Notas Práctica 2 POO

Nuria Gómez Blas

3/11/2017

Diseño UML

Se plantea la siguiente aproximación a partir de las clases que se muestran en el diagrama UML de la figura 1.

Aparecen dos nuevas clases que habrá que codificar con la funcionalidad necesaria, además será necesario modificar (añadir algún método/atributo) en alguna de las clases de la práctica anterior.

- Relación doble de asociación de `Posicion` con `Equipo` (en ambos sentidos):

```
public class Equipo {
  private Posicion posicion;

  public void setPosicion(Posicion posicion) {
    this.posicion = posicion;
  }
}
```

```
public class Posicion {
  private Equipo equipo;

  public Posicion(Equipo equipo) {
    this.equipo = equipo;
    this.equipo.setPosicion(this);
  }
}
```

Como se puede apreciar, en el constructor se hace la llamada `this.equipo.setPosicion(this)` para establecer la asociación de `Equipo` con `Posicion`.

- Relación de composición de `Clasificacion` con `Posicion` (se establece como herencia de una clase List).

```
public class Clasificacion extends ArrayList<Posicion> {

  public Clasificacion(List<Equipo> equipos) {
    for(Equipo equipo:equipos) {
      Posicion posicion = new Posicion(equipo);
      this.add(posicion);
    }
  }

}
```

Como se puede apreciar en el contructor también se hace `new Posicion(equipo)` para establecer la relación de asociación de `Posicion` con `Equipo`.

- Relación de composición de `Liga` con `Clasificacion`.

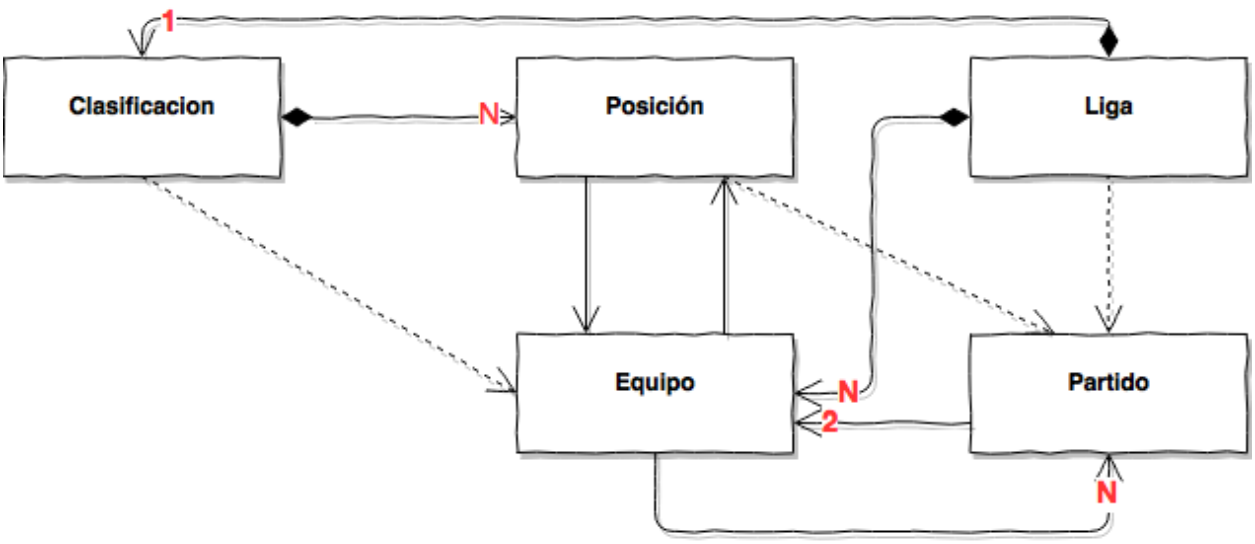

Figure 1: Diagrama de clases *UML* correspondiente a la práctica 2.

```
public class Liga {

  private List<Equipo> equipos = new ArrayList<>();
  private Clasificacion clasificacion;

  public Liga(List<Equipo> equipos) {
    this.equipos = equipos;
  }

  public Liga() {
    this.crearEquipos();
    clasificacion = new Clasificacion(equipos);
  }
```

Se puede apreciar que en el constructor de la clase `Liga` se realiza la llamada `new Clasificacion(equipos)` posteriormente a crear los equipos ya que al crear la clasificación los equipos deben estar creados para establecer la asociación de `Posicion` con `Equipo`.

La `Clasificacion` usa el array de `Equipo` para establecer correctamente todas las relaciones.

Gestión de la clasificación

Cada vez que se anota un resultado en un partido (clase `Partido` método `setResultado`) hay que actualizar los datos de la clasificación.

