “tiene”) y métodos (lo que el objeto “puede hacer”). Los métodos de comportamiento determinan cómo actúa un objeto y cómo interactúa con otros objetos o con sus propios datos internos.

Métodos de Utilitarios: Los métodos utilitarios son funciones que realizan tareas de apoyo o complementarias, generalmente no modifican directamente el estado principal del objeto, sino que ayudan a facilitar cálculos, conversiones, validaciones o manipulaciones de datos dentro del sistema.

Método de Auxiliar: En Programación Orientada a Objetos (POO), un método auxiliar es un método que se crea para ayudar a otro método principal a realizar su tarea. No suele ser llamado directamente desde fuera de la clase, sino que apoya la ejecución de otro método, dividiendo la lógica en partes más pequeñas y manejables y ayuda a mejorar la legibilidad, modularidad y mantenimiento del código.

Override: Cuando una clase hereda a otra directa o indirectamente de Object, lo que puede cambiar el comportamiento de un método heredado.

Clases

Clase Película (Movie)

- id: String
- título: String
- año: int
- duraciónMinutos: int
- clasificación: ContentRating (enum: G, PG, PG13, R, NC17)
- géneros: Set<Género>
- director: Director
- Ficha: String

Clase Actor

- id: String
- nombre: String
- filmografía: Set<Película>

Clase Director

- id: String
- nombre: String
- películasDirigidas: Set<Película>

Clase Género (Genre)

- nombre: String (ejemplo: Acción, Drama, Comedia, Terror, etc.)

Clase Catálogo (Catálogo)

- películas: Map<String, Película>
- actores: Map<String, Actor>
- directores: Map<String, Director>

Clase User (Usuario)

- id: String
- username: String
- email: String
- favoritos: Set<Película>
- calificaciones: Map<String, Integer> (idPelícula → nota 1 a 5)

Clase Admin (hereda de User)

- id: String
- username: String
- email: String
- privilegios: List<String> (ejemplo: agregar, eliminar, editar películas)

Estructura del Código

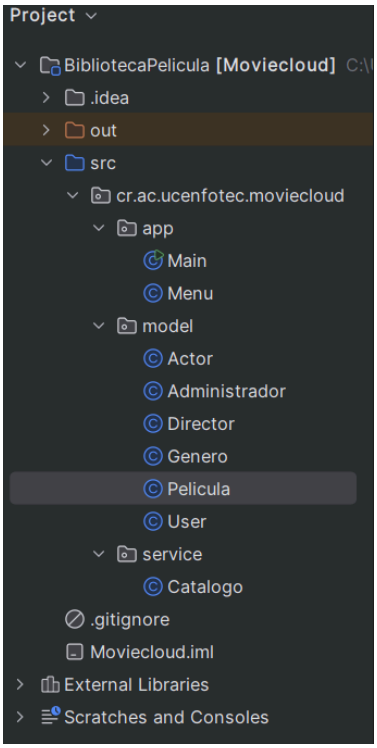
Package: Un package (paquete) en Java es básicamente una carpeta lógica donde agrupas clases que están relacionadas. En el código se utiliza de esta forma:

Paquete model: Aquí guardás las clases que representan cosas (entidades): Película, Actor, Director, Géneros (enum), User, Administrador

Paquete service: Aquí va la lógica del sistema: Catálogo (maneja las películas, búsqueda, filtros, etc.)

Paquete app (o mail): Aquí va el punto de entrada del programa: Main, Menu.

Estructura del proyecto: Se eligió esta estructura para los packages con el objetivo de lograr una organización modular y ordenada del proyecto, permitiendo separar las responsabilidades de cada parte del sistema. la lectura del código, su mantenimiento y la posibilidad de ampliarlo sin afectar otras secciones del programa.



