Informe Laboratorio N º3 Paradigma orientado a objetos

Nombre: Johan Neira Jans

Rut: 21163695-5

Profesor: Gonzalo Martínez

2.Indice

Contenido

3.Introducción

¿Alguna vez te has detenido a pensar cómo se diseñan y gestionan redes de metro modernas y qué papel juega la programación en su eficiencia? ¿Cómo puede Java, un lenguaje de programación tan robusto, ser la clave para resolver problemas complejos en estos sistemas de transporte? Para responder a estas preguntas, es esencial sumergirse en el paradigma orientado a objetos (POO). Este informe tiene como objetivo demostrar cómo el lenguaje de programación Java se aplica en la elaboración de una red de metro.

Se presentará el problema a tratar junto con su contexto, se describirá el paradigma orientado a objetos y su relación con el laboratorio, y se analizarán los desafíos que enfrenta este paradigma al crear redes de metro. Además, se detallarán los tipos abstractos de datos (TDAs) que conforman el laboratorio, el diseño de la solución y sus componentes, y los aspectos de implementación utilizados. También se proporcionarán instrucciones para el uso correcto de la aplicación, se presentarán los resultados obtenidos y se realizará una autoevaluación. Finalmente, se concluirá con las lecciones aprendidas en el laboratorio, destacando las fortalezas y debilidades del paradigma orientado a objetos.

3,1. Descripción del problema:

El metro es un sistema fundamental para el transporte urbano, que facilita la movilidad eficiente y accesible del público a través de la ciudad, a la vez que reduce la contaminación. El funcionamiento de la red de metro se basa en la gestión de diversas líneas, cada una con estaciones que sirven como puntos de entrada y salida para los pasajeros. Estas estaciones se sitúan en ubicaciones estratégicas, permitiendo a los usuarios acceder fácilmente a lugares de interés y hacer transbordos entre líneas. Una parte clave de la gestión del metro (Figura 3) es el control de los trenes que recorren estas líneas. Cada tren necesita un conductor responsable de transportar a los pasajeros de manera segura y eficiente. También es necesario hacer un seguimiento del flujo de pasajeros para garantizar que el sistema funcione de manera óptima, por lo tanto, el objetivo de este laboratorio es crear un sistema de gestión integral para una red de metro, permitiendo a los pasajeros moverse de manera eficaz y rápida por áreas urbanas.

3,2. Descripción del paradigma y relación con el laboratorio:

Para este proyecto, se utilizará el paradigma orientado a objetos (POO), el cual pertenece a la familia de los paradigmas imperativos. El paradigma orientado a objetos se basa en 4 principios fundamentales:

1. **Abstracción**: Consiste en identificar y representar las características esenciales de un objeto, ignorando los detalles no relevantes.
2. **Encapsulamiento**: Implica esconder los detalles internos de un objeto y exponer solo lo necesario a través de una interfaz pública. Esto asegura que los datos de un

objeto no puedan ser modificados directamente, sino solo a través de métodos definidos, lo cual garantiza la integridad y consistencia de los datos.

3. **Herencia**: Permite crear nuevas clases a partir de clases existentes, heredando sus características y comportamientos. Esto fomenta la reutilización de código y la creación de jerarquías de clases.
4. **Polimorfismo**: Permite que diferentes objetos respondan a la misma interfaz o método de distintas formas. Esto facilita la flexibilidad y escalabilidad del software

La Programación Orientada a Objetos (POO) es ideal para modelar sistemas complejos como una red de metro, gracias a sus características de modularidad, reutilización y mantenimiento. Este paradigma, que se centra en la creación de objetos y sus interacciones, se adapta perfectamente a la gestión de una red de metro. Proporciona un marco robusto y flexible para manejar la complejidad del sistema, garantizando coherencia y escalabilidad necesarias para su operación eficiente.

4.Análisis del problema

Para abordar la gestión integral de una red de metro, es fundamental considerar los elementos clave que la componen, tales como líneas de metro, estaciones, vagones, trenes, vías, sistemas de señalización y control, instalaciones de mantenimiento y conductores, entre otros. Por ende, se deben cumplir múltiples requerimientos funcionales que conforman las redes de metro las cuales son:

1. Creación y Gestión de Líneas de Metro
2. Añadir Estaciones
3. Planificación de Rutas y Conexiones
4. Gestión de Trenes
5. Control de Tarifas
6. Gestión de Personal
7. Mantenimiento de Infraestructura
8. Monitoreo y Control
9. Interfaz de Usuario

En resumen, la gestión integral de una red de metro abarca la creación y administración de líneas, la incorporación de estaciones, la planificación de rutas y conexiones, la gestión de trenes, el control de tarifas, la gestión de personal, el mantenimiento de infraestructuras, el monitoreo y control en tiempo real, y la provisión de una interfaz de usuario eficiente. Para garantizar una operación fluida y eficiente del sistema de metro, es crucial considerar todos estos aspectos. Por lo tanto, en este proyecto, se utilizará Java, un lenguaje de programación robusto que se adapta perfectamente al paradigma orientado a objetos. Cada uno de los componentes mencionados se representará mediante clases y objetos, lo que permite una gestión modular y escalable del sistema de metro. Además, cada

interacción directa con la red de metro se implementará como un método, asegurando así una operación organizada y coherente.

