Informe de Laboratorio

Marcelo Fort Muñoz

19 de diciembre de 2024

Resumen

Este es un informe de laboratorio.

Índice general

Capítulo 1

Introducción

1.1. Descripción del proyecto

En este proyecto se ha desarrollado un front-end para un quiosco de venta de billetes de tren para la empresa *TrainGo*. El quiosco contempla cambio en los datos del usuario, búsqueda de trenes, compra de billetes y visualización de billetes comprados. Además, se puede cambiar el idioma de la interfaz.

Aparte de esto, se ha desarrollado un sistema de persistencia de datos basado en el patrón DAO (*Data Access Object*) que se sirve de archivos XML para almacenar los datos.

1.2. Objetivos

1. Desarrollar una aplicación de escritorio en Java.
2. Usar Herencia y Polimorfismo.
3. Usar Interfaces y Clases Abstractas.
4. Usar Excepciones.
5. Usar anotaciones.
6. Usar patrones de diseño.
7. Implementar una interfaz gráfica (GUI).

1.3. Motivaciones y alcance

El proyecto se ha desarrollado con el objetivo de aprender a usar manualmente java swing y poner en práctica patrones de diseño y buenas prácticas de programación.

El alcance del proyecto es limitado, ya que no solo se procesa la interacción por parte del usuario, dejando de lado la interacción con el resto de la infraestructura de la empresa.

Por otro lado, se ha implementado un sistema de persistencia de datos basado en archivos XML.

