Universidad Nacional de Quilmes

Departamento de Ciencia y Tecnología

Programación con Objetos II

Trabajo Integrador | Terminales Portuarias

Informe

Estudiantes

Uriel Benitez unbenitez@gmail.com
Fabricio Futryk ffutryk52@gmail.com
Franco Rodriguez francototites@gmail.com

Docentes

Diego Cano
Diego Torres
Matias Butti
Matias Urbieta

Noviembre de 2025

ÍNDICE

Uriel Benitez
Fabricio Mijail Futryk
Franco Rodriguez
NOVIEMBRE, 2025
Universidad Nacional de Quilmes
Departamento de Ciencia y Tecnología

Programación con Objetos II 1037-1-G14
Trabajo Integrador | Terminales Portuarias | Informe

Introducción

El presente trabajo tiene como objetivo modelar el funcionamiento de una terminal portuaria, abordando los actores, servicios y operaciones que intervienen en los procesos de exportación e importación de contenedores, diseñando una solución orientada a objetos que permita su gestión y posterior mantenimiento de forma extensible, aplicando principios y patrones vistos a lo largo de la cursada.

Contexto

Las terminales portuarias reciben cargas que, mediante buques, se trasladan a hacia otras terminales portuarias. La operatoria de estas terminales abarca la gestión y articulación del tráfico de cargas, buques, transportes terrestres y todos los actores que forman parte del ecosistema portuario. En este contexto, la terminal gestionada representa el punto de enfoque del sistema desarrollado.

Objetivo

Desarrollar un dominio en Java que representa la operación de una terminal portuaria, incluyendo la gestión de actores, viajes, órdenes, servicios y reportes, aplicando patrones de diseño y principios de programación orientada a objetos.

Alcance

El trabajo se limita a la gestión de una "terminal gestionada", sin contemplar lo que ocurre fuera de ella. No se implementan interfaces gráficas ni persistencia de datos, tampoco procesos de carga y descarga, centrándose exclusivamente en la lógica de dominio.

Análisis del Dominio

Entidades Principales

En esta sección se describen los elementos más relevantes del dominio y su rol dentro del sistema.

La terminal gestionada está compuesta por diversos actores, objetos y servicios que interactúan en los procesos de exportación e importación de contenedores. Entre las principales entidades del dominio se pueden identificar:

- **Terminal Gestionada**: representa la terminal sobre la que se modela el sistema y donde se centraliza la gestión y flujo de información para las diferentes operaciones. Siendo así el punto de contacto principal del sistema.
- **Terminal**: representa un puerto geográficamente.
- **Naviera**: empresa marítima la cual cuenta con un conjunto de buques destinados al transporte de mercancías dentro de determinados circuitos.
- **Circuito**: recorrido circular compuesto por una secuencia de **Tramos**.
- **Tramo:** une dos terminales consecutivas dentro de un **Circuito**, incluyendo la duración y el costo del recorrido.
- **Buque**: realiza un **Viaje** siguiendo un circuito determinado.
- **Viaje**: recorrido concreto de un buque sobre un circuito con una fecha de inicio y el cronograma asociado, también se encarga de transitar diferentes fases durante el recorrido (Outbound, Inbound, Arrived, Working, Departing).
- **Container**: unidad de carga transportada por **Buques** y **Camiones**. Pueden ser de tipo Dry, Reefer o Tanque

- **Bill Of Landing** (**BL**): documento que informa el contenido de un **Container**.
- **Orden**: operación que registra una exportación o importación de un **Container**, asociándolo con un **Shipper** o **Consignee**.
- **Shipper**: exportador que envía la carga desde la terminal gestionada.
- **Consignee**: importador que recibe la carga en la terminal.
- **Empresa Transportista, Camión y Chofer**: representan el transporte terrestre de **Containers** hacia o desde la terminal gestionada.
- **Servicio**: prestación adicional realizada por la terminal sobre un container (lavado, pesado, electricidad, almacenamiento excedente).
- **Factura**: documento que detalla los servicios contratados y los costos asociados.

Relaciones y Dependencias

Las entidades del dominio se vinculan entre sí mediante asociaciones que reflejan la operatoria del puerto.

