# PRÁCTICA 1 : Uso de patrones de diseño en OO

Cristian Fernández Jiménez
Ángel Gómez Ferrer

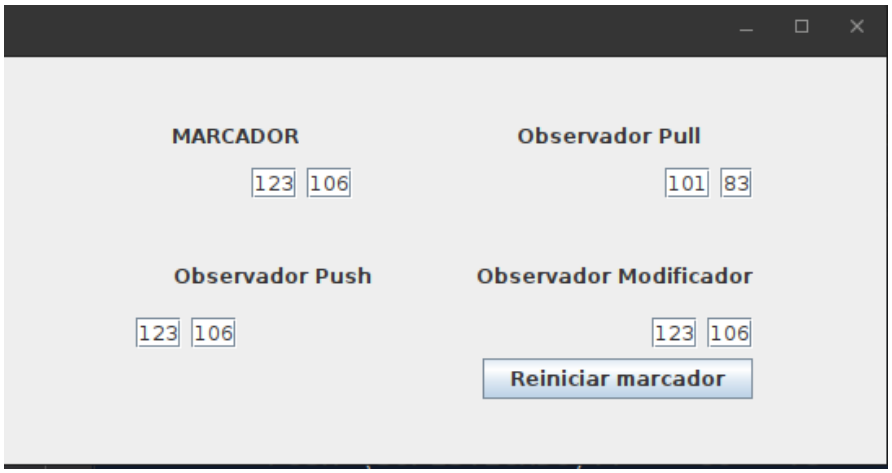
*El objeto de trabajo se trata de una aplicación para encuentros deportivos.*

