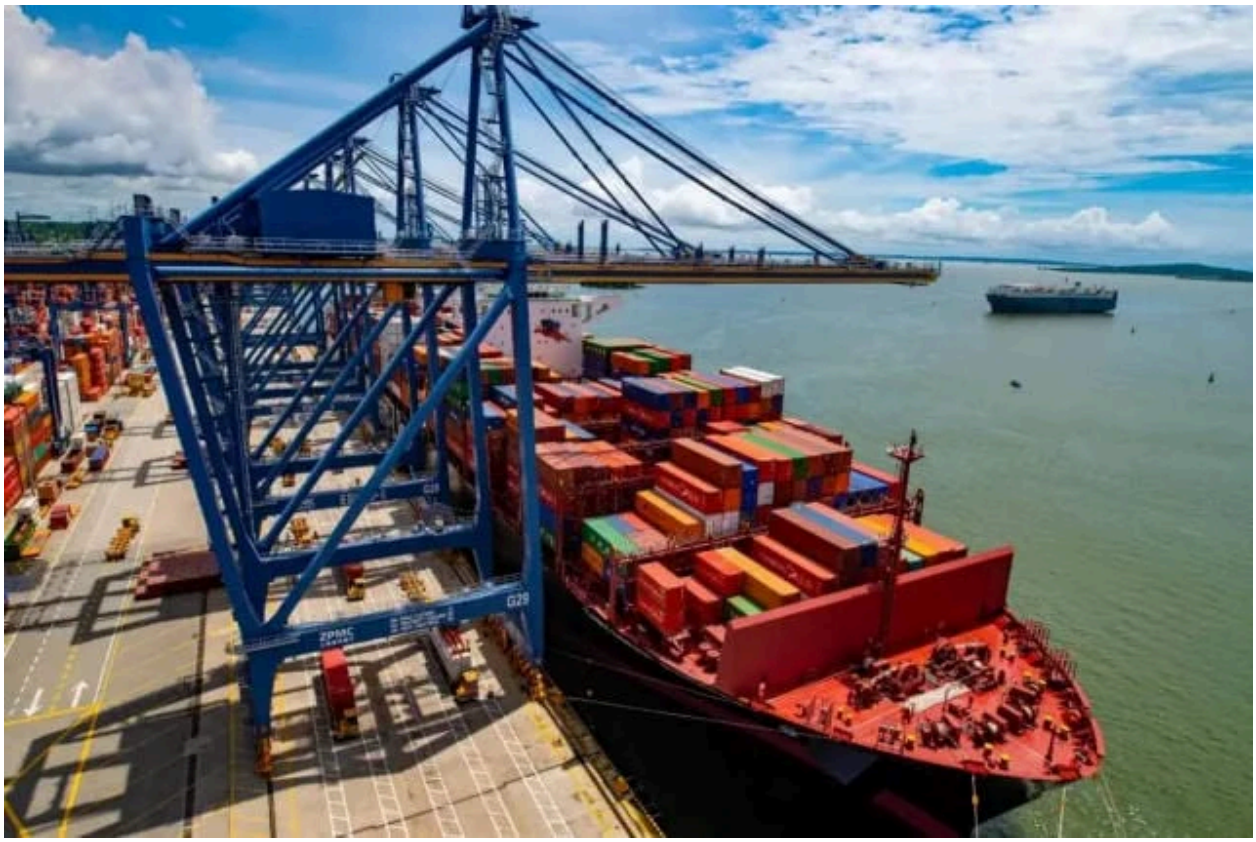
INFORME TRABAJO FINAL OBJETOS II



Integrantes:

Marco Antonio Cardozo - marcocardozo325@gmail.com

Gabriel Federico Gallo - gfgallo12@gmail.com

Sergio Ivan Zapata - serivanzapata@gmail.com

INTRODUCCIÓN

El objetivo de este proyecto fue diseñar e implementar un sistema orientado a objetos para modelar y gestionar la compleja operatoria de una terminal portuaria.

