Proyecto Final: Sistema de Observación Meteorológica

Jorge Domínguez González (alu0101330600@ull.edu.es)
Adnan Hawari Capa (Alu0100417012@ull.edu.es)
Paula Regalado De León (Alu0101330174@ull.edu.es)

January 9, 2024

Contents

List of Figures

1 Introducción

1.1 Práctica anterior

En la anterior práctica se presenta una implementación de un sistema de observación meteorológica en Java. Este sistema utiliza el patrón de diseño Observador para recibir actualizaciones de las condiciones meteorológicas de diversas ciudades a través de la API de WeatherStack. WeatherStack es un servicio de API de pronóstico del tiempo que proporciona información actualizada sobre condiciones meteorológicas actuales, incluyendo temperatura, velocidad del viento, humedad, presión atmosférica y más.

1.2 Objetivo de la Práctica

El objetivo de la práctica es implementar un historial de los distintos datos recibidos, es decir, desarrollar una aplicación de visualización de datos meteorológicos utilizando Java y el framework JFreeChart. La aplicación permite seleccionar una ciudad y visualizar datos meteorológicos como temperatura, probabilidad de precipitación, velocidad del viento, presión atmosférica y cobertura de nubes. Además, el usuario puede elegir entre dos tipos de gráficos: líneas y barras.

2 Estructura del Directorio

En la figura 1 podemos observar la estructura de nuestro proyecto.

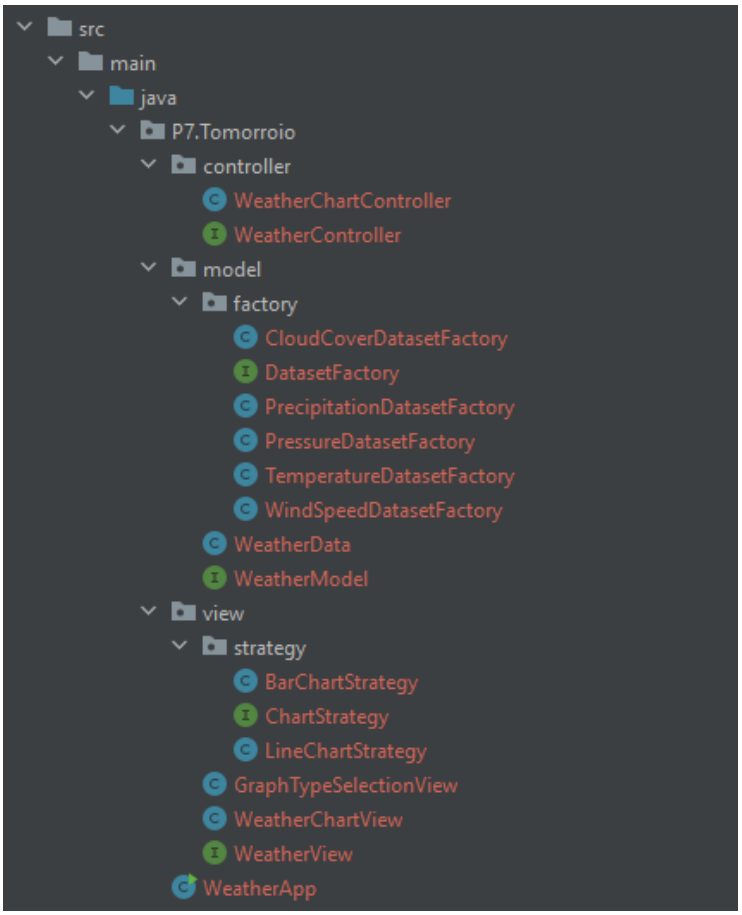


Figure 1: Estructura del directorio del proyecto.

3 Estructura del Código

3.1 Clase WeatherApp

Este archivo es el punto de entrada principal de la aplicación. Utiliza Swing para crear la interfaz de usuario y permite al usuario seleccionar la ciudad y el tipo de gráfico. Inicia la aplicación mediante la creación de instancias de las vistas y controladores necesarios.

