Universidad de Barcelona

Arthur Font / 20222613

Cristian Rodríguez / 20222381

Victor Llinares / 20222160

Proyecto de Prácticas

Disseny de Software

Práctica 3 – 31/12/2019

Barcelona

2019

Index

1. Introducción

En esta praćtica trabajaremos con la aplicación de patrones de diseño. en el desarrollo de nuestra aplicación UBFlix basada en interfaz gráfica GUI. Todo siguiendo el patrón Model-Vista-Controlador, aplicando los principios de diseño GRASP y patrones de diseño.

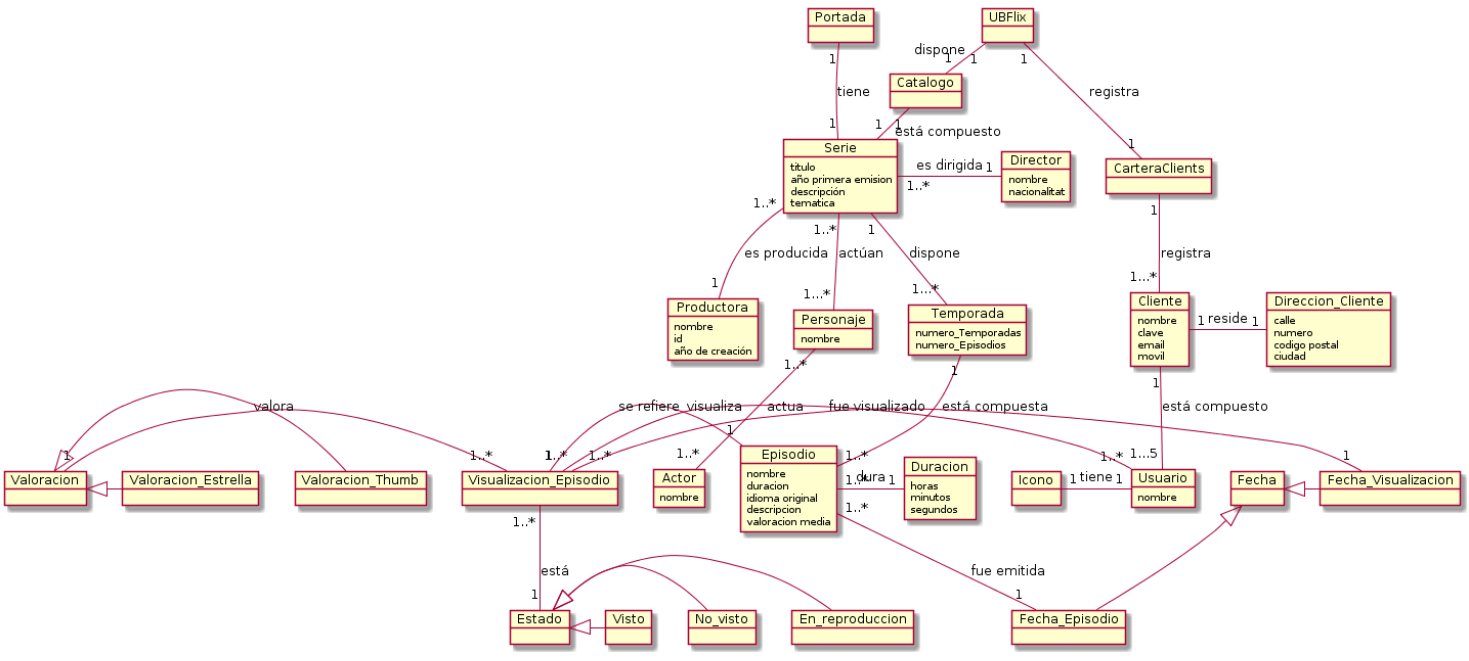
Recordamos que los principales patrones a utilizar son los siguientes:

- Observer
- Strategy
- Composite
- Singleton
- Factory

2. Modelo de Dominio

El modelo de Dominio en formato PlantUML está guardado en el proyecto subido al gitHub.

