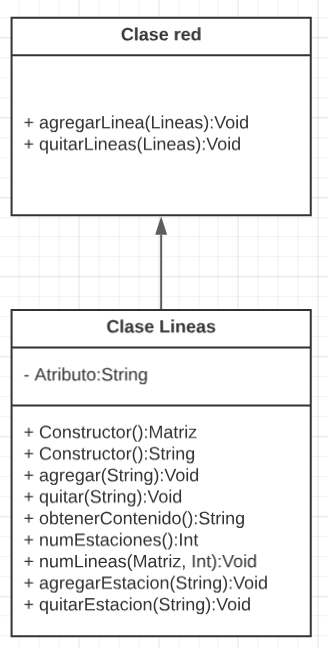
Victor Villalobos y Sebastian Rincon

1. **Análisis:**

Para la estructuración de una solución para este problema, se optó por adoptar la estructura del código previamente elaborado para la práctica 4 del laboratorio, puesto que las bases del funcionamiento del código son las mismas. Así mismo, y dada la imposibilidad de utilizar contenedores, se decidió por el uso de strings para almacenar la información de las estaciones pertenecientes a una misma línea, de esta forma cada string sería guardado como un objeto de la clase línea. También se optó por almacenar todos los objetos en una matriz de nx2, donde n equivale al número de líneas, y en donde los elementos de la primera columna son el nombre de la línea que representa el objeto string almacenado en la segunda columna, permitiendo de esta manera la facilidad de revisar y extraer información específica de una línea sin la necesidad de iterar sobre todos los objetos para encontrar la línea deseada.

1. **Diagrama de clases:**



1. **Algoritmos implementados:**

**Métodos clase líneas:**

modificarEstacionMas():

string -> void

Método que nos servirá de ayuda para el quitar y el agregar estaciones, ya que recorrer todo el contenido de la línea y hace los cambios de posición y de los tiempo de la estación anterior y estación siguiente.

* agregar():

string -> void

Este método se encarga de agregar un string al final del objeto/string sobre el cual se use.

Se le ingresa un string el cual representa la información que se agregará a la línea.

Se ingresa solo el nombre de la estación, desde el mismo método agarramos su nueva posición y tiempos, en ella se invoca el método modificarEstacionMas().

* quitar():

string -> void

Este método se encarga de quitar una sección del objeto/string sobre el cual se use.

Se le ingresa un string el cual representa la información que se le quitará a la línea.

Se ingresa solo el nombre de la estación, desde el mismo método agarramos su posición y tiempos, en ella se invoca el método modificarEstacionMas().

* obtenerContenido():

void -> string

Este método se encarga de extraer el contenido del objeto sobre el que se use.

Lee el contenido del objeto, crea una copia, y regresa dicho string.

* numEstaciones():

void -> int

Este método se encarga de contar las comas que se encuentren dentro del objeto.

Cuenta las comas y retorna el número de comas (que es equivalente al número de estaciones)

* numLineas():

Lineas, int -> void

Este método se encarga de contar la cantidad de líneas actualmente registradas al recorrer la matriz con las líneas, revisando si el contenido de la primera columna tiene algo en su interior y sumando a un contador.

Recibe el objeto de matriz para poder recorrer su primera columna y un entero con el tamaño total de la matriz para poder recorrerla por medio de un for.

Este método se sobre cargo para que a la vez hubiera otro método que nos diera el número de líneas en uso

**Funciones:**

* resize():

string, char -> string

Esta función se encarga de recortar un string, buscando la posición del char ingresado, y guardando el contenido del string posterior a dicho char.

Recibe el string a ser recortado y el char a tener en cuenta, y retorna el string recortado.

Se sobrecargó la función.

* grupito():

string, char -> string

Esta función se encarga de recortar un string, buscando la posición del char ingresado, y guardando el contenido del string previo a dicho char.

Recibe el string a ser recortado y el char a tener en cuenta, y retorna el string recortado.

Se sobrecargó la función.

* pedirLinea():

Lineas, int -> string

Esta función se encarga de realizar todo el proceso de solicitud de una línea al usuario, guardando el nombre de la línea ingresada y comparándolo con los objetos en la primera columna de la matriz para confirmar que sí se encuentre registrada, de no estarlo, vuelve a pedir al usuario que ingrese el nombre de una línea y se repite el ciclo hasta que una línea registrada sea ingresada.

Recibe el objeto de matriz para poder recorrer su primera columna y un entero con el tamaño total de la matriz para poder recorrerla por medio de un for.

* encontrarEstacion():

string, string, puntero a tring -> bool

Esta función se encarga de recorrer el string que contiene toda la información de una línea en concreto, y por medio de las funciones resize y grupito, va recortando poco a poco el string intentando encontrar la estación ingresada.

Recibe un string que contiene la línea a recorrer, un string con el nombre de la estación a encontrar, y un puntero a string que, en caso de encontrar la estación, guarda en ese string la posición de la estación. Finalmente retorna true en caso de encontrar la estación, y false en caso de no hacerlo.

cortarCadena():

Esta función se encarga de recibir toda la información de una linea y busca en qué posición se encuentra la estación que estamos buscando y devuelve la nueva cadena desde la posición que se encontró

1. **Problemas de desarrollo:**

El primer y mayor problema que afrontamos durante la elaboración del código fue a la hora de idear la manera de almacenar la información de las líneas y las líneas en sí, ya que al no estar permitido el uso de contenedores, hubo que pensar cómo elaborar esta parte del código. La solución que ideamos fue que toda la información de una línea junto con sus estaciones fueran almacenadas en un string, siendo este el objeto de la clase línea, mientras que la información se guardaría en una matriz perteneciente a la misma clase línea.

También tuvimos dificultad al momento de pensar como elaborar los procesos para agregar y quitar estaciones, así como la extracción de los tiempos de llegada y salida de las estaciones de los objetos.

1. **Evolución de la solución:**

En un inicio, se pensó en la posibilidad de guardar todos los datos de las estaciones dentro de mapas (desde que este fue el método utilizado en la práctica 4), pero una vez enterados que esto no sería posible, se pensó utilizar arreglos de tipo string para almacenar todo. En un principio se optó por esta opción, sin embargo a la hora de desarrollar el código nos dimos cuenta que esto no era del todo posible, además de la imposibilidad de distinguir entre objetos al ser creados de manera dinámica y, por ende, no tener nombres distintivos. Ante esta problemática, se decidió hacer 2 cosas, lo primero fue crear una matriz de la misma clase Lineas, la cual contara con 2 columnas y donde se pudiera almacenar los objetos, y los segundo fue el crear un segundo objeto aparte del string con el contenido de las líneas, que representara el nombre de su respectiva línea, y el cual sería almacenado a la par de su línea dentro de la matriz.