TRABAJO INTEGRADOR TERMINAL PORTUARIA

Integrantes/Email:

- **Tiago Kachorroski:** tiagokacho27@gmail.com
- **Tomás Agustin Ramos:** tomasagustinramos@gmail.com
- **Rodrigo Bedoya:** rbedoya232@gmail.com

Contenido:

Introducción

El presente trabajo elaborado por nuestro grupo, conformado por Tiago Kachorroski, Tomas Agustín Ramos y Rodrigo Bedoya, presenta diferentes decisiones de diseño que fueron tomadas para la resolución del Trabajo Práctico Integrador de la materia Programación Orientada a Objetos 2, utilizando como consigna el diseño de una Terminal Portuaria y sus interacciones con diferentes colaboradores. Las decisiones de diseño mencionadas anteriormente fueron tomadas de manera colaborativa, estando cada integrante en cada etapa de la planificación, así como en la etapa de implementación en código java.

Este informe expone los patrones de diseño utilizados, dónde se aplicaron y el por qué se utilizaron en cada caso. Además, al final del informe se explican diferentes inquietudes que quedaron respecto al enunciado.

Patrones de diseño

Entre los patrones de diseño utilizados para el modelado de este trabajo se encuentran el patrón Observer, Composite, State, Strategy y Visitor (siendo Composite aquel con más apariciones). A continuación se detallarán los lugares específicos donde aparecen dichos patrones:

Patrón Observer

Diseño de gps,buque y estados de un buque

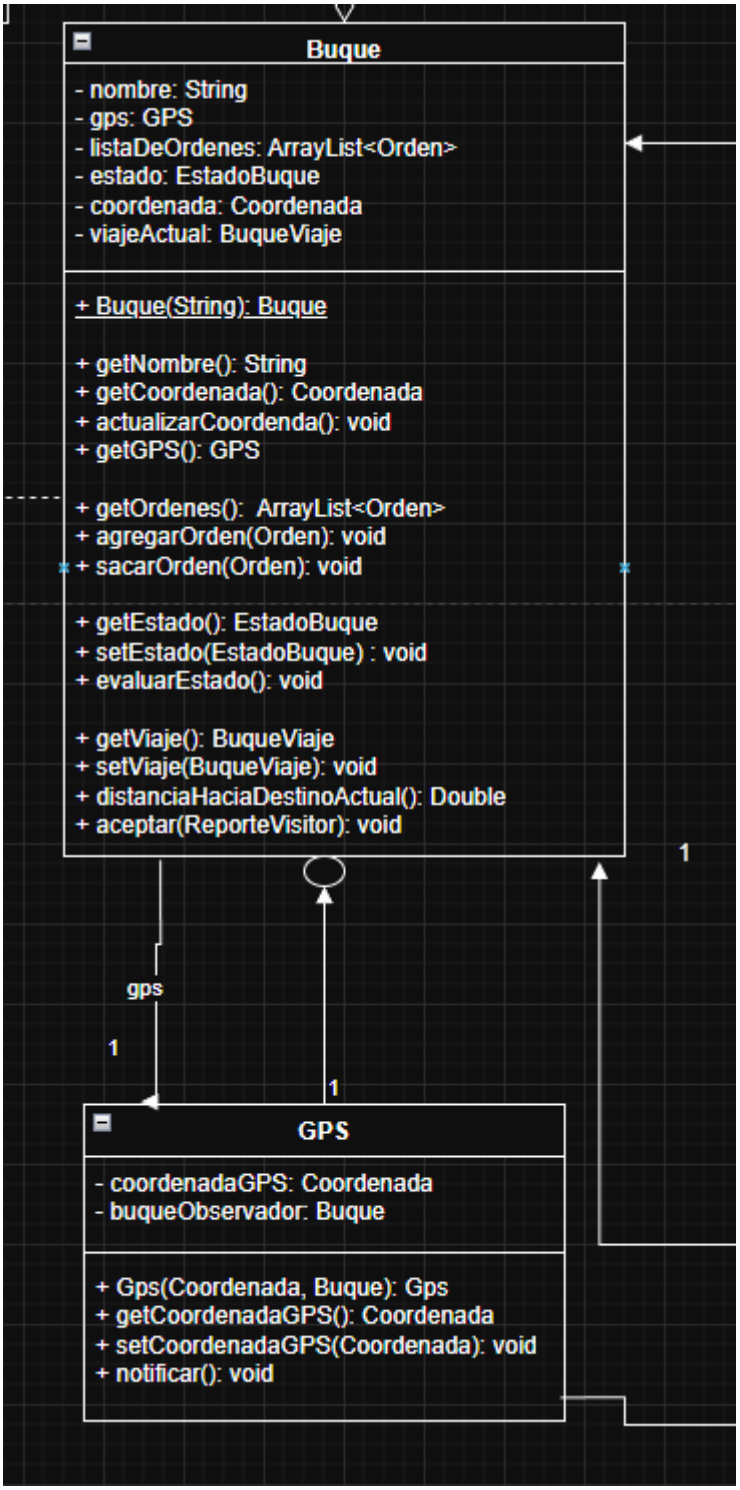
Un Buque posee una coordenada y un GPS. El GPS de un Buque también posee su propia coordenada, la cual es actualizada cada minuto. Cada vez que el GPS actualiza su coordenada debe informar al buque para obtener la nueva coordenada del GPS, y el buque basándose en la nueva coordenada determinará si tiene que cambiar de estado o no. Según el criterio del enunciado, un buque está en estado OutOfBound si se encuentra a 50km o más del destino, cuando está a menos de 50km pasa al estado InBound, y cuando la coordenada es la misma que la del destino entonces está en estado Arrived.

