Documentación: TP Integrador POO2

Terminal Portuaria

Nombre y apellido	Correo
Sanabria Cristian	crissanabria1995@gmail.com
Ezequiel Karottupullolil	ezequielkarottupullolil@gmail.com
Jeremías Ruiz	jruiz19@uvq.edu.ar

Índice

Decisiones de diseño

Introducción

Antes de comenzar a programar se interpretó el enunciado en su totalidad y se debatió cuales iban a ser los principales objetos, como así también, las tareas que debían realizar cada uno de ellos. En ese momento se decidió que los objetos `Terminal`, `Naviera`, `Circuito`, `Tramo`, `Orden` y `Viaje` serían los que resolverían la mayoría de las funcionalidades, pero a medida que se avanzó en el desarrollo del trabajo fueron surgiendo más objetos y patrones que cumplen diferentes roles para poder llevar a cabo todas las funcionalidades solicitadas en el enunciado en su totalidad, lo cual será desarrollado a continuación.

Terminal

Para la `TerminalGestionada` se decidió que la misma implemente una interfaz llamada `Terminal`, la cual permite generar las terminales que no son la gestionada, de esta forma se puede modelar de forma correcta la interacción en los tramos de un circuito, ya que un tramo está compuesto de una terminal de origen y una de destino, donde alguna o ambas pueden ser una terminal que no sea la gestionada.

Orden

En nuestro modelo inicial teníamos dos clases separadas por tipo de Orden, la clase **OrdenImportación** y la clase **OrdenExportacion** que heredan de una misma clase Orden pero finalmente en una reunión se optó por un modelo donde existiera una única clase Orden y ella determinara si era de Importacion o Exportacion.

Esto nos trajo mayores complicaciones mientras fue avanzando el proyecto, como al momento de notificar a los clientes la llegada del Buque o retirada del Buque se necesito la implementación de dos métodos diferentes que podrían haber sido un solo método implementado por las subclases OrdenExportacion y OrdenImportacion.

Búsqueda de rutas marítimas

Para la búsqueda de rutas marítimas se tomó la decisión de utilizar el patrón Composite, ya que el enunciado pedía realizar búsquedas con combinaciones lógicas mediante los operadores AND y OR, por lo que se crearon dos clases, una llamada `AND` y otra `OR`, las cuales cumplen el rol de *Composite* e implementan una interfaz llamada `Filtro` la cual cumple el rol de *Component* y define el método común *filtrar(List<Viaje>)*, esto permite generar los filtros que solicita el enunciado,

los cuales cumplen el rol de *Leaf*, además permite que en un futuro se puedan agregar otros tipos de filtros sin modificar la estructura actual.

Esto permite que las navieras puedan realizar la búsqueda sobre su lista de viajes mediante el método *buscarViajesQueCumplan(List<Filtro>)*, el cual recibe por parámetro una lista de filtros. Por otro lado, la terminal gestionada tiene un método *buscarRutasMaritimasQueCumplan(Lis<Filtro>)*, el cual recorre la lista de navieras que tiene la `TerminalGestionada` y por cada `Naviera` utiliza el mensaje anteriormente descrito.

Buscar el “mejor” circuito

Para la búsqueda de mejor circuito se tomó la decisión de utilizar el patrón Straetegy, ya que el enunciado pedía que el concepto de “mejor” debería poder ser seteado y cambiado dinámicamente en la `TerminalGestionada`. Por lo que se creó la interfaz `IBusquedaCircuito` que cuenta con un metodo llamado *seleccionarMejor(List<Naviera>,Terminal,Terminal)*, el cual recibe por parámetro una lista de navieras perteneciente a la `TerminalGestionada`, una terminal de origen y una terminal de destino. Dicha interfaz es implementada por las diferentes estrategias de búsqueda, las cuales son `MenorTiempoADestino`, `MenorPrecioADestino`, `MenorCantidadTerminalesIntermedias`.

Esto permite que la búsqueda sea llevada a cabo por cada uno de los objetos anteriormente nombrados, los cuales consumen la lista de navieras que la `TerminalGestionada` les provee. Cabe aclarar que los métodos de búsqueda específicos son responsabilidad exclusiva del objeto `Circuito`, es decir, cada objeto de búsqueda se encarga de recorrer la lista de circuitos que tiene cada `Naviera` de la `TerminalGestionada`, y por cada uno solicita el método correspondiente a lo que se busca, por ejemplo para la búsqueda de menor tiempo la clase `Circuito` cuenta con un método llamado *costoTotalDesdeHasta(Terminal,Terminal)*, el cual permite saber el costo total de ir de una terminal origen a una terminal destino.

