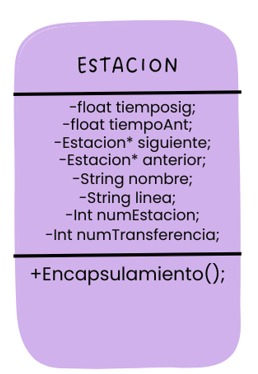
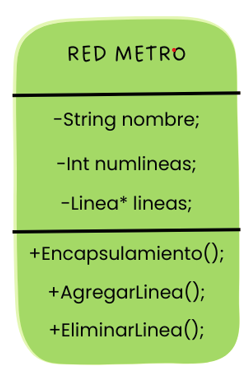
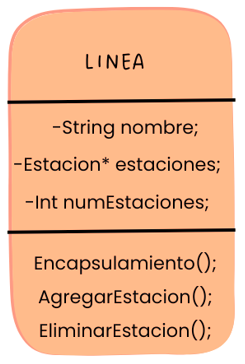
**Proceso de análisis y documentación - Desafío 2.**

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

****

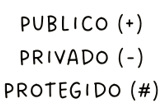
**2…n**

**1**

****

**n**b

**1**

****

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

**Clase “RED METRO”:** Hemos decidido crear una clase llamada “red metro” 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 “red metro” hemos creado los siguientes atributos:**

* **String nombre:** Este atributo es de tipo string porque almacenará un nombre; es decir, un conjunto de caracteres; en este caso particular, 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”.
* **Linea\* líneas:** Almacenara las líneas que contiene la red, siendo un arreglo dinámico de objetos tipo Linea

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

* **Encapsulamiento:** Setters, getters y constructores.
* **AgregarLinea:** Con este método vamos a poder agregar una línea a la red, agregándola al arreglo mencionado anteriormente.
* **EliminarLinea**: Con este método vamos a eliminar una línea, sacándola del arreglo (solo puede ser la primera linea ya que es la única que no tiene linas de transferencia)

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

**Clase “LINEA”:** Hemos decidido crear una clase llamada “ Linea” 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 “línea” hemos creado los siguientes atributos:**

* **String nombre:** Este atributo es de tipo string porque almacenará un nombre; es decir, un conjunto de caracteres; en este caso particular, el nombre que el usuario le pondrá a cada una de las líneas ( objetos) que contendrá la “red metro”.
* **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 compondrán cada línea(objeto) de la “red metro”.
* **Estacion\* estaciones:** Arreglo dinámica de tipo Estacion, este contendrá las estación que componen esta linea

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

* **Encapsulamiento:** Getters, settters y constructores.
* **AgregarEstacion:** Con este método vamos a poder agregar estaciones a un a línea determinada (al arreglo dinamico).
* **EliminarEstacion:** Este método fue creado con la finalidad de eliminar una estación en específico.

**¿Cuántos objetos tendrá la clase “linea”?** La clase línea tendrá la cantidad de objetos que contenga el atributo definido como **Int numlineas** en la clase “red metro”.

**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.

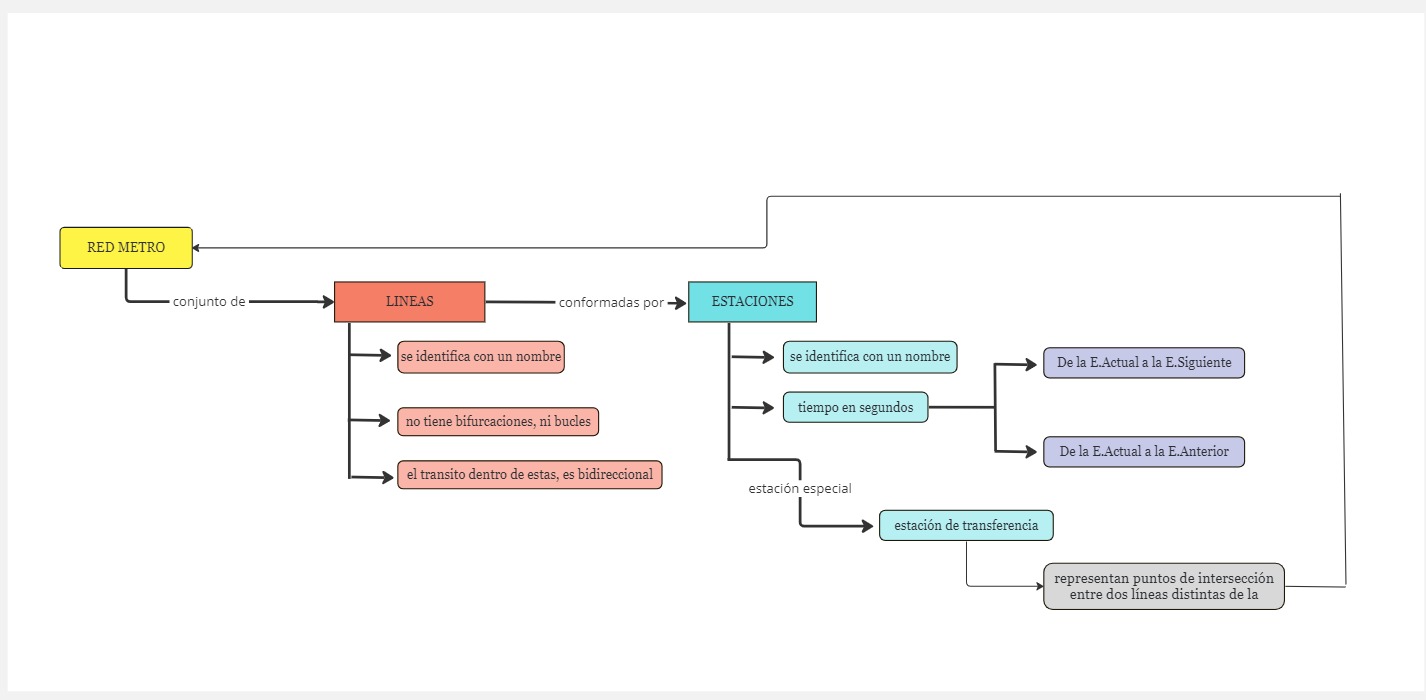
**Para la clase “estacion” hemos creado los siguientes atributos:**

* **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:** Línea 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 Estación, que contendrá las estaciones de transferencia de una estación, este arreglo siempre aumenta, pero nunca disminuye.

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

**-Encapsulamiento:** Setters, Getters, constructores.

**¿Cuántos objetos tendrá la clase “estación”?** 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 implementarla, 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 línea.

**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 la 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 línea.

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

Usando el atributo, “numLineas” de la instancia de la red, nos dará el número 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 línea.

**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.