ENTREGA 4 - DOCUMENTACIÓN Y DECISIONES DE DISEÑO

En esta iteración surgieron nuevas abstracciones.
Nos vimos en la necesidad de cosificar el Criterio, que es lo que nos permite darle un “peso” a cada jugador. Estos criterios, que surgieron en esta iteración, utilizan un procedimiento diferente para otorgarle una calificación a cada jugador, y con ello luego poder armar los equipos. Cada una de estas nuevas clases cumplen con un contrato común, la interfaz Criterio.

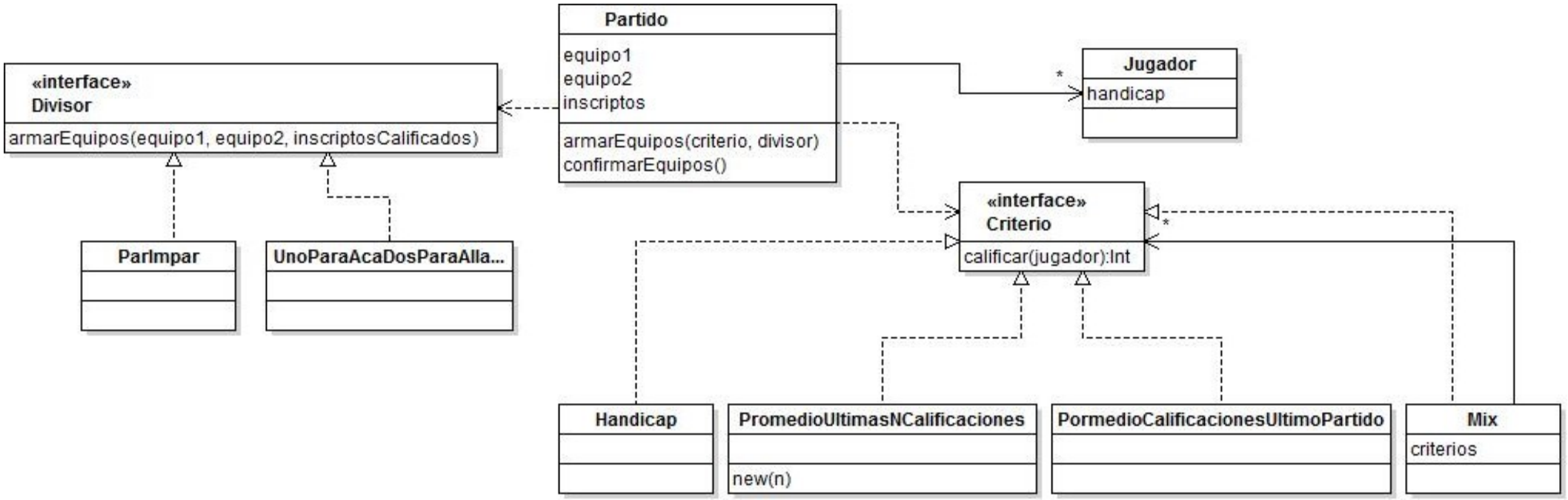
Por otra parte, también notamos que existen diferentes formas de armar los equipos. Estas diferentes formas también cumplen con un contrato entre sí, por lo que agregamos la interfaz Divisor que ambas tienen que implementar.

Estas decisiones de diseño (la de crear estas dos interfaces con sus implementaciones) se sustentaron principalmente en:
- Testeabilidad: como son módulos independientes, que no dependen de otras clases más “pesadas” (como Partido), son muy fáciles de testear de forma unitaria (ver tests realizados). Es sencillo checkear las entradas con las salidas
- Flexibilidad (extensibilidad): si se quieren agregar nuevas formas de dividir los equipos (interfaz Divisor) o nuevos criterios (interfaz Criterio), simplemente hay que implementar la interfaz que se desee. No implica ningún tipo de problema agregar nuevas implementaciones, ya que no hay que realizar ningún cambio adicional en otras clases.
- Desacoplamiento y cohesión: con las decisiones que tomamos, delegamos responsabilidad en estas nuevas entidades, responsabilidad que de otra forma hubiera caído en la clase Partido, logrando que dicha clase no sea cohesiva y se convierta en un God Object.
- Robustez: como venimos haciendo desde entregas anteriores, tratamos de que el sistema falle rápido (Fail-Fast) ante situaciones que podrían generar datos inconsistentes o un comportamiento errático. Para ello utilizamos Excepciones como mecanismo para manejar los errores que surjan.

En nuestra solución, los momentos de armado de equipos (Partido#armarEquipos(...)) y de la confirmación de los mismos (Partido#confirmarEquipos()) estan claramente diferenciadas.

Para el armado de equipos, se invoca el método armarEquipos() de la clase Partido pasándole por parámetros el criterio que se quiere utilizar, y la forma en que se quieren dividir los equipos. Este método utiliza el patrón Call & Return con el criterio, ya que le pasa un jugador y espera que éste devuelva una calificación que se corresponde con su peso.
Cuando se tienen los pesos de todos los jugadores, se envía el mensaje armarEquipos(...) a la implementación de Divisor que se recibió por parámetro, y se le pasan dos listas correspondientes a cada equipo (son atributos del partido), y la lista de jugadores calificados armada anteriormente. Aca podemos apreciar el patrón Shared Memory, ya que Divisor va a modificar los equipos recibidos, y dejar en ellos los cambios. Una vez terminada esta invocación, los equipos quedaron armados en las 2 listas que tiene la instancia de partido como atributo.

Para la confirmación de los equipos se invoca al método Partido#confirmarEquipos() (método no implementado ya que no formaba parte de la iteración). Este método puede invocarse en cualquier momento, y no es necesario que se llame inmediatamente despúes de armarse los equipos.
Los momentos del armado del equipo y de la confirmación del mismo están claramente diferenciados.


«interface»
Divisor
armarEquipos(equipo1, equipo2, inscriptosCalificados)
Partido
equipo1
equipo2
inscriptos
armarEquipos(criterio, divisor)
confirmarEquipos()
Jugador
handicap
*
«interface»
Criterio
calificar(jugador):Int
*
ParImpar
UnoParaAcaDosParaAlla...
Handicap
PromedioUltimasNCalificaciones
new(n)
PormedioCalificacionesUltimoPartido
Mix
criterios