Se utilizó este patrón dado que necesitábamos que un buque escuche los cambios de coordenada de un GPS para que pueda obtener la coordenada actualizada y luego hacer lo que tenga que hacer. La necesidad de escuchar estos eventos y actualizarse indican un patrón Observer.

Roles en el patrón según Gamma et. al:

SujetoConcreto: en este caso es “GPS”, quien cambia de estado (cambia de coordenada) y notifica a sus observadores (en este caso es solo un buque). Posee una referencia a sus observadores para poder notificarlos.

ObservadorConcreto: en este caso es “Buque”, quien posee una referencia a su GPS para poder reemplazar el estado anterior del sujeto con el nuevo estado. Internamente, el Buque tendrá toda su lógica de cambios de estado, pero eso es trabajo de otro patrón.



Si bien el patrón observer en el diagrama de Gamma et al. implica la implementación de una abstracción tanto para los observadores como para los sujetos, en este caso como un gps tiene solo un observador (un buque) no se vio la necesidad de implementar dicha abstracción.

Patrón Composite

Diseño de los(bl basico / B/L consolidado)

Un Bill of Landing (o B/L) es un documento que especifica la carga dentro de un contenedor. Dicho documento debe ser presentado por un Shipper a la hora de querer realizar una orden de exportación. En él se detallan el tipo de producto transportado y el peso del mismo. Existe además un tipo de B/L especial llamado “B/L consolidado”, el cual está conformado por una lista de B/L's (quienes a su vez pueden ser B/L’s comunes u otros B/L’s consolidados).

Se utilizó este patrón debido a que estábamos en presencia de “una hoja” (en este caso el B/L basico) y “un objeto compuesto” (un B/L consolidado, quien está compuesto de otros B/L’s), y también porque debiamos tratarlos de manera polimórfica, ya que un Contenedor Dry conoce a su carga (representada por dicho B/L) y debe poder agregarla independientemente de su tipo. Esta necesidad de polimorfismo llevó a la creación de la interfaz "B/L", la cual es implementada por dichos documentos.

Roles en el patrón según Gamma et. al:

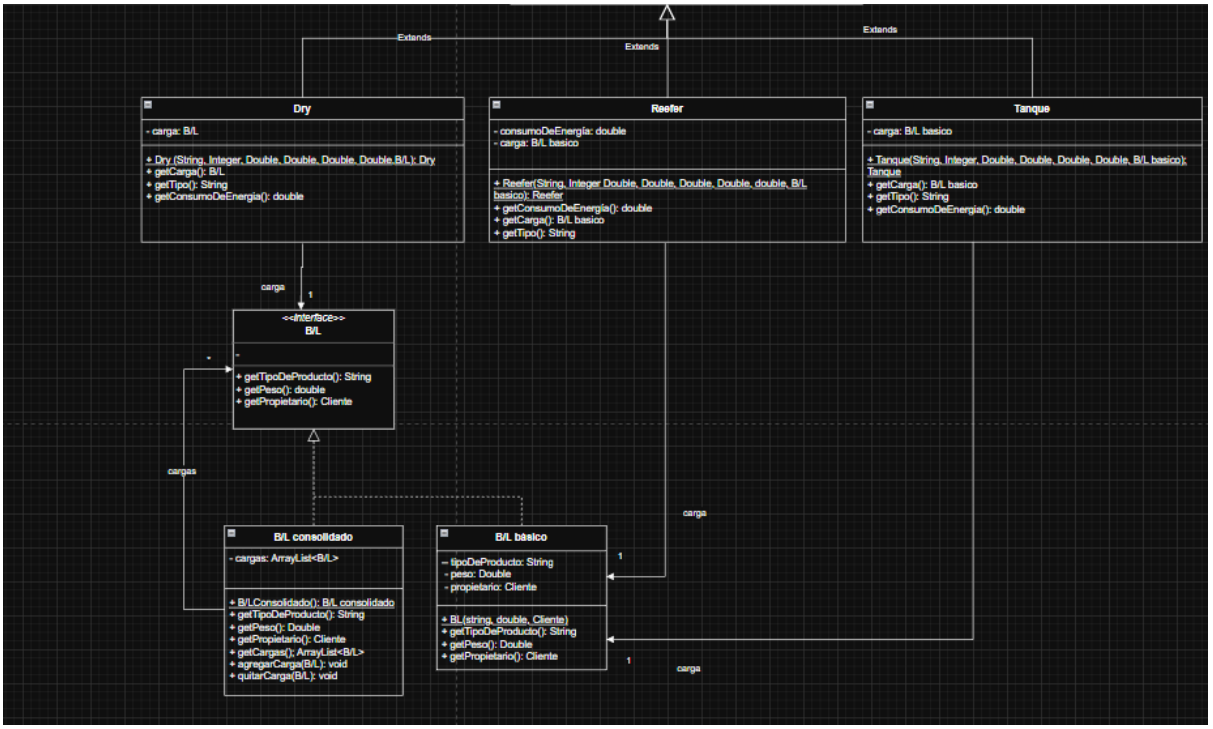
Cliente: En este caso es el contenedor “Dry”, ya que dicho contenedor puede tener uno de los B/L ya sea el básico o el compuesto entonces interactúa con los b/l tratandolos de manera polimórfica

