INFORME TERMINAL PORTUARIA (GRUPO 1)

INTEGRANTES:

- Maldonado, Benjamín.
- Oreskovic, Franco Juan.
- Sanchez, Matias Pablo Jesús.

MATERIA: Programación Orientada a Objetos II.
CICLO LECTIVO: Segundo Cuatrimestre 2025.

Introducción

El presente informe expone el desarrollo de un **sistema de gestión para una terminal portuaria**, implementado en el lenguaje de programación **Java**. El objetivo principal del trabajo es **demostrar la aplicación de diversos patrones de diseño** y **poner en práctica los conceptos teóricos** aprendidos a lo largo de la materia.

El proyecto fue desarrollado por **Benjamín Maldonado**, **Matías Pablo Jesús Sánchez** y **Franco Oreskovic**, quienes asumieron la responsabilidad de distintas secciones del sistema, trabajando de manera coordinada durante todo el proceso.

Este trabajo se realizó en el marco de la asignatura **Programación Orientada a Objetos II**, con el propósito de integrar los conocimientos adquiridos y consolidar habilidades de diseño, programación y trabajo en equipo.

Desarrollo General

El sistema desarrollado representa la operativa de una **terminal portuaria**, incluyendo la gestión de **cargas, buques, viajes y órdenes de exportación, órdenes de importación**.
A continuación, se presentan los principales participantes del modelo, junto con una descripción general de su estructura y los **patrones de diseño** aplicados.

- **Orden:** representa todo el ciclo de vida del transporte de un container. Funciona como manifiesto y/o base de datos donde tanto el container como datos asociados son una misma cosa. En este punto lo que los buques y camiones transportan son Ordenes, asociadas a un Container, y la terminal almacena Ordenes. Aplicacion de **Visitor** en la implementacion

- **Cliente**: sujeto el cual puede tener rol de shipper (quien envía la carga) o consignee (dueño de la misma) dependiendo su rol en las operaciones de importación / exportación.

- **Viaje**: representa el recorrido de un circuito, el cual tiene fecha de salida (de terminal origen) y llegada (de terminal destino), y el buque que realiza el viaje.

- **TerminalPortuaria**: representa el participante principal de todo el modelo, el cual es el encargado de gestionar el registro registrar exportaciones e importaciones, y coordinar con los buques que vienen a cargar y descargar en la misma.

- **Servicios**:: representa los costos de almacenaje de un container. Aplicación de **Visitor, enumerativos** en la implementación.

- **Buque:** representa un buque que interactúa con la terminal en acciones de transporte, carga y descarga de órdenes. Aplicación de patrones **State** en la implementación

- **Container:** representa la carga que transportan tanto buques y camiones. Representa la carga que almacena una terminal. Aplicación de patrón **Composite** en la implementación.

En conjunto, el modelo fue diseñado siguiendo los principios de la **Programación Orientada a Objetos**, priorizando la cohesión interna, el bajo acoplamiento entre clases y la reutilización del código. El uso de **patrones de diseño** permitió alcanzar una arquitectura más flexible, mantenible y extensible.

APORTE de Benjamín Maldonado

Los participantes del modelo que le fueron delegadas para desarrollar e implementar fueron los siguientes:

Camion

- **Patrón aplicado:** Ninguno.
- **Concepto:** La idea del camión es que pueda almacenar una orden (representando a la carga que transporta). No se usa de manera individual, sino que él mismo es parte de una empresa transportista que cuenta con sus servicios.
  En caso de ser contratado, el mismo puede participar en una operación de exportación, ingresando la carga a la terminal portuaria que se le indique; o en una operación de importación, yendo a retirar una carga a la terminal portuaria que se le indique.
  Sumado a todo esto, en el contexto de que exista en una empresa transportista, tiene la capacidad de responder si está disponible para aceptar un trabajo, lo cual tiene que ver con que si está transportando una carga (en realidad una orden) o no.

Chofer

- **Patrón aplicado:** Ninguno.
- **Concepto:** La idea del chofer es que pueda ser el responsable de manejar el camión aunque no participe en una relación con el mismo. No se usa de manera individual, sino que él mismo es parte de una empresa transportista que cuenta con sus servicios.
  En caso de ser contratado, el mismo puede participar en una operación de exportación o importación, siendo parte de los que ingresan a la terminal.
  Sumado a todo esto, en el contexto de que exista en una empresa transportista, tiene la capacidad de responder si está disponible para aceptar un trabajo, lo cual tiene que ver con que si fue contratado anteriormente o no.