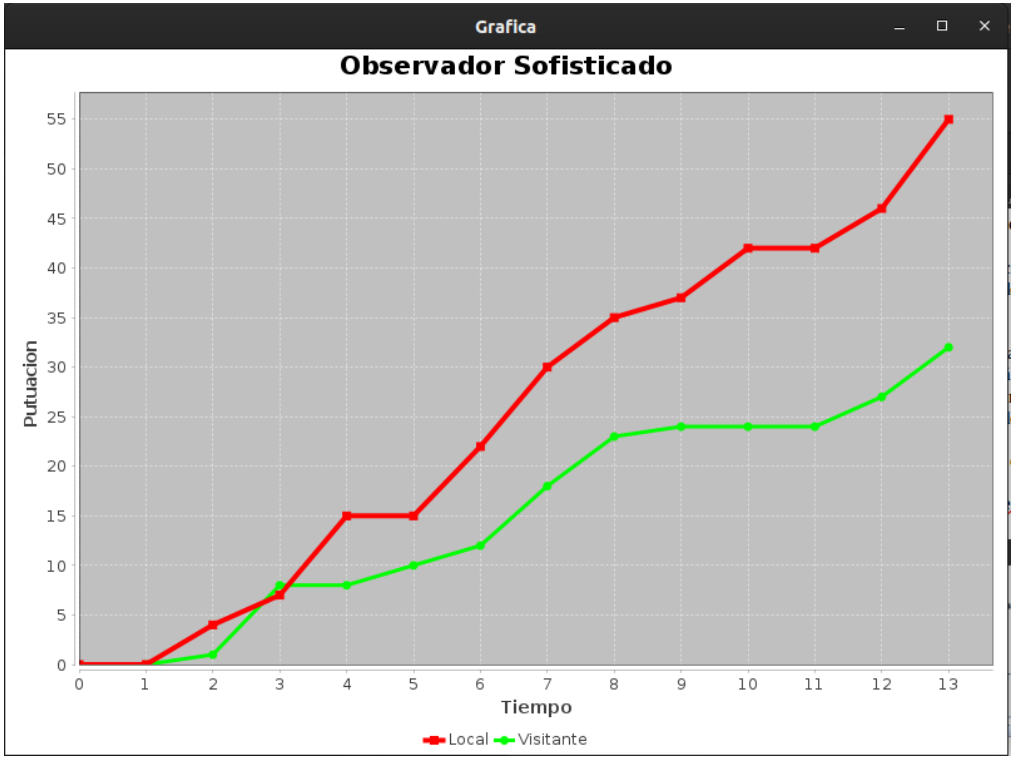
## Sesión 1: Patrón Observador

El problema que hemos escogido para poder aplicar los patrones de diseño se basa en 2 marcadores que se irán incrementando cada cierto tiempo, los datos serán generados en una hebra **Cliente** y obtenidos por el sujeto observable.
Hemos realizado 3 observadores, el primero de ellos es un observador suscrito mediante el método Push (**ObserverPistaPush**) el cuál será notificado por el Observable cuando se actualicen los datos.
El segundo es un observador no suscrito que ira comprobando cada cierto tiempo (usando hebras) si se han actualizado los datos que tiene el observable y en su caso los obtiene, mediante la estrategia Pull (**ObserverPista**).
El tercero es un observador modificador también suscrito al igual que el observador mediante Push, pero con la capacidad de modificar los datos del sujeto observable (**ObserverPistaModificador**).

Se ha creado un diseño Modelo Vista Controlador para relacionar los datos con la interfaz gráfica, en este caso una ventana que contendrá las diferentes observaciones de cada uno de los observadores, además del Marcador principal (**ObservablePista**) y con la posibilidad de reiniciar los marcadores haciendo uso del observador modificador.

Para hacer uso de la GUI y comunicar los observadores con la misma, hemos hecho uso de una clase Singleton (**Controlador**), el cual comunica al cliente con la vista problema. Se creará por tanto una instancia de nuestra clase singleton en el cliente y obtendremos los marcadores mediante su instancia desde la vista.