3.2 Clase WeatherView

Interfaz que define el método updateChart para actualizar los gráficos.

3.2.1 Clase WeatherChartView

Implementa la interfaz WeatherView. Crea y muestra una ventana que contiene gráficos JFreeChart. Utiliza estrategias de gráficos (ChartStrategy) para determinar el tipo de gráfico (líneas o barras). Proporciona métodos para agregar paneles de gráficos y actualizar los datos de los gráficos.

3.3 Clase GraphTypeSelectionView

Crea una ventana Swing para que el usuario seleccione el tipo de gráfico y la ciudad.

3.4 Clase ChartStrategy

Interfaz que define el contrato para las estrategias de creación de gráficos.

3.4.1 Clase LineChartStrategy

Implementa la interfaz ChartStrategy. Proporciona una estrategia específica para crear un gráfico de líneas utilizando JFreeChart.

3.4.2 Clase BarChartStrategy

Implementa la interfaz ChartStrategy. Proporciona una estrategia específica para crear un gráfico de barras utilizando JFreeChart.

3.5 Clase WeatherData

Implementa la interfaz WeatherModel. Obtiene datos meteorológicos de la API de Tomorrow.io utilizando la biblioteca Unirest. Convierte y almacena los datos en series temporales (XYSeries) para su uso en los gráficos.

3.6 Clase DatasetFactory

La interfaz 'DatasetFactory' define el contrato para las fábricas de conjuntos de datos meteorológicos. A través de esta interfaz, especificamos los métodos necesarios que las factorías concretas deben implementar para la creación de conjuntos de datos.

3.6.1 Clase XDatasetFactory

La clase 'XDatasetFactory' implementa la interfaz 'DatasetFactory'. Es una factoría abstracta que proporciona métodos específicos para crear conjuntos de datos basados en el tipo de dato meteorológico. Esta factoría concreta se especializa en la creación de conjuntos de datos de tipo 'XYSeriesCollection'.

3.6.2 Clase TemperatureDatasetFactory

La clase 'TemperatureDatasetFactory' es una implementación concreta de 'DatasetFactory' especializada en la creación de conjuntos de datos para la temperatura. Hereda de 'XDatasetFactory' y proporciona la implementación específica para crear conjuntos de datos de temperatura.

3.6.3 Clase HumedityDatasetFactory

La clase 'HumedityDatasetFactory' es otra implementación concreta de 'DatasetFactory'. Al igual que las otras factorías, hereda de 'XDatasetFactory' y se centra en la creación de conjuntos de datos para la probabilidad de Humedad.

3.6.4 Clase WindSpeedDatasetFactory

La clase 'WindSpeedDatasetFactory' implementa 'DatasetFactory' y se especializa en la creación de conjuntos de datos para la velocidad del viento. También hereda de 'XDatasetFactory' y proporciona la implementación necesaria para este tipo de dato meteorológico.

3.6.5 Clase PressureDatasetFactory

La clase 'PressureDatasetFactory' es una implementación concreta para la creación de conjuntos de datos relacionados con la presión atmosférica. Al igual que las otras factorías, hereda de 'XDatasetFactory' y ofrece la implementación específica para datos de presión.

3.6.6 Clase CloudCoverDatasetFactory

La clase 'CloudCoverDatasetFactory' implementa 'DatasetFactory' y se encarga de la creación de conjuntos de datos para la cobertura de nubes. Hereda de 'XDatasetFactory' y proporciona la implementación necesaria para este tipo de dato meteorológico.

Estas clases concretas de factorías implementan métodos específicos para la creación de conjuntos de datos relacionados con sus respectivos tipos de datos meteorológicos. Al emplear el patrón Abstract Factory, logramos una estructura modular que facilita la extensión del sistema al agregar nuevos tipos de datos meteorológicos en el futuro.