EmpresaTransportista

- **Patrón que aplicó:** Ninguno.
- **Concepto:** La idea de la EmpresaTransportista es que pueda ser el responsable de tener camiones y choferes en la misma, y que al mismo pueda ser registrada en una terminal portuaria. La empresa transportista es quien se encarga de gestionar los camiones y choferes a ser contratados por el shipper o consignee para transportar la carga que necesiten, ya que tiene como ventaja que la misma puede estar registrada en la terminal donde los shipper o consignee deben operar (la terminal gestionada).

Orden

- **Patrón que aplicó:** Ninguno.
- **Concepto:** La idea de la Orden es que sirva para encapsular el container (que sería la carga a transportar), el camión y chofer (quienes se van a encargar de transportar, ya sea para ingresarla o retirarla), el viaje que realiza y los servicios que se le cobran a la carga por estar en la terminal almacenada.

  El principal problema que existía, era la creación de los servicios en el momento de exportación o importación, pero fue solucionado con un Visitor en el container, el cual se encarga de visitar la orden y le devuelve los servicios que debe instanciar en base a la carga que tiene la misma.

  La misma es la que se registra en la terminal gestionada en caso de una exportación, o se retira de la misma en caso de importación. Por eso mismo, es la que representa todo el ciclo de vida de lo que se quiere transportar.

Buque (parcialmente)

- **Patrón que aplicó:** Ninguno.
- **Concepto:** La idea de la Buque es que principalmente tenga un viaje que realice, una coordenada que representa la posición donde se encuentra, y el estado del mismo (ya que tiene diferentes estados).

  Principalmente aportó las funcionalidades que se utilizan desde la terminal gestionada, como por ejemplo:
  - Devolver las órdenes que tiene cargadas el buque.
  - Iniciar trabajos, lo cual pasa el estado del buque a Working (solo si está en Arrived, de lo contrario lanza una excepción).
  - Finalizar descargas de órdenes, que elimina las ordenes que descargó del buque (solo si se encuentra en Working, de lo contrario lanza una excepción).
  - Devolver las órdenes a descargar en la terminal dada (solo si se encuentra en Working, de lo contrario lanza una excepción).
  - Cargar las ordenes dadas en el buque (solo si se encuentra en Working, de lo contrario lanza una excepción).
  - Finalizar los trabajos, lo cual pasa el estado del buque a Departing (solo si está en Working, de lo contrario lanza una excepción).

TerminalPortuaria (parcialmente)

- **Patrón que aplicó:** Ninguno.
- **Concepto:** La idea de la TerminalPortuaria era que principalmente pueda encargarse de registrar exportaciones y que al mismo tiempo el consignee pueda retirar una importación de una carga estacionada en la terminal.
  Con respecto a la exportación, la terminal se encarga de verificar que la orden dada (representando la carga, con su viaje, camión, chofer, shipper) puede ser registrada en la terminal. Para verificarlo, primeramente evalúa si está en horario para registrar la exportación (maximo 3 horas antes y no después del horario de salida del buque), y si el camión, chofer, shipper dados se encuentran registrados en la terminal, y que al mismo tiempo coincidan con los que están en la orden. Si esto sucede, la orden es registrada y la terminal comienza a brindarle los servicios que ofrece la terminal a la carga de la misma.
  Con respecto a retirar una importación, la terminal se encarga de verificar que existe una carga a retirarse del consignee dado y que además tanto el camión, chofer y el consignee se encuentren registrados en la misma. Si todo esto sucede, la terminal le cobra los servicios al consignee y el camión carga lo que se tiene que llevar, dando por finalizada la importación.
  Como funcionalidades extra (subrayado cuales implementó quien comenta este apartado), la terminal es capaz de:
  - Realizar trabajos de carga y descarga cuando llega un buque.
  - Registrar empresas transportistas (que incluye camiones y choferes), clientes (shipper y consignees), navieras, circuitos marítimos (que incluyan a la terminal).
  - Generar facturas de los servicios a cobrar.
  - Generar reportes en base al proceso de carga y descarga de un buque.
  - Envíar mails, especialmente al shipper cuando su carga se retira de la terminal.
  - Devolver el mejor circuito que une la terminal gestionada a una terminal dada, siendo el criterio de mejor seteado en la terminal.
  - Devolver cuánto tarda una naviera en llegar desde la terminal gestionada hacia otra terminal.
  - Devolver la próxima fecha de salida de un buque desde la terminal gestionada hacia otra de destino.

