Universidad Nacional de la Patagonia San Juan Bosco

Programación Orientada a Objetos

Instancia supervisada de Formación Práctica Profesional

Aplicación para la búsqueda de recorridos y gestión de redes de subtes.

Fecha: 16/12/2020

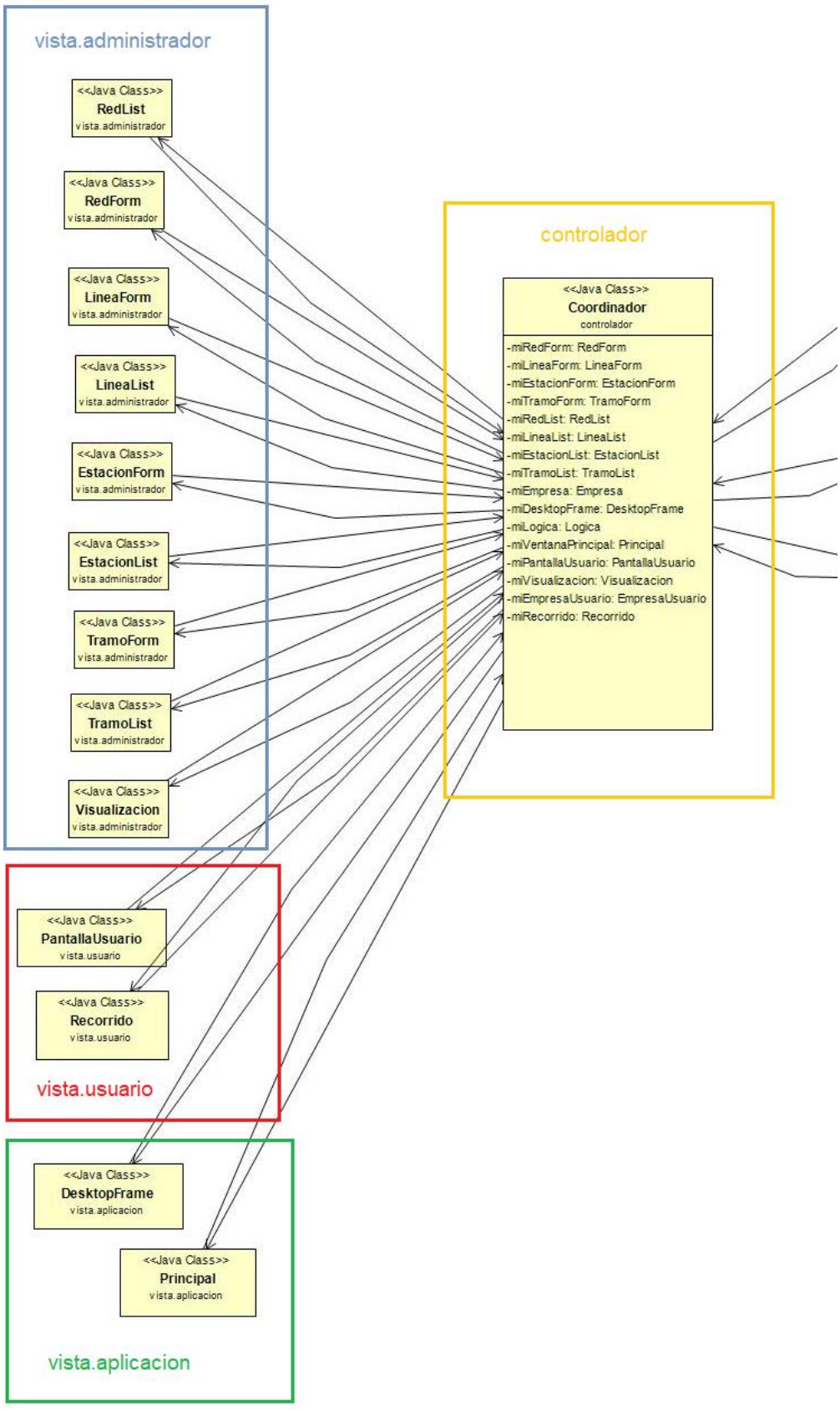
Alumna: Nicole Gianella Alvarado

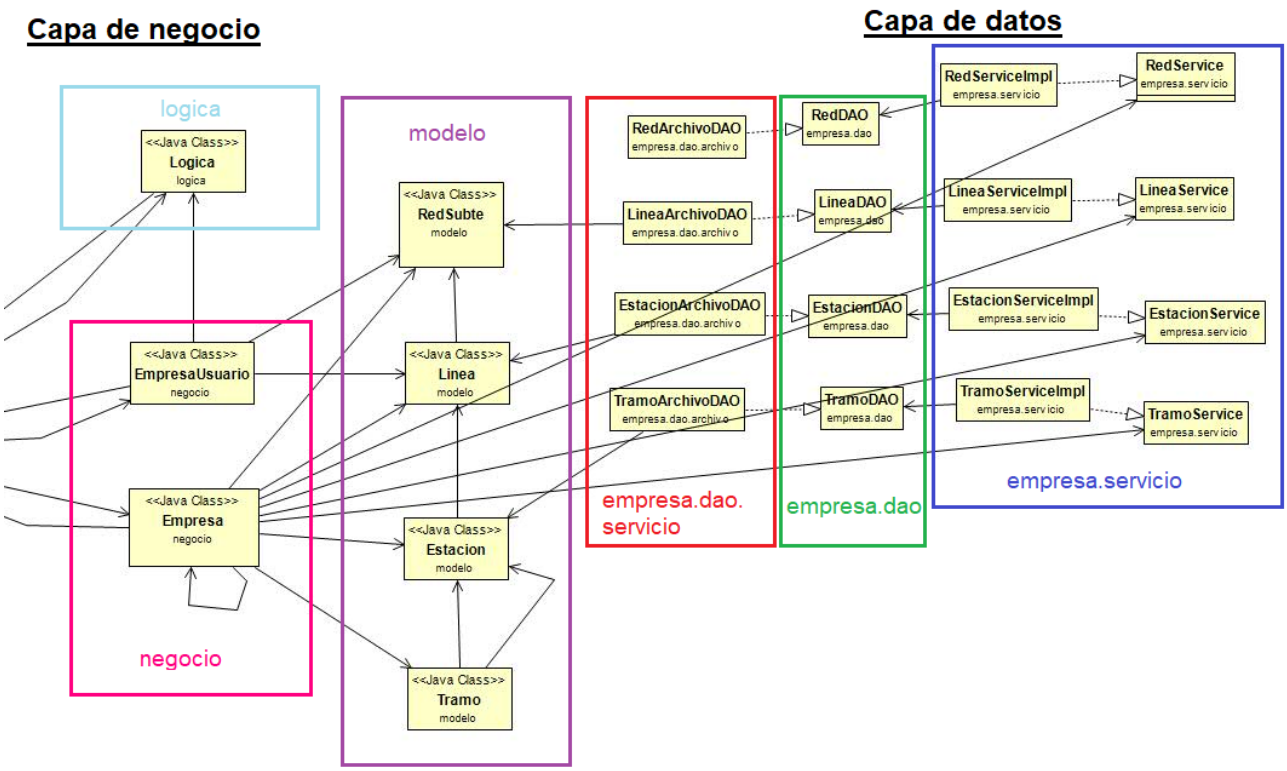
Alcance del proyecto:

El proyecto consiste en una aplicación de simulación de subtes, donde las personas pueden ingresar tanto como usuarios para consultas o administradores para la gestión de redes. En función de consulta, uno podrá seleccionar buscar recorridos (recorrido más rápido, recorrido con menor número de transbordos (combinaciones de líneas), recorrido menos congestionado), desde una estación origen a una estación destino que indique el usuario, anteriormente podrá elegir la red de subte de la cual quiere la información junto a las líneas asociadas a dicha red. Por otro lado, los administradores tendrán la posibilidad de modificar, customizar desde una interfaz gráfica, incluso eliminar redes, las líneas, las estaciones y los tramos que existen, incluyendo la posibilidad de crear nuevas redes de subtes y recorridos.

Diseño de capas de la aplicación (con sus respectivos paquetes):

Capa de interfaz





Estructuras de datos y su implementación:

La aplicación consta de dos partes, una parte funcional para la administración de redes de subte y otra parte para las consultas por parte de un usuario.