A lo largo de este informe, nos centraremos en explicar en detalle la solución que desarrollamos.

Decisiones de Diseño

El diseño del sistema se estructuró en capas claramente diferenciadas para facilitar la mantenibilidad y extensibilidad. En primer lugar, se separaron los **actores** del dominio (Shipper, Consignee, EmpresaTransportista) de las **entidades de negocio** (Buque, Container, Viaje), permitiendo una clara distinción entre quién realiza las operaciones y sobre qué se operan.

Para modelar los containers, se optó por una clase abstracta **Container** que encapsula las características comunes (dimensiones, peso, identificación), mientras que Dry, Reefer y Tanque heredan de ella para agregar comportamientos específicos. Esta decisión permite que cada tipo de container pueda tener su propia lógica de servicios sin afectar a los demás.

Las **órdenes** (OrdenDeExportacion y OrdenDeImportacion) también se modelaron mediante herencia desde una clase abstracta Orden, lo que facilita compartir la lógica común de gestión de servicios y carga, mientras se mantienen las particularidades de cada tipo de operación.

La **TerminalGestionada** fue diseñada como el orquestador central del sistema, coordinando las interacciones entre **buques**, **órdenes**, **clientes** y **servicios**. Esta clase implementa BuqueObserver para reaccionar a cambios de estado de los buques, y utiliza Strategy para permitir diferentes criterios de búsqueda de circuitos marítimos.

Los **servicios** se modelaron como una interfaz Servicio con implementaciones concretas, permitiendo que cada servicio calcule su costo de forma independiente según las características de la orden. Esta aproximación facilita agregar nuevos servicios sin modificar código existente.

Detalles de Implementación

La **transición de fases** del buque se implementa mediante el patrón State, donde cada fase (Outbound, Inbound, Arrived, Working, Departing) es responsable de evaluar las condiciones para avanzar a la siguiente. Por ejemplo, Inbound verifica la distancia GPS y cambia a Arrived cuando las coordenadas coinciden con la terminal. Las fases Outbound e Inbound pueden revertirse mutuamente para modelar cambios climáticos.

El **cálculo del precio** de un viaje se realiza sumando los precios de los tramos desde la terminal origen hasta la terminal destino dentro del mismo circuito marítimo. La implementación recorre los tramos del circuito, comenzando a sumar cuando encuentra el origen y deteniéndose al llegar al destino.

El **servicio de almacenamiento excedente** calcula los días excedentes considerando una tolerancia de 24 horas desde la fecha de llegada de la carga. Si el retiro se realiza después de este período, se calculan los días excedentes (redondeando hacia arriba) y se multiplican por el costo diario configurado.

La **facturación** se dispara automáticamente cuando un buque pasa a la fase Outbound después de haberse retirado de la terminal (distancia mayor a 1 km). Para exportaciones se facturan solo los servicios al shipper, mientras que para importaciones se agrega además el costo total del viaje al consignee.

Los **filtros de búsqueda** utilizan el patrón Composite para permitir combinaciones complejas mediante FiltroAND y FiltroOR. Cada filtro compuesto delega la evaluación a sus filtros hijos, permitiendo construir expresiones lógicas arbitrarias sin que TerminalGestionada necesite conocer la estructura interna de los filtros.

Patrones de Diseño y los Roles de las Clases

Patrón Composite

Rol(según GoF)	Clase en el proyecto	Descripción
Component	FiltroDeBusqueda *(*interfaz*)*	Define la interfaz que declara las operaciones que deben implementar los Filtros.
Composite	FiltroAND, FiltroOR	Implementan la interfaz FiltroBusqueda y contienen una colección de otros filtros (componentes).
Leaf	FiltroPuertoDestino, FiltroFechaLlegada, FiltroFechaSalida	Implementan la interfaz FiltroBusqueda, pero son filtros concretos que no contienen otros filtros.
Cliente	TerminalGestionada	Conoce la interfaz FiltroBusqueda y la utiliza para aplicar criterios de filtrado sin distinguir si el filtro es simple o compuesto.