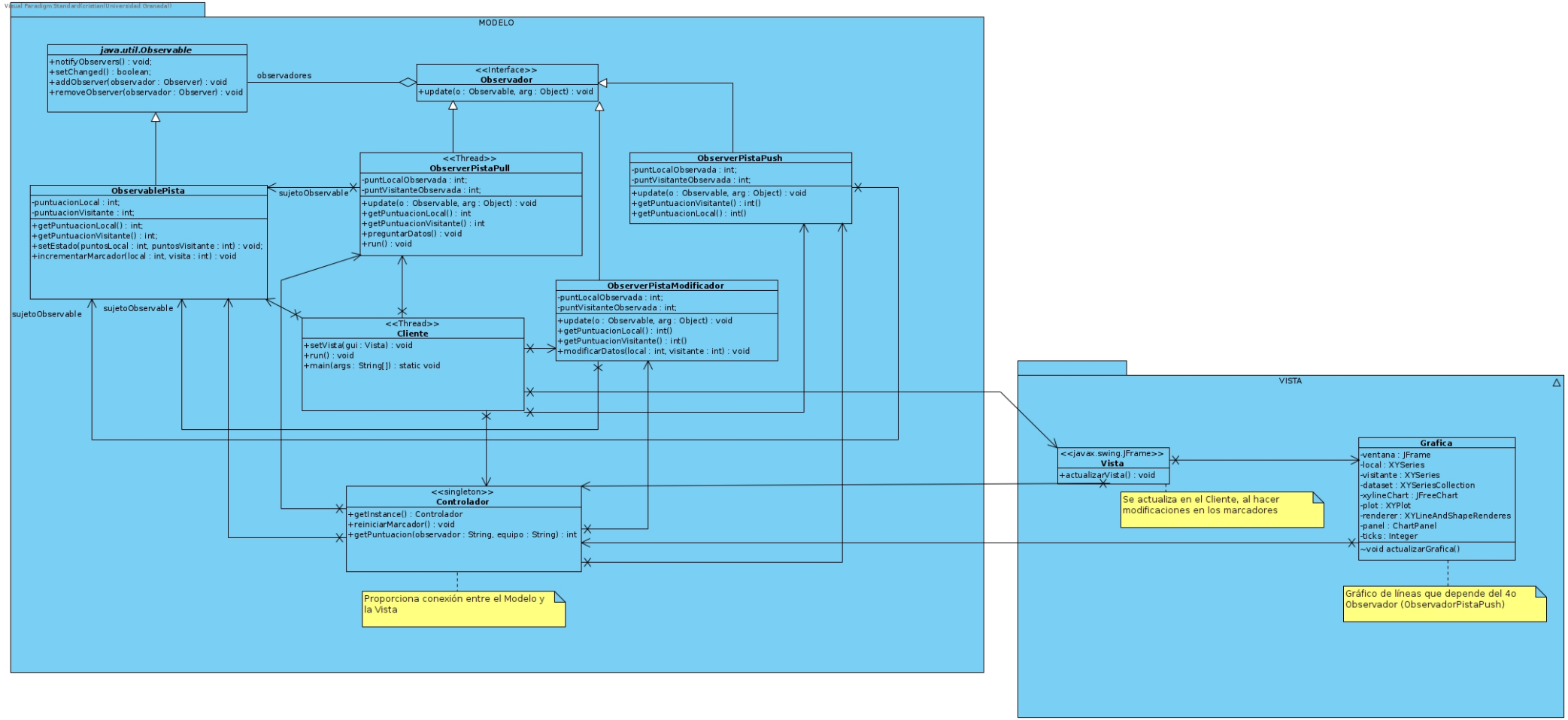



Además, se ha creado otra instancia del **ObservadorPistaPush** que será monitorizada mediante una gráfica de líneas, implementada en la clase **Gráfica** en la zona de la Vista.

