**INFORME DESAFIO II**

* **Análisis del problema y consideraciones para la alternativa de solución propuesta:**

Después de leer el documento del desafío II varias veces para poder tener certeza de las diferentes problemáticas que podemos llegar a tener, llegamos a pensar en las principales problemáticas que puede llegar a tener el problema para considerar su respectiva solución y así poder avanzar con el desafío planteado en clase.

**Estructura de clases:**

Uno de los principales problemas a la hora de poder empezar el desafío II fue como se iba a estructurar las clases y si llegaba a tener una jerarquía no solamente cuantas clases, sino que tendría adentro cada clase, que parámetros, métodos, constructores, etc. Una de las alternativas que optamos como podemos ver en el diagrama de clases, fue crear 3 clases. Clase estación, Clase línea, Clase RedMetro. Donde en cada una de las clases tendrá un nombre del objeto que se está creando. Para la clase estación se tiene pensado guardar los tiempos de las estaciones, tanto para la estación siguiente como la estación anterior, además guardar si llega o no a ser una estación de transferencia. Para la clase línea se tiene pensado guardar todas las estaciones que contiene la línea que se está creando en un arreglo dinámico, además de poder agregar o eliminar una estación, además de conocer la cantidad de estaciones que puede llegar a tener una misma línea y conocer si una estación es o no perteneciente a la línea que se está viendo. Para la clase RedMetro, se tiene planeado también tener un arreglo dinámico para saber cuántas líneas hay dentro de la red que nosotros estamos trabajando, además de poder tanto agregar como eliminar una línea, conocer cuantas líneas en total hay adentro de la red y el total de estaciones que puede llegar a tener la misma red.

**Uso de memoria dinámica:**

Además de considerar que contendría cada clase, también una de las problemáticas grandes era saber cómo guardar cada una de las clases o los elementos que tendría sin poder usar ningún tipo de contenedor de la STL ya vistos en clase, ya que el propio desafío nos impedía usarlas, para ello como alternativa para la problemática, pensamos en usar arreglos con el uso de memoria dinámica para así poder contener cada uno de los elementos que necesitamos, tanto agregar como eliminar o conocer la cantidad de objetos dependiendo de la clase que nosotros estamos mirando como alternativa al uso de vectores o mapas o multimapas etc.

* **Diagrama de clases de la solución planteada.**

**1. Clase Estación:** Representa una estación dentro del sistema de metro. Puede tener atributos como nombre, tiempo de llegada a la siguiente estación en la misma línea, y un indicador si es una estación de transferencia. Los atributos de índole privado son los siguientes:

**string nombre, int tiempoSiguiente, Estacion\* siguiente**. Estos parámetros de tipo privado tienen como objetivo guardar el nombre de la estación, conocer el tiempo de una estación a la siguiente y un puntero a una instancia de la clase estación y lo que almacena es la dirección de memoria de la siguiente estación en la secuencia de estaciones para así poder conservar el orden de las estaciones con sus líneas.

Para los atributos de índole publico tenemos 4 métodos, 1 constructor y 1 destructor, los cuales son los siguientes:

**Estacion(string nombre, int tiempoSiguiente), string obtenerNombre() const, int obtenerTiempoSiguiente() const, Estacion\* obtenerSiguiente() const, void establecerSiguiente(Estacion\* siguienteEstacion), ~Estacion()**. Para el constructor recibe dos parámetros, los cuales son el nombre y el tiempo siguiente, el cual sería en minutos, usa la lista de inicialización de miembros para inicializar los atributos, se asigna nullptr al puntero siguiente para indicar que inicialmente la estación no tiene una estación siguiente. Para el método string obtener nombre simplemente devuelve el nombre de la estación. Para el método obtener tiempo siguiente simplemente devuelve el tiempo en minutos hasta la siguiente estación. Para el método establecer siguiente recibe un puntero a una instancia de estación como parámetro, establece el puntero siguiente de la estación actual para apuntar a la siguiente estación y para el destructor simplemente se encarga de liberar la memoria asignada dinámicamente para la siguiente estación y elimina la siguiente estación llamando a delete

**2. Clase Línea:** Representa una línea dentro del sistema de metro. Contendría una lista de estaciones que pertenecen a esa línea, así como métodos para agregar y eliminar estaciones, y para verificar si una estación pertenece a esa línea. Los atributos privados son los siguientes:

**String nombre, Estacion\* primeraestacion.** En estos dos parámetros lo que hacemos es almacenar el nombre de la línea y un puntero a la primera estación de la línea.