Componente: En este caso es “bl”, es una abstracción que permite tratar tanto a las hojas como al compuesto de manera polimórfica

Hoja: En este caso es “B/L basico”, quién es un objeto primitivo que no tiene hijos y representa el eslabón más chico.

Compuesto: En este caso es “B/L Consolidado”, quien almacena hojas y es capaz de agregar y quitar hojas de su composición.

A continuación se muestra el diseño UML de esta sección:



Diseño de los filtros del buscador de rutas marítimas

Un buscador de rutas marítimas conoce a su terminal, por ende conoce los viajes de las navieras asociadas a ella, y debe ser capaz de realizar búsquedas sobre esos viajes en base a diversos filtros específicos que se le digan, todo esto para determinar la opción que cumpla con todos ellos. El dominio otorgado indica tres filtros concretos: **puerto destino, fecha de salida y fecha de llegada** (aunque en el futuro pueden agregarse otros). Estos filtros a su vez deben poder ser capaces de concatenarse para formar otros más complejos ,utilizando para ello conectores lógicos **AND** y **OR** (quienes también son filtros y están compuestos de dos filtros, que a su vez también pueden ser otros filtros AND u OR, etc). Dado que un buscador debería poder tratar de manera polimórfica a estos filtros, sin importarle si se le pasa un filtro AND o un filtro primitivo, es necesario tener una estructura polimórfica de filtros, que permita al buscador poder utilizar cualquiera de forma polimórfica.

Se utilizó este patrón debido a dichas especificaciones, tenemos filtros concretos (hojas) y filtros compuestos (compuestos de filtros hoja u otros filtros compuestos), y la necesidad de un polimorfismo por parte del buscador. Para obtener un polimorfismo se decidió crear una abstracción llamada “FiltroRuta” de la cual dependen todos los filtros, en este caso es una interfaz porque solo se necesita el mensaje, no hay una implementación en común y

tampoco se tienen atributos en común, por lo que solo con la nomenclatura de un mensaje “cumple(viaje)” es suficiente. Internamente en cada filtro, dicho mensaje poseerá en su implementación la condición que debe cumplir un viaje para ser seleccionado por éste.

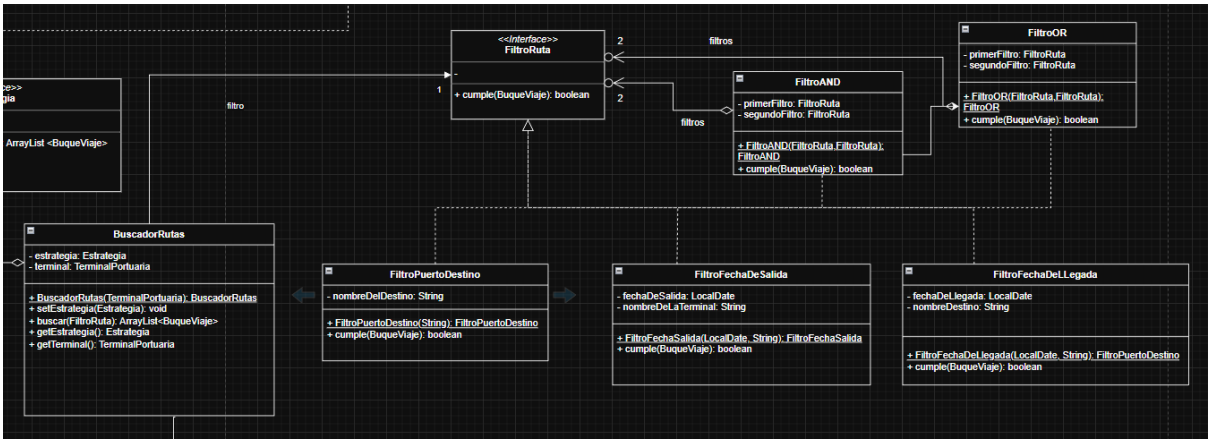
Roles en el patrón según Gamma et. al:

Cliente: En este caso es “BuscadorRutas”, quien interactúa con un filtro polimórficamente, sin importar que sea simple o compuesto.

