Diseño y Programación Orientada a Objetos

Taller 5

Jaime Esteban Moreno Jaramillo

Código: 202220189

El patrón seleccionado para este taller fue el flyweight. El objetivo principal de este patrón es el ahorro de memoria y lo hace al evitar la creación de objetos con tipo de información idénticos. Este patrón tiene 2 objetos: flyweight y flyweightFactory. Flyweight es el objeto que será implementado por todos los objetos similares el cual llamaremos "A", mientras que el otro se encarga de crear y gestionar todos los demás objetos. Es decir, evitar que se creen objetos idénticos y crearlos cuando no existan. Para revisar si el objeto ya existe, se usa un mapa que contendrá objetos de tipo "A", esto para evitar la creación de objetos innecesarios y poder devolverlo con facilidad si ya existe.

Link proyecto seleccionado: https://github.com/peterm85/design-patterns/tree/master/src/flyweight/examples/counterstrike

Hablando sobre el proyecto que seleccione, se llama "Counter Strike" y lo que hace es simular un spawn de personajes del counter strike. Como en el juego, hay 2 bandos terroristas y antiterroristas (en el programa se llaman terrorists y counter terrorists), ademas cada personaje cuenta con una arma y una tarea la cual varia del bando, si es terrorists su tarea es: "PLANT A BOMB" mientras que si es counter terrorists su tarea es "DIFFUSE A BOMB". Entonces el programa crea objetos de tipo "Player" los cuales cuentan con un arma, un equipo y una tarea. La arma y el equipo se eligen de manera aleatoria, mientras que la tarea se crea cuando se crea el objeto terrorists o counter terrorist. Cada vez que crea un objeto lo añade a un mapa que tiene por llave un string y por valor un objeto "Player"

Sobre las clases del programa. Hay una clase de tipo interfaz que tiene 2 metodos: assignWeapon y mision. AssignWeapon le asigna el arma al objeto y mision le asigna la misión al objeto. Además de esta, hay una clase para terrorist y otra para counter terrorists las cuales implementasn player. Eso en cuanto los objetos, sobre la ejecución están: CounterStrikeTest y PlayerFactory. CounterStrikeTest ejecuta el programa, es decir que tiene el main y además las listas de posibles armas y de posibles jugadores, por ende, también tiene 2 metodos: getRandPlayerType y getRandWeapon.Por otra parte, PlayerFactory guarda el mapa y tiene un método getPlayer(). Este método devuelve el objeto asociado al tipo de jugador y en caso de que no exista añade otro.

El patron se usa a la hora de crear jugadores. Como los jugadores se crean usando el método get, es decir que intenta obtener el objeto asociado al jugador del mapa y si no existe lo crea. El usar get evita que se creen objetos idénticos. También, se puede ver en la implantación de Player, gracias a que terrorists y counter terrorists implementar Player en el mapa solo toca guardar objetos de tipo Player lo que ayuda mucho a la hora de hacer el get antes mencionado.

El patrón se ve en los siguientes fragmentos:

```
public class Terrorist implements Player {
```

archivo Terrorist.java linea 3

```
public class CounterTerrorist implements Player {
```
archivo counterterrorist.java linea 3

El hecho que las clases que crean los objetos es algo muy característico del patrón flyweight.

```
10    public static Player getPlayer(String type) {
11          Player p = null;
12
13          /* If an object for TS or CT has already been
14          created simply return its reference */
15          if (hm.containsKey(type)) {
16
17                p = hm.get(type);
18
19          }else {
20                switch(type){
21                      case "Terrorist":
22                         System.out.println("Terrorist Created");
23                         p = new Terrorist();
24                         break;
25                      case "CounterTerrorist":
26                         System.out.println("Counter Terrorist
Created");
27                         p = new CounterTerrorist();
28                         break;
29                      default :
30                         System.out.println("Unreachable code!");
31                }
32
33                // Once created insert it into the HashMap
34                hm.put(type, p);
35          }
36          return p;
37    }
```
Archivo PlayerFactory.java en la parte izquierda se ven las líneas.

El patron flyweight se caracteriza por evitar que existan 2 objetos iguales y para agregar objetos se con un get, haciendo que si quiere añadir un objeto que ya existe el programa le devuelve ese objeto y no cree otro.

Ahora se hablará sobre las ventajas de usar este patrón en este proyecto. La primera ventaja es en cuanto a espacio. Debido que para crear objetos se usa un get, si el objeto ya existe se obtendrá ese objeto y no creara otro idéntico haciendo que se ahorre el espacio que desperdiciaría si se crearan 2 objetos iguales. Otra ventaja es la flexibilidad para añadir jugadores. Si se quiere añadir otro jugador solo toca crear su clase y en el switch otro case, esto gracias a que los tipos de jugadores están implementados con Player.

Sobre desventajas no se logran notar muchas ni muy graves. Sin embargo, hay una que podría ser la flexibilidad para añadir ítems a los jugadores. Esta flexibilidad se ve reducida por el hecho que toca hacer un paso de más, añadir el nombre del método en la interfaz. Pero no es algo muy grave ni que sea una gran pérdida de tiempo.