Para los atributos de tipo publico tenemos los siguientes:

Tenemos 1 constructor, 1 destructor y 3 métodos los cuales son:   
**~Linea(), Linea(string nombre), void agregarEstacion(const string& nombreEstacion, int tiempoSiguiente), void eliminarEstacion(const string& nombreEstacion), void mostrarEstaciones() const. Para el destructor** simplemente se encarga de liberar la memoria asignada para las estaciones de la línea, para el constructor toma 1 parámetro donde inicializa el nombre de la línea con el valor proporcionado y establece el puntero primera Estacion en nullptr, indicando que la línea no tiene estaciones inicialmente. El método agregar estación permite agregar una nueva estación a la línea, recibe el nombre de la estación y el tiempo que le toma a la siguiente, Si la línea ya tiene estaciones, recorre la lista de estaciones hasta encontrar la última y agrega la nueva estación al final. Para el método eliminar estación, nos permite eliminar una estación de la línea, recibe el nombre de la estación Recorre la lista de estaciones y busca la estación con el nombre especificado Si encuentra la estación, la elimina ajustando los enlaces de las estaciones adyacentes. Y el método mostrar estaciones, nos muestra todas las estaciones de la línea junto con el tiempo hasta la siguiente estación, Itera sobre la lista de estaciones e imprime el nombre de cada estación con su respectivo tiempo.

**3. Clase Red Metro:**

En la clase Red Metro, tenemos tanto parámetros privados como públicos, en los atributos privados podemos encontrar:

**Linea\*\* líneas, int cantidadLineas, int capacidad.** En estos parámetros tenemos, un puntero doble que apunta a un array dinámico de punteros a objetos de tipo Linea. Este array contiene las líneas de metro en la red. Cada puntero apunta a una instancia de la clase Linea. También tenemos un entero que almacena el número actual de líneas en la red del metro. Y también un entero que indica la capacidad actual del array dinámico.

Para los atributos de tipo publico tenemos lo siguiente:

1 constructor, 1 destructor y 5 métodos públicos los cuales son los siguientes, **RedMetro (), ~RedMetro (), void agregarLinea (const string& nombreLinea), void agregarEstacionALinea (const string& nombreLinea, const string& nombreEstacion, int tiempoSiguiente), void agregarEstacionDeTransferencia (const string& nombreEstacion, const string& nombreLinea, const string& nombreEstacionTransferencia, int tiempoSiguiente), void mostrarEstacionesDeLinea(const string& nombreLinea), void eliminarEstacionesDeLinea(const string& nombreLinea).** El objetivo principal del constructor es inicializar los miembros de la clase y preparar el objeto para su uso, se inicializa con el valor nullptr, lo que indica que al principio no hay ninguna línea en la red del metro, y la cantidad de líneas y capacidad Se inicializa con el valor 0, ya que al inicio no hay ninguna línea en la red del metro y que el array líneas no tiene ninguna capacidad asignada al principio, por eso empezando la capacidad en 0. Para el destructor se utiliza un bucle for para recorrer cada uno de los punteros a objetos Linea almacenados en el array líneas, para cada puntero a objeto Linea, se llama al operador delete para liberar la memoria asignada al objeto Linea apuntado por ese puntero, después de liberar la memoria para cada objeto Linea, se libera la memoria asignada al array líneas utilizando el operador delete []. Para el método agregar línea permite agregar una nueva línea de metro a la red. Toma como argumento el nombre de la línea que se va a agregar. Si la capacidad actual de la red no es suficiente para almacenar la nueva línea, el método redimensiona el array de líneas para acomodar más líneas. Para el método agregar estación a línea permite agregar una nueva estación a una línea existente en la red del metro. Toma como argumentos el nombre de la línea a la que se va a agregar la estación, el nombre de la estación y el tiempo hasta la siguiente estación. Para el método agregar estación de transferencia permite agregar una estación de transferencia entre dos líneas de metro. Toma como argumentos el nombre de la estación de transferencia, el nombre de la línea de origen, el nombre de la línea de destino y el tiempo hasta la siguiente estación. Para el método mostrar estaciones de línea permite mostrar las estaciones de una línea específica de metro. Toma como argumento el nombre de la línea cuyas estaciones se van a mostrar. El método es constante, lo que significa que no modifica el estado interno de la clase RedMetro. Para el método eliminar estaciones de la línea permite eliminar todas las estaciones de una línea específica de metro y la elimina de la red del metro. Toma como argumento el nombre de la línea que se va a eliminar.

* **Algoritmos implementados debidamente intra-documentados.**

Además de conocer los algoritmos ya utilizados en el diagrama de clase (parámetros privados, parámetros públicos, métodos, constructores y destructores etc.), utilizamos un menú para poder abordar todos los puntos que se piden en el desafío, se implementaron también 3 clases para poder dar una óptima solución a las problemáticas planteadas, algoritmos adicionales a los conocidos en el diagrama no fueron creados ya que no se vio completamente necesario ya que con los métodos anteriormente creados podían solucionar los diferentes problemas planteados en el documento del desafío