Lab2

INTRODUCCIÓN

En este tercer laboratorio hemos afrontado la implementación de una nueva mecánica para nosotros. A la vez que hemos diseñado muchos más objectos que interactuaban entre sí, hemos aprendido a manejar el concepto de la herencia. En este laboratorio nuestro programa implementa nuevos tipos de competiciones las cuales heredan atributos i métodos de un padre en común. También hemos implementado dos tipos de jugadores que se detallaran más adelante, así como dos tipos diferentes de equipos de los cuales solo un ha necesitado redefinirse como un nuevo objecto independiente. Finalmente, hemos diseñado dos tipos nuevos de metodologías de partidos i hemos extendido el código del objecto "country".

En primer lugar, tenemos uno de los nuevos objectos más importantes que es el objecto "competition". Este dará herencia a los objectos "league", "groupPlay" y "cup", los cuales tienen métodos y atributos en común. El objecto "competition" no se usa de forma directa ya que son sus tres hijas las que se darán en nuestro código main.

El primer objecto hijo sería el "League" el cual ya estaba definido en el anterior lab. En segundo lugar, tenemos al "cup", este objecto sin duda uno de los más difíciles objectos en cuanto a la complejidad de sus métodos.

El objeto "cup" consta de un array de objectos "team" y otro de objectos "match". En cuanto a sus métodos, los más sofisticados son el "generateMatches" el cual, dada una lista de equipos, las mezcla entre ellas i después las va agrupando en pares para guardarlas en un objecto "match" y almacenarlo en la lista de partidos "matchList". En caso de que se den impares equipos de entrada, el que no pueda jugar en esa ronda, jugará en la siguiente y así iterativamente. Finalmente, el método "simulateMatches" se encarga de recibir los resultados de cada partido y almacenarlos. Cada equipo que pierda se irá removiendo de la lista de equipos, de esta manera tendremos siempre actualizada nuestra lista.

El objecto "GroupPlay" se puede abstraer como un conjunto de ligas ya que la disposición de cada partido es igual que la de un partido de liga. En el método "generateMatches" se trata de forma iterativa con hasta tres bucles "for" los cuales permite que para cada liga se juega todos los partidos en esa lista de equipos. Finalmente, su método "simulateMatches" es el mismo que presenta el objecto "League".

```
public void generateMatches(){
    for (int i = 0; i<noGroups; i++){
        for (int a = 0; a < teamList.size(); a++) {
            Team homeTeam = teamList.get(i);
            for (int j = a + 1; j < teamList.size(); j++) {
                Team awayTeam = teamList.get(j);
                Match match1 = new Match(homeTeam, awayTeam)
                matchList.add(match1);
            }
        }
    }
}
```

Ilustración 1: Objecto "GroupPlay" método "generateMatches"

Por otro lado, hemos tenido que diseñar dos tipos de equipos de equipos. Uno sería el diseño de equipo estándar que es el que teníamos hasta ahora y otro sería el diseño de equipo nacional o como lo hemos nombrado "NationalTeam". Este objecto solo consta de su constructor i su método que lo hace particular, el "addPlayer". Este método llama simplemente al método "equals" del objecto "Country", el cual evaluará si ese jugador que se quiere añadir al equipo tiene a misma nacionalidad que el equipo.

En cuanto a aquello que implica la gestión de los partidos esta vez se ha definido una relación de herencia entre el objecto “Match”, el cual ya definimos anteriormente, y su hijo “CupMatch”. Este hijo está especializado para las competiciones de clase “Cup”. En ella, tenemos redefinido el método “simulateMatch”. Este método no es más complejo que el que define su padre, solo se asegura que no se dé un caso de empate ya que una competición “Cup” no puede terminar en empate.

Finalmente, una de las actualizaciones más importantes en nuestro laboratorio ha sido el definir dos subclases que heredan de “Player”. “Outfielder” y “Goalkeeper”, refiriéndose a jugador de campo y portero respectivamente se han conceptualizado en dos clases distintas para tal de aportarle los atributos que define cada uno. “Outfielder” redefine el método “updateStats” de tal forma que hace lo que para antes era el “updateStats” convencional, pero sin la actualización de nuestro número de partidos. En segundo lugar, el objecto “Goalkeeper” redefine el concepto del “updateStats” en base a sus nuevos atributos, el número de paradas i el número de goles asistidos.

METODOLOGÍA

En el transcurso de este laboratorio, nos hemos visto inmersos en la tarea de abordar y resaltar la significativa importancia de la herencia entre clases, lo cual ha representado un desafío considerable. Exploramos varias alternativas en relación con la organización general del código, y en algunas de nuestras propuestas iniciales, la aplicación de herencias no fue considerada como una intervención necesaria, razón por la cual decidimos descartarlas. No limitándonos únicamente a implementar la herencia, también hemos incorporado un enfoque claro de polimorfismo.

A través del concepto de "overriding" o sobreescritura, hemos tenido la capacidad de modificar ciertos métodos según las necesidades específicas de cada subclase. Este nivel de flexibilidad no solo mejora el modularidad del código, sino que también permite adaptar el comportamiento de cada clase derivada de manera más precisa. El resultado es un diseño más robusto y adaptable que aprovecha al máximo las ventajas de la herencia y el polimorfismo en el contexto del desarrollo de software. Este proceso ha profundizado en nuestra comprensión de los principios fundamentales de la programación orientada a objetos, enriqueciendo así nuestra experiencia en este ámbito.

CONCLUSIÓN

Finalmente, cuando teníamos todo el código ya diseñado tocaba implementarlo en un código main lo cual sabíamos que derivaría en muchas situaciones donde nos daríamos cuenta de múltiples errores que hubiéramos cometido. En primera instancia, declarábamos una serie de países, así como de jugadores tanto porteros como de campo, y una serie de equipos, ya fueran nacionales o clubes.

En nuestra implementación ponemos "trampas" en nuestro programa como por ejemplo añadir el equipo nacional español a la copa del rey, algo que evidentemente debería resultar en error. Con estas implementaciones testeamos los límites de nuestro código.

```
kingsCup.addTeam(barcelona);
kingsCup.addTeam(realMadrid);
kingsCup.addTeam(sevilla);
kingsCup.addTeam(valencia);
kingsCup.addTeam(barcelonaFem);     // Ha de sortir malament (genere)
kingsCup.addTeam(espanya);          // Ha de sortir malament (no és un club)
kingsCup.addTeam(boca);             // Ha de sortir malament (nacionalitat)
```

A partir de este punto, nos dedicamos a implementar cada competición y a través de diversos ensayos y pruebas, logramos resolver algunos errores, ya sea relacionados con la ejecución incorrecta de LinkedList o con la implementación errónea de ciertos tipos de datos.

En conclusión, hemos logrado con éxito incorporar los conceptos fundamentales de herencia y polimorfismo en nuestro proyecto, demostrando así nuestra habilidad para desarrollar un código robusto y funcional.