Rol(según GoF)	Clase en el proyecto	Descripción
Component	BillOfLanding*(*interfaz*)*	Define la interfaz que declara las operaciones que deben implementar los conocimientos de carga (bill of landing)
Composite	CompuestoBL	es una instancia concreta que implementa la interfaz BillOfLanding y contiene una colección de BillOfLanding

Leaf	SimpleBL	Son instancias concretas que implementan la interfaz BillOfLanding pero no contienen otros BillOfLanding
Cliente	Container(abstract)	Conoce una instancia de BillOfLanding que indica si el Container tiene un conocimiento de carga simple o compuesto

Patrón Observer

Rol(según GoF)	Clase en el proyecto	Descripción
Subject	Buque	Mantiene una referencia a BuqueObserver y provee los métodos para registrarlos, removerlos y notificarles cambios de estado.
Observer	BuqueObserver(interfaz)	Define la interfaz que deben implementar todas las clases interesadas en recibir notificaciones del Buque.
ConcreteObserver	TerminalGestionada	Implementa la interfaz BuqueObserver. Define la acción concreta que debe realizarse cuando el Buque notifica un cambio

Patrón Strategy

Rol(según GoF)	Clase en el proyecto	Descripción

Strategy	CriterioDeBusqueda*(*interfaz*)*	Define la interfaz que declara el método a implementar por los distintos Criterios.
ConcreteStrategy	CriterioMenorTiempo, CriterioMenorCantidadTerminales, CriterioMenorPrecio	Implementan la interfaz CriterioDeBusqueda , donde cada una define un propio algoritmo.
Context	TerminalGestionada	Mantiene una referencia a un objeto del tipo CriterioDeBusqueda

Patrón State

Rol(según GoF)	Clase en el proyecto	Descripción
State	FaseBuque*(*interfaz*)*	Define la interfaz que declara el comportamiento común para todos las fases del Buque.
ConcreteState	Outbound, Inbound, Arrived, Working, Departing	Implementan la interfaz FaseBuque, proporcionando comportamiento específico según la fase.
Context	Buque	Mantiene una referencia a un objeto del tipo FaseBuque.

Patrón Visitor

Rol(según GoF)	Clase en el proyecto	Descripción
Visitor	Visitor(interfaz)	Define la interfaz que declara las operaciones que deben implementar los ConcreteVisitor.

ConcreteVisitor	ReporteMuelleVisitor, ReporteAduanaVisitor, ReporteBuqueVisitor	Implementan la interfaz Visitor, proporcionando operaciones específicas que se ejecutan al visitar cada elemento.
Element	Visitable(interfaz)	Declara el método accept(Visitor v) que permite a un visitante acceder a la instancia del elemento.
ConcreteElement	TerminalGestionada, Naviera, Buque, OrdenDeImportacion, OrdenDeExportacion, Dry, Reefer, Tanque	Implementan el método accept(Visitor v) invocando el método correspondiente del visitante.

CONCLUSIÓN

A lo largo del desarrollo de este Trabajo Final, logramos diseñar e implementar un sistema funcional para la gestión de una terminal portuaria.

El proceso implicó un análisis detallado del dominio, que se tradujo en el diagrama de clases UML y la posterior implementación en Java. Sin embargo, el desafío principal fue la identificación y correcta aplicación de los patrones de diseño.

El uso de patrones como Composite, Strategy, State, Visitor y Observer no fue un ejercicio meramente académico, sino una necesidad fundamental para resolver los problemas de diseño de forma robusta, mantenible y extensible.

REFERENCIAS

1. Libro Design Patterns: Elements of Reusable Object-Oriented Software. Erich Gamma Et At