Víctor López García y Ana Mosquera Docampo

Informe ejercicio 2 – Incursiones Navales

En este ejercicio diseñamos un sistema de incursiones navales en el cual utilizamos clases abstractas y herencias para definir las diferentes estrategias que utilizamos en las incursiones como son las tormentas marinas, ataques aéreos y los avistamientos Utilizamos las herencias e implementamos interfaces para definir las estrategias y simular el efecto de estas en la flota y también utilizamos polimorfismos para simular dichas estrategias a través de la interfaz 'Estrategia'.

• Principios de diseño SOLID:

Para nuestro ejercicio utilizamos varios principios de diseño SOLID:

- Principio de Responsabilidad Única (SRP): Cada clase tiene una responsabilidad única y se centra en un solo aspecto del problema. Por ejemplo, la clase `Flota` se centra en la información de la flota, la clase `Nodo` se centra en la estructura del árbol y la clase `Estrategia` se centra en la simulación de estrategias de batalla. Esto hace que cada clase sea más fácil de entender, mantener y extender.

- Principio de Abierto/Cerrado : este principio establece que las clases deben estar abiertas a la extensión pero cerradas a la modificación. En el ejercicio utilizamos este principio en las clases que implementan la interfaz `Estrategia` que son abiertas a la extensión, lo que permite agregar nuevas estrategias de batalla sin modificar el código existente. Además, la clase `Nodo` está cerrada a la modificación, ya que su estructura y comportamiento las definimos en la clase base y no se pueden modifican en las subclases.

- Principio de Sustitución de Liskov : este principio establece que las subclases deben ser sustituibles por sus clases base sin afectar el comportamiento del programa. En el ejercicio, las subclases que implementan la interfaz `Estrategia` son sustituibles por la clase base `Nodo` sin afectar el comportamiento del programa. Esto permite que las estrategias se traten de manera uniforme y que se puedan agregar nuevas estrategias sin afectar el comportamiento del programa.

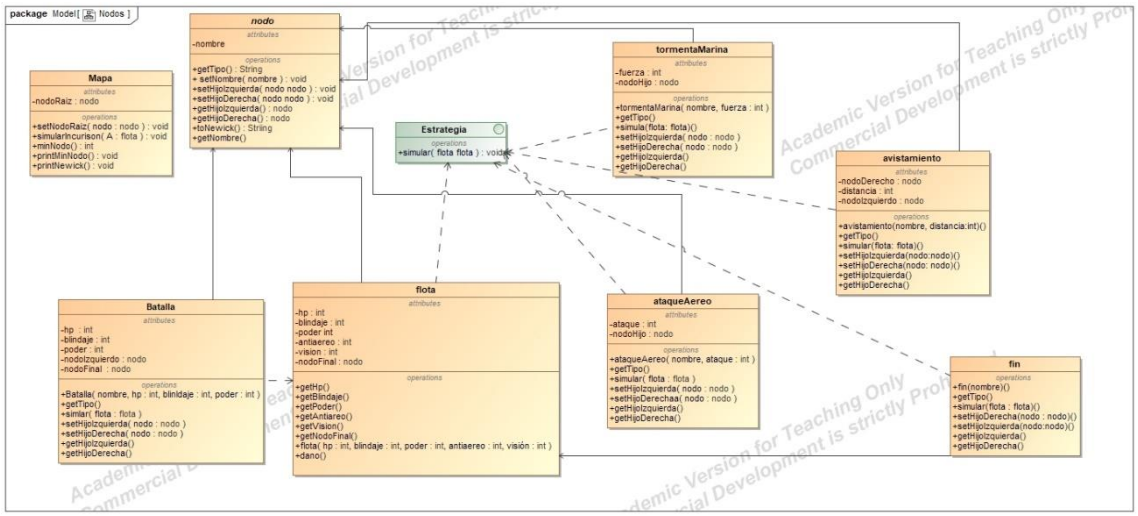
- Principio de Segregación de Interfaces: establece que las interfaces deben ser específicas para los clientes que las utilizan. En el ejercicio utilizamos la interfaz `Estrategia` donde definimos solo los métodos necesarios para simular una estrategia de batalla, lo que permite que las clases que implementan la interfaz se centren solo en la simulación de estrategias y no en otros aspectos del problema.

• Patrón de diseño - Patrón Estrategia

Escogimos el patrón estrategia ya que se utiliza para definir una familia de algoritmos, encapsular cada uno de ellos y hacerlos intercambiables. En este caso, las clases que implementan la interfaz `Estrategia` representan diferentes estrategias de batalla, como tormentas marinas, ataques aéreos y avistamientos.

Cada estrategia encapsula un algoritmo específico para simular su efecto en una flota, y estas estrategias son intercambiables, lo que permite que la flota pueda utilizar diferentes estrategias en tiempo de ejecución.

También utilizamos el patrón composición aunque no lo utilicemos explícitamente, al utilizar una estructura de árbol para representar y simular las estrategias sugiere que exista la presencia de este patrón. La clase `Nodo` actúa como componente base que define la interfaz común para todos los nodos del árbol, ya sean estrategias individuales o nodos compuestos que contienen otras estrategias. Esto permite tratar tanto a las estrategias individuales como a las composiciones de estrategias de manera uniforme.



• Diagrama dinámico - Diagrama de secuencia:

Para este ejercicio elegimos el diagrama de secuencia, ya que, al usar el patrón estrategia, nuestro código se adapta perfectamente a este tipo de diagrama, decidimos descartar el diagrama de estados y el de comunicación ya que no nos parecía que nuestro código se adaptase a alguno de estos. Una vez escogido el tipo de diagrama realizamos uno básico donde se pueden apreciar todas las iteraciones realizadas.