3.7 Clase WeatherChartController

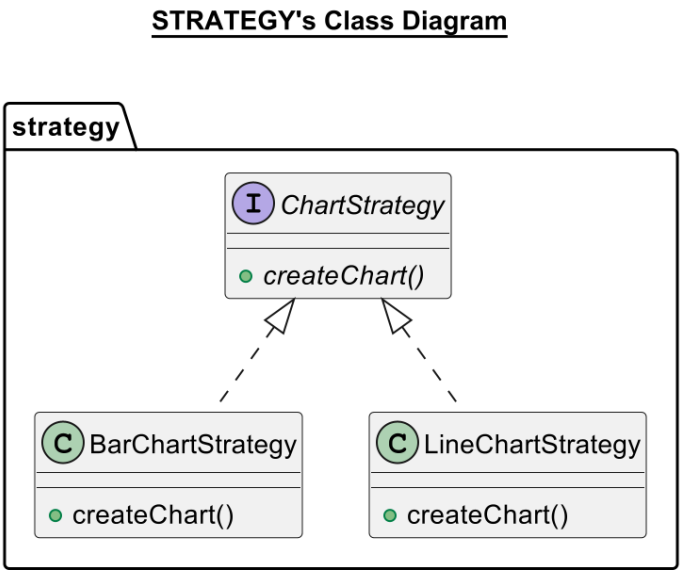
Implementa la interfaz WeatherController. Coordina la actualización de los gráficos mediante la creación de instancias de las fábricas de conjuntos de datos y la actualización de la vista. El proyecto utiliza de manera efectiva patrones de diseño como el patrón Estrategia para los diferentes tipos de gráficos y el patrón Fábrica para la creación de conjuntos de datos meteorológicos.

4 Diagrama UML y Patrones

En esta práctica usamos varios patrones que hemos visto a lo largo de la asignatura, como puede ser el patrón estrategia, el abstract factory y el modelo de vista controlador que iremos señalando en el siguiente apartado:

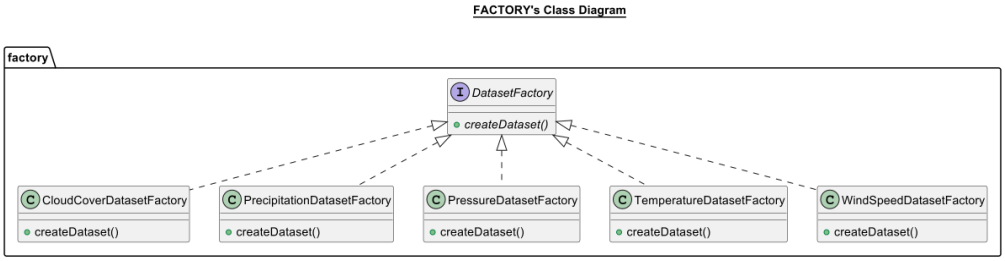
4.1 Patrón Estrategia

El patrón de diseño Estrategia se ha utilizado para la elección de gráficos en la aplicación debido a su capacidad para encapsular algoritmos intercambiables y permitir que el cliente seleccione dinámicamente uno de ellos. Como podemos observar en la image 2, el usuario puede seleccionar la gráfica de líneas o la gráfica de barras.



