Universitat Pompeu Fabra Barcelona

Laboratorio 2

El objetivo del laboratorio es:

Implementar parte de la aplicación de fútbol diseñada en el seminario 3 utilizando el concepto de herencia.

Entrega:

- Código del proyecto (Programas en Java)
- Documentación de la implementación.

Universitat Pompeu Fabra Barcelona

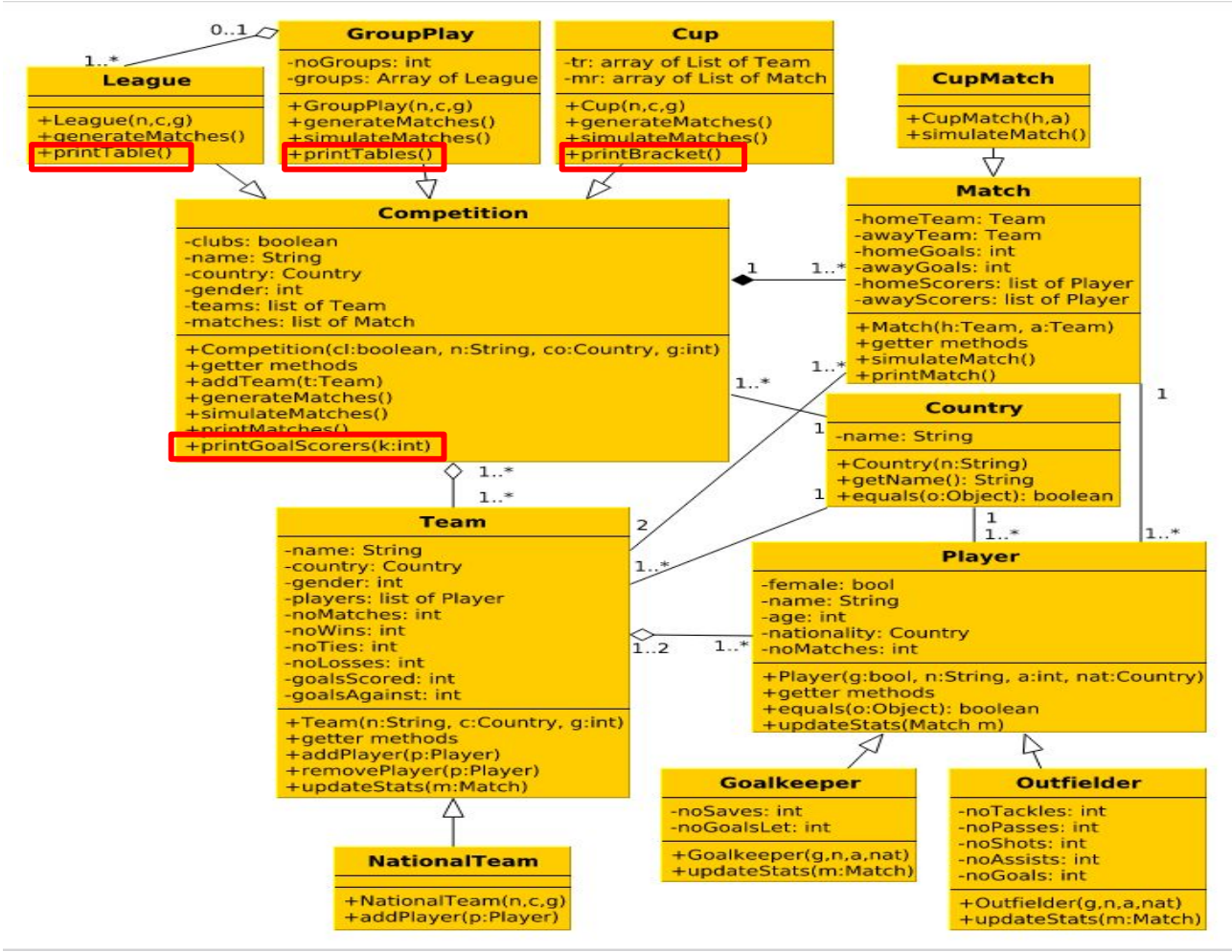
Documentación

1. Una **introducción** donde se describe el problema.
   a. ¿Qué hace el programa?
   b. ¿Qué clases se definieron?
   c. ¿Cuáles métodos se implementaron?
2. Una **descripción** de **soluciones alternativas** que se discutieron, y una descripción de la **solución elegida**, así como el motivo de la elección de esta solución en lugar de otras. También es una buena idea mencionar los **aspectos teóricos** de los conceptos de programación orientada a objetos que se aplicaron como parte de la solución.
3. Una **conclusión** que describa el **funcionamiento** de la práctica, es decir, ¿las pruebas mostraron que las clases se implementaron correctamente? Se puede mencionar cualquier **dificultad** durante la implementación, así como la solución.