Componente: En este caso es “FiltroRuta”, es una abstracción que permite tratar a todos los filtros polimórficamente.

Hoja: En este caso son los filtros concretos “FiltroPuertoDestino”, “FiltroFechaDeLLegada” y “FiltroFechaDeSalida”, son filtros primitivos sin hijos.

Compuesto: En este caso son “FiltroAND” y “FiltroOR”, quienes están compuestos por otros filtros. En este caso un filtro AND u OR solo puede tener dos filtros a la vez, por lo que no es posible tener un mensaje para agregar y quitar un filtro. Estos compuestos son instanciados con ambos filtros cuando son necesarios pero, al no ser guardados, luego de su uso son destruidos, evitando así el problema de crear un filtro compuesto por cada 2 condiciones distintas.



Patrón State

Diseño de los Estados de Buque

Outbound es un estado donde indica que el buque se encuentra en la fase inicial de un viaje, en otras palabras refleja que el buque se encuentra muy lejos de la terminal a la que debe llegar. Cuando un buque está en dicho estado recibe la información de las exportaciones pero no se puede realizar pagos de servicios del contenedor.
Inbound es el estado donde indica que el buque se encuentra a menos de 50 kms de distancia de la terminal a la que debe llegar. Al entrar en esta fase, el buque da aviso a la terminal sobre su inminente llegada y cuando dicho aviso la recibe la terminal envía un mail a todos los consignees notificando de que la carga está llegando. En caso de que el buque se aleje de su destino por cuestiones climáticas, el estado del buque pasa del estado Inbound al estado Outbound.
Arrived es el estado donde indica que el buque llegó a su destino, esto se confirma mediante las coordenadas, cuando las coordenadas del buque y de la terminal son exactamente iguales, después la terminal debe dar la orden de poder iniciar el trabajo y para ello lo hace mediante el uso del mensaje ponerEnWorking() que le permite pasar el estado de un buque a **Working.**
Working es el estado que indica el inicio del trabajo de descargar y cargar los contenedores (Proceso del cual no nos interesa implementar), y una vez que ya no haya más contenedores para realizar el trabajo la terminal da la orden depart mediante el mensaje ponerEnDepart() que lo lleva a la siguiente y última fase.
Departing es el estado que indica que el buque está saliendo de la terminal estando menos de un kilómetro de distancia de la misma, en el caso de que la distancia entre el buque y la terminal es mayor a un kilómetro entonces el buque da aviso a la terminal y pasa de vuelta al estado de la fase inicial Outbound. Cuando la terminal recibe dicho aviso, enviará un enviará un mail a todos los shippers avisando que todas las órdenes de exportación que están asociadas a ese viaje, ya han salido de la terminal. Cabe destacar también que al momento de que el buque está en la fase Departing se debe realizar el pago de los servicios de importación y exportación de los contenedores operados.
Se decidió usar este patrón debido a que el buque puede o no hacer determinadas cosas dependiendo de su estado, y si el comportamiento depende de su estado entonces es un patrón state.
Para obtenerlo se implementó una abstracción, en este caso, la interfaz EstadoBuque; este método de polimorfismo es idéntico al que aplicamos en