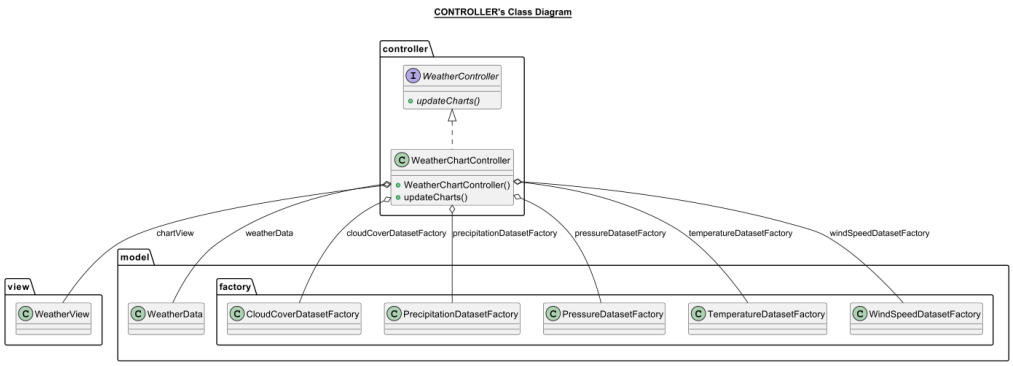
4.2 Patrón Abstract Factory

El patrón de diseño Abstract Factory se ha utilizado como se puede observar en la figura 4.2 para crear de forma dinámica los diferentes tipos de datos de forma ordenada y flexible.



4.3 Patrón Modelo Vista Controlador

El patrón de diseño Modelo Vista Controlador se ha utilizado para separar por capas los diferentes modelos de la aplicación.



5 Pasos del Flujo del Programa

El programa sigue estos pasos:

1. El usuario interactúa con la interfaz gráfica, seleccionando el tipo de gráfico y la ciudad en GraphTypeSelectionView.
2. GraphTypeSelectionView envía eventos al controlador (WeatherChartController).
3. WeatherChartController utiliza la información proporcionada por el usuario para obtener datos meteorológicos actualizados de WeatherData.
4. Se crean conjuntos de datos utilizando fábricas (DatasetFactory) para diferentes tipos de datos meteorológicos.
5. WeatherChartController actualiza la vista (WeatherChartView) con los nuevos conjuntos de datos.

6 Interfaz Gráfica para el Usuario

La interfaz gráfica incluye un formulario para que el usuario seleccione el tipo de gráficos como vemos en la figura 2 y la ciudad de interés como vemos en la figura 3 y una pantalla para mostrar el historial de temperatura, precipitación, viento, presión y nubes, en la figura 4 podemos ver un ejemplo para Tokio representado en líneas y en la figura 5 para Tokio representado en barras, donde se actualiza en tiempo real la información.

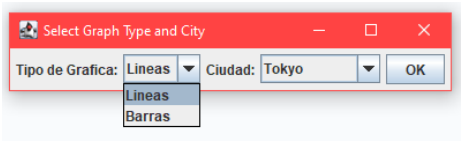


Figure 2: Interfaz gráfica para el usuario.

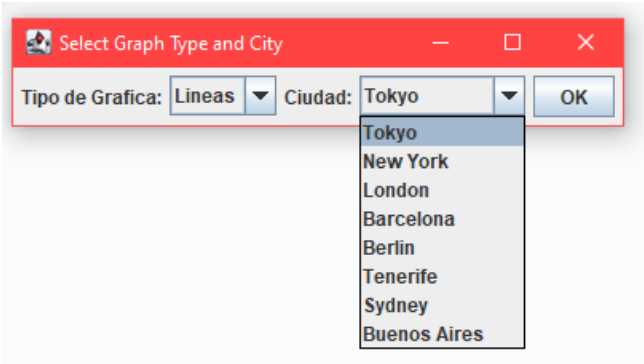


Figure 3: Interfaz gráfica para el usuario.

