Facultad de Ciencias, UNAM
Modelado y Programación
Proyecto Final

Hernández Morales José Ángel
No. de cuenta: 315137903
Rubí Rojas Tania Michelle
No. de cuenta: 315121719

04 de septiembre de 2020

1. Descripción del proyecto

   El proyecto final consiste en la simulación de un juego de soldaditos. Al inicio del juego, podemos elegir entre tres diferentes ejércitos para poder combatir a un enemigo. Después de que se nos despliega una pantalla con las características de cada uno de nuestros soldados, inicia el juego. Tenemos tres acciones disponibles:

   - Atacar. Al seleccionar esta opción, los comandantes mandarán la órden de atacar al enemigo. Si la distancia de los soldados con respecto al enemigo es igual a 0, entonces pueden atacar.
   - Mover. Los soldados se encuentran a una distancia inicial del enemigo, así que al seleccionar esta opción lo que pasa es que los comandantes mandarán la órden de que los soldados se muevan en dirección al enemigo.
   - Reportar. Al seleccionar esta opción, los comandantes mandarán la órden de reportarse a los soldados. Esto simplemente nos mostrará algunas características de los soldados, como lo son el nombre, ID, puntos de vida, distancia con respecto al enemigo y el tipo de soldado que es.

   Esta rutina se repetirá hasta que los puntos de vida del enemigo sean igual a 0. En este momento, el juego termina. Para jugar otra partida, bastará con volver a ejecutar el programación principal.

2. Funcionalidad del programa

   a) Primero, debemos posicionarnos dentro de la carpeta **src**. Aquí se encuentran todas las clases de nuestro proyecto.

   b) Compilamos todas nuestras clases.

   ```
   $ javac *.java
   ```

   c) Ejecutamos nuestro programa principal.

   ```
   $ java Main
   ```

3. Patrones de diseño utilizados

   - Factory

     Con ayuda de este patrón podemos delegar la responsabilidad de la creación de soldados a la clase `Factory`, para que no seamos nosotros los que decidamos qué clase instanciaremos; sino que confiaremos en que `Factory` nos regrese el soldado que deseamos. `Factory` decidirá qué `Soldado` crear por medio de una condición en tiempo de ejecución.

     Tenemos la clase abstracta `AbstractFactory`, la cual define el comportamiento por default de los `Factory`. La clase `Factory` representa una *fábrica* concreta, la cual es utilizada para la creación, gracias al patrón `Strategy`, de los soldados `DeArtillería`, `DeCaballería` y `DeInfantería`; los cuales corresponden a la clase `ConcreteProduct`.

   - Composite

     Con ayuda de este patrón creamos y manejamos una jerarquía (en forma de árbol) entre los comandantes y los soldados. Esto resuelve la comunicación entre éstos dos.

     La clase `Componente` en este caso es la interfaz `Soldado`, la cual contiene el comportamiento mínimo que debería tener un soldado. Ésta implementa cuatro clases: `Comandante`, `DeArtillería`, `DeCaballería` y `DeInfantería`. Fue diseñada así ya que todos ellos son soldados, y en particular, para poder combinarla junto con el patrón `Strategy` en el caso de los soldados subordinados. La clase `Composite` correspondería a la clase `Comandante`, ya que éste es el de mayor jerarquía entre los soldados. Puede agregar, eliminar y obtener la lista con todos los soldados que tiene como subordinados. Gracias a cómo está implementada la interfaz `Soldado`, al realizar las acciones propias de un soldado, éste hace que todos sus soldados subordinados realizen dicha acción. Finalmente, la clase `Hoja` corresponde a las clases `DeArtillería`, `DeCaballería` y `DeInfantería` ya que éstos no contienen a otros elementos.Así, cada uno realiza la acción que les corresponde, y nada más.

   - Strategy

     Con ayuda de este patrón creamos un `Soldado` que pueda comportarse de formas diferentes (lo cual se define en el momento de su creación). Aunque todos los soldados realizarán las acciones mínimas correspondientes a un `Soldado`, ellos lo hacen de diferente modo, dependiendo de sí es un soldado `DeArtillería`, un soldado `DeCaballería` o un soldado de `DeInfantería`.

     La interfaz `Estrategia` corresponde a la interfaz `Soldado`. Ésta contiene el comportamiento mínimo que debe realizar cada soldado. Las clases `EstrategiaConcretaN` corresponden a las clases `DeArtillería`, `DeCaballería` y `DeInfantería`.

   - Observer

     Con ayuda de este patrón podemos notificar a los comandantes cuando la acción que quiere realizar el usuario cambia. Esto resuelve la comunicación entre el usuario y los comandantes (los cuales, a su vez, le comunicarán a sus soldados subordinados la acción que deseea el usuario).

     Tenemos las interfaces `Sujeto` y `Observador`, las cuales implementan las clases `Usuario` y `Comandante`, respectivamente. La clase `Usuario` puede agregar, eliminar y notificar a los observadores, que en este caso se tratan de los comandantes. Éstos a su vez, logran notificar a los soldados la acción que deben realizar, de acuerdo a los deseos del usuario.

   Para la clase principal `Main` no utilizamos algún patrón de diseño ya que considerámos que no era necesario.