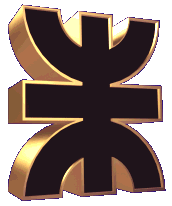
|  |
| --- |
| **Nombre y Apellido** |
| Cappellini Darío |
| Carella Germán |
| Centurion Damian |
| Fage Victor |
| Gonzalez Santiago |

|  |  |
| --- | --- |
| **Fecha De Presentación:** | 04/06/2014 |
| **Fecha de Devolución:** |  |
| **Calificación** |  |
| **Firma Profesor** |  |



**GRUPO N° 7**

TP Anual: 3ra Entrega

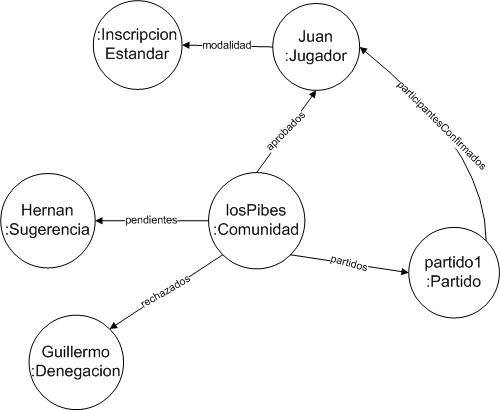
***Diseño de Sistemas 2014***

***Miércoles mañana***

**Sugerencia de nuevos jugadores:**

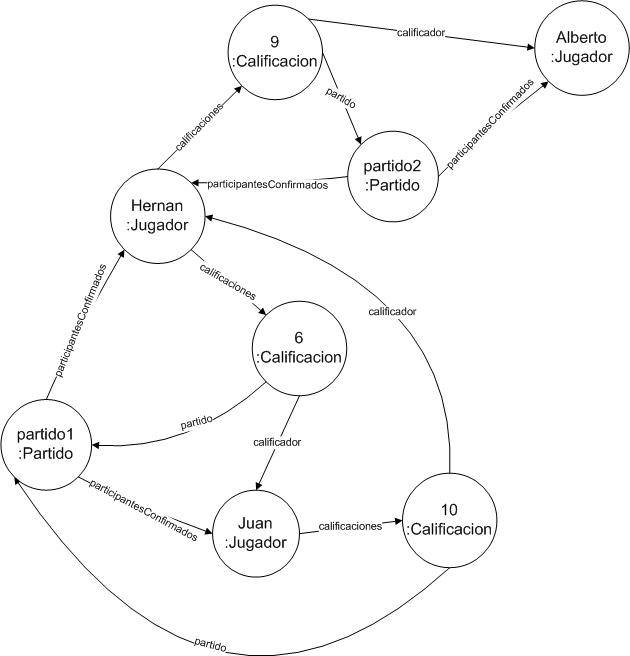
Para cumplir con este caso de uso se tuvo que modelar primero la “comunidad de amigos” que es manejada por algún admin, el que modera la entrada o no de jugadores a esa comunidad. Una comunidad es simplemente un conjunto de jugadores (miembros), que organiza ciertos partidos, con la lógica, por ahora, de aceptar sugerencias de jugadores y luego tratarlas.

A una comunidad le puedo sugerir un prospecto de jugador (una *sugerencia*) y queda guardado en una colección de sugerencias pendientes. Luego la comunidad es capaz de aprobarlo, operación mediante la cual la sugerencia se convierte en jugador y se agrega a los miembros (*aprobados*). Asimismo, también puede rechazarlo, agregando los datos, el motivo y la fecha a una colección de *denegaciones*, que son estructuras de datos que contienen la información que se quiere conservar de un jugador rechazado. Esto se vería más o menos así:



Como muestra el diagrama, los jugadores pueden inscribirse a los partidos normalmente. Un detalle no menor es que la comunidad no tiene la lógica necesaria para convertir una sugerencia en jugador o en denegación, sino que al recibir la orden de aprobar o rechazar, delega este paso en sugerencia, que hace las operaciones necesarias y devuelve el objeto esperado. Luego la comunidad agrega este objeto a la colección correspondiente.