5.Diseño de la solución

5,1. Idea principal:

Dado que el núcleo de este problema radica en el uso de clases, métodos, interacciones entre objetos, el uso de interfaces o herenecias, la solución se fundamenta en la programación orientada a objetos (POO) y en la capa constructora de los tipos abstractos de datos (TDA) para gestionar y modificar la red de metro. Se diseñó un menú interactivo visualizable a través del CMD, proporcionando al administrador de la red una herramienta cómoda para resolver problemas y realizar consultas. Este menú interactivo permite también cargar archivos en formato ".txt" para integrar líneas, trenes, combinaciones y conductores en el sistema.

La solución para el modelado de la red de metro se centra en categorizar cada componente como una clase y definir las interacciones entre ellos a través de métodos. El diseño inicial se basó en un diagrama UML de análisis (Figura 5), estructurado antes de comenzar a manipular los objetos. A medida que avanzamos en el proyecto, observamos que el UML original era insuficiente, lo que nos llevó a expandirlo con más clases y métodos de interacción entre objetos para satisfacer las necesidades emergentes. Además, se consideraron todos los aspectos y condiciones que debía cumplir cada elemento de la red de metro, como IDs únicas, secciones y líneas duplicadas, asegurando así la integridad y eficiencia del sistema.

5,2. UML de análisis y diseño

El UML de análisis inicialmente propuesto resultó ser insuficiente para abordar el problema completo, ya que no contemplaba la implementación de un menú interactivo y la carga de archivos a través de este. Esta ampliación del problema requirió añadir varios atributos extra a las clases existentes y crear nuevas clases para los nuevos elementos. Durante el desarrollo del programa, realizamos múltiples ajustes y mejoras que reflejaron la evolución de nuestra comprensión del problema.

Conforme avanzábamos en la implementación, se hicieron evidentes las necesidades adicionales del sistema. Estos ajustes permitieron satisfacer plenamente los requisitos del proyecto y culminaron en la creación del UML de diseño. Este diagrama contempla la solución final al problema, cubriendo todos los requerimientos especificados.

El UML de diseño (Figura 6) detalla todas las interacciones del programa, incluyendo atributos, interacciones y métodos de cada clase. Este nivel de detalle asegura que todos los componentes del sistema estén adecuadamente representados y que sus relaciones e interacciones se comprendan claramente. Además, este UML de diseño no solo muestra la

estructura del sistema, sino que también ilustra cómo se integran los nuevos elementos, como el menú interactivo y la carga de archivos, proporcionando una visión completa y detallada de la solución final.

6.Aspectos de implementación

Para este laboratorio, todos los archivos fueron implementados en el "IDE IntelliJ IDEA Community Edition 2024.1" versión 64 bits, utilizando OpenJDK versión 11, específicamente “Amazon Corretto” versión 11.0.23. El sistema de construcción empleado fue “Gradle” con el DSL de Gradle basado en “Groovy”.

Cada TDA se representa como una clase dentro del programa. Además, cinco clases específicas: CargaDeDatos, Line, Menu, Subway y Train, contienen un archivo de “Interfaz”. Se implementó un menú interactivo funcional a través de una GUI, que incluye la capacidad de cargar archivos por la línea de comando. Por otro lado, se consideró la prohibición del uso de bibliotecas externas a Java, permitiéndose únicamente el uso de la biblioteca estándar de Java.

7.Instrucciones de uso

Como se implementó una interfaz GUI a través de CMD, integrada con una red de metro lista para hacer consultas, es importante seguir las instrucciones de cada opción cuidadosamente. Si se solicita un número, no se debe introducir una cadena de texto y viceversa. Si se requiere un archivo de texto, este debe ser estrictamente un archivo .txt, y debe incluirse su nombre completo con la extensión ".txt".