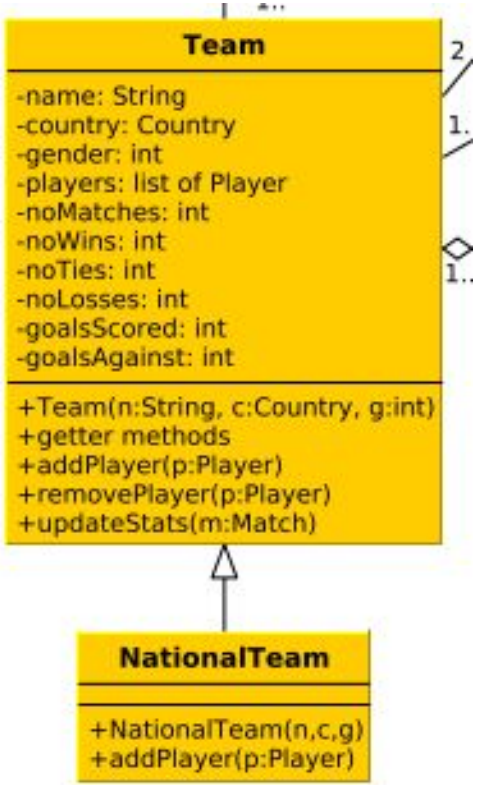
El código fuente y la documentación deben cargarse en un directorio de nombre **Lab3** de su repositorio Git ***antes de la próxima sesión de laboratorio***. ***Importante, el nombre del directorio deberá ser exactamente Lab3.***

Universitat Pompeu Fabra Barcelona

Diagrama propuesto



Clase NationalTeam

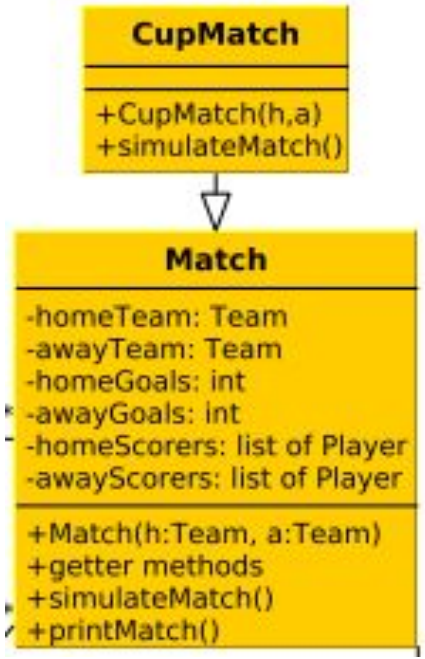



```
public class NationalTeam extends Team {
```

- La clase NationalTeam representa los equipos nacionales ampliando la clase de equipo. La única diferencia es que al intentar agregar un jugador al equipo, el código debe comprobar que el jugador tiene la nacionalidad correcta. Por eso es necesario sobreescribir el método **addPlayer** de equipo.
- Para comprobar si dos países son iguales, es una buena idea sobreescribir el método **equals** que el País hereda del Objeto, similar al método **equal** visto en la teoría. Dos países son iguales si tienen el mismo nombre.