Articulación de operaciones: Buque, Terminal y Ordenes

El modelo fue pensado de forma tal, que las notificación a los clientes por parte de la terminal estuviera completamente determinadas por el Viaje que tenía el Buque al momento de notificar a la terminal.

El buque conoce únicamente el Viaje y el Tramo que está recorriendo en ese mismo momento, al pasar al estado **Inbound** le notifica a la terminal que tiene como destino actualmente por el mensaje *`avisarLlegada(Buque)`*, la terminal va a recibir el mensaje *`notificarLlegada(Buque)`* y va a notificar a sus Consignees la inminente llegada del Buque con su respectiva carga.

La terminal notificar a todas las Ordenes por el metodo *`notificarLlegada(Buque)`* y las órdenes son las que determinan si deben o no notificar al cliente asociado a la Orden.

Al igual que antes, cuando el Buque pasa al estado **Outbound** y se aleja más de 1 kilómetro de la terminal origen el Buque le notifica a la terminal su partida por el mensaje *`avisarPartida(Buque)`*, la terminal va a recibir el mensaje *`notificarPartida(Buque)`* y va a notificar a sus Shippers la partida del Buque con su respectiva carga.

Shipper, Consignee y Cliente

En nuestro modelo el Shipper y el Consignee están modelados de forma indirecta por la clase Cliente. Nos referimos a que están modeladas de forma indirecta en la aplicación porque un mismo Cliente puede ser Consignee y/o Shipper dependiendo de la Orden específica.

Un cliente es identificado como Shipper si la Orden tiene asociado un origen definido, en contraparte un Cliente goza de condición de Consignee dentro de la Orden si está tiene un destino definido.

Cuando un Cliente registra una Orden en la TerminalGestionada dependiendo de qué mensaje se use la Orden se le va a asignar un destino, o un origen.

`exportarHacia(Orden,Terminal)` → El cliente goza de condición de Shipper, Orden utiliza la terminal actual como origen
`importarDesde(Orden,Terminal)` → El cliente goza de condición de Consignee, Orden utiliza la terminal actual como destino

Esto es importante porque la clase Orden (única para exportación e importación) asume la responsabilidad de gestionar su propio ciclo de notificaciones, determinando cuándo sí y cuándo no notificar al cliente.

Camion, Chofer y Empresa Transportista

La empresa Transportista no está modelada dentro de la aplicación, principalmente porque su única finalidad era almacenar Camiones y Choferes, está decisión de modelo nos dio libertad para situaciones donde los Clientes buscarán hacer retiros particulares sin el uso de las Empresa Transportistas.

Facturación

Consecuencia de no haber creado una jerarquía de clase en Cliente, para Consignee y para Shipper se creó una Jerarquía de clase en Facturación, siendo **FacturaConsignee** la específica para las órdenes de tipo importacion y **FacturaShipper** para las órdenes de tipo exportación.

Patrones de diseño y roles

Composite

Se utilizó este patrón para la búsqueda de rutas marítimas, ya que permite tener filtros anidados y una composición entre filtros del tipo AND y OR.

Los roles son:

- **Component**: *Filtro*.
- **Leaf**: *FiltroFechaSalida*, *FiltroFechaLLegada*, *FiltroPuertoDestino*.
- **Composite**: AND, *OR*.
- **Client**: *TerminalGestionada*, *Naviera*.

Strategy

Se utilizó este patrón para la búsqueda del mejor circuito, ya que permite que el concepto de “mejor” sea seteado y cambiado de forma dinámica en la *TerminalGestionada*.

Los roles son:

- **Context**: *TerminalGestionada*.
- **Strategy**: *IBusquedaCircuito*.

- ➔ **ConcreteStrategies**: *MenorTiempoADestino*, *MenorPrecioADestino*, *MenorCantidadTerminalesIntermedias*.

State

El Buque implementa estados específicos en relación a su distancia respecto a la Terminal destino, Terminal origen y al proceso que se está llevando a cabo en la terminal.

Los roles son:

- ➔ **Contexto:** Buque
- ➔ **Estado:** EstadoGPS (Abstracta)
- ➔ **Estados concretos:**
  - ◆ **Outbound**
  - ◆ **Arrived**
  - ◆ **Working**
  - ◆ **Departing**
  - ◆ **InBound**

Visitor

Se utilizó este patrón para emitir reportes sobre el *Buque*.

Los roles son:

- ➔ **Visitor**: *Reporte*.
- ➔ **Concrete Visitor**: *ReporteMuelle, ReporteAduana, ReporteBuque*.
- ➔ **Element**: Reportable.
- ➔ **Concrete Element**: *Buque*.