- Ya que cada equipo está asociado a su posición (que contendrá todos los datos de goles, puntos, etc. . .) tan sólo hay que notificar a cada equipo que actualice su clasificación.

```
public void setResultado(int golesLocal, int golesVisitante) {
  this.golesLocal = golesLocal;
```

```
  this.golesVisitante = golesVisitante;

  this.equipoLocal.actualizaClasificacion(this);
  this.equipoVisitante.actualizaClasificacion(this);
}
```

- Y el método `actualizaClasificacion(Equipo equipo)` accede a la posición del equipo para actualizar los datos de acuerdo al resultado del partido.

```
public void actualizaClasificacion(Partido partido) {
  this.posicion.actualizaPosicion(this, partido);
}
```

- En la clase `Posicion` se actualizan todos los datos.

```
public void actualizaPosicion(Equipo equipo, Partido partido) {
  if ( partido.haGanado(equipo) ) {
    puntos += 3;
    partidos_ganados++;
  } else if (partido.haEmpatado(equipo)) {
    puntos++;
    partidos_empatados++;
  } else partidos_perdidos++;

  goles_favor += partido.golesFavor(equipo);
  goles_contra += partido.golesContra(equipo);
}
```

Polimorfismo básico

Se plantea a modo didáctico la siguiente jerarquía de herencia.

La figura 2 muestra la jerarquía de herencia planteada.

- Clase `PosicionAscenso`

```
public class PosicionAscenso extends Posicion {
  public PosicionAscenso(Posicion posicion) {
    super(posicion);
  }

  public String toString() {
    return "ASC ---> " + super.toString();
  }
}
```

- Clase `PosicionDescenso`

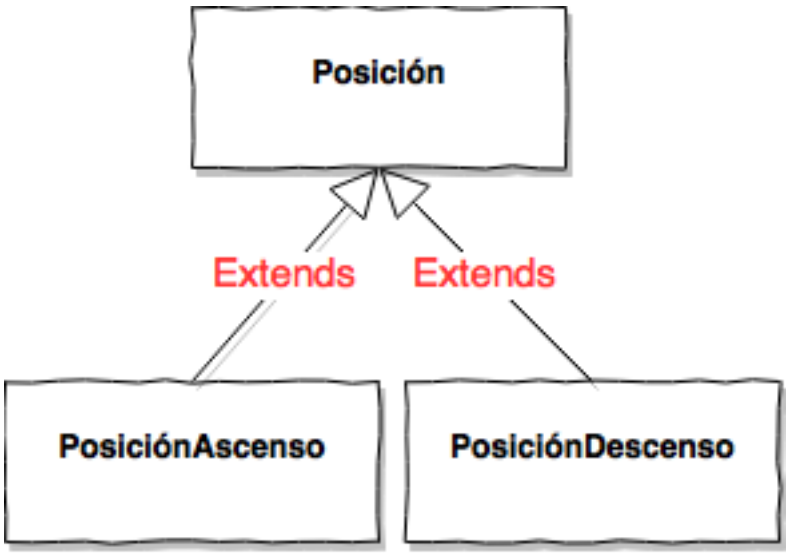

Figure 2: Diagrama de clases *UML* correspondiente a la práctica 2.

```
public class PosicionDescenso extends Posicion {
  public PosicionDescenso(Posicion posicion) {
    super(posicion);
  }

  public String toString() {
    return "DSC ---> " + super.toString();
  }
}
```

- Se ha modificado la clase `Posicion` para añadir un constructor de copia.

```
 public Posicion(Posicion posicion) {
  this.equipo = posicion.equipo;
  this.equipo.setPosicion(this);
  this.goles_favor = posicion.goles_favor;
  this.goles_contra = posicion.goles_contra;
  this.partidos_perdidos = posicion.partidos_perdidos;
  this.partidos_ganados = posicion.partidos_ganados;
  this.partidos_empatados = posicion.partidos_empatados;
  this.puntos = posicion.puntos;
}
```

Notesé que se ha vuelto ha establecer la relación de asociación `Posicion` con `Equipo` para no perder las referencias.

A la hora de gestionar la clasificación (clase `Clasificacion`) se ha modificado el método `visualizarClasificacionOrdenada()` de manera que:

1. Primero se ordena la lista de posiciones de acuerdo a los puntos obtenidos,
2. posteriormente todos los elementos se convierten a objetos ´Posicion´empleando el constructor de copia. Ya

que al ganar/perder un equipo puede subir/bajar a puestos de ascenso/descenso y desplazar a otro equipo a un puesto *intermedio*,

3. luego se ajustan los equipos que ascienden y descienden,
4. y por último, se visualiza la clasificación empleando `toString()`.

```
public void visualizarClasificacionOrdenada() {
 this.sort(Comparator.comparing(Posicion::getPuntos).reversed());

 for(int i=0; i<this.size(); i++)
   this.set(i, new Posicion((Posicion)this.get(i)));

 for(int i=0; i<3; i++) {
   // ascenso
   this.set(i, new PosicionAscenso(this.get(i)));
   // descenso
   this.set(this.size()-i-1, new PosicionDescenso(this.get(this.size()-i-1)));
 }

 System.out.println(this.toString());
}
```

De acuerdo a los principios de herencia y polimorfismo el código siguiente de clase `Clasificacion`:

```
public String toString() {
  String msg = "";
  for(Posicion posicion:this)
      msg += posicion + "\n";
  return msg;
}
```

reconoce la clase del objeto que hay en el array de posiciones `this` y redirige los mensajes `toString()` a los objetos adecuados (`Posicion`, `PosicionAscenso` o `PosicionDescenso`).