Entre ellas se encuentran:

- Una **Naviera** posee varios **Circuitos** y cada uno está formado por múltiples **Tramos**.
- Un **Tramo** posee una **Terminal** origen y una de destino.
- Una **Naviera** mantiene un cronograma de **Viajes.**
- Un **Buque** pertenece a una **Naviera** y realiza **Viajes** que recorren un **Circuito**.
- Cada **Orden** vincula un **Container** con un **Shipper** o **Consignee,** proviene de un **Viaje** (importación) o será enviada en uno (exportación) y puede tener varios **Servicios** contratados mientras permanece en la **Terminal Gestionada**.
- Un **Container** tiene asociado su **Bill Of Landing** correspondiente.

Procesos

- **Exportación:**
    1. El **Shipper** solicita una orden de exportación indicando el destino, el **Container**, el **Camión** que la llevará a la terminal y el **Chofer** del mismo.
    2. La terminal asigna un turno para llevar la carga la misma.

3. Cuando el **Camión** llega con la carga a la terminal, ésta verifica que su turno no difiera en más de tres horas respecto al turno asignado. Además de verificar la autorización por parte del **Shipper**.
4. Una vez pasadas todas las verificaciones, el **Container** queda almacenado en la terminal a la espera de la llegada del **Buque**.
5. Cuando un **Buque** se acerca a la terminal da el preaviso correspondiente unas horas antes para que la terminal se prepare para el arribo. Al llegar, se procede a la descarga y carga de **Containers**, proceso el cual no entra en el alcance del trabajo.

- **Importación:**
    1. La terminal envía un mail al **Consignee** indicando la fecha y hora de llegada.
    2. El **Consignee** informa a la terminal el camión y el chofer que retira su carga.
    3. Cuando llega el **Camión**, la terminal verifica la autorización del mismo.
    4. Una vez verificado, el camión se lleva el **Container** y se registra su retiro.

Diseño de la solución

Decisiones de Diseño

Durante el desarrollo y diseño del sistema se tomaron ciertas decisiones de diseño en base a nuestro entendimiento del enunciado y del dominio planteado para poder garantizar una coherencia, extensibilidad y claridad en el diseño propuesto. Entre estas decisiones se encuentran:

- **No tener una jerarquía de Cliente para *Shipper* y *Consignee*.** Decidimos no subclasificar a los *shippers* y a los *consignees* ya que su estructura y comportamiento eran equivalentes. En su lugar, decidimos determinar el rol que toma un cliente según el contexto de la operación (importación o exportación).
- **Modelar los productos de un *Bill of landing* como una tupla.** Decidimos representar los productos de un *Bill of landing* como una tupla `(nombre, peso)` en lugar de tener una clase "Producto" para evitar el *code smell* asociado a las *data classes* sin comportamiento.
- **No implementar un *observer* para los avisos basados en la ubicación del buque.** Si bien el enunciado sugiere la posibilidad de utilizar el patrón de diseño *Observer* para que la terminal reaccione según a la distancia del buque, descartamos esta opción ya que no cumple con la relación propuesta por Gamma et. al (1994), donde un "Sujeto" notifica a múltiples "Observadores". En nuestro caso, la terminal debería observar múltiples viajes, invirtiendo dicha dependencia y contradiciendo el propósito original del patrón.
- **La fecha de llegada y partida de un viaje son las mismas.** Dado el alcance del sistema, que no contempla los procesos de carga y descarga de contenedores, decidimos que la fecha de llegada y la de partida de un viaje sean iguales, ya que no existe una forma determinística de calcular la fecha de partida dentro del diseño propuesto.

Patrones de Diseño

Para mejorar la claridad, flexibilidad y mantenibilidad del código, empleamos patrones de diseño para poder comunicarlo de manera más efectiva y reducir la complejidad del sistema. Los patrones de diseño utilizados fueron los siguientes:

Strategy

Propósito: Según *Gamma et al. (1994)*, el patrón *strategy* define una familia de algoritmos, encapsula cada uno de ellos y los hace intercambiables. Permitiendo que un algoritmo varíe independientemente de los clientes que lo usan.