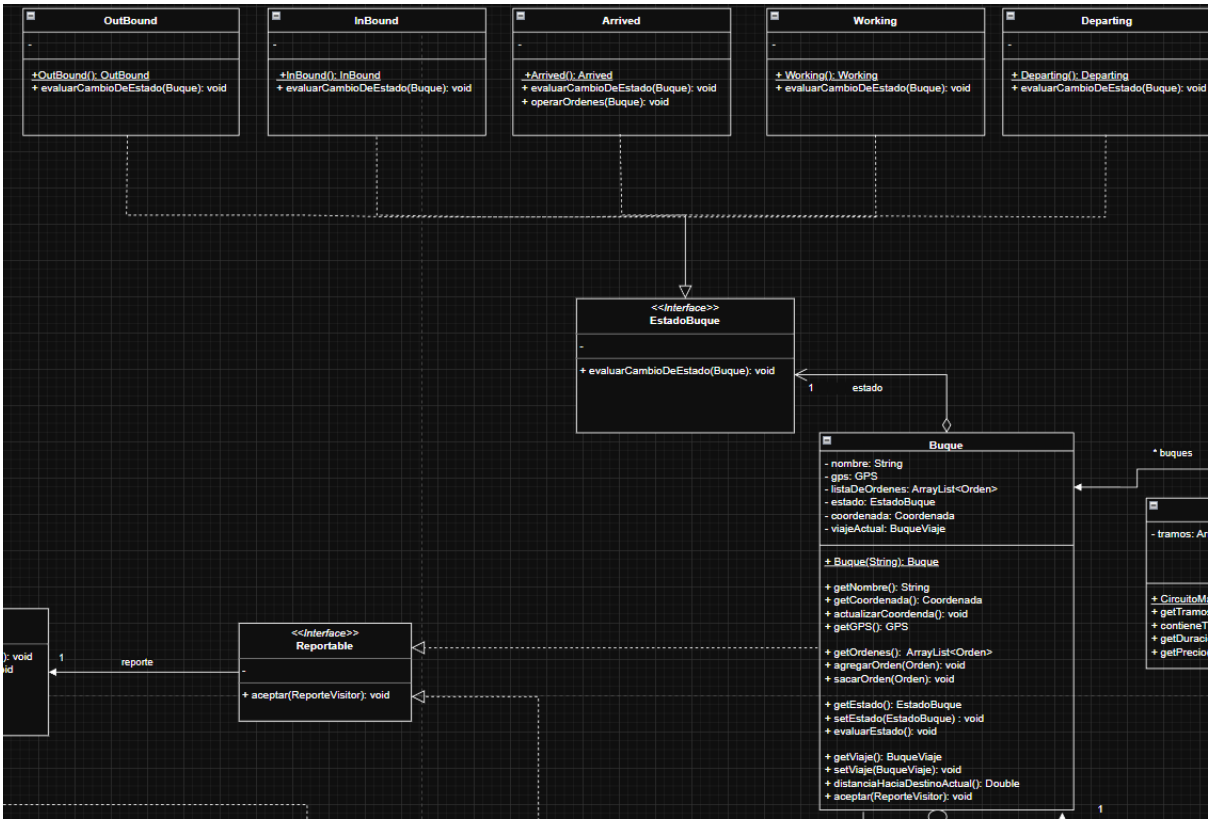
los casos de Composite que explicamos anteriormente, con la diferencia de que en esta interfaz tiene únicamente un solo mensaje llamado “evaluarCambioDeEstado(Buque)” que se comporta de forma polimorfica, donde su implementación solamente se define en cada estado concreto teniendo en cuenta los diferentes casos en que se ejecuta el cambio.

Roles en el patrón según Gamma et. al:

Contexto: En este caso es “Buque”, ya que interactúa con una interfaz de EstadoBuque que le permite poder cambiar su estado mediante el mensaje setEstado(EstadoBuque). Para lograr el cambio, el buque posee los mensajes de cambio por cada estado, los cuales internamente delega a su estado pasándose a sí mismo como parámetro, esto con el objetivo de que el estado pueda utilizar su mensaje “setEstado”. Por lo tanto, cada estado en la implementación de dichos mensajes sabe si puede transicionar a un estado o no. Por ejemplo, se puede pasar de un estado InBound a un estado Arrived, pero si se hace al revés el estado Arrived debería lanzar una excepción, ya que no se puede pasar de Arrived a InBound.

Estado: En este caso es “EstadoBuque”, es una interfaz que le permite encapsular el comportamiento asociado con un determinado estado del Buque.

Subclases de EstadoConcreto: En este caso son los estados concretos son “Outbound”, “Inbound”, “Arrived”, “Working” y “Departing son subclases del estado buque cada una con su propia implementación que se ejecuta según el estado en el que se encuentra el buque



Patrón Strategy

Diseño de las Estrategias para el Buscador de Rutas

EstrategiaMenorTiempo es una de las estrategias que se utilizan para poder averiguar cuáles de los viajes de la terminal tienen un menor tiempo de recorrido. En este caso, dicha estrategia se encarga de ordenar, de menor a mayor, aquellos viajes de las navieras registradas en la terminal gestionada en base al tiempo de recorrido del viaje. En caso de no existir devuelve una lista vacía.

EstrategiaMenorPrecio es otra de las estrategias que se pueden utilizar para averiguar los viajes cuyo precio total sea menor, ordenados tambien de menor a mayor. En caso de que dichos viajes en la lista dada no existan devuelve una lista vacía.

EstrategiaMenorTransbordo es la última de las estrategias sobre el buscador de rutas para ordenar una lista de viajes, de menor a mayor, basandose en la cantidad de tramos de dicho viaje.