Capítulo 2

Diseño del programa

2.1. Prototipado de la interfaz gráfica

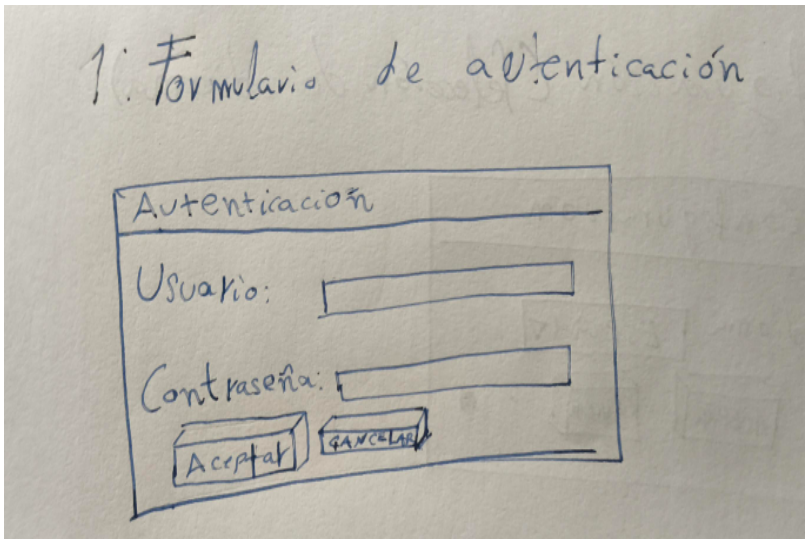

Figura 2.1: Pantalla 1: Formulario de autenticación

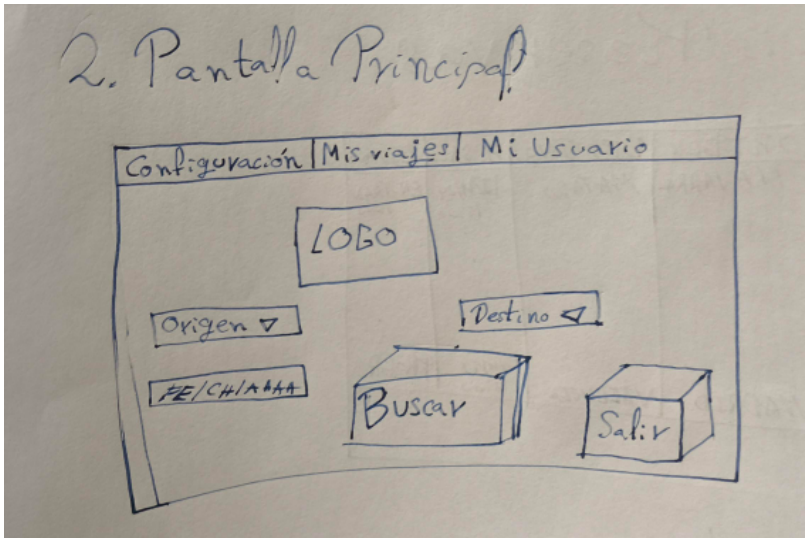

Figura 2.2: Pantalla 2: Pantalla principal

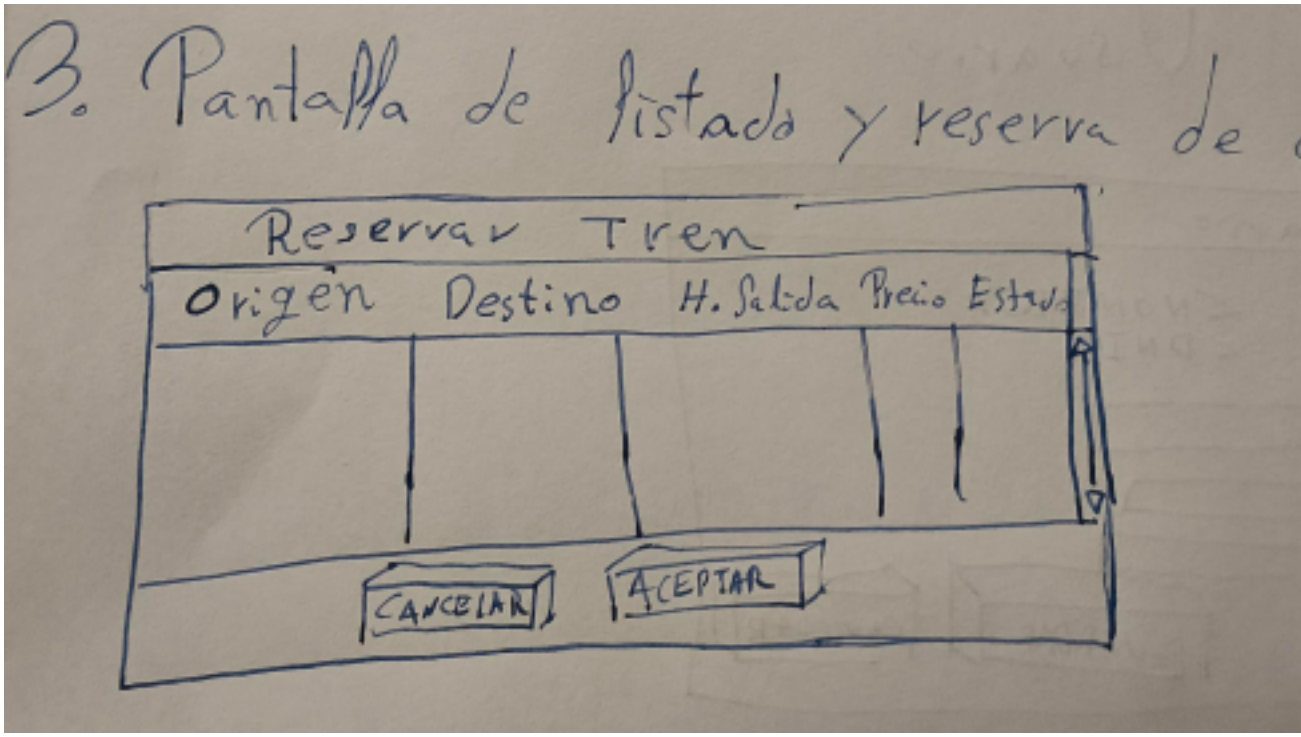

Figura 2.3: Pantalla 3: Lista de trenes y reserva de billetes

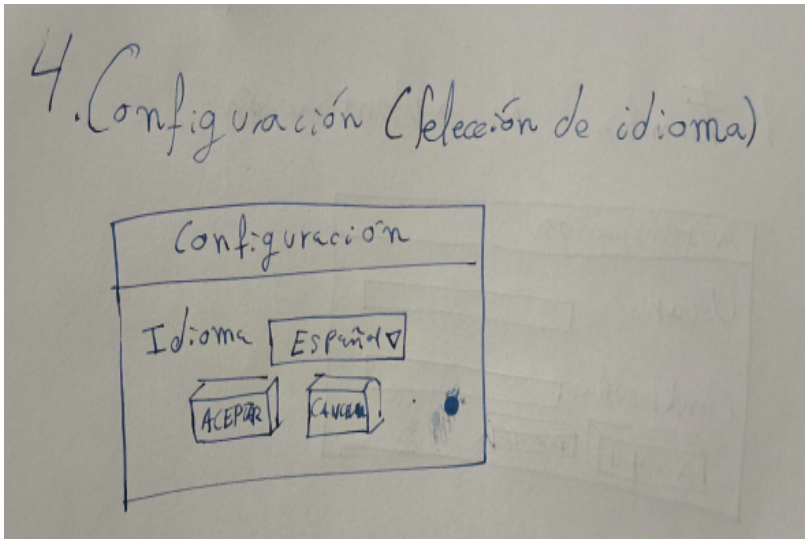

Figura 2.4: Pantalla 4: Configuración

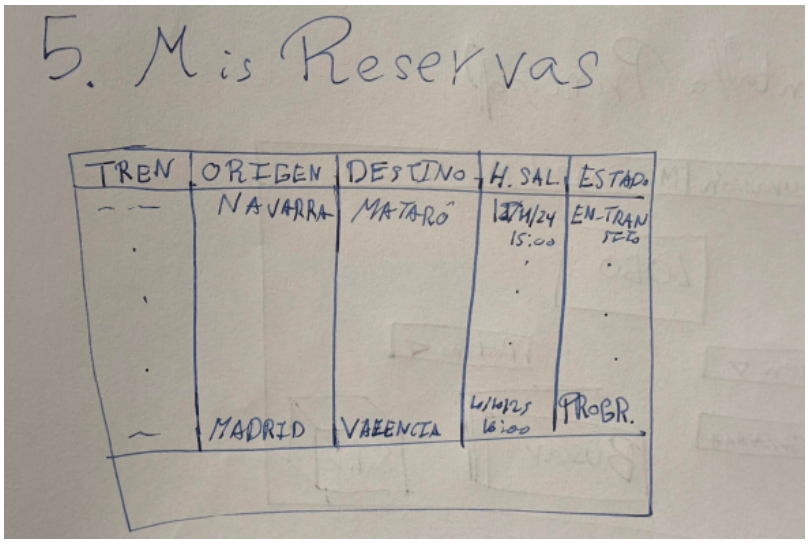

Figura 2.5: Pantalla 5: Mis Reservas

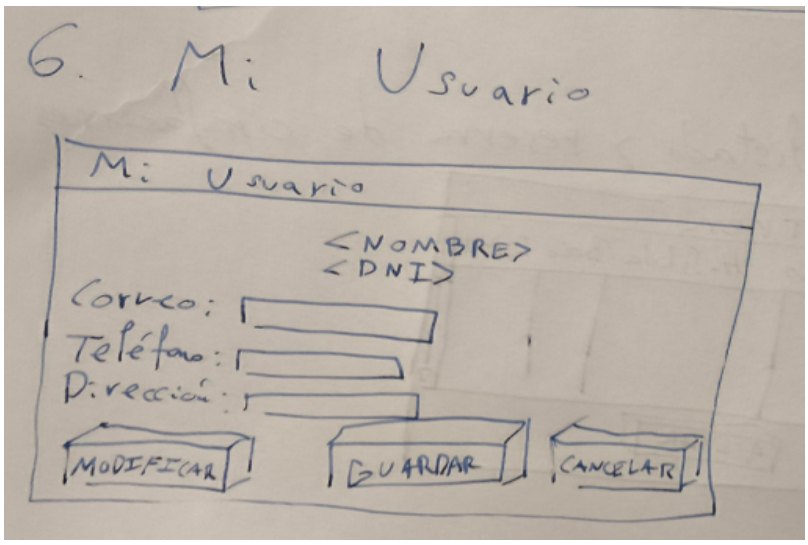

Figura 2.6: Pantalla 6: Mi usuario

