**Proceso de análisis y documentación - Desafío 2.**

1. **Análisis del problema y consideraciones para la alternativa de solución propuesta**
2. **Diagrama de clases**

|  |
| --- |
| **RedMetro** |
| -string nombre - int numLineas -Linea\* lineas  **1** |
| +void AgregarLinea() +void EliminarLinea() +RedMetro(string) |

|  |
| --- |
| **Estacion** |
| -string nombre -string linea -int numTranferencias -float tiempoAnterior -float tiempoSiguiente -Estacion\* lineasTranferencia |
| +Estacion(string,string) |

**1…n**

|  |
| --- |
| **Linea** |
| -string nombre -int numEstaciones -Estacion\* estaciones |
| +void AgregarEstacion(Estacion) +void AgregarEstacionAtras(Estacion) +void AgregarEstacionPosicion(Estacion,int) +void EliminarEstacion() +void mostrarEstaciones() +Linea(int) |

**1…n**

**n**

|  |
| --- |
| **Utilidades** |
|  |
| +Estacion\*agregarEstacionArregloFinal(Estacion,Estacion\*,int) +Linea\* agregarLineaArregloFinal(Linea,Linea\*,int) +Estacion\* agregarEstacionArregloInicio(Estacion, Estacion\*, int) +Linea\* agregarLineaArregloInicio(Linea, Linea\* , int) +Estacion\* agregarEstacionArregloMedio(Estacion, Estacion\*, int, int) +Linea\* agregarLineaArregloMedio(Linea , Linea\* , int , int ) +Linea\* EliminarLinea(Linea , Linea\* , int ) +Estacion\* EliminarEstacion(Estacion\*, Estacion\*, int ) |

**Explicación detallada de las clases, sus atributos y sus métodos.**

**Clase “RedMetro”**

Hemos decidido crear una clase llamada **“RedMetro”** con el fin de que esta contenga un conjunto de atributos y métodos que describan cómo se verá y se comportará un objeto de este tipo.

**Para la clase “RedMetro” hemos creado los siguientes atributos:**

* **String nombre:** Este atributo es de tipo string porque almacenará el nombre que el usuario le pondrá a la “red metro”.
* **Int numlineas**: Este atributo almacenará un número entero positivo y diferente de cero; el cuál, representará la cantidad de líneas que compondrán la “red metro”. (recordar hacer validaciones para que el número ingresado siga las características)
* **Linea\* líneas:** Este atributo es un puntero llamado “líneas” que apunta a un arreglo unidimensional; el cuál, almacena objetos tipo “linea”; en otras palabras, este arreglo almacena el nombre de las líneas que contendrá la red metro.

**Para la clase “RedMetro” hemos creado los siguientes métodos:**

* **void AgregarLinea( ):** Este método permite que el usuario agregue una nueva linea a la red metro, pues despliega un menú en pantalla que muestra las lineas que hacen parte de esta, enumerando cada una de ellas. Posterior a esto, sé solicita al usuario que ingrese un número para escoger una linea determinada; luego, se desplegará otro menú, igual al anterior pero con las estaciones que conforman la linea que fue seleccionada, pues desde esa estación seleccionada, comenzará la creación de una nueva línea. Finalmente se pide al usuario que ingrese por pantalla el nombre de la nueva linea que desea crear.
* **void EliminarLinea( )**: Este método se ha creado con la finalidad de eliminar una línea; es decir, eliminándola arreglo que contiene las líneas (solo puede ser eliminada la primera línea, ya que es la única que no tiene estaciones de transferencia) PREGUNTAR A MIGUEL.
* **RedMetro(string nombre**): Este método es un contructor de la clase “RedMetro”, el cuál inicializa un objeto de la clase RedMetro con un nombre dado, posterior a esto crea un arreglo dinámico para almacenar las líneas que pertenecerán a la red metro. También inicializa el contador de líneas en “0”, indicando que aún no hay líneas en el sistema cuando se crea un objeto en la clase RedMetro.

**¿Cuántos objetos tendrá la clase “RedMetro”?** Según nuestro análisis, la clase “red metro” tendrá una sólo onjeto, pues nuestro programa sólo se encargará de modelar una red.

**Clase “Linea”**

Hemos decidido crear una clase llamada **“Línea”** con el fin de que esta tenga un conjunto de atributos y métodos que describan cómo se verá y se comportará un objeto de este tipo.