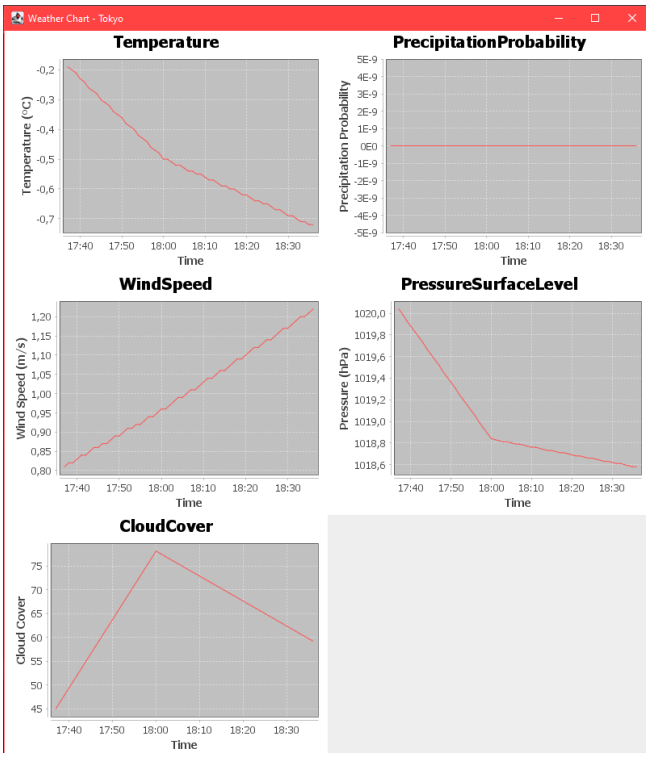


Figure 4: Interfaz gráfica para el usuario.

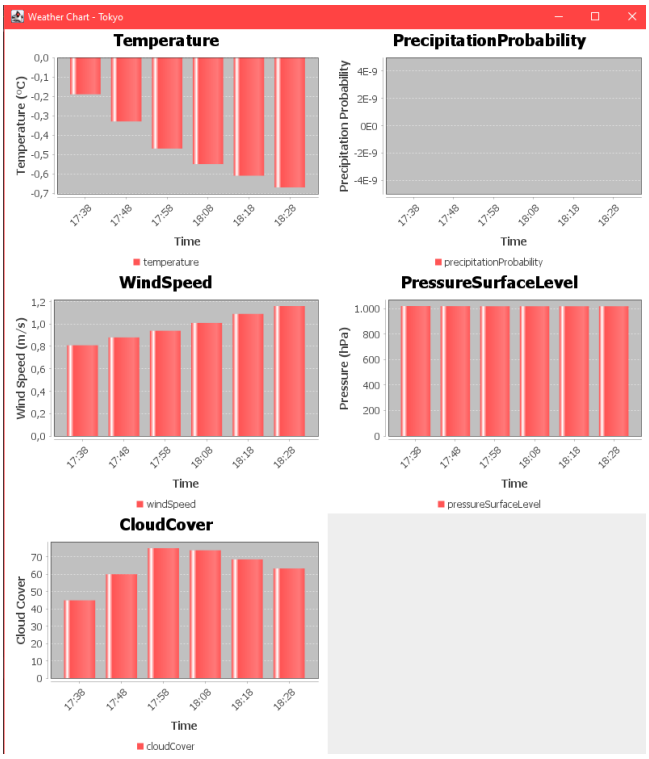


Figure 5: Interfaz gráfica para el usuario.

7 Diagrama UML

El diagrama UML del proyecto se muestra en la Figura 6.

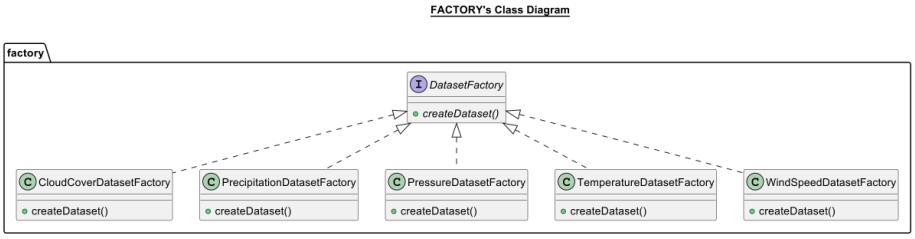


Figure 6: Diagrama UML del proyecto.

8 Errores y Desafíos Encontrados

Durante el desarrollo de este proyecto, nos enfrentamos a varios desafíos que requirieron soluciones creativas y resolución de problemas. A continuación, destacamos algunos de los errores y desafíos más significativos y cómo abordamos cada uno:

8.1 Conexión con la API de Tomorrow.io

Uno de los desafíos iniciales fue establecer una conexión efectiva con la API de Tomorrow.io para obtener datos meteorológicos en tiempo real. En algunos casos, experimentamos problemas de tiempo de espera y errores de conexión. Para superar esto, implementamos un manejo robusto de errores y optimizamos nuestras solicitudes para mejorar la eficiencia.

