Universidad Nacional de Quilmes

Departamento de Ciencia y Tecnología

PROGRAMACIÓN CON OBJETOS II

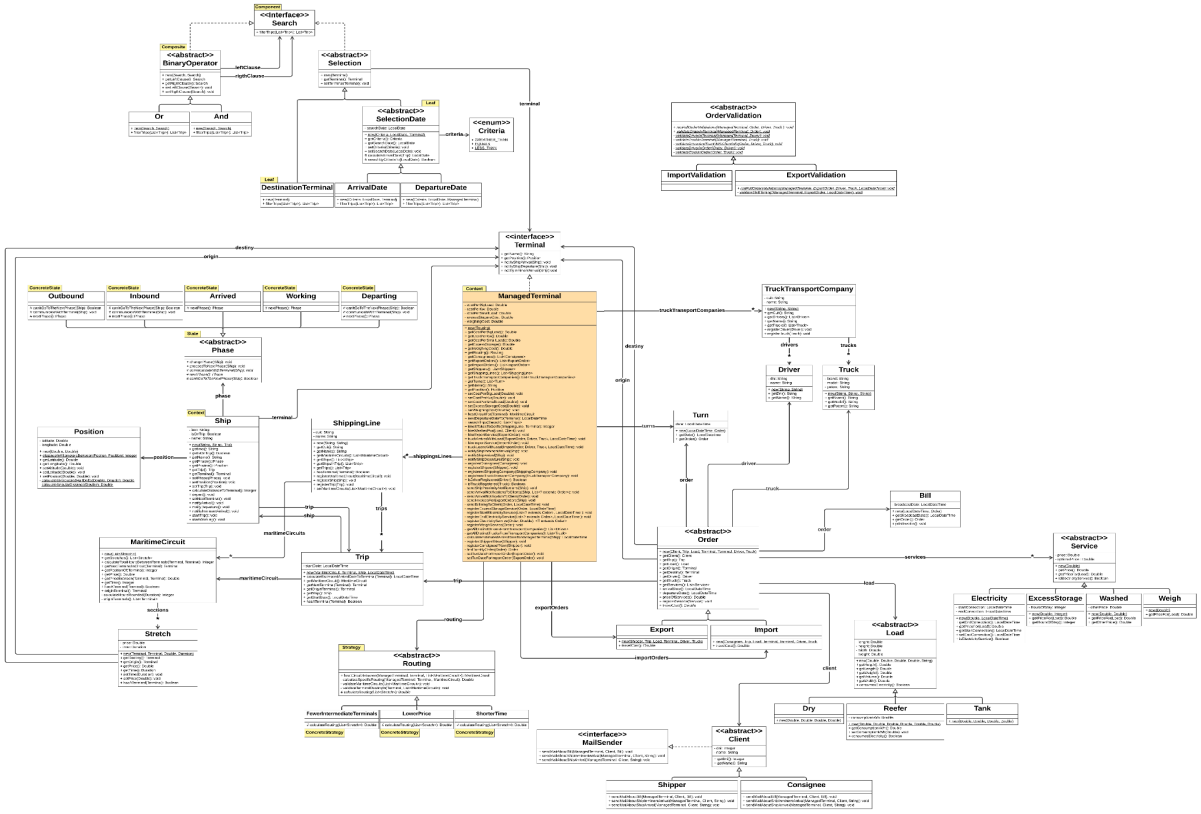
Trabajo Final | Terminal Portuaria

2° Cuatrimestre 2023

INTEGRANTES

- Besel, Alejandra | alejandrabesel@gmail.com
- Fascetta, Gabriela | gfascetta@gmail.com
- Ventoso, Yoel | yoel.ventoso@gmail.com

UML



DECISIONES DE DISEÑO

Terminal Gestionada

En la terminal gestionada, se propuso que la terminal contenga una lista de turnos, además de las órdenes (exportación e importación), ya que la terminal le otorgará un turno al shipper o consignado con la fecha que deberá llevar o retirar la carga a la terminal.

Comunicación con la terminal

Cuando el estado del barco sea Departing, se comunicará con la terminal, donde la terminal debe enviarle la factura con los desgloses al Shipper y Consignee. Nosotros solo lo implementamos para el Shipper, debido a que no corresponde mandarle la factura al Consignee, porque no retiró la carga de la Terminal Gestionada, y por lo tanto si se demora en retirar la misma, se debe cobrarle el servicio por “Almacenamiento Excedente.”

Servicios

En esta parte, se consideró que la clase abstracta Service contenga un mensaje que diga si es un servicio eléctrico. El motivo de esto surge de la necesidad de poder distinguir los

servicios, ya que solamente el servicio eléctrico se lo debe agregar cuyas cargas corresponden a Reefer porque son las únicas que consumen electricidad.

Carga

Es similar a lo anterior, ya que la clase abstracta Load contiene un mensaje que sabe decir si la carga consume electricidad o no, dado que se necesita poder distinguir las cargas de tipo Reefer de las demás.

SUPOSICIONES SOBRE EL DISEÑO

- Al crear una instancia de un circuito marítimo, el conjunto de tramos corresponden a uno válido, que consiste en un secuencia unidireccional y cerrada de terminales (donde la última se conecta con la primera).
- Al añadir una empresa de transporte en la terminal gestionada, los choferes y camiones de esa empresa, ya se consideran registrados en la terminal gestionada.
- Al crear una instancia de carga, sus dimensiones y su peso son positivos.