**Para la clase “Linea” hemos creado los siguientes atributos:**

* **String nombre:** Este atributo almacenará un nombre; en este caso particular, el nombre que el usuario le pondrá a cada uno de los objetos de la clase “Linea”.
* **Int numEstaciones:** Este atributo almacenará un número entero positivo, diferente de cero, y mayor o igual a 2; el cuál, representará la cantidad de estaciones que tendrán cada una de las líneas de la red metro.
* **Estacion\* estaciones:** Este atributo es un puntero llamado “estaciones” que apunta a un arreglo unidimensional que contiene objetos de la clase “estación”. Este arreglo almacenará el número de estaciones que compondrán a cada una de las líneas

**Para la clase “Línea” hemos creado los siguientes métodos:**

* **void AgregarEstacion(Estacion estacion):** Este método permite agregar una estación a una línea de la red metro. Si es la primera estación, la agrega directamente al principio de la línea. Si no es la primera estación, ajusta los tiempos de espera entre las estaciones existentes y la nueva estación, y luego agrega la estación al final de la línea. Finalmente, imprime un mensaje indicando que la estación se ha agregado con éxito.

**¿Cuántos objetos tendrá la clase “Linea”?** La clase línea tendrá la cantidad de objetos que almacene el atributo definido como **Int numlineas** en la clase “red metro”. ¿De qué manera haremos eso? Preguntar a miguel.

**Clase “Estacion”**

Hemos decidido crear una clase llamada “Estacion” con el fin de que esta contenga un conjunto de atributos y métodos que describan cómo se verá y se comportará un objeto de este tipo.

**Para la clase “estacion” hemos creado los siguientes atributos:**

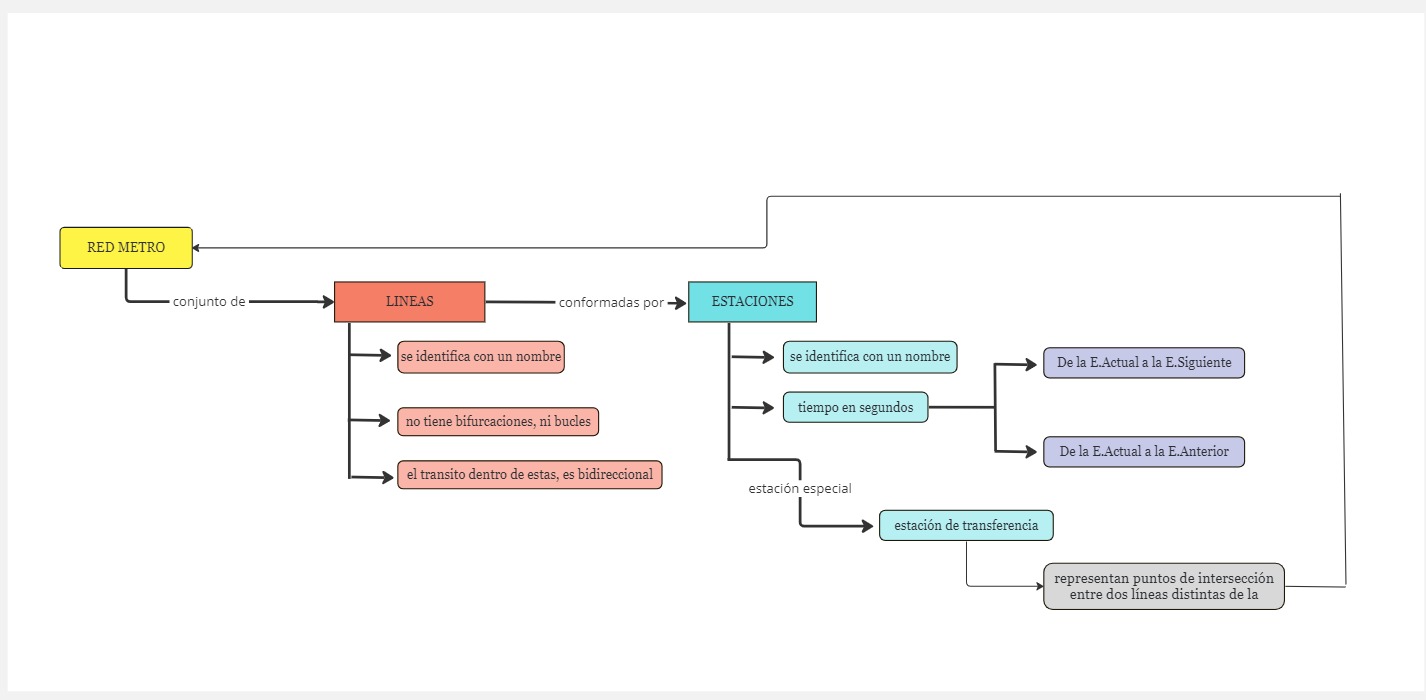
* **Float tiemposiguiente:** Tiempo que se demorara de la estación actual a la siguiente estación.
* **Float tiempoAnterior:** Tiempo que se demorara de la estación actual a la estación anterior.
* **String nombre**: Nombre de la estación.
* **String línea:** Linea a la que pertenece.
* **Int numEstacion:** Indice en orden de la estación con respecto a las otras en el arreglo.
* **Int numTransferencia**: Numero de transferencia que tiene la estación
* **Estacion\*:** arreglo de tipo Estacion, que contendrá las estaciones de transferencia de una linea, este arreglo siempre aumenta pero nunca disminuye