Es fundamental leer el archivo “léeme.txt”, ya que contiene consejos para una mejor visualización y uso del programa. Para compilar correctamente el programa, se deben utilizar los siguientes comandos:

Linux/Unix:

1. Compilación: ./gradlew build
2. Ejecución: ./gradlew run

Windows:

1. Compilación: gradlew.bat build
2. Ejecución: gradlew.bat run

8.Resultados y Auto Evaluación

Se lograron cumplir los 25 requerimientos funcionales establecidos (Figura 1). Esto se debió en gran parte a la reutilización de ideas provenientes de laboratorios anteriores, lo cual facilitó la creación de las clases y métodos que representan la red de metro. Sin embargo, es importante destacar que hubo dos métodos que requirieron más tiempo que los demás: WhereIsTrain y TrainPath. Estos métodos involucraron el uso de una biblioteca de tiempo que complicó el proceso de finalización del laboratorio, pero se logró completar satisfactoriamente. La precisión en la ejecución de los métodos solicitados puede observarse en la (Figura 4), donde se analiza la frecuencia con la que funcionaron.

9.Conclusión

A lo largo del proyecto, hemos desarrollado una comprensión profunda y habilidades prácticas en el uso del paradigma orientado a objetos en Java, contrastándolo con nuestros conocimientos previos en Prolog y Scheme. Java, siendo un lenguaje orientado a objetos robusto y ampliamente utilizado, nos ha permitido manejar eficazmente la complejidad de la red de metro que estamos implementando. A diferencia de Prolog y Scheme, que adoptan un enfoque declarativo y funcional respectivamente, Java se centra en la creación de objetos y en la interacción entre ellos, lo cual es fundamental para modelar sistemas complejos como una red de metro.

Java nos ha obligado a adoptar una forma diferente de pensamiento comparado con Prolog y Scheme. Mientras que Prolog nos enseñó a especificar qué queremos lograr de manera declarativa, Java nos ha llevado a pensar en cómo implementar esos objetivos utilizando clases, métodos y objetos. Esta transición ha enriquecido nuestra perspectiva sobre la programación y la resolución de problemas, proporcionándonos herramientas para abordar problemas complejos de manera modular y escalable.

Durante el proyecto, no experimentamos conflictos de paradigmas significativos. El cambio desde Scheme, un lenguaje funcional, hacia Prolog, centrado en reglas y hechos, preparó el terreno para nuestro trabajo actual en Java. Aunque Java requiere un ajuste hacia una estructura más orientada a objetos, hemos podido aprovechar las lecciones aprendidas de los paradigmas anteriores para implementar eficazmente los requisitos funcionales del proyecto. Esta adaptabilidad demuestra nuestra capacidad para integrar diferentes enfoques y herramientas de programación según las necesidades del proyecto, asegurando así el éxito en la implementación del sistema de gestión de una red de metro utilizando el paradigma orientado a objetos en Java.

10.Anexos

A. *Auto evaluación, Figura 1*

```
TDA station-constructor/ station / 1
TDA section-constructor / section/ 1
TDA line-constructor / line / 1
TDA line-line-length / lineLength / 1
TDA line-line-section-length / lineSectionLength / 1
TDA line-line-cost / linecost / 1
TDA line-line-section-cost / linesectioncost / 1
TDA line-line-add-section / lineaddsection / 1
TDA line-line? / line? / 1
TDA PassengerCar -constructor / PassengerCar / 1
TDA train-constructor/ train / 1
TDA train-addCar / addCar / 1
TDA train-removeCar / removeCar / 1
TDA train-isTrain / isTrain / 1
TDA train-fetchCapacity / fetchCapacity / 1
TDA driver-constructor / driver / 1
TDA subway-constructor / subway / 1
TDA subway-addTrain/ addTrain / 1
TDA subway-addLine / addLine / 1
TDA subway-addDriver / addDriver / 1
TDA subway-toString / toString / 1
TDA subway-assignTrainToLine / assignTrainToLine / 1
TDA subway-assignDriverToTrain / assignDriverToTrain / 1
TDA subway-whereIsTrain / whereIsTrain/ 1
TDA subway-trainPath/ trainPath / 1
```