2.2. Diagramas UML

2.2.1. Diagrama de clases

Para comenzar, en la Figura 2.7, se muestran las clases que representan a la información que se maneja en la aplicación.

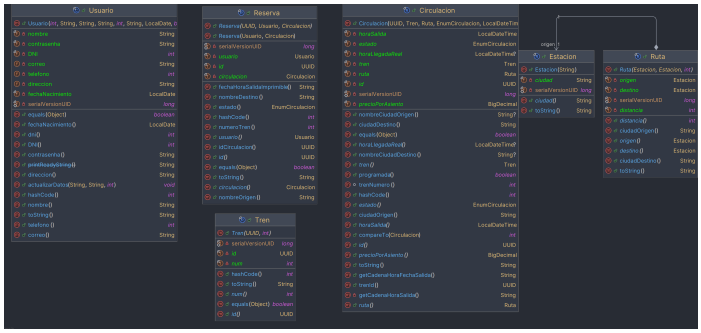

Figura 2.7: Modelo de datos

2.2.2. Diagrama de clases DAO

Relacionado con el modelo de datos, para implementar una persistencia o «serialización» de los datos, se ha implementado un patrón DAO (*Data Access Object*). El patrón DAO se encarga de abstraer la lógica de acceso a los datos (ver Sección 2.3). En la Figura 2.8 se muestra el diagrama de clases de este patrón.