Principios de Programación Orientada a Objetos aplicados en el portafolio

Abstracción: La abstracción se aplicó al definir clases que abstraen sus propias responsabilidades que representan entidades del mundo real, ocultando detalles innecesarios y solo mostrando lo esencial.

```
public class Pelicula {  40 usages
    private String id;  4 usages
    private String titulo;  6 usages
    private int anio;  6 usages
    private int duracionMinutos;  6 usages
    private String clasificacion;  6 usages
    private Set<Genero> generos;  4 usages
    private Set<Actor> elenco;  4 usages
    private Director director;  4 usages
```

Encapsulamiento: El encapsulamiento se refleja en el uso de atributos privados y métodos públicos para proteger la información de las clases.

```
private String id;  4 usages
private String titulo;  6 usages
```

Modularidad: La modularidad se logra dividiendo el programa en paquetes y clases especializadas, donde cada una cumple una función específica.

```
import cr.ac.ucenfotec.moviecloud.model.Administrador
import cr.ac.ucenfotec.moviecloud.model.Director;
import cr.ac.ucenfotec.moviecloud.model.Genero;
import cr.ac.ucenfotec.moviecloud.model.Pelicula;
import cr.ac.ucenfotec.moviecloud.model.User;
```

Dependencia: La dependencia se aplica cuando la clase usa temporalmente a otra para realizar una tarea, por ejemplo en el CLUB del main, en el método actualizarPelicula.

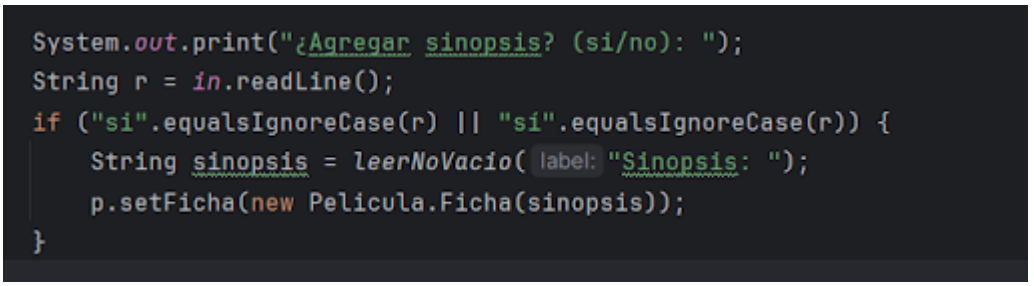
```
static void actualizarPelicula() throws IOException {  1 usage
    if (totalPeliculas == 0) {
        System.out.println("No hay películas.");
        return;
    }
    String id = leerIdExistentePelicula( label: "ID a actualizar: ");
    int idx = indicePelicula(id);
    Pelicula p = peliculas[idx];

    String nuevoTitulo = leerConDefault( label: "Nuevo título (enter = mantener): ", p.getTitulo());
    Integer nuevoAnio = leerEnteroConDefault( label: "Nuevo año (enter = mantener): ", p.getAnio(),  min: 1888,  max: 3000);
    Integer nuevaDur = leerEnteroConDefault( label: "Nueva duración (min, enter = mantener): ",
            p.getDuracionMinutos(),  min: 1,  max: 10000);

    System.out.print("Nueva clasificación (enter = mantener): ");
    String s = null;
    try { s = in.readLine(); } catch (IOException ignored) {}
    String nuevaClas = (s == null || s.trim().isEmpty()) ? p.getClasificacion() : validarClasificacion(s.trim());

    p.setTitulo(nuevoTitulo);
    p.setAnio(nuevoAnio);
    p.setDuracionMinutos(nuevaDur);
    p.setClasificacion(nuevaClas);
```

Asociación: La asociación está presente en la relación entre clases y película, una película tiene un director, pero ambos pueden existir de forma independiente.

Agregación: La agregación se da cuando la clase agrupa a otras sin que dependan completamente de ella, esto ocurre en los géneros.

```
private Set<Genero> generos;  4 usages
```

Composición: La composición aparece en la película con la clase interna de ficha, que almacena la sinopsis, sin la película, la ficha desaparece.



```
System.out.print("¿Agregar sinopsis? (si/no): ");
String r = in.readLine();
if ("si".equalsIgnoreCase(r) || "si".equalsIgnoreCase(r)) {
    String sinopsis = leerNoVacio( label: "Sinopsis: ");
    p.setFicha(new Pelicula.Ficha(sinopsis));
}
```

CRUD: El término C.R.U.D hace referencia a las cuatro operaciones básicas que se pueden realizar sobre los datos en un sistema: Create (Crear), Read (Leer), Update (Actualizar) y Delete (Eliminar). Estas operaciones representan la base del manejo de información en la mayoría de los sistemas informáticos. En este portafolio, el concepto de CRUD se aplica dentro del sistema tipo Tienda de Películas, ya que permite gestionar la información de las distintas entidades del programa, como Películas, Usuarios, Actores y Directores. Gracias a estas operaciones, el sistema puede mantener un catálogo dinámico y actualizado, simulando cómo funcionaría una aplicación real de gestión o alquiler de películas.