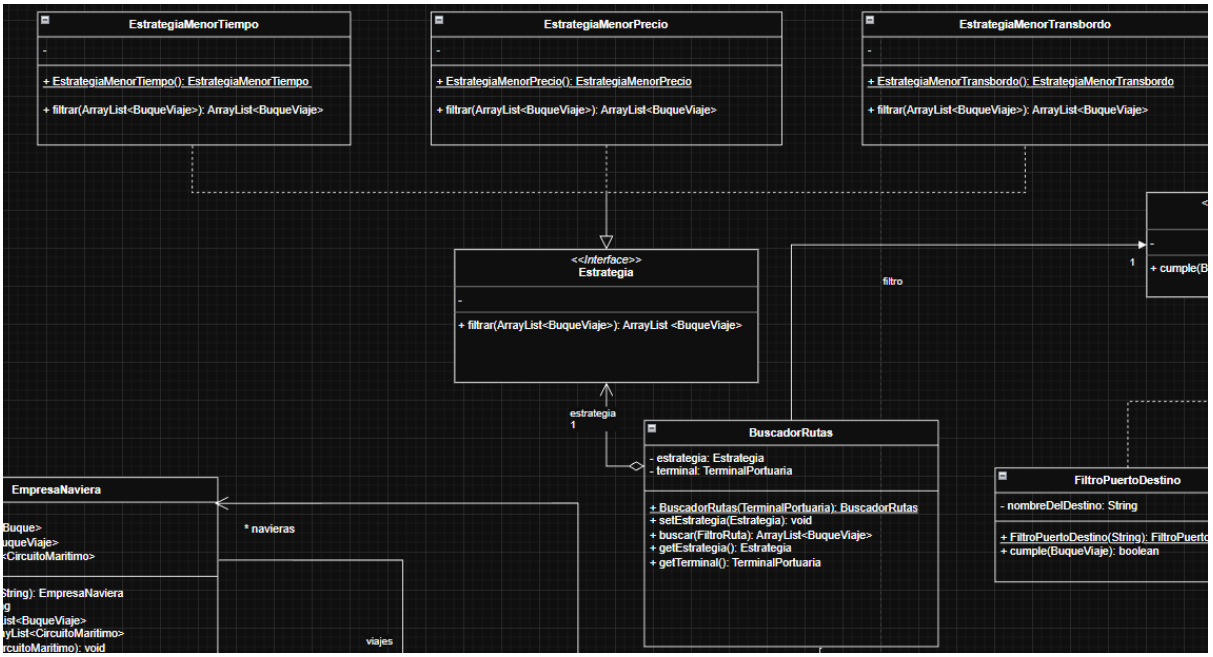
Se decidió usar este patrón debido a que el buscador de rutas posee diferentes algoritmos para ordenar viajes segun un criterio dado. difieren su comportamiento para poder ordenar los viajes.
Para obtenerlo se implementó una abstracción, en este caso, la interfaz Estrategia; este método de polimorfismo es idéntico al que aplicamos en los casos de Composite y en el caso de State que se mencionaron anteriormente, con la única diferencia de que en esta interfaz se encarga de encapsular los diferentes algoritmos teniendo en cuenta el comportamiento de cada algoritmo, para que un objeto pueda elegir un solo algoritmo y poder cambiarlo en tiempo de ejecución.

Roles en el patrón según Gamma et. al:

Contexto: En este caso es “BuscadorRutas”, ya que interactúa con la interfaz de Estrategia que le permite a dicha interfaz acceder a los datos del BuscadorRutas para poder determinar cual es la estrategia que tiene el Buscador para ordenar los viajes de la terminal.

Estrategia: En este caso es “Estrategia”, es una interfaz que le permite encapsular el comportamiento asociado con una estrategia del Buscador de Rutas.

EstrategiasConcretas: En este caso las estrategias concretas son “EstrategiaMenorTiempo”, “EstrategiaMenorPrecio” y “EstrategiaMenorTransbordo” son subclases de la estrategia donde cada una tiene su propia implementación que se ejecuta según la estrategia que tenga el buscador de rutas. Dado que se implementa una interfaz que hace que las estrategias sean polimorfismo, sería posible en un futuro agregar una nueva estrategia sin problemas.



Patrón Visitor

Diseño de los reportes Muelle, Aduana y Buque

ReporteMuelle es uno de los reportes que se utilizan para brindar información a los participantes que operan en la terminal. Este reporte es un documento en texto plano donde indica los siguientes datos: el nombre del buque, la fecha de arribo y la fecha de partida de un buque y la cantidad de contenedores que fueron importados y exportados al buque
ReporteAduana es otro de los reportes que tienen la función brindar información a los participantes que operan en la terminal. Este reporte es un documento con estructura HTML donde indica los siguientes datos: el nombre del buque, la fecha de arribo y la fecha de partida de un buque y la lista de contenedores incluyendo el tipo de contenedor que es y su id.
ReporteBuque es el último de los reportes que cumple la misma función que los otros reportes, dicho reporte se encarga de confirmar que los registros del buque coincidan con los de la terminal, para ello, se listan los identificadores de los contenedores descargados y cargados.

