DISEÑO DE SOFTWARE

PRÁCTICA 3

DANIEL YESTE LÓPEZ — 16884276
CHRISTIAN GUERRERO GÓMEZ — 16847191
RODRIGO CABEZAS QUIRÓS — 20100426

28 DE DICIEMBRE DE 2018

Índice

1. Objetivos

El objetivo en esta última entrega es aprender a aplicar patrones de diseño en el desarrollo de una aplicación basada en interfaces gráficas. Para lograrlo se implementará una GUI siguiendo el patrón Model – View – Controller y aplicando los principios GRASP y algunos patrones de diseño.

Las funcionalidades que debe implementar la GUI de StUB son las siguientes:

1. Registro de un nuevo usuario en el sistema.
2. Inicio de sesión de un usuario en el sistema.
3. Vista del catálogo de series, dependiendo de la etiqueta seleccionada:
    - Lengüeta de la lista Watched List (series vistas y acabadas).
    - Lengüeta de la lista Watch Next (series con episodios por ver).
    - Lengüeta de Not Started Yet (series cuyos episodios ya han estado visualizados pero que todavía tiene algunos pendientes por estrenar).
    - Lengüeta del catálogo de series, ordenada por fecha de estreno.
4. Acceso desde el catálogo a las diferentes temporadas y episodios de una serie.
5. Visualizar un episodio.
6. Valorar un episodio según el criterio de emoción.
7. Suscripción a un episodio.

2. Patrones aplicados

2.1. Observer:

El patrón observador ha sido aplicado en las clases Controller y User, siendo la primera la clase observadora y la segunda la observada. El uso que le hemos dado ha sido para actualizar las listas de WatchNext y Watched.

Cuando todos los episodios de una serie han estado visualizados, y no tiene pendientes de estrenar, esta pasa a la lista Watched donde se guardan todas aquellas series que ya se han visto. Cuando el usuario visualiza el último episodio pendiente, notifica a la clase Controller, que se encarga de mover la serie a la lista Watched.

Utilizando este patrón conseguimos actualizar las listas de usuario automáticamente cuando se produce un cambio de estado.

2.2. Strategy:

El patrón Strategy se ha usado para estructurar la utilización de la inserción en las listas en la clase Stub. Este patrón se utiliza sobretodo para no delegar el uso de los algoritmos directamente a los métodos de clase si no para hacerlo mediante una interfaz. Por este mismo motivo, hemos creado una interfaz Strategy y las distintas clases con Strategy para poder asignarlas a los distintos métodos. Consideramos apropiado el uso de Strategy en estos casos por la similar implementación del algoritmo.

2.3. Composite:

Este patrón se ha usado en las clases Actor, Director, Producer y Staff. Esta última era una clase abstracta, pero para aplicar Composite la hemos convertido en la interfaz Component.

Hemos considerado que era apropiado porque la organización de estas entidades es apropiada, es decir, el director de la serie tiene a su cargo a los diferentes actores, que serían las clases Leaf, y utilizando este patrón de diseño podemos estructurar la jerarquía más fácilmente.

2.4. Singleton:

El patrón Singleton sirve para asegurar que una clase tiene una única instancia y para proporcionar un acceso global a esta. Lo hemos aplicado en la clase Controller, así siempre tendremos una única instancia de Controller con su correspondiente.

2.5. Factory:

En este caso, se ha aplicado el patrón Factory a las clases DAO_XML_STUB y StaffFactory, ya que con esta implementación, evitamos violar un Open-close Principle de los principios SOLID: De esta forma se consigue que al crear un nuevo artista que puede ser un director o un actor, la clase StaffFactory se encarga de ver y crear el objeto que corresponde. Si se quisiera añadir o modificar estas clases, solo habría que ir a StaffFactory y hacer los cambios allí en vez de tocar código del DAO_XML_STUB.

Por desgracia, se sigue violando el principio Open-close todavía, pero su solución es demasiado complicada como para ser útil, pero al menos se ha mejorado un poco más y tenemos localizado el fragmento de código responsable.