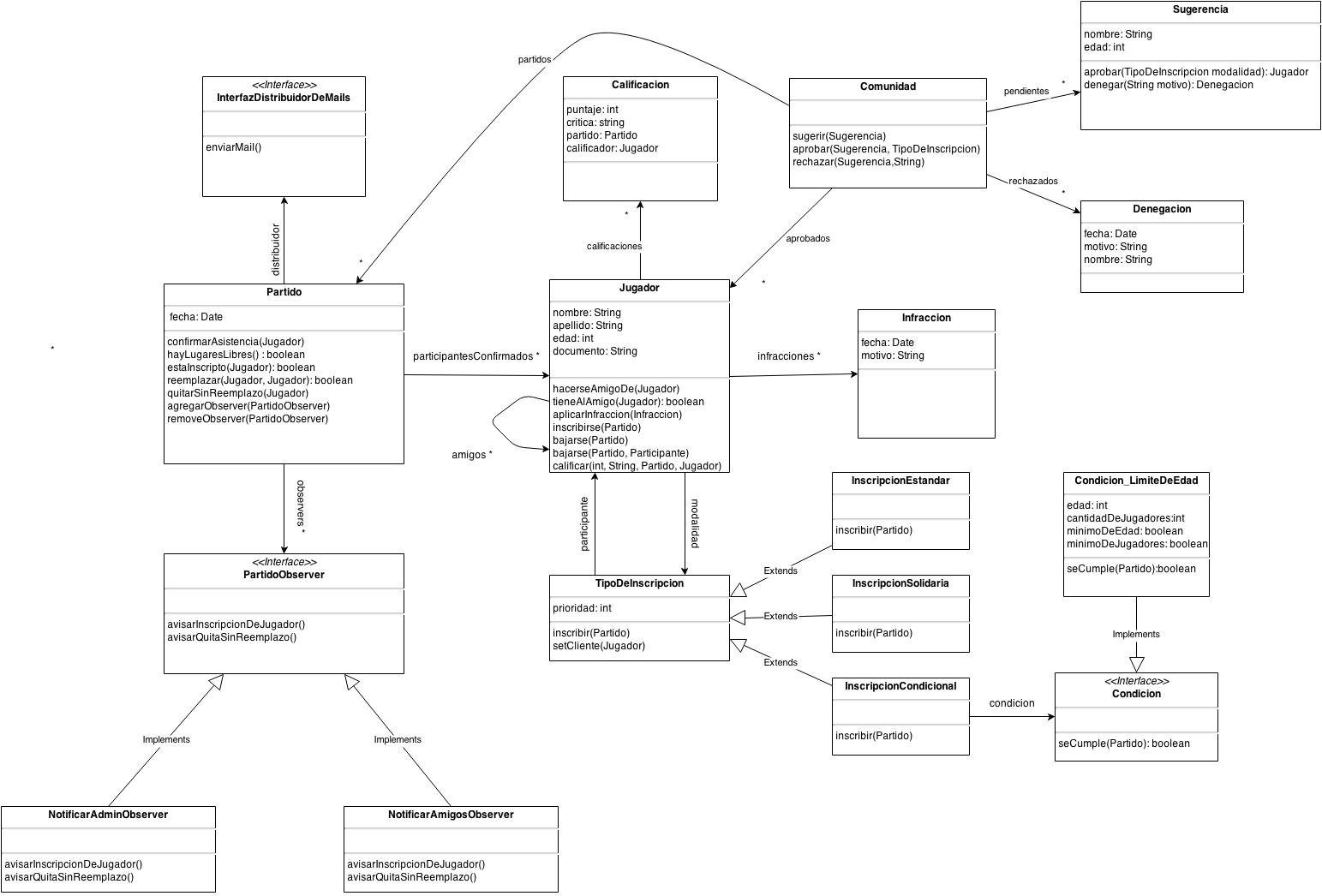
**Calificaciones:**

Para calificar un jugador, simplemente se invoca el método calificar, que recibe el puntaje, la reseña en forma de string, el partido para el cual se lo está calificando, y quién lo está calificando. En base a estos datos se hacen algunas verificaciones (como por ejemplo que un jugador no se califique a sí mismo - ver tests para más detalles) y si todo está OK se agrega la *calificación* a una colección de calificaciones que tiene cada jugador sobre sí. Éstas son objetos por el solo hecho de que Java no ofrece structs, pero funcionan como estructuras de datos. En un diagrama de objetos se ve así:

En este ejemplo Juan calificó a Hernán con un 6. El mensaje que se utilizó para lograr esto fue:

hernan.calificar(6, "bien en la marca pero le falta manejo del balón", partido1, juan).

**Vista estática:**



**Explicaciones adicionales:**

Identificamos dos conceptos muy relacionados entre sí que permitieron bajar el acoplamiento en esta parte del diseño: delegación y encapsulamiento.

La comunidad delega en la sugerencia la responsabilidad de aprobarse para convertirse en un jugador, o de denegarse para convertirse en una denegación. Esto baja el acoplamiento entre la comunidad y la sugerencia: aunque obviamente deben estar acoplados porque son componentes relacionados del sistema, la comunidad puede conseguir lo que necesita de la sugerencia por medio de mensajes sencillos. Dicho de otra forma, la sugerencia encapsula la lógica de crear un jugador a partir de una sugerencia, y por lo tanto la comunidad no necesita saber cómo es la sugerencia por dentro ni cómo es el jugador.

Si no delegaramos este paso la comunidad tendría que conocer cómo es el estado interno de la sugerencia y qué operaciones tiene que hacer con éste para, por ejemplo, convertirla en un jugador al aprobarse. Si hubiera un cambio en la estructura interna de la sugerencia, no alcanzaría con modificar la sugerencia sino que el cambio se propagaría a la comunidad, haciendo que el sistema sea un poco menos mantenible. Otra consecuencia de este acoplamiento adicional es que el sistema queda menos flexible: si el día de mañana nos enteramos de que hay varias formas de sugerir y cada una se convierte en jugador de forma distinta nos basta con poner nuestra clase Sugerencia atrás de una interfaz y agregar más implementaciones. Si esta responsabilidad estuviera toda en la comunidad, esto no sería posible.

Como esta alternativa presenta los inconvenientes mencionados, y no vemos que nos presente ninguna ventaja, creemos que es un nivel de acoplamiento poco conveniente para el caso.

**Notas adicionales:**

-Entre la última entrega y esta, debido a los nuevos requerimientos, se realizó un refactor importante: dejó de existir la clase participante. Lo que significa esto es que el tipo de inscripción del jugador (lo que llamamos *modalidad*) ya no es propio de un jugador en un cierto partido, sino que es algo propio de cada jugador en general. Por lo tanto toda la responsabilidad del participante pasó al jugador, y se cambiaron algunos constructores en consecuencia. A pesar de eso, la inscripción se sigue delegando como un strategy stateful. El jugador recibe el mensaje jugador.inscribirse(partido) y ahí delega en la modalidad, que realiza las operaciones necesarias, comunicándose con el partido y con el jugador, para averiguar si el jugador puede inscribirse: si puede lo inscribe y si hay algún problema levanta una excepción.

-Los casos de prueba de esta entrega se encuentran directamente en el código ya que son lo suficientemente autodescriptivos. Los separamos en dos archivos, uno para probar el mecanismo de calificaciones y uno para el de las sugerencias de nuevos jugadores.