Adicionalmente, colaboró en el desarrollo de la clase Servicio y Viaje. En Viaje, aportó la idea de que la fecha de llegada como la de salida tenía que ser con la clase LocalDateTime, porque al poder definir una fecha junto a un horario, eran mucho más sencillas las diferentes operaciones que requerían de este dato con exactitud (como sucede con la terminal gestionada a la hora.
En Servicio, aportó la idea de utilizar la fecha de instanciación del servicio para poder definir el costo. En general funciona en base a lo que cuesta el servicio a cobrar en la terminal gestionada, pero sucede en algunos casos, como en el de almacenamiento excedente, que dependiendo de la fecha y hora puede cobrarte más o menos.

En relación a test unitarios, aportó en el desarrollo de la cobertura de los siguientes participantes:

- Camion.
- Chofer.
- EmpresaTransportista.
- TerminalPortuaria (parcialmente).
- Buque (parcialmente).
- Coordenada (parcialmente).
- Estados del buque (parcialmente).

Para finalizar, a pesar de no haber tenido la necesidad de implementar ningún patrón de diseño, las funcionalidades implementadas fueron necesarias para el desarrollo completo del trabajo en cuestión, tal como sucede en la terminal gestionada.

SECCIÓN de Franco Oreskovic

Los participantes del modelo que le fueron delegadas para desarrollar e implementar fueron los siguientes:

GeneradorDeReportes

- **Patrón aplicado:** Visitor.

- **Concepto:** El patrón Visitor nos permite representar operaciones sobre los elementos de estructuras de objetos.
  En este caso, los objetos son Ordenes y las estructuras que los contienen son listas.

  En cuanto a las operaciones, se definió una que procesa un elemento (Orden) y devuelve un String destinado a ser parte de un reporte de tipo aduana, y otra que procesa un elemento y devuelve un String destinado a ser parte de un reporte de tipo buque.
  Estas operaciones están representadas por los concrete visitors VisitorReporteAduana y VisitorReporteBuque.

- **Clases**:
  -VisitorReporte (visitor)
  -VisitorReporteAduana (concrete visitor)
  -VisitorReporteBuque (concrete visitor)
  -Orden (element)

BuscadorDeCircuito

- **Patrón aplicado:** Strategy.

- **Concepto:** Se le puede pedir a la terminal un CircuitoMaritimo que una a dicha terminal con otra pasada por parámetro.

  Sobre aquellos CircuitosMaritimos que unan a dichas terminales, se buscará el más adecuado en base a un criterio que aplica sobre los tramos desde la terminal de origen hasta la terminal de destino.
  Este criterio puede ser cambiado dinámicamente por el usuario.

  Se modelaron 3 criterios concretos (búsqueda por menor precio, por menor tiempo y por menor cantidad de terminales), pero se podrían definir nuevos y ser utilizados sin romper el Open-Closed Principle.

- **Clases**:
  -BuscadorDeCircuito (strategy)

-BuscadoPorPrecio (concrete strategy)
-BuscadoPorTiempo (concrete strategy)
-BuscadoPorCantidadTerminales (concrete strategy)

BuscadorDeViaje

- **Patrón aplicado:** Composite.

- **Concepto:** Se le puede pedir a la terminal una lista de Viajes que cumplan con una Condicion pasada por parámetro.

  Esta Condicion puede ser una expresión lógica simple o una expresión lógica compuesta formada por otras expresiones unidas por conectores.

  Se le delega a la propia Condicion la tarea de revisar si un cierto Viaje cumple o no con su condición, ya que es esta quien tiene la mayor información para determinar esto.

- **Clases**:
  -BuscadorDeViaje (le delega la tarea a la condición)
  -Condicion (component)
  -Conector (composite)
  -Or (subclase composite)
  -And (subclase composite)
  -FiltroSimple (leaf)
  -FiltroPuertoDestino (subclase leaf)
  -FiltroFechaSalida (subclase leaf)
  -FiltroFechaLlegada (subclase leaf)

CircuitoMaritimo

- **Patrón aplicado:** Ninguno

- **Concepto:** Es la clase que representa una secuencia de terminales visitadas. Esta secuencia está representada mediante una lista de Tramos, los cuales contienen una terminal origen y una terminal destino.

  No cualquier conjunto de tramos es válido para formar un CircuitoMaritimo.
  Por ejemplo, si la lista es vacía, no es válida para formar un CircuitoMaritimo.
  También, si para la lista de tramos no se cumple que, a lo largo de toda la secuencia de tramos, la terminal destino de un tramo es la terminal origen en el siguiente tramo de la secuencia, entonces esa lista no es válida para formar un CircuitoMaritimo.

  Tiene métodos que son utilizados por el buscador de circuitos, como decir el tiempo que lleva recorrer desde una terminal portuaria a otra en el circuito, y

esto mismo para el caso del precio y de la cantidad de terminales en sus respectivos métodos.