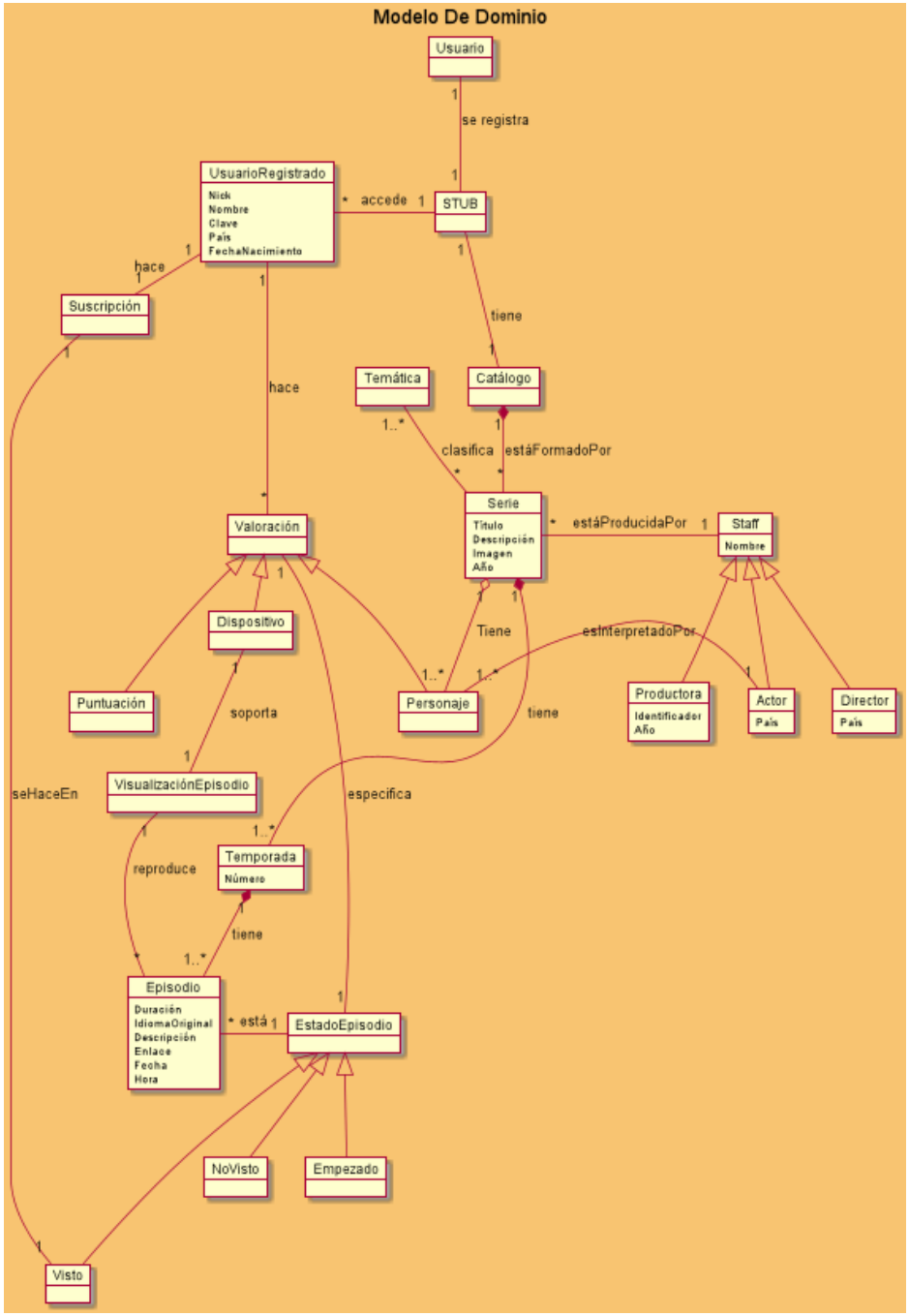
2.6. Facade:

El patrón facade ha sido utilizado para estructurar nuestra delegación de clases en capas, en nuestro caso para la relación entre Controller y User. Dada la necesidad de dsitribuir las clases de una manera mas homogénea y clara, hemos

decidido usar nuestra clase FacadeUser como puerta de entrada entre controller y user con la finalidad de simplificar clases y delegar a otras intermedias para hacer el código más legible.

