# INFORME DE PARCIAL II

**Sergio Andrés Berbesí Builes – Salome Bermúdez Macias**

## a. Análisis del problema y consideraciones para la alternativa de solución propuesta:

Se requiere modelar una red de metro (como modelo genérico, no de una red existente de metro). Esta red contiene líneas y estaciones, las cuales poseen características como:

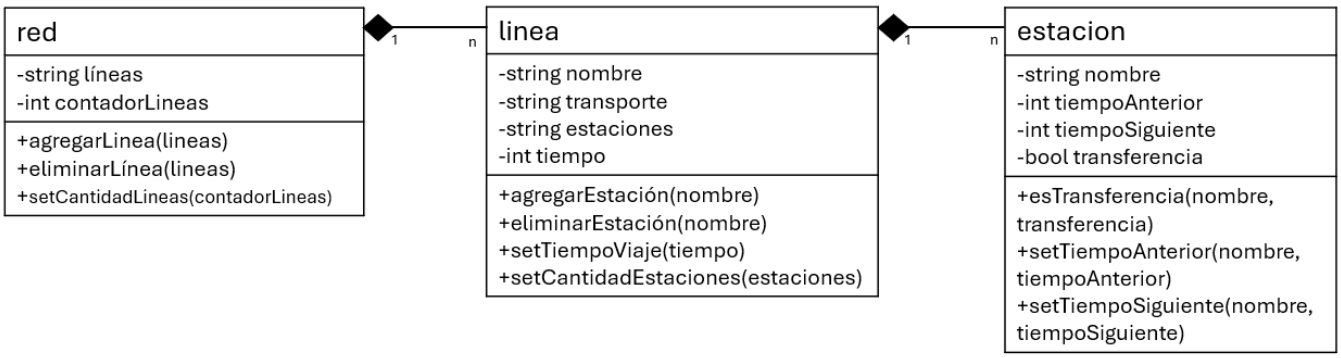
*Línea:* tipo de transporte (tren o tranvía) y dirección (bidireccionales). No tienen bifurcaciones ni bucles.

*Estación:* nombre, tiempo de desplazamiento (en segundos, hasta la estación anterior y siguiente en la misma línea) y categoría (pertenecer a una o más líneas).

Específicamente, la simulación debe permitirle al usuario plantear una red de metro desde cero, ofreciendo las opciones para agregar, eliminar y conocer la cantidad de líneas y estaciones que conforman el diseño que está elaborando. Así mismo, se solicita que se ofrezca un subprograma para que el usuario conozca los tiempos de desplazamiento entre estaciones de la misma línea, que deben extraerse partiendo de la red de metro creada.

*Solución propuesta:* Se va a crear un simulador, compuesto por un modelo de una sola red de metro, la cual, va a manejar diferentes líneas compuestas por estaciones, conexiones entre estaciones (estaciones de transferencia) y opciones para modificar la red. Se van a manejar las estaciones como objetos, puesto que, por el planteamiento del problema resulta como la opción más eficiente. Adicionalmente, se van a manejar las líneas como clases, las cuales van a contener los objetos estaciones y permitirán manejar a cada una de ellas de forma independiente.

## b. Diagrama de clases:




## c. Algoritmos implementados:

**Clase Estación:**

- **Atributos:**
  - **nombre:** Una cadena que representa el nombre de la estación.

- **lineas:** Un arreglo dinámico de punteros a objetos de la clase Linea, que almacena las líneas a las que está conectada la estación.
- **capacidadLineas:** Un entero que representa la capacidad del arreglo lineas.
- **numLineas:** Un entero que representa el número actual de líneas conectadas a la estación.
- **anterior:** Un puntero a la Estación anterior en la misma línea.
- **siguiente:** Un puntero a la Estación siguiente en la misma línea.

- **Métodos:**
  - **Estacion(string _nombre):** Constructor que inicializa el nombre de la estación y otros atributos.
  - **~Estacion():** Destructor que libera la memoria del arreglo de punteros lineas.
  - **getNombre():** Devuelve el nombre de la estación.
  - **setAnterior(Estacion* estacion):** Establece la estación anterior en la misma línea.
  - **getAnterior():** Devuelve la estación anterior en la misma línea.
  - **setSiguiente(Estacion* estacion):** Establece la estación siguiente en la misma línea.
  - **getSiguiente():** Devuelve la estación siguiente en la misma línea.
  - **agregarLinea(Linea* linea):** Agrega una nueva línea a la estación.
  - **tieneLinea(Linea* linea):** Verifica si la estación está conectada a una línea específica.
  - **eliminarEstacion(Estacion* estacion):** Elimina una estación de la red.
  - **tiempoDeViaje(Estacion* origen, Estacion* destino):** Calcula el tiempo de viaje entre dos estaciones.

**Clase Linea:**

- **Atributos:**
  - **nombre:** Una cadena que representa el nombre de la línea.
  - **transporte:** Una cadena que representa el tipo de transporte de la línea.
  - **primeraEstacion:** Un puntero a la primera Estación en la línea.
  - **ultimaEstacion:** Un puntero a la última Estación en la línea.
  - **siguiente:** Un puntero a la siguiente Línea en la red.
  - **capacidad:** Un entero que representa la capacidad de la línea.
  - **numEstaciones:** Un entero que representa el número actual de estaciones en la línea.