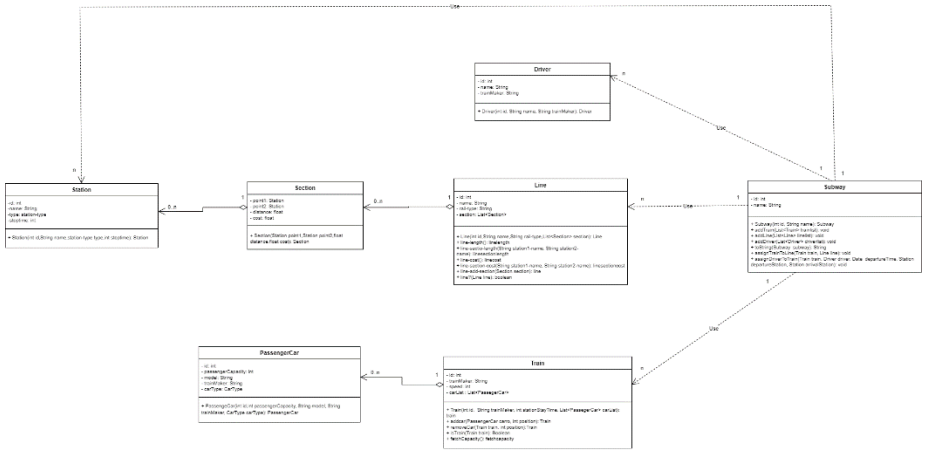
B. *Red de metro de Santiago. Figura 2*

C. *línea del metro de Santiago. Figura 3*

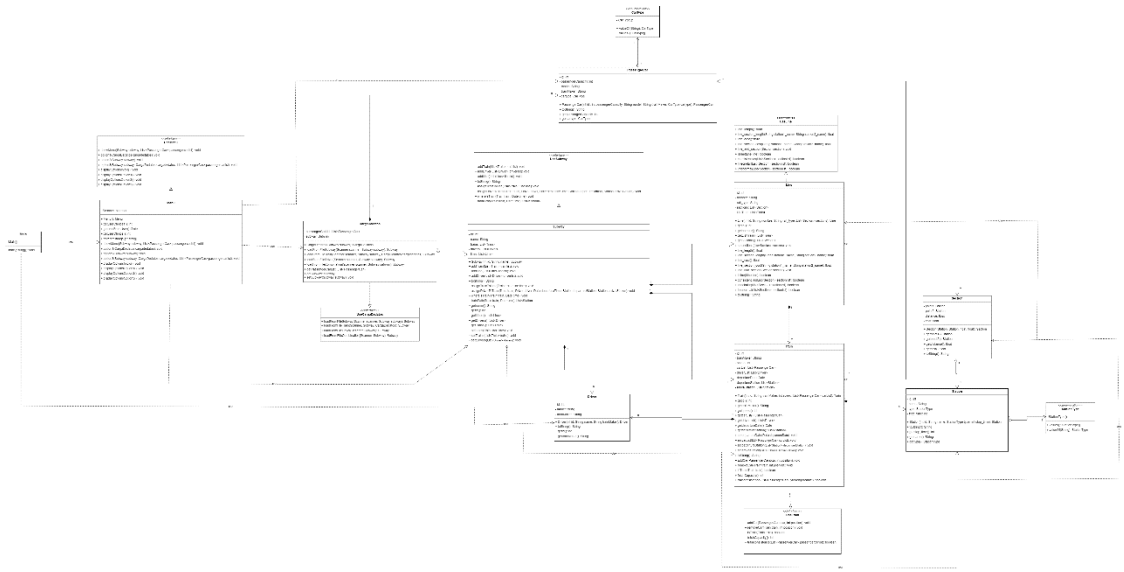
D. *Tabla de resultados de ejecución de las reglas solicitadas. Figura 4*

Metodo	Conteo Aprobación	Conteo de fallo
station	5	0
section	5	0
line	5	0
lineLength	10	5
lineSectionLength	10	5
lineAddSection	5	1
isLine	10	3
PassengerCar	5	0
train	5	0
AddCar	10	3
RemoveCar	10	3
isTrain	10	0
trainCapacity	5	0
driver	5	0
subway	5	0
subwayAddTrain	10	4
subwayAddLine	10	4
subwayAddDriver	10	4
subwayToString	10	0
subwaySetStationStoptime	10	5
subwayAssignTrainToLine	10	5
subwayAssignDriverToTrain	10	5
whereIsTrain	20	8
subwayTrainPath	20	8

E. *Diagrama de análisis. Figura 5*



F. *Diagrama de diseño. Figura 6*



11.Bibliografía:

1. *CAMPUS VIRTUAL:* Paradigma Orientado a Objetos.(s. f.). https://uvirtual.usach.cl/moodle/course/view.php?id=10036§ion=20
2. Google Drive: Slides POO. (s. f.). Recuperado de https://drive.google.com/drive/u/2/folders/1Dl4k6qJ6pzCEvD2POW1mQJca-t3UwE9-
3. Learn Java - dev.java. (s. f.). Dev.java: The Destination For Java Developers. https://dev.java/learn/
4. Miriam Martínez (2020, noviembre 2) ¿Qué es la programación orientada a objetos? Recuperado de https://profile.es/blog/que-es-la-programacionorientada-a-objetos/
5. Jorge López (2023, Octubre 27) Introducción a POO en Java: Objetos y clases Recuperado de https://openwebinars.net/blog/introduccion-a-poo-en-java-objetos-y-clases/#:~:text=La%20Programación%20Orientada%20a%20Objetos%20(POO)%20es%20un%20enfoque%20fundamental,una%20instancia%20de%20una%20clase
6. *Qué es el lenguaje unificado de modelado (UML)*. (s. f.). Lucidchart. https://www.lucidchart.com/pages/es/que-es-el-lenguaje-unificado-de-modelado-uml