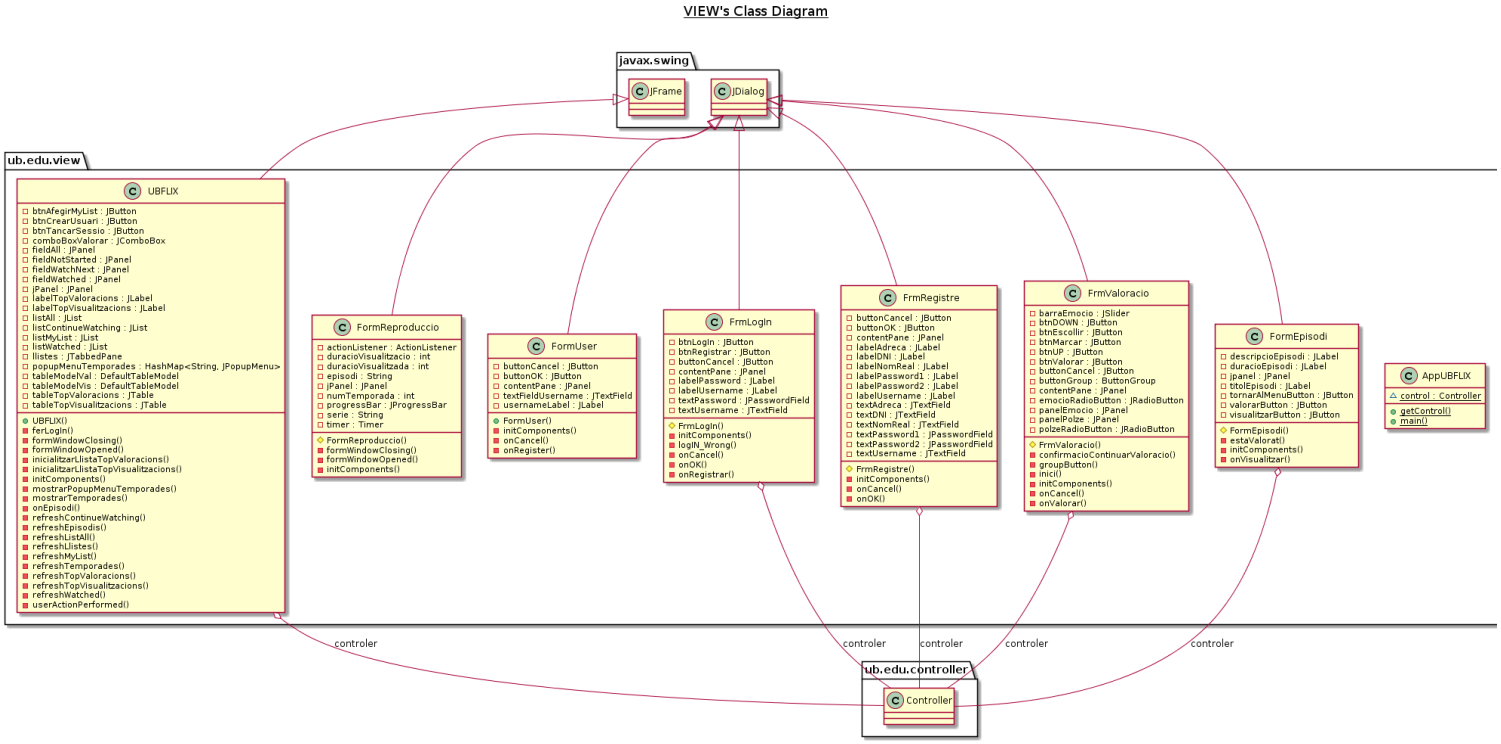
Sigue abajo fotos del modelo de Dominio clases del paquete Model de la aplicación.