- CREATE (Crear):

Se utiliza para agregar nuevos elementos al sistema, como registrar una nueva película, actor o usuario dentro del catálogo.

- READ (Leer):

Permite consultar y visualizar la información almacenada, por ejemplo, mostrar la lista completa de películas o los detalles de un director.

UPDATE (Actualizar):

Se aplica cuando se requiere modificar información existente, como actualizar los datos de una película o editar los privilegios de un usuario administrador.

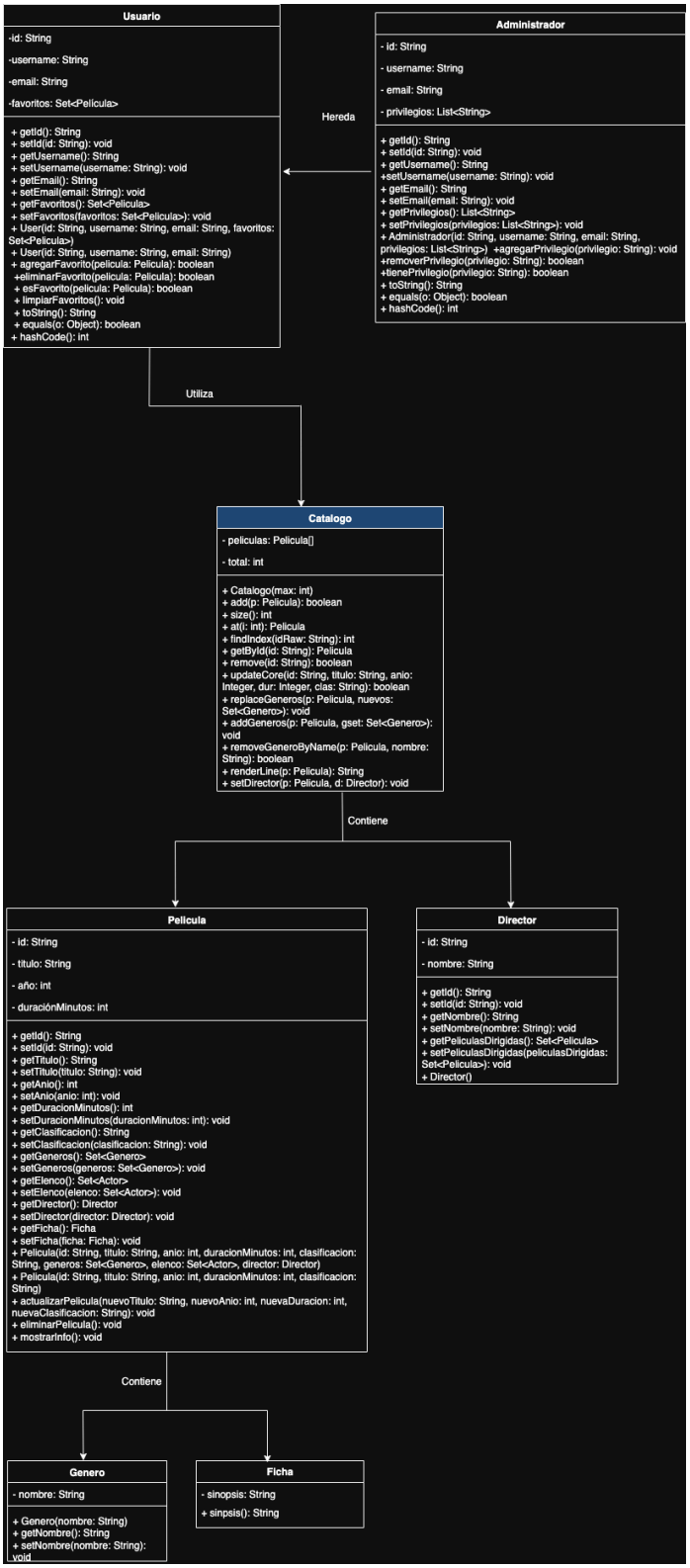
- DELETE (Eliminar):

Se usa para borrar registros que ya no son necesarios, como eliminar una película del catálogo o un usuario del sistema.