3. Diagramas

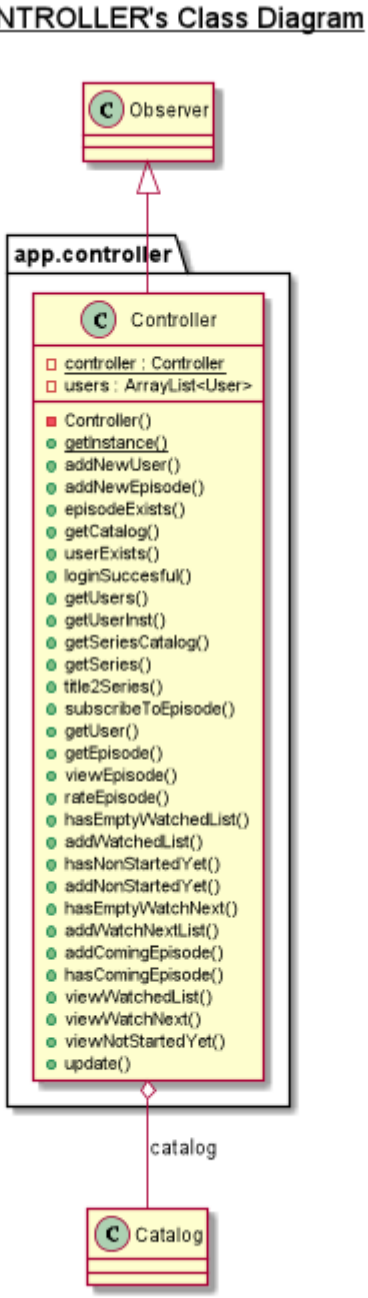
3.1. Modelo de dominio:



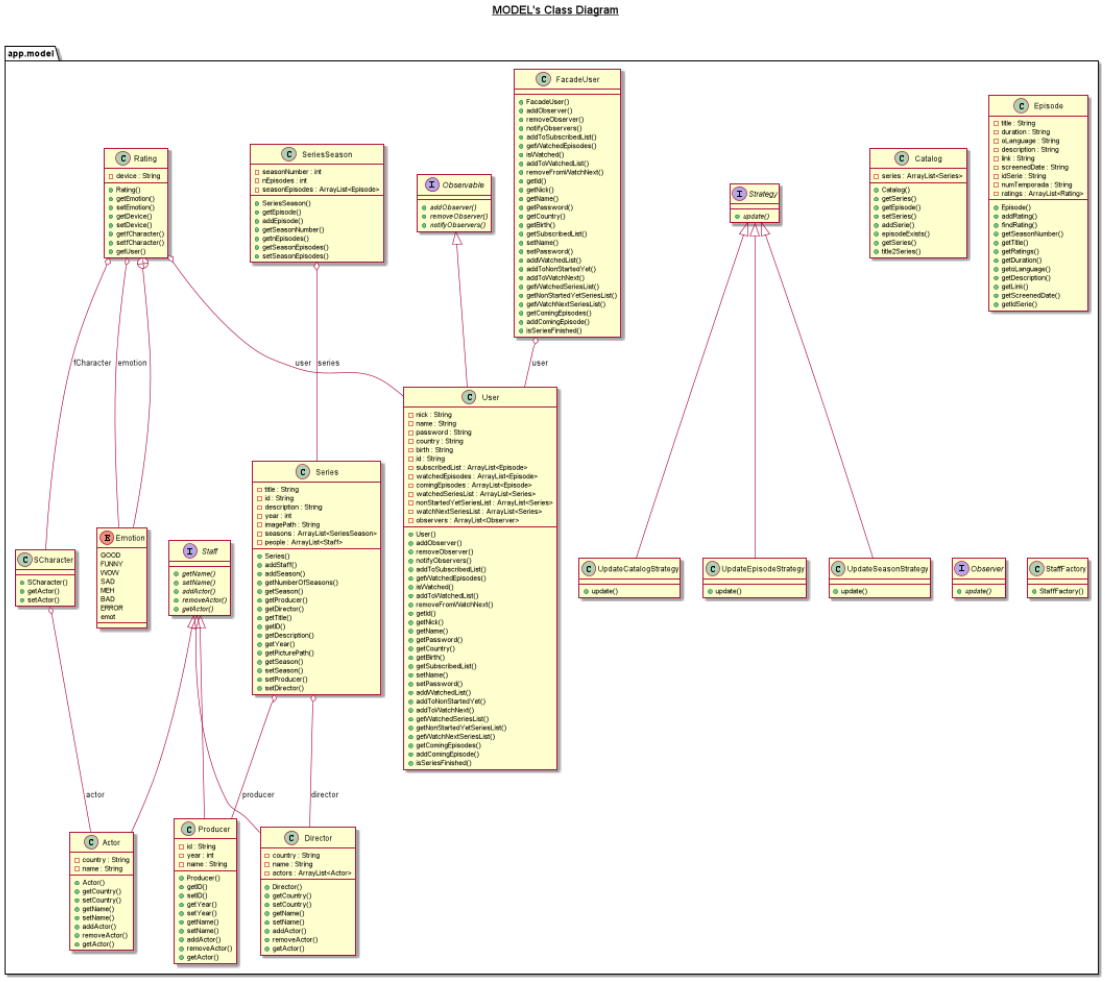
El modelo de dominio lo hemos mantenido intacto, sin modificar nada respecto a la anterior entrega pues no nos ha hecho falta.

