Bolaños, Rodrigo
Cuevas, Leandro leandroagustin.cuevas@gmail.com
Moreira Toja, Ignacio ignaciomoreiratoja@outlook.com.ar

# Decisiones de diseño

El TP final de Terminal Portuaria consta los patrones de diseño **strategy**, **template method**, **state** y **composite**.

La Terminal Gestionada tiene un Criterio Abstracto, dentro del cual tiene un template method. El metodo esqueleto es elMejor(), que posee el algoritmo completo del método. A su vez, cada subclase, que son clases de Criterio concretas, realizan un comportamiento diferente en el metodo comparar(Circuito, Circuito).

A su vez, las clases concretas de Criterio operan como un Strategy donde la Terminal Gestionada es el contexto y el Criterio es la estrategia para definir el mejor Circuito. La terminal gestionada contará con sólo una de las estrategias proporcionadas dejando abierta la posibilidad de proveer nuevas estrategias. Esto cumple el Open-Closed Principle. A su vez, los criterios contarán con una variable de instancia de ComparadorCircuito, que dados dos circuitos los compara y devuelve un int dependiendo de si el primero es más grande, igual o menor que el segundo. Elegimos esta doble delegación en Criterio y Comparadores y no solo en comparadores en la Terminal ya que consideramos que deja abierta a la extensión en caso de requerirse un nuevo criterio que no solamente busque el mínimo.

El Buque tiene un estado según la distancia a la que se encuentra de la Terminal, por lo cual se emplea el patrón state. Cada estado del buque conoce su siguiente, pero no está implementado dentro del diseño cuál es su siguiente sino que debe indicársele al momento de instanciar el objeto. Esta decisión es más performática porque solo se debe contar con una instancia de cada estado. Sin embargo, debe instanciarse correctamente cuál es el siguiente de cada uno ya que, de no realizarse de esta manera ocurrirían errores de comportamiento indeseados. La otra opción, no elegida, era que cada estado oficiase de constructor de su siguiente. El problema de performance de esta decisión es que, de pasar al primer estado y de nuevo al último una indeterminada cantidad de veces, iría dejando regadas instancias de memoria que no se reutilizarían más.

La clase abstracta de Estado de Buque tiene un template method que es activarGPS(Buque) donde se encuentra el esqueleto del algoritmo y luego cada Estado de Buque Concreto se encarga de definir el comportamiento. Notamos que algunos estados no dependen de su posición con respecto a la terminal, sino que dependen de un llamado externo. Es por esto que en el template establecimos una condición de pasar fase, para que en los que depende de su posición la evalúe y en los que dependen de un llamado externo (de la Terminal), devuelva siempre False para que el esqueleto del template nunca se ejecute.

La Terminal Gestionada puede realizar búsquedas marítimas con alguno o todos de los filtros, puerto destino, fecha de salida (desde la terminal gestionada) y fecha de llegada (a puerto destino). Para esta búsqueda se utiliza el patrón composite donde la interfaz Condición es el componente del patrón. Las clases Or y And, que implementan la interfaz Condición, funcionan como clases compuestas del patrón y recursionarán sobre sus dos instancias de Condición, que al ser tipado con la interfaz, pueden ser tanto componentes como compuestos. El caso base de la Condición son hojaFechaLlegada, hojaFechaSalida, hojaPuertoDestino. El cliente de esta estructura es, nuevamente, la Terminal Gestionada.

Otras decisiones de diseño no asociadas a los patrones son el uso de las librerías LocalDateTime que proporciona el manejo de fechas y horarios y Point que permite el manejo de puntos y distancias en un eje cartesiano, utilizado para medir las distancias del Buque respecto de la Terminal Gestionada.

No se solicitó la implementación de carga y descarga de los buques pero hemos decidido implementarlo de igual manera ya que nos facilitaba el seguimiento del flujo de comportamiento de los demás métodos y objetos involucrados en esa actividad.

Decisiones de diseño por fuera de patrones:

En la jerarquía de clases de los containers, decidimos que todos los Containers entiendan el mensaje getConsumo() y todos retornan 0 ya que no consumen electricidad, menos el caso del Container Reefer, que lo re-implementa y retorna su instancia de costo. Esto nos facilita al momento de desde la terminal aplicar servicios, ya que le podemos aplicar el servicio de Electricidad a cualquier container, pero en el único que va a modificar su costoDeServicios() es en el Container Reefer, ya que devuelve algo distinto a 0.