El uso del modelo CRUD fue escogido porque permite representar de forma clara y eficiente la gestión de datos dentro del sistema. Este modelo fue previamente investigado al inicio del desarrollo del portafolio, con el objetivo de aplicar una estructura sólida para el código. El enfoque CRUD facilita mantener un control completo sobre las entidades del sistema y garantiza que la información pueda ser creada, consultada, modificada y eliminada según las necesidades del sistema.

La implementación del CRUD se realizó dentro de la clase Catalogo, ubicada en el package de servicio, ya que actúa como núcleo funcional del sistema. De esta forma, todas las operaciones de gestión se centralizan en un solo punto, manteniendo un orden lógico y una estructura coherente. Es un modelo que resultó ideal para este proyecto, ya que trata de un sistema de gestión de películas que requiere precisamente las cuatro operaciones fundamentales que el CRUD ofrece.

Diagrama de Flujo



Resultado del Código

Los resultados del código cumplieron su propósito, logrando implementar una biblioteca de películas utilizando Programación Orientada a Objetos (POO), métodos y el modelo CRUD. El sistema cuenta con cuatro tipos de menús: uno para registrar a un administrador o un usuario normal, otro para iniciar sesión, un menú de administrador y un menú de usuario normal. En el menú de administrador se hace uso del CRUD, permitiendo registrar, listar, actualizar y eliminar películas. Cada película contiene información como nombre, género y clasificación, manteniendo un catálogo organizado y completo. El menú de usuario normal permite listar las películas disponibles, agregarlas a favoritos y consultar su lista de favoritos, siendo un código funcional y sencillo.

Aunque el sistema es efectivo y cumple con los objetivos planteados, presenta oportunidades de mejora, como la posibilidad de buscar películas por director y listar todas las películas de un mismo director, lo que podría hacerlo más completo y práctico o agregar actores y listar películas por su actor. En resumen, el sistema logró implementar de manera clara y organizada los conceptos de POO y CRUD, diferenciando las funcionalidades según el tipo de usuario y ofreciendo una base sólida para futuras mejoras.

Reflexión Personal

Durante este portafolio logré comprender de manera más detallada el concepto de la programación orientada a objetos y a cómo estructurar programas complejos de forma ordenada. Al inicio tuve dificultades para entender conceptos como la dependencia y la composición, en el caso de esta última, tuve que agregar un fragmento de código para poder interpretarla correctamente dentro del sistema. A medida que fui modelando el sistema, pude identificar cada tipo de relación y concepto, así como el propósito que cumplen dentro del programa. También aprendí a organizar el código en paquetes y a pensar en la reutilización, algo que al inicio no comprendía completamente.

Al inicio, algunos fragmentos de código eran necesarios, pero no entendía completamente su funcionamiento. Con la práctica, aprendí a implementarlos correctamente y a aplicar el concepto de CRUD y sus principios, lo que me fue de gran ayuda para desarrollar el sistema de películas y el catálogo.

Este sistema resultó interesante de implementar, ya que tenía muchos atributos o “variantes” que podían volverse complicadas al analizar la problemática. Por ejemplo, la gestión del elenco o de los actores era una clase que, si se agregaba directamente, podía complicar demasiado el código; sin embargo, es algo que se podría incorporar en futuras mejoras. Además, al final agregué otra clase en el paquete de la aplicación: una clase de menú, donde centralice el código de los menús para evitar que el método main se volviera demasiado extenso. Esta es otra mejora que podría optimizarse en el futuro, buscando formas de reducir la longitud y complejidad del main.

En conclusión, este proyecto fue un proceso de aprendizaje profundo en el que comprendí mejor la programación orientada a objetos y la organización de sistemas complejos. A través de la práctica, pude implementar correctamente fragmentos de código, aplicar los principios de CRUD y estructurar el sistema de películas y el catálogo de manera ordenada. También aprendí a manejar la complejidad de los atributos y relaciones entre

clases, así como a mejorar la organización del código para que sea más limpio y reutilizable. Aunque el proceso fue desafiante, me permitió consolidar conocimientos fundamentales y sentar bases sólidas para futuras mejoras en el desarrollo de software.