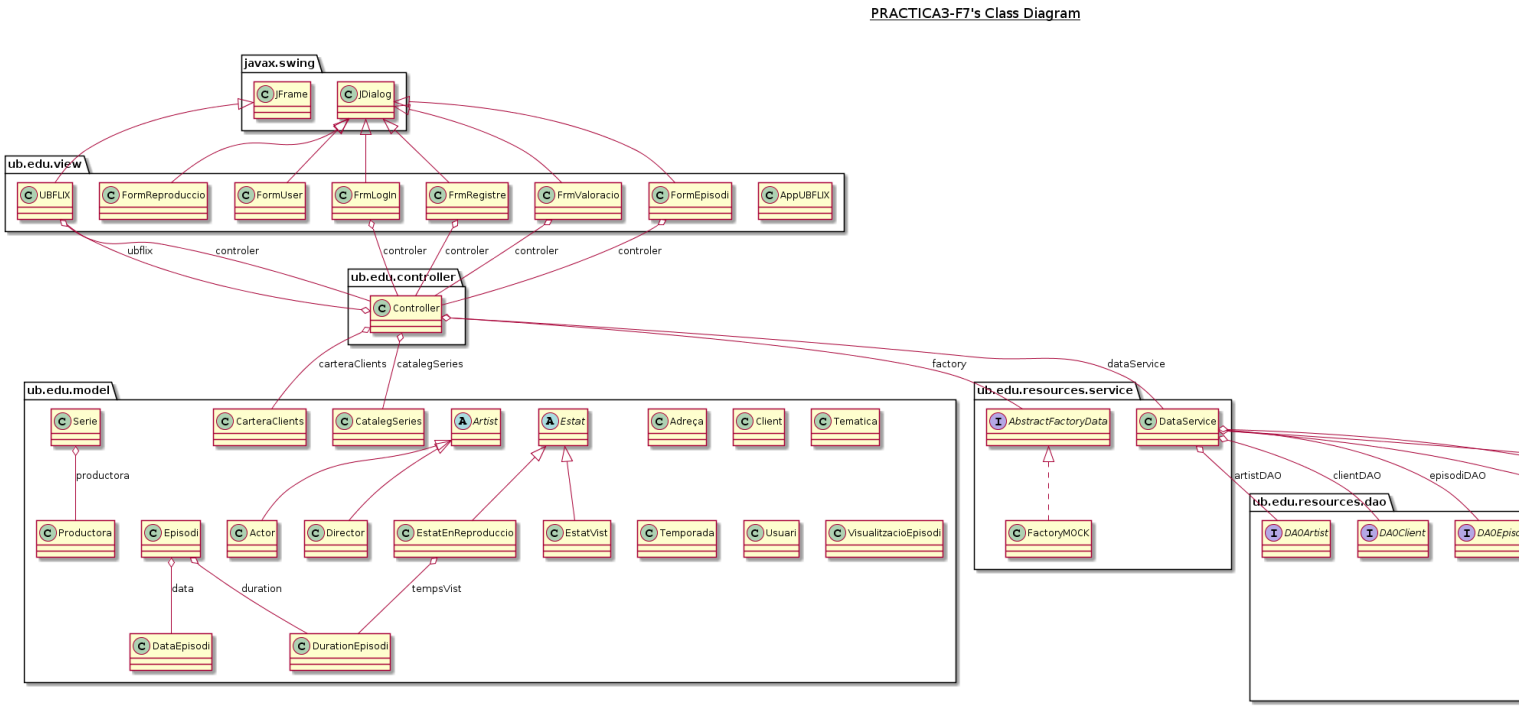


3. Modelo de Classes

El modelo de clases en formato PlantUML está guardado en el proyecto subido al gitHub.

Sigue abajo fotos del modelo de clases del paquete View y de todas las clase de la aplicación.





```
public class Controller {
        ...
        public String createUser(String clientname, String username){
        return carteraClients.createUser(clientname, username);}

}

public class CarteraClients {
        ...
        public String createUser(String clientname, String username) {
                Client client = this.find(clientname);
                if (client!=null) return client.addUser(username);
                else return "Client Unknown";
        }
}

public String addUser(String username) {
        ...
        if (nombreUsuaris < MAX_USUARIS) {
        if (!exist(username)) {
                Usuari nouUsuari = new Usuari(username);
                llista.add(nouUsuari);
                nombreUsuaris += 1;
                return "User Created";
        } else return "The user already exist";
        } else return "Users limit reached";
}
```

2) **Singleton** en la clase Controlador

El patrón Singleton se usa para asegurar que una clase sea instanciada una sola vez.

Sigue abajo un trecho de código que ejemplifica el patrón utilizado:

```
public class AppUBFLIX {

   public static void main(String[] args) {
      //TODO Cal crear la GUI a Controller, i aqui tenir instanciat el
Controller
      Controller control = Controller.getInstance(); // Singleton
   }
}
```