Esto se ha realizado mediante la librería JFREECHART, creando una nueva ventana cuyo panel será el gráfico en sí.

En resumen, el diagrama de clases del proyecto es el siguiente:



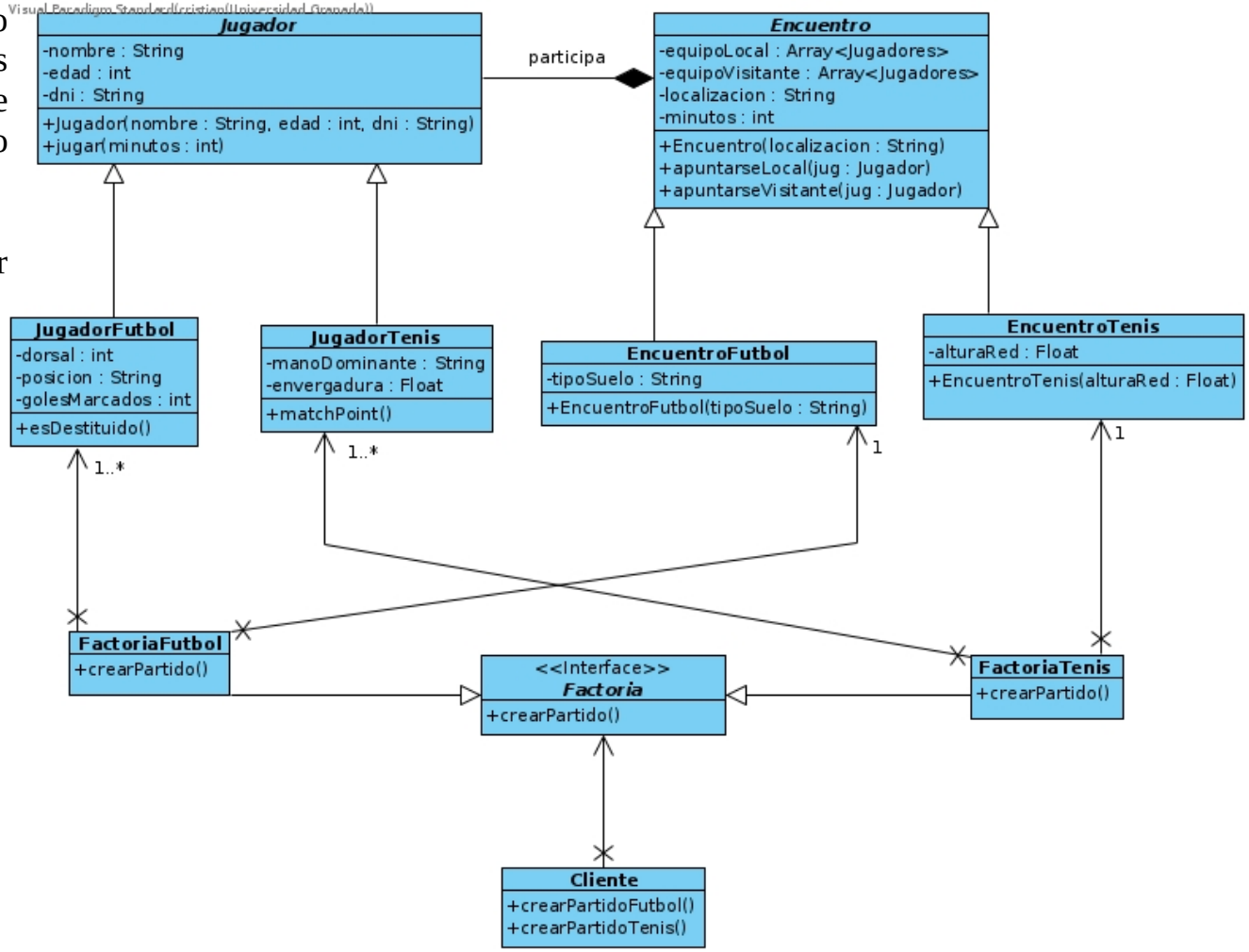
## Sesión 2: Patrones *Factoría Abstracta*, *Método Factoría* y *Prototipo*