Para el primer caso, se trabajó con las siguientes estructuras de datos:

- ArrayList: se crearon ArrayList's para el almacenamiento de cada uno de los objetos. Existirá una relación con el modelo, de tal forma que guardará los objetos para realizar los ABM's (para poder agregar, borrar o modificar). Por ejemplo, en mi clase Empresa.java tengo un ArrayList para cada uno de los objetos del paquete modelo.
- HashTable: otra estructura de datos que se utilizó para identificar datos mediante una llave o clave, en este proyecto, se los reconoce por códigos (el código de una red, línea, etc.).
- TreeMap se utilizaron para almacenar datos junto a claves que están asociadas a un único valor, guardando así partes del grafo "subte", como líneas, estaciones y congestiones. En el último caso, se creó un TreeMap, para contener los niveles de congestión como claves, y los multiplicadores asociados como valores, lo cual sirvió aparte de calcular el recorrido menos congestionado, tener un código más extensible.

Para la simulación de consultas (usuario), se decidió trabajar con un Grafo general por ser una de la mejor posible representación de una red de subte, al poder almacenar en los vértices las estaciones, y en los arcos, una relación entre dos estaciones, en este caso un tramo, y se implementó la clase Tramo.java, el cual contiene parámetros generales (duración, tipo (nivel de congestión), etc.).

- Grafo: se usaron para almacenar las diferentes redes de subtes de acuerdo a las preferencias del usuario (recorrido más rápido, menor número de transbordos, menos congestionado). Ej.: para recorrido más rápido, se generó un grafo el cual copia los vértices que son de tipo Estacion.java del grafo general, cambiando los arcos por enteros que representan la duración en minutos entre dos estaciones continuas.
- List: como el método shorttestPathList retorna una lista posicional de estaciones, con el recorrido generado, los métodos encargados de la impresión de los distintos recorridos la necesitaban como parámetros.

Patrones de diseños utilizados:

Para mi proyecto he implementado cinco patrones de diseño (FACTORY METHOD, SINGLETON, DAO, VO, MVC).

- Factory Method: el patrón permite la creación de objetos de un subtipo determinado a través de una clase Factory. Fue utilizado para la búsqueda de instancias y saber a dónde llamar a las clases correspondientes del patrón DAO.

Por ejemplo, en este proyecto, cuando se crea algo de tipo RedArchivoDAO.java, para saber cuál es la instancia que tiene que hacer, se llama al FACTORY y se obtiene la instancia, la cual leyó de un archivo de propiedades, en tal archivo se encuentra la ruta de la clase de la instancia que tenía que seleccionar.

- Singleton: permite la creación de una única instancia de clase permitiendo que sea global para toda la aplicación. En este caso, Empresa.java va a encargarse de todas las operaciones básicas para relacionarse con el modelo. Por ejemplo, como dicho anteriormente, almacenara los objetos del modelo, para luego poder agregarlos, modificarlos o eliminarlos. En esta clase aplico el patrón para darle solamente a una única instancia de una clase, el poder de crear, modificar o eliminar objetos.

- DAO (Data Access Object): se usa para acceder a los datos desde diversas fuentes, manteniendo de forma abstracta la manera en la que se accede a ellos. Se crean interfaces de servicios (RedService.java, LineaService.java, EstacionService.java, TramoService.java) las cuales definen que comportamiento van a tener las implementaciones. En la implementación (RedServiceImpl.java, LineaServiceImpl.java, EstacionServiceImpl.java, TramoServiceImpl.java) se define desde que fuente se van a acceder a los objetos, en este caso, archivos de acceso aleatorio, y en la concreción de esa interfaz se crea una instancia DAO (RedDAO.java, LineaDAO.java, EstacionDAO.java, TramoDAO.java) y se define como va a acceder a los objetos como "ABM" (RedArchivoDao.java, LineaArchivoDao.java, EstacionArchivoDao.java, TramoArchivoDao.java).

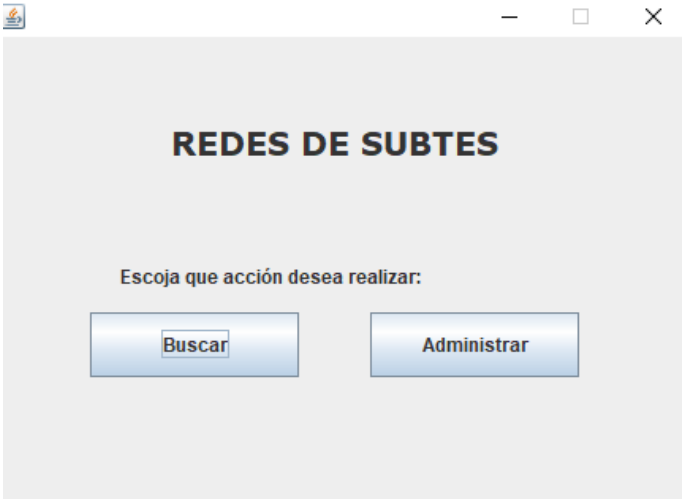
- VO (Value Object): se realiza una especie de implementación del patrón DAO y el patrón VO, el DAO, mencionado anteriormente, para el acceso a los objetos (datos de información) y en

este caso, el patrón VO, hace referencia a las vistas (RedList.java, LineaList.java, EstacionList.java, etc.), la visualización de los objetos creados mediante el DAO.

