Trabajo Práctico
Objetos II
Terminal Portuaria

Integrantes y mails:
Banegas Nicolás: banegasnico@gmail.com
Rodriguez Fennema Sabrina: sabrifennema@gmail.com
Correa Nicolás: nicolas.correa97@hotmail.com

Decisiones de diseño

El diseño giró en torno a la clase *TerminalGestionada*, donde están implementadas la mayoría de las funcionalidades del sistema, ya que es la que se encarga de planificar y manejar horarios, viajes y almacenar sus containers.
Al enfrentarnos con la problemática de esta sobrecarga de mensajes que portaba la clase *TerminalGestionada*, decidimos crear distintas interfaces para clasificar en distintas utilidades sus mensajes. Estas fueron *GestionLogistica* para todas las funcionalidades que tuvieras que ver con el manejo de camiones, containers y verificaciones de cargas y transportes. Y *GestiónEnvio* que se encargaba de importaciones y exportaciones y todo lo que tuviera que ver con la interacción *Cliente-Terminal*.
Además, para evitar el acople de *BuqueViaje* a la terminal, decidimos que la *TerminalGestionada* implemente la interfaz *Notificable* (aparte de *Localizable* para todas las terminales), así el *BuqueViaje* podría reportar sus cambios de estado y demás a otra entidad separada de la *Terminal* destino a la que estuviera yendo, con sus respectivas coordenadas.
Para asegurar el correcto uso de los mensajes de cada clase, y poder avanzar paralelamente en el grupo a la hora del desarrollo, implementamos una Interfaz para cada clase del sistema, donde cada clase implementa su propia interfaz pública, y solo se pueden comunicar con dichas clases a través de dichas interfaces.
Decidimos evitar el uso de las llamadas “Data Class” en todo momento. Está enfatizada la idea de que cada clase tuviera algún comportamiento.

Detalles de implementacion que merezcan ser explicados

El enunciado plantea que todas las facturas deban ser enviadas a los clientes en el mismo momento que el *BuqueViaje* pasará a fase *Outbound* luego de *Departing*, pero no creemos que tuviera sentido ya que en la factura se cobran los servicios aplicados a los containers, pero para ese momento no se sabrían todos los servicios para el *Consignee*, ya que recién hubiera llegado su container a la terminal como para cobrarle por el tiempo allí. Por ende decidimos que solo se genere la *Factura* para el *Shipper* en ese momento y recién en el retiro del *Container* por el camión para el *Consignee*.

Aunque en el enunciado se decía que la carga y descarga de containers no iba a ser contemplada en la implementación, se nos hizo más cómodo implementarlo para la coherencia del flujo de envíos.
Acordamos que si bien el sistema está pensado únicamente para la *TerminalGestionada*, quizás en un futuro esta se pudiera expandir y podrían haber más de una. Para esto, la instanciación del *BuqueViaje* ocurre desde el inicio del viaje, donde le avisa a cada terminal en el circuito de ese Viaje que acaba de iniciar, dando la oportunidad a que cada TerminalGestionada pudiera instanciar su propio BuqueViaje o su propia implementación de dicha gestión.
Los servicios especiales especificados como “servicio de desconsolidado” y “servicio especial” no están explicados en el enunciado, así que quedaron implementados como servicios con sus montos sin más complejidad. Aún así el servicio de desconsolidado creemos que no puede ser aplicado en ningún momento ya que el enunciado se contradice. “El contenedor puede transportar carga de **un solo** importador pero para optimizar el espacio, se puede utilizar un tipo de BL especial que agrupa a otros BLs. Este BL especial solo puede conocer la carga a partir de los BL agrupados” y posteriormente “Si la carga corresponde a varios importadores...”.
También nos encontramos con el inciso del enunciado que plantea que en ciertas fases de un *BuqueViaje*, la terminal tiene limitadas sus acciones, por ejemplo el pago de las facturas de los containers operados en ese viaje. Creemos que tampoco era posible de implementar ya que permite el pago en *Departing* pero recién en *Outbound* se envían las facturas, así que están implementadas las funcionalidades pero quedaron sin efecto en la versión final del sistema.

Patrones de diseño utilizados y los roles según la definición de Gamma et. al
Para la resolución del enunciado decidimos utilizar los siguientes patrones:

Strategy:

Context:	GestorDeRutas
Strategy:	MejorRuta
ConcreteStrategy:	FechaMasProxima
	MenorTiempo
	MenorPrecio
	MenorCanidadDeTerminales

Template Method:

AbstractClass:	MejorRuta
ConcreteClass:	FechaMasProxima
	MenorTiempo
	MenorPrecio
	MenorCantidadDeTerminales

Composite: (Se utilizó este patrón en 2 ocasiones)

BL

Component:	BillOfLading
Leaf:	BLSimple
Composite:	BLEspecial

Filtro

Component:	Filtro
Leaf:	FiltroSimple (Clase Abstracta con subclases concretas)
Composite:	FiltroCompuesto (Clase Abstracta con subclases concretas)

State:

Context:	BuqueViaje
State:	FaseBuqueViaje
ConcreteState:	Outbound
	Inbound
	Arrived
	Working
	Departing