- **Métodos:**
  - **Linea(string _nombre, string _transporte, int _capacidad):** Constructor que inicializa el nombre, tipo de transporte y capacidad de la línea.
  - **getNombre():** Devuelve el nombre de la línea.
  - **numeroEstaciones():** Devuelve el número de estaciones en la línea.
  - **agregarEstacion(Estacion* estacion):** Agrega una estación a la línea.
  - **eliminarEstacion(Estacion* estacion):** Elimina una estación de la línea.

- **tiempoDeViaje(Estacion* origen, Estacion* destino):** Calcula el tiempo de viaje entre dos estaciones en la misma línea.
- **getSiguiente():** Devuelve la siguiente Línea en la red.
- **setSiguiente(Linea* _siguiente):** Establece la siguiente Línea en la red.

**Clase Red:**

- **Atributos:**
  - **primeraLinea:** Un puntero a la primera Línea en la red.
  - **ultimaLinea:** Un puntero a la última Línea en la red.
  - **numLineas:** Un entero que representa el número actual de líneas en la red.

- **Métodos:**
  - **Red()**: Constructor que inicializa los atributos de la red.
  - **agregarLinea(Linea* linea):** Agrega una nueva línea a la red.
  - **eliminarLinea(Linea* linea):** Elimina una línea de la red.
  - **contarLineas():** Devuelve el número de líneas en la red.
  - **mostrarMenu():** Muestra un menú de opciones para interactuar con el sistema.
  - Además de las clases, hay una función calcularTiempoDeLlegada que simula el cálculo del tiempo de llegada entre dos estaciones en la misma línea.

El programa principal main() se implementa para mostrar un menú de opciones al usuario y gestionar las interacciones con el sistema de gestión de la red de metro.

**d. Problemas de desarrollo que afrontó:**

Durante el desarrollo del programa, nos enfrentamos a varios desafíos. Uno de los principales desafíos fue el manejo de la programación orientada a objetos. En este sentido, la creación y gestión de clases, así como la implementación de constructores, destructores, getters, setters y operaciones de despliegue, estas fueron fundamentales para la estructura del código de manera coherente y eficiente. La correcta definición de estas estructuras fue crucial para garantizar la modularidad y reutilización del código, así como para facilitar su mantenimiento y escalabilidad en el futuro.

Otro desafío importante fue el manejo de la memoria dinámica. El manejo de este debe ser esencial para evitar fugas de memoria o errores de acceso que podrían hacer daños en el programa. Garantizar la correcta reserva y liberación de memoria para cada objeto creado dinámicamente.

Además, la implementación de la lógica del programa presentó otro desafío importante. Desarrollar un algoritmo eficiente para la red del metro, que se adaptara a las necesidades y requerimientos del cliente, demandó un análisis detallado y una planificación cuidadosa. Fue necesario considerar diversos factores, como la conectividad entre estaciones, los tiempos de viaje, las posibles rutas alternativas y las restricciones del sistema, para diseñar un algoritmo funcional.

Por último, la validación de las entradas del usuario, verificar que las condiciones ingresadas por el usuario fueran válidas y cumplieran con los requisitos del problema fue crucial para garantizar la integridad y consistencia de los datos procesados por el programa.

**e. Evolución de la solución y consideraciones para tener en cuenta en la implementación:**

- Una estación puede pertenecer a varias líneas, lo cual, la convierte en una estación de transferencia y cambia su nombre.
- Una línea solo puede pertenecer y estar una vez en una red.
- Una estación solo puede estar una vez en una línea.
- Si una red tiene más de una línea, estas deben estar siempre conectadas (es decir, al menos debe existir una estación de transferencia).
- Pueden existir dos estaciones con el mismo nombre, siempre que se cumpla la condición de que cada estación esté en líneas distintas.
- El tiempo entre estaciones siempre se debe manejar en segundos.
- Cuando se agrega una estación entre dos estaciones ya establecidas, se debe actualizar el tiempo de desplazamiento desde dichas estaciones hacia la estación actual. La suma del tiempo que se define entre estaciones no necesita ser igual al tiempo de desplazamiento establecido antes de la creación de la estación actual.
- Cuando se elimina una estación ubicada entre dos estaciones ya establecidas, se debe actualizar el tiempo de desplazamiento entre ambas estaciones sumando los tiempos que tenía la estación eliminada hacia sus estaciones anterior y posterior.
- No pueden existir estaciones aisladas (excepción de la primera línea, que va a estar aislada de forma obligatoria).
- No se pueden eliminar las estaciones de transferencia.
- Al crear estaciones, siempre se le pregunta al usuario por el nombre, línea a la que va a pertenecer y tiempos de transferencia a sus estaciones vecinas.
- Si una línea tiene estaciones de transferencia, no se puede eliminar.
- No hay límite de líneas en las que pueda estar una estación de transferencia.
- No pueden existir dos estaciones de transferencia con las mismas líneas (genera un bucle).