PATRONES DE DISEÑO

COMPOSITE

PARTICIPANTES

- COMPONENT
  - Search
- LEAF
  - DestinationTerminal
  - ArrivalDate
  - DepartureDate
- COMPOSITE
  - And
  - Or

USO DEL PATRÓN

Se utiliza principalmente para realizar búsquedas en los viajes gestionados por una compañía naviera. En este contexto, se aplican diversos criterios para filtrar los viajes que cumplen con condiciones específicas. Por ejemplo:

Destination Terminal (Terminal de Destino):

Se encarga de buscar los viajes que contienen la terminal de destino especificada. Esto permite identificar todos los viajes que tienen como destino la terminal deseada.

Arrival Date (Fecha de Llegada):

Filtra los viajes que llegan en una fecha determinada a un destino específico. Además, ofrece la flexibilidad de utilizar criterios de búsqueda como menor, mayor o igual para encontrar los viajes que cumplen con condiciones temporales específicas.

Departure Date (Fecha de Salida):

Busca los viajes que salen de la terminal gestionada en una fecha particular. Similar al criterio de llegada, también permite utilizar condiciones de búsqueda, como menor, mayor o igual, para encontrar los viajes según las fechas de salida deseadas.

STRATEGY

Utilizamos dicho patrón de diseño ya que entendimos que la terminal debía proveer la posibilidad de elegir las mejores rutas a los clientes en base a distintos conceptos. Al tener que proveer uno solo a la vez, el patrón Strategy nos permite ir cambiando entre las distintas estrategias para darle a cada cliente su mejor ruta dependiendo del precio, el tiempo, o las terminales de interés. El algoritmo correspondiente a cada estrategia se modela en una clase concreta independiente e intercambiable. De esta manera, podemos modificar cada una de estas estrategias sin alterar a las demás, cómo así también crear nuevas estrategias de rutas en un futuro, sin tener que cambiar el código ya existente. (Principio de open-closed)
Además, cada una de las clases es polimórfica ya que responden al mismo mensaje exhibiendo cada una un comportamiento diferente. (Tercer principio de SOLID)

PARTICIPANTES

- STRATEGY
  - Routing
- CONCRETE STRATEGY
  - FewerIntermediateTerminals
  - LowerPrice
  - ShorterTime
- CONTEXT
  - ManagedTerminal

USO DEL PATRÓN

Esta funcionalidad está diseñada principalmente para buscar el circuito marítimo óptimo para una terminal destino específica. Se implementan tres criterios clave:

Fewer Intermediate Terminals (Menos Terminales Intermedias):

Esta función busca el mejor circuito marítimo evaluando aquellos que tienen menos terminales intermedias en su ruta hacia el destino.

Lower Price (Precio Menor):

La búsqueda se realiza considerando el precio asociado a cada circuito marítimo. La funcionalidad prioriza y selecciona el circuito que ofrece el costo más bajo para alcanzar la terminal destino.

Shorter Time (Tiempo Menor):

Esta característica se centra en encontrar el mejor circuito marítimo evaluando aquellos que presentan el menor tiempo total desde la salida hasta la llegada a la terminal destino.

STATE

Entendemos que el buque debe contar con fases que pueden ser interpretadas como estados. Creemos que es responsabilidad de las Fases (concrete states) conocer su próxima fase y cambiar el estado del buque. De esta forma, en caso en que se agregue en un futuro una nueva fase, el buque ni se enterará y simplemente irá pasando de fase en fase como antes. (Encapsulamiento y principio de open-closed).
Los estados se modelan con el modificador final dado que no precisamos que los mismos puedan ser heredados por alguna subclase.

PARTICIPANTES

- CONTEXT
  - Ship
- STATE
  - Phase
- CONCRETE STATE
  - Outbound
  - Inbound
  - Arrived
  - Working
  - Departing

TEMPLATE METHOD

PARTICIPANTES

- ABSTRACT CLASS
  - Routing
- CONCRETE CLASS
  - Fewer Intermediate Terminals
  - Lower Price
  - Shorter Time

USO DEL PATRÓN

```
private final Double calculateSpecificRouting(ManagedTerminal
origin, Terminal destiny, MaritimeCircuit maritimeCircuit) {

Integer startPosition = maritimeCircuit.getPositionOf(origin);

Integer endPosition = maritimeCircuit.getPositionOf(destiny);

return
calculateRouting(maritimeCircuit.getStretches().subList(startP
osition, endPosition));
}
```

Este método se encuentra en la clase abstracta Routing, donde lo que se hace es primero obtener la posición donde se encuentra la terminal origen y destino en el circuito marítimo, y lo luego el método calculateRouting() que es una operación primitiva, donde cada subclase sobreescribe ese método para resolver un calculo específico por cada subclase.

PARTICIPANTES

- ABSTRACT CLASS
  - Phase
- CONCRETE CLASS
  - Outbound
  - Inbound
  - Arrived
  - Working
  - Departing

USO DEL PATRÓN

```
public void proceedToNextPhase(Ship ship) {
     if(this.canItGoToTheNextPhase(ship)) {
          this.communicateWithTerminal(ship);
          this.changePhase(ship);
     }
}
```

Este método se encuentra en la clase abstracta Phase, donde para pasar a la siguiente fase el buque, se deben cumplir los criterios que establece cada subclase de fase, ya que canItGoToTheNextPhase es un método primitivo, y para la comunicación con la terminal también es un método primitivo.