- MVC (Modelo Vista Controlador): separa la información y lógica de negocio de la parte visual de la aplicación y la lógica que gestiona las relaciones y eventos del sistema. Sin la implementación del patrón MVC, el patrón VO se conectaría directamente con el patrón DAO. Las vistas no llamarían a un coordinador, sino que crearían objetos de tipo RedService, LineaService, EstacionService. Gracias a la implementación del patrón, se llama a un coordinador (como se aplicó Singleton y Factory Method, puede llamar tanto a Empresa.java, EmpresaUsuario.java, Logica.java), para conectar todas las clases y no estar creando instancias de todas las clases en cada una de ellas. Se respeta el manejo de vistas y modelo.

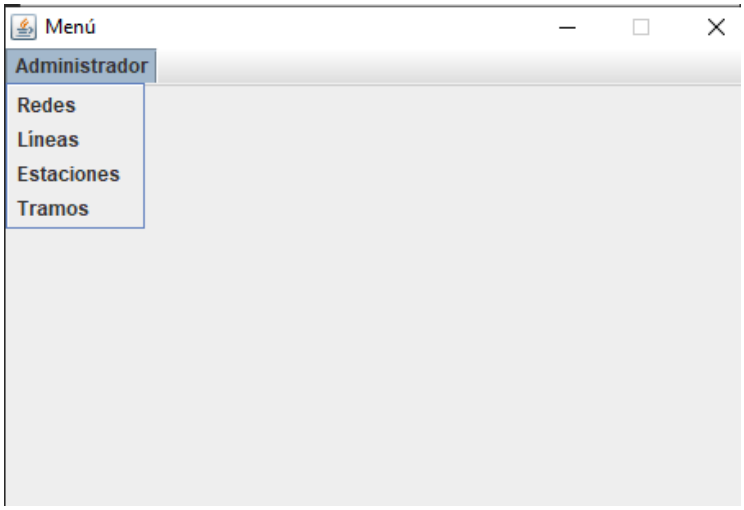
Manual de usuario:

Al iniciar el programa, se mostrará la siguiente ventana principal, la cual contiene dos botones, el botón BUSCAR para los usuarios, realizar la búsqueda de recorridos y el botón ADMINISTRAR para los administradores que deseen customizar o crear una nueva de red de subte.



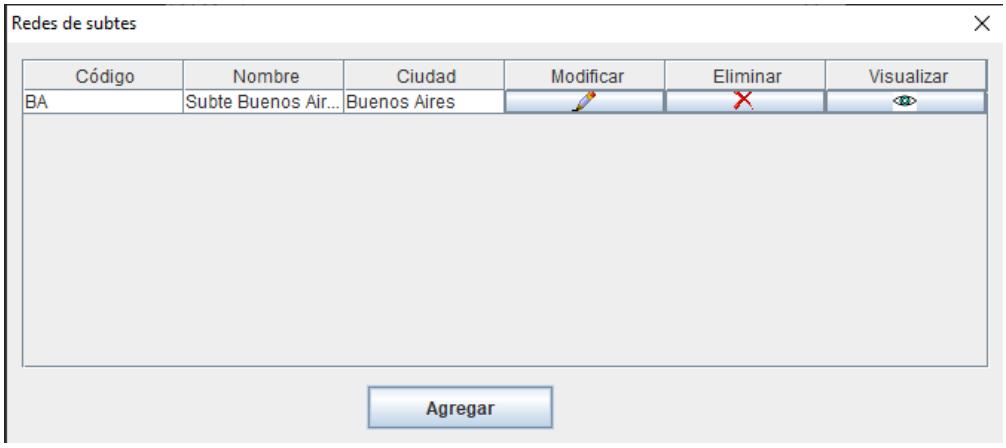
Si el evento pulsado fue el botón BUSCAR, entonces se abre otra ventana, en la cual, el usuario primero deberá seleccionar una red de la lista desplegable, seguido deberá seleccionar las líneas que tengan asociadas las estaciones destino y origen del cual se desea el recorrido. Para concretar la búsqueda, deberá pulsar el botón BUSCAR para mostrar los tres tipos de recorridos (recorrido más rápido, recorrido con menos combinaciones de líneas y el recorrido menos congestionado), los cuales se mostrarán en una nueva ventana.