Para que estos reportes puedan enviar información adecuadamente a las clases relacionadas con la operación de la terminal se tuvo que implementar dos interfaces para poder acceder a los datos necesarios, que son los siguientes:

ReporteVisitor Es la interfaz que define las operaciones primitivas de cada reporte que debe poder visitar las clases de dominio necesarias, dichas operaciones son ejecutadas por los reportes definidos anteriormente, donde cada una solicita los datos de las clases que quiere visitar y así poder implementar un reporte adecuado con todos los datos correspondientes a dichas clases.
Reportable Es la interfaz implementada por las clases del dominio que pueden ser visitadas, en nuestro caso las clases son Terminal y Buque donde ejecutan el mensaje definido en la interfaz que es aceptar(ReporteVisitor) donde cada una tiene su propia implementación. El mensaje aceptar permite a uno de los reportes visitar las clases de dominio para poder acceder a sus datos fácilmente y lograr crear reportes adecuados con el fin de informar a los participantes de la operatoria de la terminal. Para esto, dicho reporte se pasa como parámetro con el objetivo que el reporte pueda saber de la existencia de dicho objeto y obtener sus datos.

Se decidió usar este patrón Visitor para la generación de reportes con el fin de desacoplar la lógica de negocio del proceso de obtención y presentación de datos ya que los reportes no contaban con toda la información necesaria para transmitir a los participantes fundamentales que operaban en la terminal. Para lograrlo se implementaron dos interfaces que se mencionaron anteriormente donde Reportable es implementada por Terminal y Buque y los reportes Aduana, Muelle y Buque implementan la interfaz ReporteVisitor, de esta manera, cada reporte accede a la información de los objetos visitados mediante el método aceptar(visitor), permitiendo agregar nuevos tipos de reportes sin modificar las clases del dominio, respetando el principio Open/Closed.

Roles en el patrón según Gamma et. al:

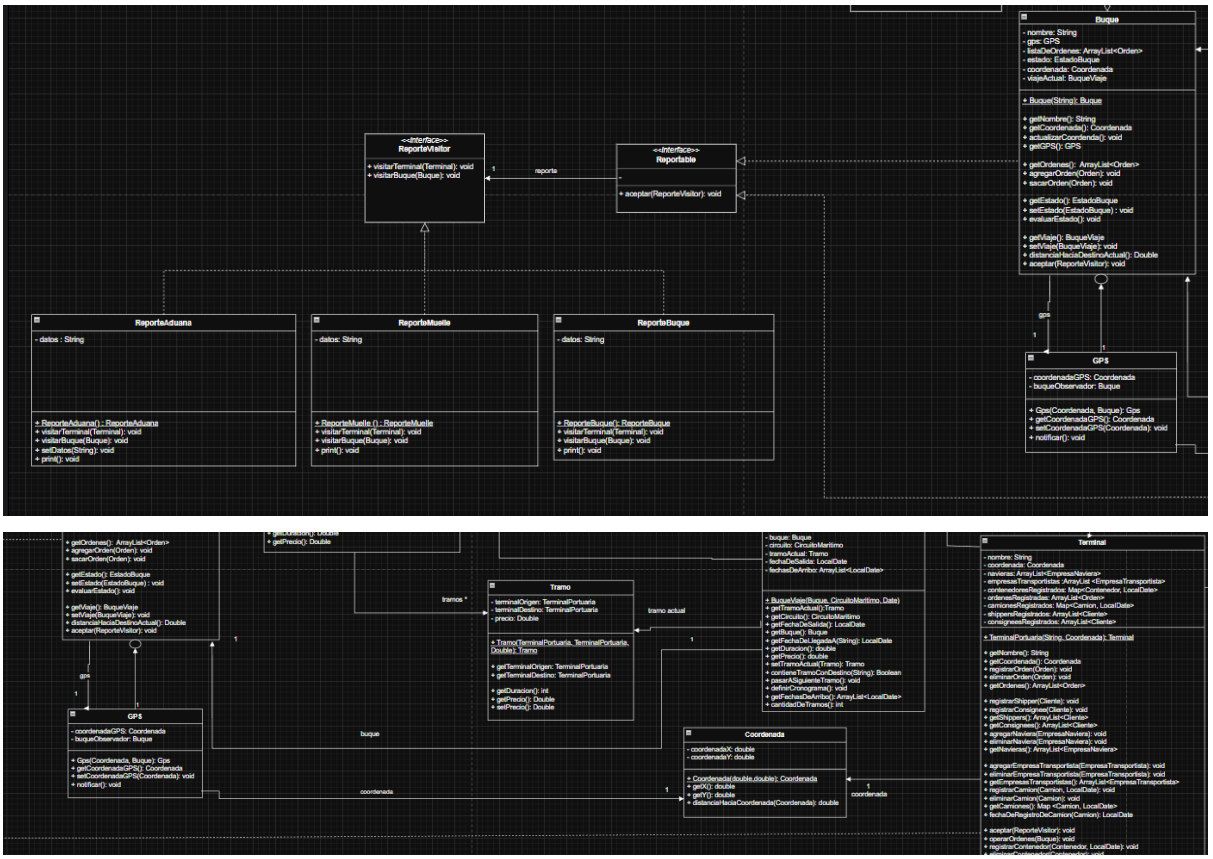
Visitante: En este trabajo, “ReporteVisitor” tiene el rol de visitante ya que declara operaciones de Visitar para las clases que tienen el rol de ElementoConcreto, el nombre y la asignatura de las operaciones permiten identificar a la clase que solicita visitar a un elemento concreto, en este caso los reportes solicitan visitar a la terminal y/o al buque