8.2 Sincronización de Datos en Tiempo Real

La sincronización de datos en tiempo real para proporcionar actualizaciones constantes en la interfaz gráfica fue un desafío importante. Implementamos un mecanismo de actualización basado en eventos para garantizar que los gráficos reflejaran con precisión los datos meteorológicos más recientes. Esto implicó coordinar eficientemente la obtención de datos, su procesamiento y la actualización de la interfaz de usuario.

8.3 Manejo de Excepciones

En el proceso de desarrollo, identificamos la necesidad de un manejo exhaustivo de excepciones para garantizar la estabilidad y la usabilidad de la aplicación. Implementamos una estrategia sólida de manejo de excepciones para notificar al usuario sobre posibles problemas y evitar fallos inesperados.

Estos desafíos contribuyeron significativamente a nuestro aprendizaje y resiliencia durante el desarrollo del proyecto. La capacidad para superar estos obstáculos mejoró nuestras habilidades de resolución de problemas y fortaleció la calidad general del sistema.

9 Estimación de Plazo

Table 1: Estimación de Plazo

Tarea	**Tiempo Estimado**	**Tiempo Real**
Elegir propuesta	30 minutos	15 minutos
Implementar JFreeChart	1 hora	1 hora y media
Obtener datos historial	30 minutos	45 minutos
Implementar patrones	1 hora	45 minutos
Implementar interfaces	2 horas	2 horas y media
Pruebas y depuración	2 horas	3 horas
Documentación	1 hora	1 hora y media
Implementación Github	1 hora y media	1 hora y media

10 Conclusiones

En conclusión, el uso de patrones de diseño ha simplificado el diseño y la implementación de la aplicación. Ha proporcionado una estructura modular y extensible, facilitando la incorporación de nuevas funcionalidades en algún futuro. Además de utilizar diferentes herramientas, como el JFreeChart, para realizar un proyecto bastante práctico y elaborado.

11 Bibliografía

References

[1] omorrow.io API: https://www.tomorrow.io/

[2] nirest: http://kong.github.io/unirest-java/

[3] FreeChart: https://www.jfree.org/jfreechart/

[4] FreeChart Developer Guide: https://www.jfree.org/jfreechart/api/javadoc/org/jfree/chart/JFreeChart.html

[5] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/index.htm

[6] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_quick_guide.htm

[7] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_overview.htm

[8] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_xy_chart.htm

[9] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_bar_chart.htm

[10] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_line_chart.htm

[11] FreeChart Tutorial: https://www.tutorialspoint.com/jfreechart/jfreechart_pie_chart.htm

[12] eatherStack API: https://weatherstack.com/

[13] FreeChart: https://sourceforge.net/projects/jfreechart/

[14] bservador (patrón de diseño): https://es.wikipedia.org/wiki/Observer_(patr%C3%B3n_de_dise%C3%B1o)

[15] strategia (patrón de diseño): https://es.wikipedia.org/wiki/Estrategia_(patr%C3%B3n_de_dise%C3%B1o)

[16] ábrica abstracta: https://es.wikipedia.org/wiki/F%C3%A1brica_abstracta

[17] odelo–vista–controlador: https://es.wikipedia.org/wiki/Modelo%E2%80%93vista%E2%80%93controlador

[18] atrones de diseño: https://es.wikipedia.org/wiki/Patr%C3%B3n_de_dise%C3%B1o_de_software

[19] atrones de diseño: https://www.tutorialspoint.com/design_pattern/index.htm

[20] atrones de diseño: https://www.geeksforgeeks.org/software-design-patterns/

[21] atrones de diseño: https://www.journaldev.com/1827/java-design-patterns-example-tutorial

[22] atrones de diseño: https://www.javatpoint.com/design-patterns-in-java