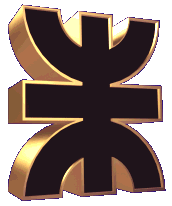
|  |
| --- |
| **Nombre y Apellido** |
| Cappellini Darío |
| Carella Germán |
| Centurion Damian |
| Fage Victor |
| Gonzalez Santiago |

|  |  |
| --- | --- |
| **Fecha De Presentación:** | 18/06/2014 |
| **Fecha de Devolución:** |  |
| **Calificación** |  |
| **Firma Profesor** |  |



**GRUPO N° 7**

TP Anual: 4ta Entrega

***Diseño de Sistemas 2014***

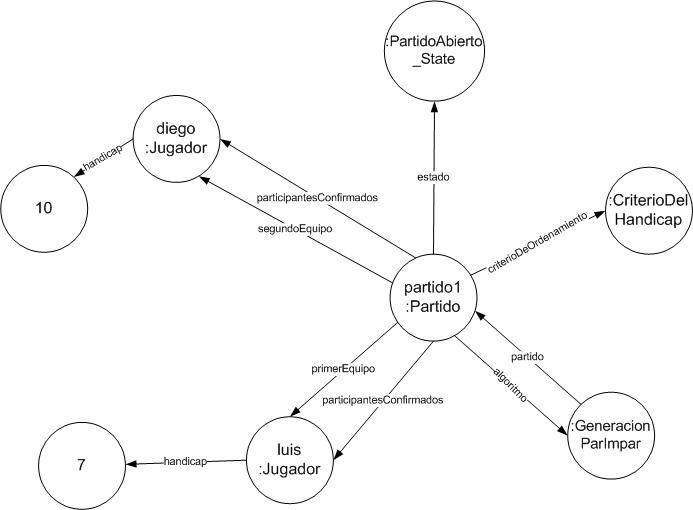
***Miércoles mañana***

**Introducción**

Para esta nueva entrega se agregó el requerimiento de generar los equipos. Hay dos asuntos principales a resolver para llevarlo a cabo:

* Evaluar a los jugadores según una variedad posible de criterios
* Repartir a los jugadores entre los dos equipos según los resultados de esa evaluación, también de distintas formas posibles.

Para realizar estas operaciones se le envían mensajes al partido, ya que los equipos se generan para un partido. El partido conoce a ciertos objetos que le ayudan a realizarlas. Incluimos un diagrama que muestra estos objetos y más adelante pasamos a explicarlos.



**Evaluación de jugadores**

El partido puede ordenar a sus jugadores según distintos criterios de evaluación. El partido tiene un atributo criterio que apunta a un objeto capaz de hacer el ordenamiento. Por dentro, ese objeto hace las operaciones necesarias para ordenar la lista de jugadores del partido. En realidad no es una delegación “clásica” en la que A llama a B, sino que combinamos la idea de strategy con las herramientas que nos da Java. El partido no le hace llamadas al criterio, pero lo que sí hace es pasar al criterio como argumento de Collections.sort() que es un artilugio de Java que nos permite ordenar una colección de cualquier tipo de objetos según un criterio. Lo que resuelve el requerimiento es que la clase genérica criterio implementa el comparador que pide Java, pero al hacerlo se pide a sí misma evaluar al jugador. Esta evaluación es lo que se delega a cada subclase, siempre con la misma interfaz. Por lo tanto nuestros distintos criterios se comportan polimórficamente con los componentes de Java que los utilizan y que no sabemos cómo funcionan por dentro.

Por otro lado, cada criterio específico hace las operaciones que solicita el enunciado, preguntándole al jugador lo que necesita saber de el (recibe al jugador por parámetro). Esto incluye revisar las calificaciones que tuvo el jugador. En el caso particular del puntaje promedio obtenido en el último partido, el crtirerio debe crearse con respecto de una comunidad: no es lo mismo un criterio de último partido para la comunidad X que para la comunidad Y ya que su último partido no fue el mismo. Sobre el criterio que pide promediar los resultados de otros criterios aplicamos de forma directa el patrón composite, permitiéndonos anidar criterios de forma transparente.

**Generación de los equipos**

Con esto nos referimos a repartir a los jugadores en los distintos equipos. Creamos una clase que representa la generación de los equipos, al que le podemos pedir simplemente generarEquipos(). Ésta agrupa algo de comportamiento común (validaciones y comunicación con el partido) y delega en sus subclases, por medio del mensaje designarJugadores(), la repartija definitiva. Al implementar este método en las subclases que cumplen lo que menciona el enunciado, es verdad que utilizamos índices sobre una lista. Sin embargo, el propio requerimiento habla de que “los jugadores pares van a un lado y los impares al otro”, o de que “los jugadores en la enésima posición van para tal lado”, por lo que usar índices sobre una lista ordenada (que viene ordenada tras haber aplicado el criterio) nos pareció lo más natural y lógico.

Además de esto, la consigna pide aplicar algún patrón para distinguir los momentos en que un administrador pide armar los equipos y cuando finalmente los confirma. Lo más sencillo nos pareció aplicar un patrón state muy simple. Distinguimos que un partido donde todavía se pueden hacer cosas y un partido donde están confirmados los equipos se comportan de formas distintas, aunque la diferencia no es muy compleja. Simplemente existen operaciones que no se pueden hacer en un partido con equipos confirmados (inscribir jugadores, bajar jugadores y generar equipos), por lo que agregamos una validación preguntándole al objeto estado del partido. Si quiero hacer una operación que no está permitida para un partido con equipos confirmados, se alza una excepción.

La generación de equipos funciona de la siguiente manera: al haber 10 jugadores inscriptos el administrador puede pedir generar equipos. Si no le gusta el resultado, puede variar como quiera el criterio de evaluación y el algoritmo de reparto de los jugadores. Cuando vuelva a pedir generar equipos, los equipos que había dejan de existir (en realidad los dejamos de referenciar y se los llevará el garbage collector) y se generan los nuevos. Una vez conforme, el admin puede solicitar confirmarEquipos() lo que cambia el estado del partido.