Aplicación: El patrón es utilizado principalmente en la sección relacionada al motor de búsqueda, siendo utilizado por los filtros, particularmente por el filtro de fecha, ya que se debía de poder establecer la forma por la cual se deseaba la comparación, siendo las disponibles: *AntesDeFecha*, *DespúesDeFecha* y *DuranteLaFecha*. Y también siendo utilizado directamente por el motor de búsqueda, permitiendo diferentes algoritmos para la búsqueda de circuitos, siendo estos los siguientes: *MásBarata*, *MásRapida*, *ParadasMinimas*

Participantes:

- En los filtros:
  - **Contexto** → FiltroPorFecha
  - **Estrategia** → EstrategiaComparacionFecha
  - **Estrategia Concreta** → AntesDeFecha, DuranteLaFecha, DespuesDeFecha
- En el motor de búsqueda:
  - **Contexto** → MotorDeBusqueda
  - **Estrategia** → EstrategiaBusquedaCircuito
  - **Estrategia Concreta** → MasBarata, MasRapida, ParadasMinimas

State

Propósito: Según *Gamma et al. (1994)*, el patrón *state* Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno. Parecerá que cambia la clase del objeto.

Aplicación: El patrón es utilizado para determinar el comportamiento durante las diferentes fases del buque definidas por el enunciado. Sin embargo, a pesar del nombre "Fases de Buque" en el enunciado, decidimos que el contexto del patrón sea un Viaje ya que nuestro diseño, un Buque puede no estar realizando un viaje, por lo tanto no sería viable tener el contexto del *state* en el buque ya que esto implicaría tener un estado "Nulo" o "En Espera" sin ningún comportamiento determinado.

Participantes:

- **Contexto** → Viaje
- **Estado** → FaseDeViaje
- **EstadoConcreto** → *Arrived*, *Departing*, *Working*, *Outbound*, *Inbound*

Visitor

Propósito: Según *Gamma et al. (1994)*, el patrón *visitor* representa una operación sobre los elementos de una estructura de objetos. Permitiendo definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

Aplicación: El patrón es utilizado para la generación de reportes ya que se necesitaba extender el comportamiento de ciertas clases del dominio las cuales no deberían tener dicha responsabilidad. A través de este patrón, se recorren las entidades del sistema: buque, orden importación y orden exportación, aplicando una lógica específica para la generación del reporte correspondiente a ella mediante *Double Dispatching*. De esta manera, el sistema puede ser extendido para incorporar nuevas formas de reporte utilizando la jerarquía de "VisitanteReporte".

Participantes:

- **Visitante** → VisitanteReportable
- **Visitante Concreto** → VisitanteBuque, VisitanteAduana, VisitanteMuelle
- **Elemento** → IReportable, Orden
- **Elemento Concreto** → Buque, OrdenImportacion, OrdenExportacion

Composite

Propósito: Según *Gamma et al. (1994)*, el patrón *composite* permite la composición de objetos en estructuras de árbol para representar jerarquías de parte-todo. Permitiendo que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

Aplicación: El patrón es utilizado principalmente en aquellas estructuras del dominio que poseen una naturaleza inductiva o recursiva, siendo estas la jerarquía de filtros de búsqueda y la correspondiente a los *Bill of landing*. En el primer caso, se aplicó debido a que diseñamos los filtros tomando inspiración en la lógica proposicional, la cual está definida de manera inductiva. En el segundo caso, lo utilizamos dado que los contenedores, particularmente los de tipo "*Dry*" tienen la capacidad de contener otros *Bill of landing* en su interior.

Participantes:

- En los filtros:
  - **Componente** → IFiltroViaje
  - **Hoja** → FiltroPorDestino, FiltroPorFecha
  - **Compuesto** → FiltroCompuesto, FiltroAnd, FiltroOr
- En los *Bill of landing*:
  - **Componente** → IBillOfLanding
  - **Hoja** → BillOfLanding
  - **Compuesto** → BillOfLandingCompuesto

Bibliografía

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (**1994**). *Design patterns: Elements of reusable object-oriented software.* Addison-Wesley.