Viaje

- **Patrón aplicado:** Ninguno

- **Concepto:** Es la clase que representa una realización de un CircuitoMaritimo por un determinado Buque con una determinada fecha de salida.

Naviera

- **Patrón aplicado:** Ninguno

- **Concepto:** Es la clase que representa a una empresa que cuenta con múltiples Buques, múltiples CircuitosMaritimos que pueden recorrer sus Buques, y múltiples Viajes (que realizan alguno de sus Buques sobre alguno de sus CircuitosMaritimos).

TerminalPortuaria (parcialmente)

- **Patrón aplicado:** Ninguno

- **Concepto:**

  - Se implementaron los métodos relacionados a la búsqueda de circuitos: setBuscadorDeCircuito() y buscarCircuito()

  - Se implementó el método relacionado a la búsqueda de viajes: buscarViaje()

  - Se implementaron los métodos relacionados a la generación de reportes: generarReportes() y el método privado ordenesDelViaje().

  - Se implementó el método tiempoEntre(), que nos devuelve un Duration que representa el tiempo que tomaría recorrer el CircuitoMaritimo más eficiente de una determinada Naviera que conecta a la terminal gestionada y a una determinada terminal portuaria .

  -Se implementó el método proximaFechaHacia(), que nos dice cuál es la fecha de salida más próxima para un Viaje con un determinado Buque que conecte a una terminal origen (la gestionada) y a otra terminal destino. La fecha de salida refiere a cuándo se parte de dicha terminal origen.

SECCIÓN de Sanchez Matias Pablo Jesús

Esta sección tiene como objetivo describir de manera técnica algunos requerimientos del Sistema de Gestión de Terminal Portuaria. Se detalla por cada uno su propósito, funcionalidades esperadas y la resolución del mismo en base al uso de Patrones aprendidos en la materia. A continuación quedan listados los requerimientos implementados:

1.Servicios-Container-Orden
2.Containers-CargaBL
3.Buque-EstadosBuque
4.Orden
5-Precio Servicio Terminal
6-Cobro-Servicios-EnvioMail

1.**Servicios**

- **Patrones aplicados: Visitor -**

El requerimiento consiste en implementar los Servicios que se deben cobrar a los clientes por el almacenaje de los Containers en la Terminal. Se implementa entonces la Clase Servicio que funciona como un manifiesto con los datos del container,su dueño, y su horario de ingreso a la terminal. El servicio cuenta entonces con el método **precioServicio**(**args**) donde interactúa con la terminal para calcular el monto que deberá pagar el cliente en el momento que se requiera generar la factura.

El patrón **Visitor** es utilizado de esta manera:
Cuando la **Orden** llama al método **serviciosACobrar** envía un **visitante** a un **container.** El container recibe al visitante y le comunica al mismo los servicios que debe crear en base al Tipo que tenga el Container. De esta manera el visitante crea una Colección de Servicios, que varía en función del Container que los requiere.

2-**Containers**.

- Patrones aplicados - **Composite**

El requerimiento consiste en representar los Containers que son las cargas que transporta el buque. En particular funcionan como un Data Class donde solo contienen datos asociados a la carga y al dueño de la misma.

Se representan entonces los containers **Tanque**, **Reefer** y **Dry**

-El patrón **Composite** es utilizado de esta manera:
El Container Dry posee una carga especial denominada BL, que puede ser de un cliente o de varios, y esta carga puede a su vez estar compuesta de otro **BL's**. Se crea entonces la interfaz **CargaBL** cuya implementación es realizada por las clases **CargaBLCompuesta** y **CargaBLHoja.** Según el libro de Gamma los roles son:
-Componente : **CargaBL**
- Hoja : **CargaBLHoja**
- Compuesto **: CargaBLCompuesta**

De esta manera la carga de un Container Dry se obtiene a partir de una **CargaBL**

3 -**Buque-EstadosBuque**

Patrones aplicados **State, Observer**
El Buque se comporta a partir de su **EstadoBuque**. El buque cambia de estado dependiendo su interacción con la Terminal Gestionada. De esta manera se implementan los siguientes estados

- **Outbound** : En este punto el buque conoce la **terminal** que tiene que arribar. Solo se le permite moverse. Una vez que se encuentra a menos de 50km de la **terminal** pasa a la siguiente fase.

