Desafío#2: Simulación Red Metro

Jorge Enrique Rueda Urrea - 1.020.488.162
Edison Fredy Serrano Álvarez - 1.055.273.822
Ingeniería de Telecomunicaciones

Fecha: Mayo 2024

Keywords: Simulación, Red metro, Punteros, Estación, Métodos, Clases, Arreglos

1 ANÁLISIS DEL PROBLEMA

El problema se centra en la gestión de una red de metro, incluyendo la creación, modificación y consulta de sus elementos (líneas, estaciones y estaciones de transferencia). Además, se requiere un subprograma para simular el tiempo de llegada de un tren a una estación específica.

2 CONSIDERACIONES

- Se utiliza el paradigma de POO para encapsular la información y el comportamiento de los elementos de la red de metro.
- Se definen clases para representar cada elemento (RedMetro, Linea y Estacion) con sus atributos y métodos correspondientes.
- Las relaciones entre las clases se establecen mediante asociaciones.

Los subprogramas se diseñan para realizar las operaciones específicas sobre la red de metro, incluyendo la gestión de líneas y estaciones, la consulta de información y la simulación del tiempo de llegada.

3 TIPO Y ESTRUCTURAS DE DATOS A USAR

- **Arreglos:** Se utilizarán arreglos dinámicos para almacenar las colecciones de elementos (líneas en una red, estaciones en una línea).
- **Atributos:** Se utilizarán variables de tipo String para almacenar nombres, de tipo int para almacenar tiempos y de tipo booleano para indicar si una estación es de transferencia.
- **Punteros:** Se utilizarán punteros para gestionar las relaciones entre las clases y para almacenar las referencias a los elementos.

4 DIAGRAMA DE CLASES

- **CLASE#1: RedMetro**
  - **Atributos:**
    - nombre: String
    - lineas: Arreglo dinámico de Linea*
  - **Métodos:**
    - agregarLinea(Linea* linea): void
    - eliminarLinea(Linea* linea): void
    - getNumeroLineas(): int
    - getNumeroEstaciones(): int
    - isEstacionEnLinea(Estacion* estacion, Linea* linea): boolean
    - toString(): String
- **CLASE#2: Linea**
  - **Atributos:**
    - nombre: String
    - redMetro: RedMetro*
    - estaciones: Arreglo dinámico de Estacion*
  - **Métodos:**
    - agregarEstacion(Estacion* estacion, Posicion posicion): void
    - eliminarEstacion(Estacion* estacion): void
    - getNumeroEstaciones(): int
    - toString(): String
- **CLASE#3: Estacion**
  - **Atributos:**
    - nombre: String
    - linea: Linea*
    - tiempoAnterior: int
    - tiempoSiguiente: int
    - esTransferencia: boolean
  - **Métodos:**
    - getTiempoLlegada(Estacion* destino): int
    - toString(): String

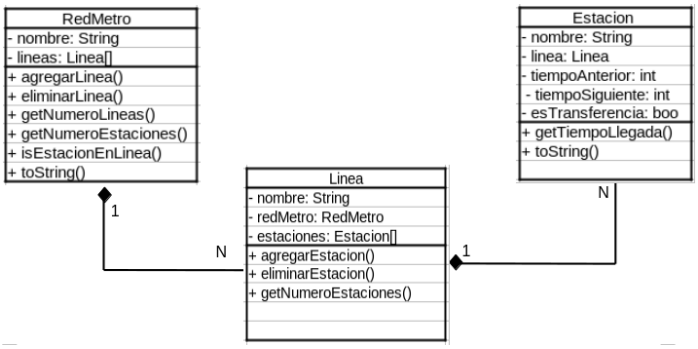

Figure 1: Diagrama de clases simplificado.

En este diagrama se utilizan los especificadores de acceso ”-” para indicar atributos privados y ”+” para indicar métodos públicos.

5 A TENER EN CUENTA:

Estos detalles son importantes a la hora de analizar y posteriormente procesar nuestras clases:

- RedMetro tiene un arreglo dinámico de Linea.
- Linea pertenece a una RedMetro.
- Linea tiene un arreglo dinámico de Estacion.
- Estacion pertenece a una Linea.
- Estacion tiene un atributo TipoTransporte.

6 LOGICA DE LAS TAREAS NO TRIVIALES

Para cada una de nuestras clases se deben implementar varias tareas cuyo desarrollo tiene un analisis mas profundo

- **Agregar una estación a una línea:** Se debe verificar si la estación ya existe en la línea.
  Si la estación no existe, se crea una nueva instancia de Estación y se agrega al arreglo dinámico de estaciones de la Línea.
  Se actualizan los tiempos de llegada de las estaciones posteriores a la nueva estación.
- **Eliminar una estación de una línea:** No se permite eliminar estaciones de transferencia.
  Si la estación no es de transferencia, se elimina del arreglo dinámico de estaciones de la Línea.
  Se actualizan los tiempos de llegada de las estaciones posteriores a la estación eliminada.
- **Cálculo del tiempo de llegada:** Se obtiene la estación de origen y la estación de destino.
  Se recorre el arreglo dinámico de estaciones de la Línea desde la estación de origen hasta la estación de destino, sumando los tiempos de llegada de cada estación.
  Se retorna el tiempo total de llegada a la estación de destino.

La implementación del modelo de datos se realizará haciendo uso POO. Se debe tener en cuenta la gestión de la memoria y la liberación de recursos para evitar fugas de memoria, debido a esto haremos uso exhaustivo de punteros.

A medida que vayamos avanzando en la solución de nuestro Desafío se irán realizando pruebas unitarias para verificar el correcto funcionamiento de cada clase y subprograma. Seguido a esto haremos pruebas de integración para verificar la interacción entre las diferentes clases.