VisitanteConcreto: En este caso, los que tienen el rol como visitantes concretos son los reportes: ReporteMuelle, ReporteAduana y ReporteBuque, donde cada uno implementa cada operación que está en el ReporteVisitor,

así cada visitante concreto podrá solicitar los datos requeridos de la clase necesaria para implementar el reporte completo.

Elemento: El rol de elemento lo tiene la interfaz Reportable ya que la misma define la operación aceptar donde toma a un visitante como argumento que pueden ser uno de los 3 reportes definidos anteriormente para saber cual reporte es el que va a visitar a una clase.

ElementoConcreto: Este rol lo tienen la clase Buque y Terminal ya que, además de sus operaciones primitivas, implementan la operación aceptar visitante que toma a un visitante como argumento que le permite dar el permiso al visitante de poder acceder a sus datos con el fin de que puedan tener toda la información necesaria para realizar el reporte.



Conclusión y algunas observaciones

Ya habiendo presentado las diferentes decisiones de diseño y el por qué diseñar su resolución de la manera mostrada, se procederá a explicar diferentes inquietudes sobre el enunciado.

En este trabajo se intentó desde el inicio entender el dominio completamente con el objetivo de diseñar el diagrama UML eficientemente, lo cual tomó la mayoría de la etapa principal de desarrollo. Una de las cosas

que era vaga es *“¿En que momento se crea una orden de importación?*”.Si bien el enunciado dice que cuando una carga está llegando a una terminal y entra en estado InBound, **se avisa a aquellos consignees que vayan a recibirla por medio de un email**. Quedaba claro que un Cliente de rol Shipper es quien crea una Orden de Exportación, pero no se explicaba si con esa creación de la Exportación implícitamente debían crearse las órdenes de importación con su consignee, o es que el Cliente de rol Consignee debía a su vez crear una orden de importación en la otra terminal (aquella que sería la terminal destino en la orden de Exportación). Debido a esta inquietud se decidió optar por la segunda opción, cuya implementación es más sencilla. Sin embargo plantea un problema, ya que si Shipper como Consignee no sincronizan la creación de las órdenes (por ejemplo, un shipper crea una exportación para el consignee, pero dicho consignee no crea su importación del otro lado) puede haber muchos errores (cargas recibidas por nadie, por ejemplo).

Otra inquietud es a la hora de cargar y descargar los contenedores en un buque. Según el enunciado, el proceso de carga y descarga no se tiene en cuenta, dado que se explicó que este proceso conlleva el manejo de grúas y todo un protocolo de seguridad complejo. Si este proceso no se tiene en cuenta entonces *“¿Cómo sabemos que contenedores se cargan en el buque, y cuales se descargan?”*. Para implementar esto se pensó en aplicarlo dentro del mensaje de la terminal que provoca el cambio al estado working en un buque, así como también en el mensaje de “registrar contenedor” de la terminal, pero al hacer esto se estaría simulando un “proceso de carga y descarga”, entonces la inquietud sigue.

Otra de las inquietudes fue el tema de las fechas, al no estar muy familiarizado con el tema de las fechas y/o los objetos de tipo LocalDate nos costó encarar como implementar el código en base a las fechas para poder realizar el cálculo de costo de los servicios electricidad y almacenamiento excedente.

Por último, relacionado a la anterior inquietud, queda hablar del patrón Visitor. Fue un patrón que no se explicó en clase y se tuvo que entender por nuestra cuenta. Si bien creemos que lo entendimos, muchas de las cosas que se piden están relacionadas al proceso de carga y descarga de un buque (contenedores cargados y descargados), que según el enunciado, es un proceso que no se tiene que modelar.

A modo de conclusión, este trabajo nos ayudó a entender la importancia de los patrones de diseño orientados a objetos, ya que plantean resolución de problemas en alto nivel sin importar la implementación concreta

en cualquier lenguaje de código. Todos los patrones vistos resultan útiles en el caso apropiado y permiten abstraernos de la sintaxis.

Esperamos que el trabajo les haya gustado. Muchas gracias.