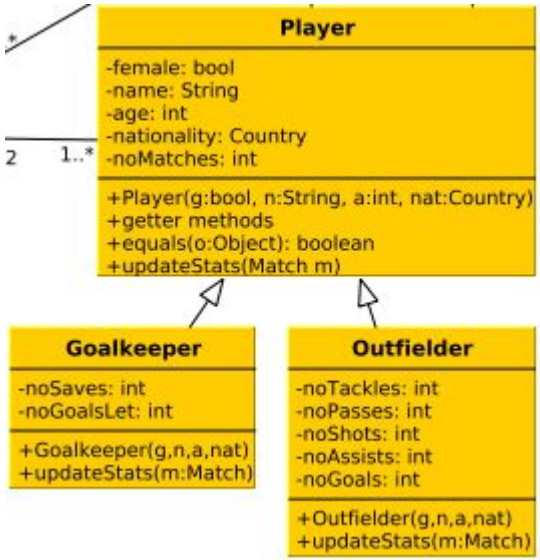
```
public void addPlayer(Player p) {
    if (country.equals(p.getNationality()))
        super.addPlayer(p);
}
```

Clase CupMatch



- La clase CupMatch representa partidos de copa ampliando la clase Match La única diferencia es que al simular un partido, no se puede terminar en empate. Por lo tanto, en caso de empate, la simulación debe continuar (por ejemplo, tiempo extra, penales) hasta que haya un ganador. Para ello se sobreescribe el método **simulateMatch** de Match. Asegurarse de generar goleadores también por los goles extra que se marquen.

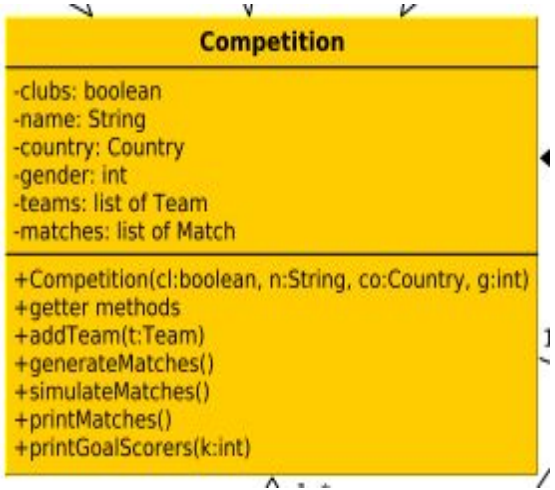
Clase Goalkeeper and Outfielder



- Las estadísticas que fueron creadas en la clase de Jugador debe trasladarse a la clase de outfielder, ya que sólo este tipo de jugador tiene las estadísticas asociadas.
- Los porteros deberían en lugar de eso, definir sus propias estadísticas.

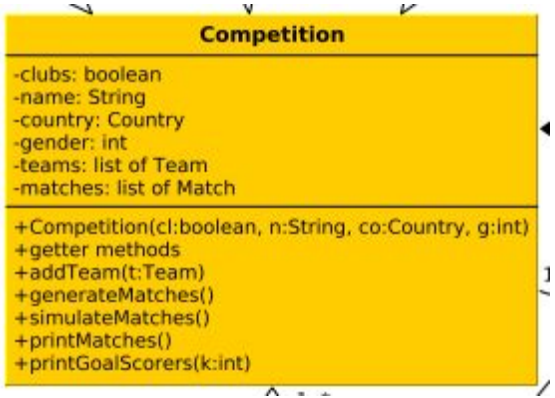
Universitat Pompeu Fabra Barcelona

Clase Competition



- La clase Comptetition se puede definir reutilizando los atributos y métodos de la clase Liga.
- Sólo algunos métodos (como generateMatches y printTable) deberan permanecer en League (que ahora hereda de Competition).
- Por ahora, el método generateMatches se puede dejar vacío en Competición, ya que la idea es siempre utilizar una de las subclases.