En la implementación de este patrón, se han usado varias funcionalidades clave de Java. Por un lado, la clase *AbstractDAO* es una clase abstracta que implementa la interfaz *IDAO*. La clase AbstractDAO a su vez tiene una relación de herencia con los DAOs individuales.

2.2.3. Diagrama de clases GUI

Llegados a este punto, se ha implementado la interfaz gráfica de usuario (GUI). Se ha usado una estructura formulario-modelo para las clases GUI con tablas y elementos con información. En la Figura 2.9 se muestra el diagrama de clases de la GUI.

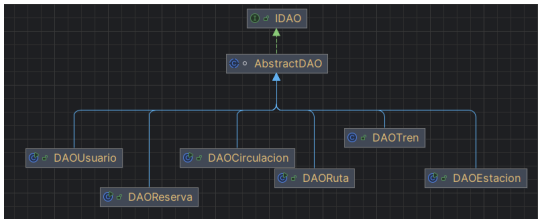

Figura 2.8: Diseño de la estructura de clases del DAO

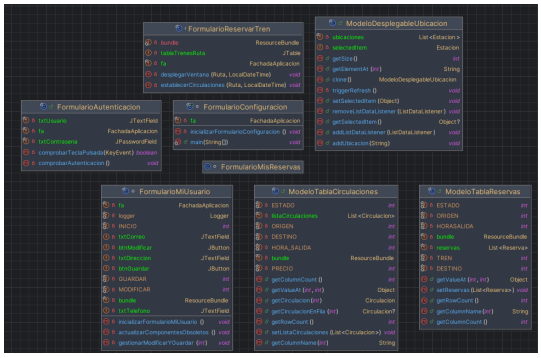

Figura 2.9: Diagrama de clases para la interfaz gráfica

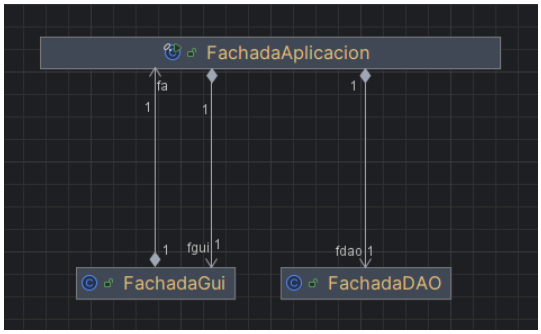

Figura 2.10: Jerarquia de clases para el patrón façade

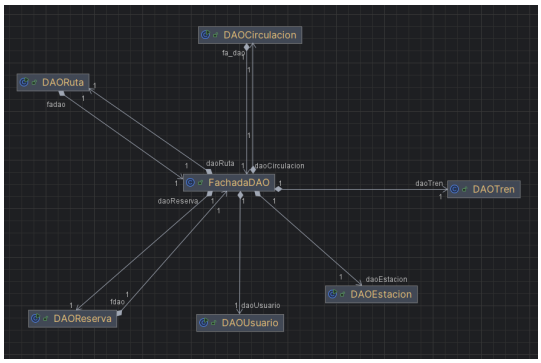

Figura 2.11: Relaciones entre la façada de los DAOs y los mismos DAOs

2.2.4. Façade

Para simplificar la interacción entre la GUI y el modelo de datos, se ha implementado un patrón Façade. No solo simplifica la interacción, sino que también permite una mayor flexibilidad en la implementación de la GUI respecto al modelo de datos. En la Figura 2.10 se muestra el diagrama de clases del Façade respecto a su jerarquía interna.

Para poner un ejemplo, se puede ver como interactúan los DAOs con el Façade en la Figura 2.11.

Estos son todos los diagramas UML más relevantes. Se hablará más en detalle de la implementación en la Capítulo 3. Respecto a la estructura del proyecto, se detalla a continuación en la Sección 2.3.

2.3. Estructura del proyecto

Para este apartado, seguiremos una lógica de aproximación top-down. Analizaremos primero la estructura general del proyecto, para luego ir descendiendo en detalle.

De esta forma iremos viendo cómo se ha estructurado el proyecto, desde la raíz hasta las clases más concretas. A continuación, se muestra la estructura de directorios del proyecto.

```
TrainGo
├── datos
└── src
```

Dentro del directorio *datos*, habremos de hallar los archivos de datos que se usan en la aplicación. Estos están escritos en un formato XML.