3.2. Modelo de clases:

3.2.1. Paquete controlador:

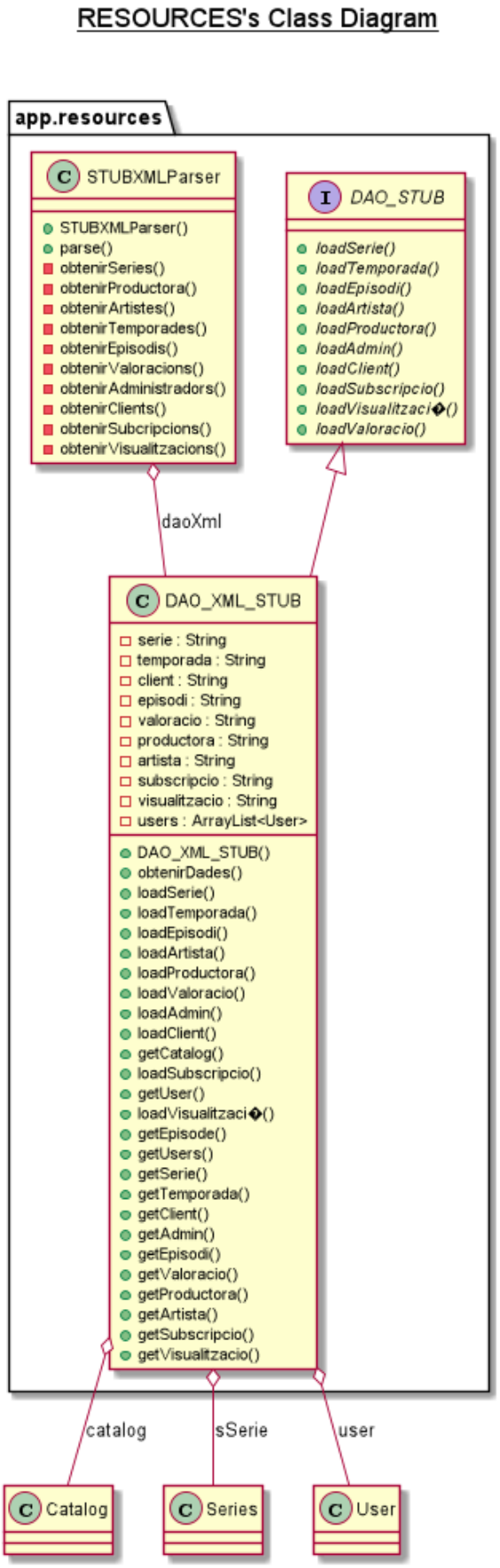


3.2.2. Paquete modelo:

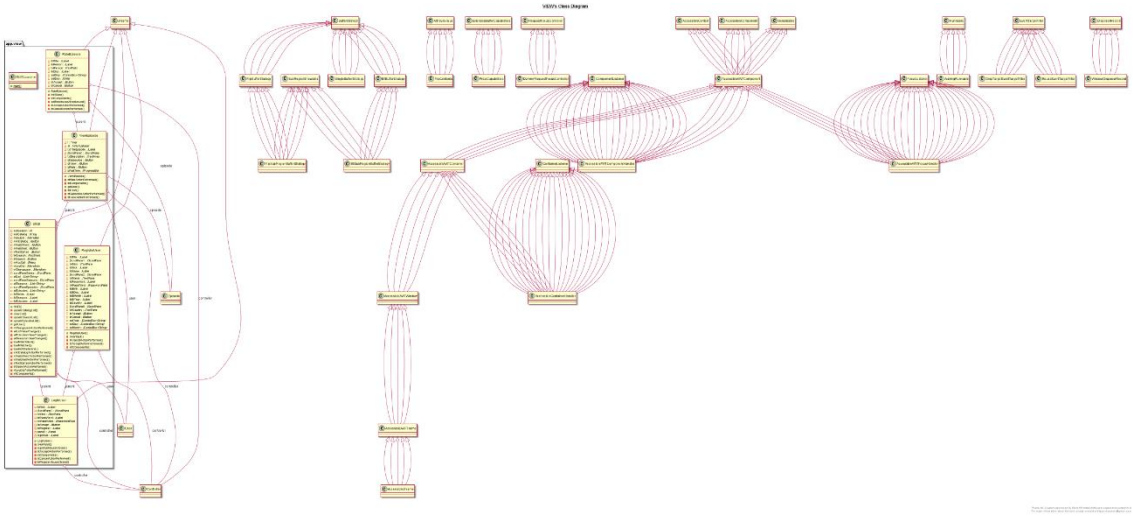


Seguir el hipervínculo para ver la imagen más ampliada.

3.2.3. Paquete recursos:



3.2.4. Paquete vista:



Seguir el enlace para ver la imagen más ampliada. Los diagramas también están incluidos en el proyecto.

En las clases del paquete vista, hemos incluido un atributo ‘parent’, que nos sirve para saber desde donde se ha creado la nueva ventana. De este modo, podemos realizar operaciones sobre la clase que crea instancia la nueva ventana, como por ejemplo cerrar o abrirla o acceder a los metódos públicos que esta implementa.

El paquete de recursos no lo hemos modificado en esta entrega, y las clases del paquete modelo las hemos cambiado para implementar los patrones de diseño, tal y como hemos explicado en el apartado anterior.

4. Observaciones

Hemos realizado tres puntos opcionales en la entrega: temporizador de 5 segundos, opción de búsqueda y valoración por personaje y/o dispositivo.

Para el widget del temporizador hemos utilizado la clase Timer. Este temporizador tiene un delay de un segundo y a cada “tic” decrementa el valor de la barra de progreso hasta llegar a 0. Una vez llega a 0, habilita el botón de valorar para que el usuario pueda dar su opinión sobre el episodio. Hemos entendido que la espera de los 5 segundos solo es para la primera visualización, siendo las siguientes visualizaciones inmediatas.

La búsqueda de series por nombre la hemos implementado utilizando un método del catalogo que devuelve una serie dado su título. Para mostrarla en la lista, hemos utilizado el mismo método que para el catalogo completo, sólo que en vez de pasarle un array completo se le pasa un array con la serie buscada.