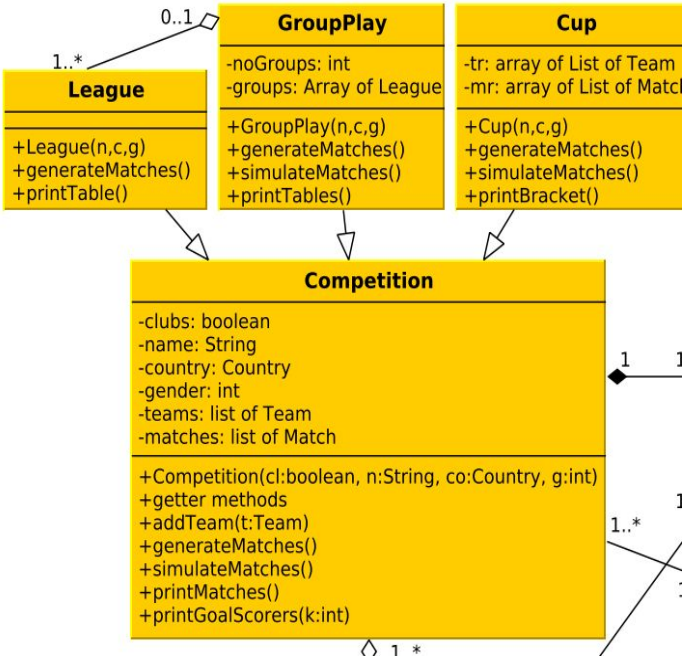
Clase Competition



- La clase Competition necesita una forma de distinguir entre competiciones de clubes y competiciones internacionales (que se realiza en el diseño utilizando el atributo booleano clubs).
- Tener en cuenta que el método **addTeam** debe tener en cuenta el tipo de competición, es decir, solo permitir equipos de clubes en competiciones de clubes o selecciones nacionales en competiciones internacionales.

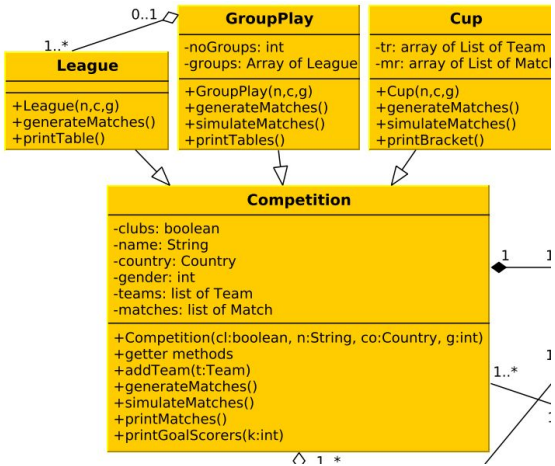
Universitat Pompeu Fabra Barcelona

Clase League



- La clase Liga ahora hereda de Competición, y contiene el método generateMatches que específicamente genera los partidos de una liga.

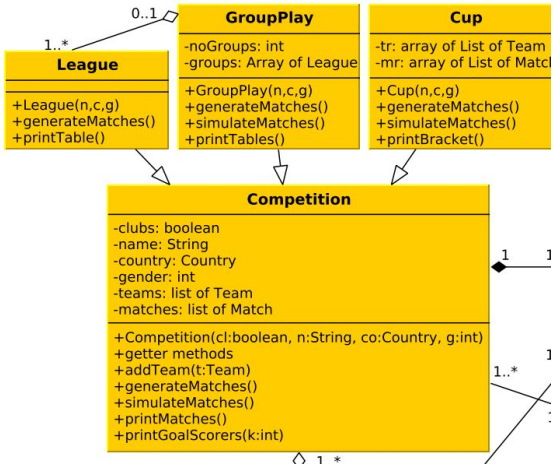
Clase Cup



- La clase de Cup debería generar y simular partidos para múltiples rondas, hasta que solo quede un ganador. Para hacerlo, es una buena idea mantener matrices. **tr** almacena la lista de equipos de cada ronda, y **mr** almacena la lista de partidos de cada ronda.

- En la primera ronda, la lista de equipos incluye a todos los equipos participantes. Es una buena idea aleatorizar los equipos de la primera ronda (por ejemplo, usando el método **Collections.shuffle).** En rondas posteriores, la lista de equipos es aquellos que ganaron sus partidos en su ronda anterior (más cualquier equipo que fuera no emparejado con otro equipo, en el caso de un número impar de equipos).

Clase GroupPlay



- Finalmente, la clase GroupPlay se puede implementar manteniendo una matriz de Liga, de modo que cada grupo es una instancia de liga. De nuevo, es una buena idea aleatorizar los equipos que se asignan a cada liga. Los métodos de League se puede utilizar para generar y simular partidos.