De caso contrario, si el usuario decide entrar como administrador, en la ventana principal al pulsar el botón ADMINISTRAR, aparecerá una ventana la cual contiene un menú de todos los componentes de una red, incluyéndola.



El usuario podrá elegir cualquiera de las opciones, preferiblemente se aconseja comenzar creando una red nueva, ya que el programa viene cargado previamente con información de la red de subte de Buenos Aires.

La tabla de las redes de subte contiene además la opción de visualizar una red, se mostrará un listado de la red de todas sus líneas, estaciones y especificando las combinaciones de líneas.



Igualmente, el administrador podrá realizar cambios limitados en los distintos formularios, pero, por ejemplo, no podrá eliminar una línea que contenga estaciones asociadas a ella.

Errores detectados:

Cuando se comenzó a ejecutar las pruebas para localizar errores, uno surgió al momento de agregar una nueva línea en el ABM de líneas. El agregado visualmente se vio, pero al querer eliminarla o modificarla, me percaté de que no sucedía ninguna de las opciones seleccionadas. Revisando el archivo lineas.dat, noté que se guardaba tal línea con el código de red de la primera red cargada en el cbListRedes (JComboBox perteneciente LineaList.java). Haciendo un seguimiento del código, detecté que el problema era una poscarga de las redes en el cbListRedes. La solución que implemente fue que al método agregarLineaForm(), que se encontraba en Coordinador.java, cargarle las redes nuevamente con una invocación del método cargarRedes() de LineaList.java.

Finalmente, para enlazar los JComboBox's de redes de LineaList.java con el de LineaForm.java, a la hora de llamar al formulario correspondiente, le pasaba la red de la línea a agregar como parámetro, con la cual se debía setear el JComboBox.

Posibles mejoras:

- La aplicación para cargar datos podría tener la opción de modificar los parámetros y editar el archivo.properties, mediante la creación de un Form en donde se mostrarían los valores actuales y que tenga la opción de permitirlos cambiar y grabar por otros.
- Se podría agregar en las listas desplegables el autocompletado de palabras, o sea que indicando algunas letras que se permita rápidamente encontrar la estación que se desea seleccionar en la consulta.
- Se podría agregar también congestiones para las líneas.

Extensiones:

- Se podría extender la interfaz gráfica agregando un mapa geográfico con las estaciones y las líneas, donde se pueda visualizar cada tipo de recorrido, en vez de mostrar sólo nombres de las estaciones por las que pasa cada subte.

- El sistema podría tener la opción de internacionalización (por ejemplo, en español/ inglés/ portugués).
- En la visualización de los recorridos, se podría indicar si el recorrido está habilitado para personas con discapacidad.

Conclusión:

En general, la realización del proyecto ha sido todo un desafío para mí, ya que hubo temas vistos en clase que tuve que reforzarlos para poder implementarlos, como lo fue el caso de “Hilos”.

A pesar de ello creo que el proyecto se ha solventado bien, tratando de seguir con todo lo aprendido y visto en la cursada.

Gracias a la ayuda de las clases de consultas virtuales, indicaciones y de las devoluciones de entrega del seguimiento del proyecto por parte de los profesores, me dieron otro punto de vista
de cómo plantear cada parte del proyecto u observaciones donde surgían fallos en el programa.

Una jugada en contra que tuve fue el hecho de estar realizando dos proyectos de formación práctica profesional al mismo tiempo. El lado bueno de ello, fue el adquirir más experiencia de lo visto en este ámbito y una mejor preparación para futuros proyectos. En mi opinión, si hubiera tenido más tiempo del dado, probablemente haya implementado más cosas, como el hecho de trabajar con una base de datos, tener la opción de internacionalización, entre otras. Tal vez no sean temas tan complicados de implementar, pero al no verlos bien en detalle, decidí no arriesgar y poner en práctica todos los conocimientos vistos en la materia Programación Orientada a Objetos.