- **Inbound :** El buque se encuentra a menos de 50km de la **terminal**. Se notifica a la misma el inminente arribo a la terminal a través de la interfaz del buque observado, aplicando el patrón **Observer.** Cuando el buque coincide en coordenadas con las de la terminal. Pasa a la siguiente fase.
- **Arrived :** El buque queda a la espera de que la terminal le notifique que está en condiciones de iniciar la carga y descarga. Una vez notificado pasa a la siguiente fase. Durante este fase no puede cambiar sus coordenadas
- **Working:** el buque está en condiciones de comenzar con la carga y descarga de los containers. Queda a la espera de que la terminal habilite su partida. Cuando esto sucede pasa a la siguiente fase. Durante esta fase no puede cambiar sus coordenadas
- -**Departing:** El buque se encuentra saliendo de la terminal. Una vez que se encuentra a más de 1 km, notifica a la terminal, y esta envía los mails requeridos a los shippers asociados a las Ordenes de exportación
- -**OutBoundFinal:** El buque deja de tener asociada la terminal y se dirige al siguiente destino. No tenemos control de lo que ocurre a partir de aquí

4-Orden

- patrones aplicados: **Visitor**

  El requerimiento consiste en implementar la funcionalidad:
  -**serviciosACobrar**

  La implementación consistió en que la orden tenga asociado una clase **ConcreteVisitorContainer** que se encarga de obtener la lista de servicios requeridos de un **container** asociado a una **Orden.**

5-Precio Servicio Terminal

El requerimiento consiste en implementar los precios que las terminales deben brindar a los **Servicios** para calcular el costo de los mismos. No

se utilizó patrones, sino que se implementó con datos de tipo **ENUM.** Donde cada elemento de **PrecioServicioTerminal** cumple la función de decir su costo. De esta manera la terminal comunica el precio de los servicios.

6-**Cobro-Servicios-EnvioMail**

Requerimiento asociado a **terminal**

El requerimiento consiste en generar la factura de servicios, cuyo desglose de conceptos está compuesto por el **nombre** del cliente, y el dato de cada **servicio**, con su **costo** asociado. La implementación del método consiste en **generarLaFactura** haciendo un mapeo de la Orden y obteniendo los datos correspondientes. Se devuelve entonces un **String** que representa la factura a enviar.

El requerimiento **enviarMail** consiste en enviar a los clientes asociados a la orden un main con el envío de la factura o el aviso de llegada de carga, según sea el caso. Depende qué tipo de orden puede ser o a un **Shipper** o a un **Consigne.** La terminal realiza esta acción en 3 momentos:

-El buque está por llegar: Al recibir la notificación del **buque,** la **terminal** notifica a los **consignee** que sus cargas están próximas a llegar

-El buque parte de la terminal: Al recibir la notificación del **buque,** la **terminal** envía por correo la **factura** de los servicios del **container** al **shipper** correspondiente

-El camión viene a buscar carga: Al llegar un camión, una vez entregada se envía por correo la factura de los servicios al **consignee,** que es dueño de la misma.

Conclusiones

A lo largo del desarrollo del proyecto, se aplicaron diversos **patrones de diseño** con el objetivo de lograr un sistema modular, extensible y fácil de mantener. El uso de estos patrones permitió estructurar el código de forma clara, favoreciendo la reutilización y la separación de responsabilidades entre los distintos componentes.

El trabajo se llevó a cabo mediante **reuniones y coordinaciones semanales**, lo que facilitó la comunicación continua dentro del grupo y permitió una organización eficiente de las tareas. Este proceso colaborativo fue clave para la integración de las distintas partes del sistema y para la resolución conjunta de los desafíos que surgieron durante el desarrollo.

Asimismo, el proyecto experimentó **numerosos cambios sobre la marcha**, tanto en los requerimientos como en el diseño. Estas modificaciones pusieron a prueba nuestra capacidad de adaptación y fortalecieron nuestra comprensión del ciclo de vida del software, desde el análisis inicial hasta la implementación final.

En conjunto, el trabajo realizado no solo permitió aplicar conceptos teóricos a un caso práctico, sino también desarrollar habilidades de **trabajo en equipo, comunicación y flexibilidad**, esenciales para cualquier proyecto de desarrollo de software.

Palabras Finales

Este proyecto representó una valiosa oportunidad para aplicar distintos patrones de diseño aprendidos a lo largo de la cursada, fortalecer la coordinación grupal y adaptarnos a los cambios surgidos durante el desarrollo.

Nos despedimos satisfechos con el trabajo realizado y con la convicción de haber adquirido nuevos aprendizajes técnicos y de trabajo en equipo.

Atentamente

El equipo de desarrollo:

Maldonado, Benjamín
Franco Oreskovic
Sanchez Matias Pablo Jesús