**Para la clase “estacion” hemos creado los siguientes métodos:**

**-Encapsulamiento:** La palabra “encapsulamiento”, se está utilizando para generalizar la creación de **setters, getters y constructores.**

**¿Cuántos objetos tendrá la clase “estacion”?**

Esta clase se instanciará de manera indefinida ya que depende de la cantidad de elementos que defina el usuario

1. **Diagrama simplificado de las condiciones del problema.**

**Condiciones a tener en cuenta por parte del problema:**

1. Agregar una estación a una línea, en los extremos o en posiciones intermedias.
2. Eliminar una estación de una línea. No se pueden eliminar estaciones de transferencia.
3. Saber cuántas líneas tiene una red Metro.
4. Saber cuántas estaciones tiene una línea dada.
5. Saber si una estación dada pertenece a una línea específica.
6. Agregar una línea a la red Metro.
7. Eliminar una línea de la red Metro (sólo puede eliminarse si no posee estaciones de transferencia).
8. Saber cuántas estaciones tiene una red Metro (precaución con las estaciones de transferencia).

Teniendo en cuenta el esquema y las condiciones anteriores, hemos diseñado un plan de desarrollo con el fin de resolver el problema planteado en este desafío.

Se pondrán las ideas a manera de ítems; en donde cada uno de ellos, corresponderá a la descripción de las estrategias que se utilizarán para resolver cada una de las tareas.

**Estrategia de solución para el inciso “A”:**

Primero podemos decir que nuestra primera opción fue usar una estructura nodal como una lista doblemente ligada, pero nos dimos cuenta que era un problema al momento de implementar, entonces decidimos trabajar con arreglos e índices, entonces vamos a agregar una instancia del objeto Estacion en el atributo “Estaciones” de la instancia correspondiente Linea, actualizando los tiempos correspondientes y agregando 1 a el atributo numEstaciones de la instancia de la linea

**Estrategia de solución para el inciso “B”:**

Verificamos que no tenga estaciones de transferencia y luego, usando los arreglos dinámicos vamos a actualizar el arreglo “Estaciones” de la instancia correspondiente de línea y sin incluir las estación que nosotros decidimos eliminar y actualizamos los tiempos de la estación anterior y siguiente de la actual, también restamos uno a el atributo numEstaciones de la instancia de la linea

**Estrategia de solución para el inciso “C”:**

Usando el atributo, “numLineas“ de la instancia de la red, nos dará el numero de líneas que posee la red

**Estrategia de solución para el inciso “D”:**

Usando el atributo, “numEstaciones“ de la instancia de la linea, nos dará el numero de estaciones que posee la linea

**Estrategia de solución para el inciso “E”:**

Comparamos el atributo “linea” de la instancia de Estacion con el nombre de la línea dada

**Estrategia de solución para el inciso “F”:**

Partiendo de una estación vamos a volver esa estación una estación de transferencia y luego vamos a agregarle 1 al atributo “numLineas” de la instancia de la red, y agregamos la instancia de la línea al arreglo “Lineas” de la instancia de la red.

**Estrategia de solución para el inciso “G”:**

Verificamos que no tenga estaciones de transferencia y luego la sacamos del arreglo de líneas y restamos 1 al número de líneas de la instancia de la red