Finalmente para valorar por personaje, en el JFrame de la valoración hemos añadido un campo de texto para que el usuario escriba su personaje favorito. Para seleccionar el dispositivo, el usuario selecciona uno de los disponibles en una JComboBox en el mismo JFrame.

5. Conclusiones

Para terminar, comentaremos como hemos asolido nuestros objetivos en la práctica y los problemas y conocimientos adquiridos en esta. Los objetivos principales de la práctica era aplicar los patrones de diseño, añadir una GUI a nuestro proyecto siguiendo el modelo MVC y aprender a identificar y aplicar los patrones demandados en nuestro código.

Con todo esto, hemos conseguido seguir nuestro modelo MVC ampliando la Vista y el Modelo para cumplir los requisitos. En la parte de la Vista hemos creado lo necesario para que funcionase nuestro GUI y en la parte del modelo hemos creado nuevas interfaces y clases para satisfacer nuestros patrones de diseño.

Nos hemos repartido el trabajo de una manera más o menos equitativa con Rodrigo haciendo toda la interfaz, Singleton y Observador, Christian haciendo Factory y Composite y Daniel haciendo Strategy y FacadeUser. El trabajo de la memoria nos lo hemos repartido cada uno asumiendo la parte correspondiente.

Por nuestra parte la implementación del GUI ha sido sencilla y sin problemas. Hemos tenido algunas objeciones de como aplicar los distintos patrones de diseño por tal de que el código se adaptara a estos pero finalmente las hemos solucionado de forma satisfactoria. El mayor problema de comprensión fue para la aplicación de Strategy donde preguntamos a los docentes para finalmente aplicarlo en la vista. Se utilizó para los métodos que trabajaban con las listas de la clase Stub, proporcionándoles diversas implementaciones para los problemas que se planteaban.

En conclusión, hemos aprendido como funcionan los patrones de diseño asignados, a escribir código de manera limpia y clara usandolos y a como distribuir este para que sea mas legible, portátil y comprensibe.