En el directorio *src*, se encuentran los archivos fuente del proyecto, donde se encuentra la implementación de la aplicación.

El código fuente de esta aplicación sigue una modificación **patrón Modelo-Vista-Controlador** (MVC). En este caso, se ha modificado el patrón para que se ajuste a las necesidades de la aplicación. La estructura de directorios de la aplicación se muestra en la Sección 2.3.

```
src
└── main
    ├── java
    │   └── com
    │       └── marceSoft
    │           └── trainGo
    │               ├── aplicacion
    │               ├── dao
    │               ├── gui
    │               └── util
    └── resources
        └── images
```

En el directorio *aplicacion*, se encuentran las clases que implementan la lógica de la aplicación y los modelos de datos. (es decir, modelo y controlador).

En el directorio *dao*, se encuentran las clases que implementan el patrón DAO. Es importante separar la lógica de acceso a los datos de la lógica de la aplicación, ya que al hacerlo, se facilita la implementación de la persistencia de los datos.

En el directorio *gui*, se encuentran las clases que implementan la interfaz gráfica de usuario. Estas clases se encargan de mostrar la información al

usuario y de recibir la información del usuario (vista).

En el directorio *util*, se encuentran las clases que implementan utilidades para la aplicación.

En el directorio *resources*, hallamos los *bundles* de internacionalización, archivo de configuración del *logger* y las imágenes que se usan en la aplicación. Sin embargo, es mejor no adelantarse, ya trataremos estos aspectos en el Capítulo 3.

2.3.1. Estructura del paquete *aplicacion*

Volviendo a la estructura de paquetes, en el paquete *aplicacion* se encuentran las clases que implementan parte de la lógica de la aplicación.

Por un lado, tenemos la *FachadaAplicacion*, que se encarga de abstraer la lógica de la aplicación. [1] Al mismo nivel, se encuentran las clases «contenedoras» de los modelos de datos.

Bajando un nivel nos podemos encontrar con la anotación *NoNegativo* y su validador, los dos enums, las excepciones y formateadores personalizados. Esto se ve muy bien en la Subsección 2.3.1.

```
aplicacion
├── anotaciones
│   ├── validadores
│   │   └── ValidadorNoNegativo.java
│   └── NoNegativo.java
├── enums
│   ├── EnumCirculacion.java
│   └── EnumIdioma.java
├── excepciones
│   ├── CargaArchivoFallidaException.java
│   ├── CierreArchivoXMLErroneoException.java
│   ├── LecturaSiguienteEventoException.java
│   ├── NoHayUsuariosRegistradosException.java
│   ├── ProcesadoSiguieteEventoException.java
│   ├── SituacionDeRutasInesperadaException.java
│   └── UsuarioNoEncontradoException.java
├── formato
│   └── FormattedFecha.java
├── FachadaAplicacion.java
├── Circulacion.java
├── Estacion.java
├── Reserva.java
├── Ruta.java
├── Tren.java
└── Usuario.java
```

De esta zona, lo más reseñable a nivel de diseño, es, tal vez, es la decisión de poner Usuario como clase contenedora modificable, mientras que el resto de clases son inmutables (*Records*). Esto se debe a que los usuarios son los únicos datos que son susceptibles a cambios.

[1] En la Figura 2.10 se muestra la jerarquía de clases de la Fachada.

Otra vez, se me puede (con cierta razón) acusar de adelantarme. Permitidme, entonces, que me disculpe y que os remita a la Capítulo 3 para más detalles. Sin embargo, considero que es una decisión de diseño importante, ya que aunque hay datos que pueden ser ≪renovados≫ (como las reservas), una reserva ≪modificada≫ es, en realidad, una reserva nueva (en nuestro caso). Por este motivo, se ha decidido que las reservas sean inmutables.

2.3.2. Estructura del paquete *dao*

En el paquete *dao*, se encuentran las clases que implementan el patrón DAO, que se encarga de abstraer la lógica de acceso a los datos (persistencia).

Este paquete está estructurado como se muestra en la Subsección 2.3.2.

```
dao
├── FachadaDAO.java
├── IDAO.java
├── AbstractDAO.java
├── DAOCirculacion.java
├── DAOEstacion.java
├── DAOReserva.java
├── DAORuta.java
├── DAOTren.java
└── DAOUsuario.java
```

En este paquete, se ha implementado un patrón DAO para abstraer la lógica de acceso a los datos. Este ha sido implementado siguiendo una versión simplificada de las directrices oficiales de ORACLE **??**, que también es usado por otros autores **??**.

Capítulo 3

Implementación

Apéndice A

Anexos