En esta sesión hemos decidido implementar el patrón factoría involucrando los **jugadores y los encuentros**. Crearemos **dos factorías, para Fútbol y Tenis**. Cada una de ellas se encargará de crear un Encuentro del deporte correspondiente, y añadir jugadores específicos a cada uno de ellos. Los Jugadores y los Encuentros trabajarán como hebras, paralelamente.

### Patrón *Factoría Abstracta* y *Método Factoría*

En el primer caso, aplicamos el método factoría, haciendo que las factorías específicas creen distintas instancias de jugadores y encuentros y devolviendo el encuentro ya creado.

La clase cliente se encargará de llamar a las factorías y recibir lo pedido.

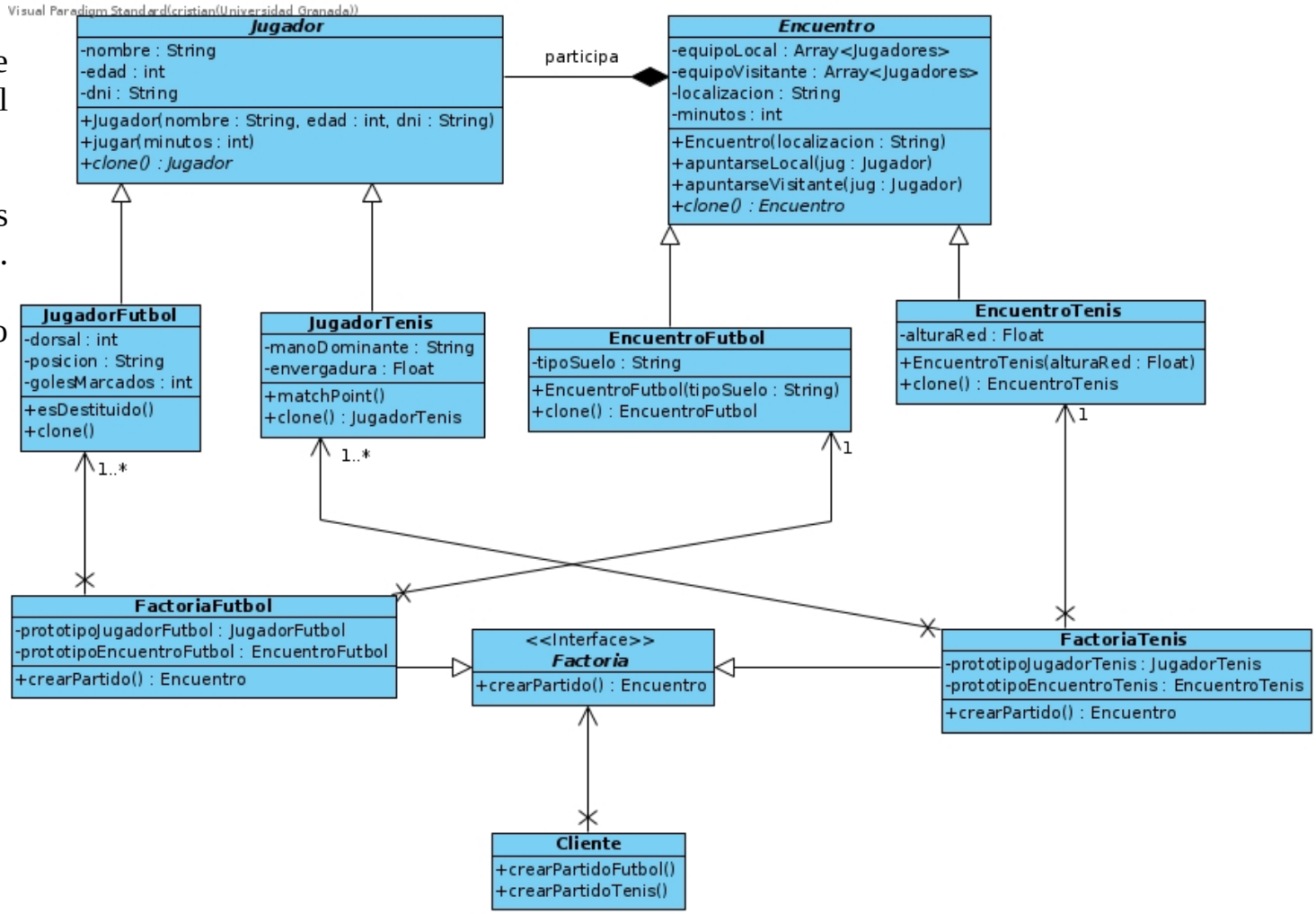


## Patrón *Factoría Abstracta* y *Prototipo*

En el segundo caso se nos pide realizar lo mismo pero con el patrón Prototipo.