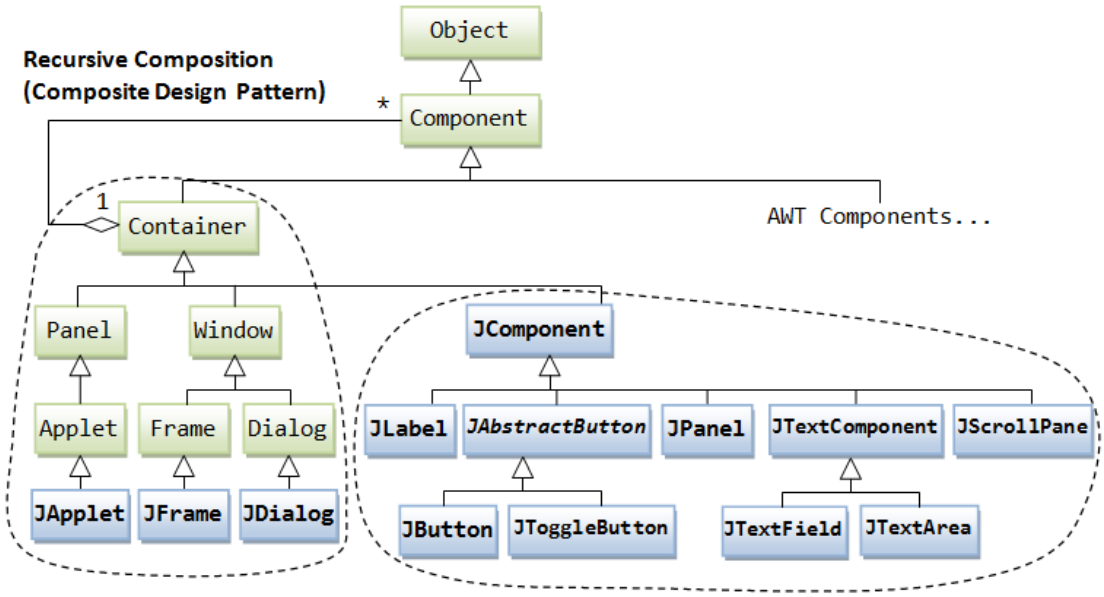
```
public class Controller {

   private AbstractFactoryData factory;      // Origen de les dades
   private DataService dataService;         // Connexio amb les dades
   private CarteraClients carteraClients;   // Model
   private CatalegSeries catalegSeries;     // Model
   private UBFLIX ubflix;                   // -- GUI (JFrame)
   private volatile static Controller uniqueInstance; // Singleton
   public Controller() { }
   public static Controller getInstance() {
      if (uniqueInstance == null) {
         synchronized (Controller.class) {
            if (uniqueInstance == null) {
               uniqueInstance = new Controller();
               uniqueInstance.factory = new FactoryMOCK();
               uniqueInstance.dataService = new
                                 DataService(uniqueInstance.factory);
               uniqueInstance.iniCarteraClients();
               uniqueInstance.initCatalegSeries();
               uniqueInstance.ubflix = new UBFLIX();
            }
         }
      }
      return uniqueInstance;
}
```

Así tenemos el Controlador inicializado apenas una vez, dado que necesitamos apenas un controlador para manejar nuestra aplicación.

3) **Composite** en las clases del paquete View

Las clases GUI de java siguen en patrón Composite. Una vez que los componentes de más alto nivel contienen otros componentes de más bajo nivel.



5. Repartición del trabajo

El trabajo ha sido delegado de la siguiente forma:

Arthur:

- Conexión Practica 2 → Practica 3
- Aplicación del patrón Expert en el Controlador (Pendiente Practica 2)
- Aplicación del patrón Singleton en el Controlador
- Log In y Registro
- Llengüeta del Catàleg de sèries
- Llengüeta del ContinueWatching
- Llengüeta del MyList
- Llengüeta del WatchedList
- Memória

Cristian:

- Top watched series list
- Top liked series list

6. Conclusiones

Debido a las fiestas, a los trabajos de las demás asignaturas, a los exámenes finales y mayoritariamente a nuestra falta de organización y dedicación al trabajo, no tuvimos el objetivo de aplicación de padrones totalmente alcanzado. Ya que nos costó hacer la integración de la interfaz gráfica y nuestro programa.

Por otra banda, pudimos donar funcionalidad al nuestro programa y trabajar un poco con la aplicación práctica de los padrones.

Realizar este trabajo nos dejó claro la importancia y las ventajas de un buen diseño de un programa. Una vez que buen diseño de software nos proporciona un un código limpio, robusto, mantenible y principalmente la capacidad de incluir cambios sin tener que retocar muchos trozos de código. Además de proporcionar una visión clara del problema, y así poder dividirlo en subproblemas de menor complejidad.