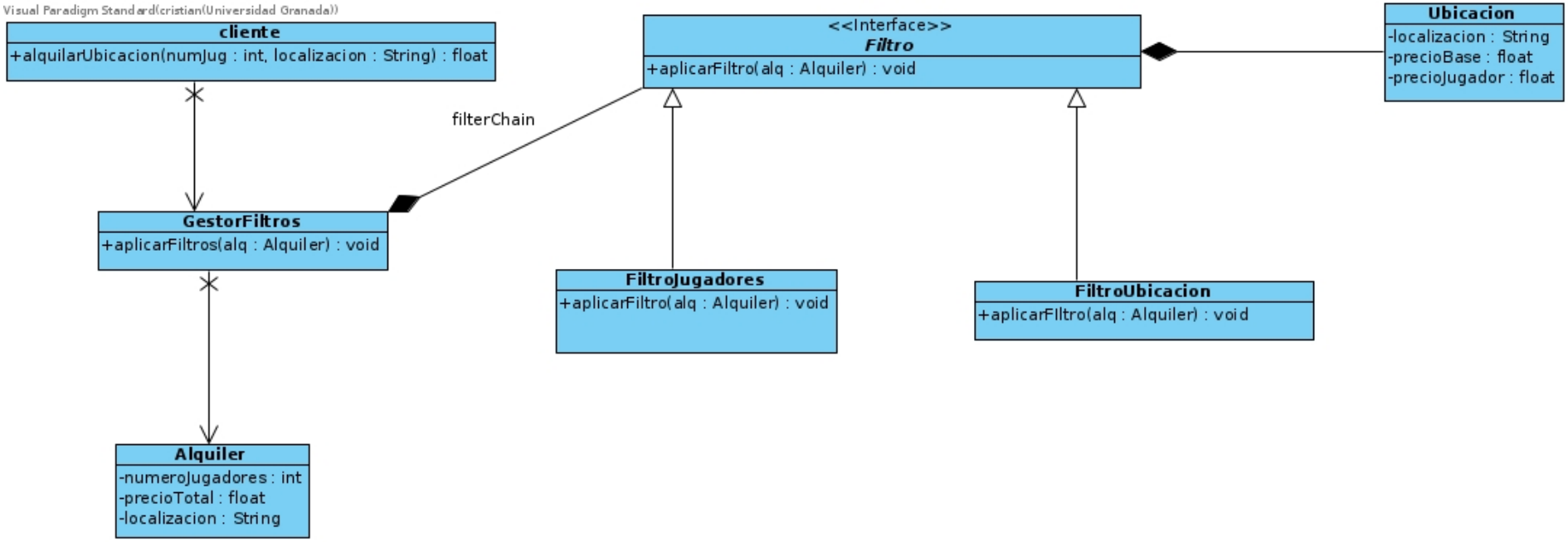
Haremos entonces a los jugadores y encuentros clonables.

Al ser implementado en Ruby, no hemos hecho uso de Hebras.



## Sesión 3: Estilos arquitectónicos: el estilo *Filtros de Intercepción*

En esta sesión hemos decidido implementar el estilo Filtros de Intercepción involucrando el alquiler de pistas deportivas, según los **jugadores y las ubicaciones**.




El objetivo a conseguir será un objeto Alquiler, cuyo coste sea definido según los filtros. Sin estos, el objetivo sería funcionando, puesto que se parte de un coste por el uso de la aplicación de 1.
Crearemos **dos filtros, para Jugadores y Ubicaciones**. Partimos de una interfaz Filtro, con un método aplicar filtro que será implementado por dos filtros concretos. Además, se relacionará con las ubicaciones.
El primer filtro consultará una ubicación según dónde se realice el encuentro y obtendrá el precio de alquiler de esa ubicación, añadiéndolo al Alquiler objetivo.

El segundo, observará el número de jugadores participantes en el alquiler y añadirá un bonus a pagar al Alquiler objetivo.

Así, tendremos finalmente un objeto Alquiler con el coste correcto y completo, con el que realizaremos estadísticos para nuestra aplicación.

```
4 jugadores alquilan el Polideportivo Maracena: 5.5€

22 jugadores alquilan el Campo de Futbol de Granada: 87€

4 jugadores alquilan Pista de Tenis Atarfe: 1€

4 jugadores alquilan la Pista de Tenis Albolote: 12€

14 jugadores alquilan la Pista de Futbol Illora: 34€

2 jugadores alquilan la Pista de Tenis de Cogollos: 7€

En el día de hoy se han realizado 6 con un coste medio de 24.4167
Cada jugador participante ha pagado de media 2.93 por disfrutar de un encuentro deportivo
```

“Localización”, precio base, precio jugador

```
Ubicacion maracena("Polideportivo Maracena", 4.5, 0);
Ubicacion granada("Campo de Futbol Granada", 20, 3);
Ubicacion atarfe("Pista Tenis Atarfe",0,0);
Ubicacion albolote("Pista de Tenis Albolote", 7, 1);
Ubicacion illora("Pista de Futbol Illora", 5, 2);
Ubicacion cogollos("Pista de Tenis de Cogollos", 6, 0);
```

Como parte opcional se ha optado por implementar el programa en Dart mediante Android Studio.
(Documentos adjuntados en la práctica)