

TÍTULO DE GRADO EN

INGENIERÍA INFORMÁTICA

**Tecnologías de Desarrollo de Software**

Práctica final

**Documentación**

CURSO 2018 / 2019

CONVOCATORIA DE ENERO

Grechyshkin Vladyslav – vladyslav.grechyshkin@um.es

Valero Leal Enrique – enrique.valerol@um.es

Grupo 2.2

**Índice general**

[I. Diagrama de clases del dominio 1](#_Toc534457021)

[II. Diagrama de colaboración para añadir un vídeo 2](#_Toc534457022)

[III. Arquitectura de la aplicación 3](#_Toc534457023)

[IV. Patrones de diseño utilizados 4](#_Toc534457024)

[**4.1 – Patrón Factoría Abstracta** 4](#_Toc534457025)

[**4.2 – Patrón Adaptador** 4](#_Toc534457026)

[**4.3 – Patrón Singleton** 4](#_Toc534457027)

[**4.4 – Patrón Fachada** 4](#_Toc534457028)

[V. Componentes utilizados 5](#_Toc534457029)

[**1.1 – Títulos** 5](#_Toc534457030)

[VI. Tests unitarios 5](#_Toc534457031)

[**1.1 – Títulos** 5](#_Toc534457032)

[VI. Manual de usuario 5](#_Toc534457033)

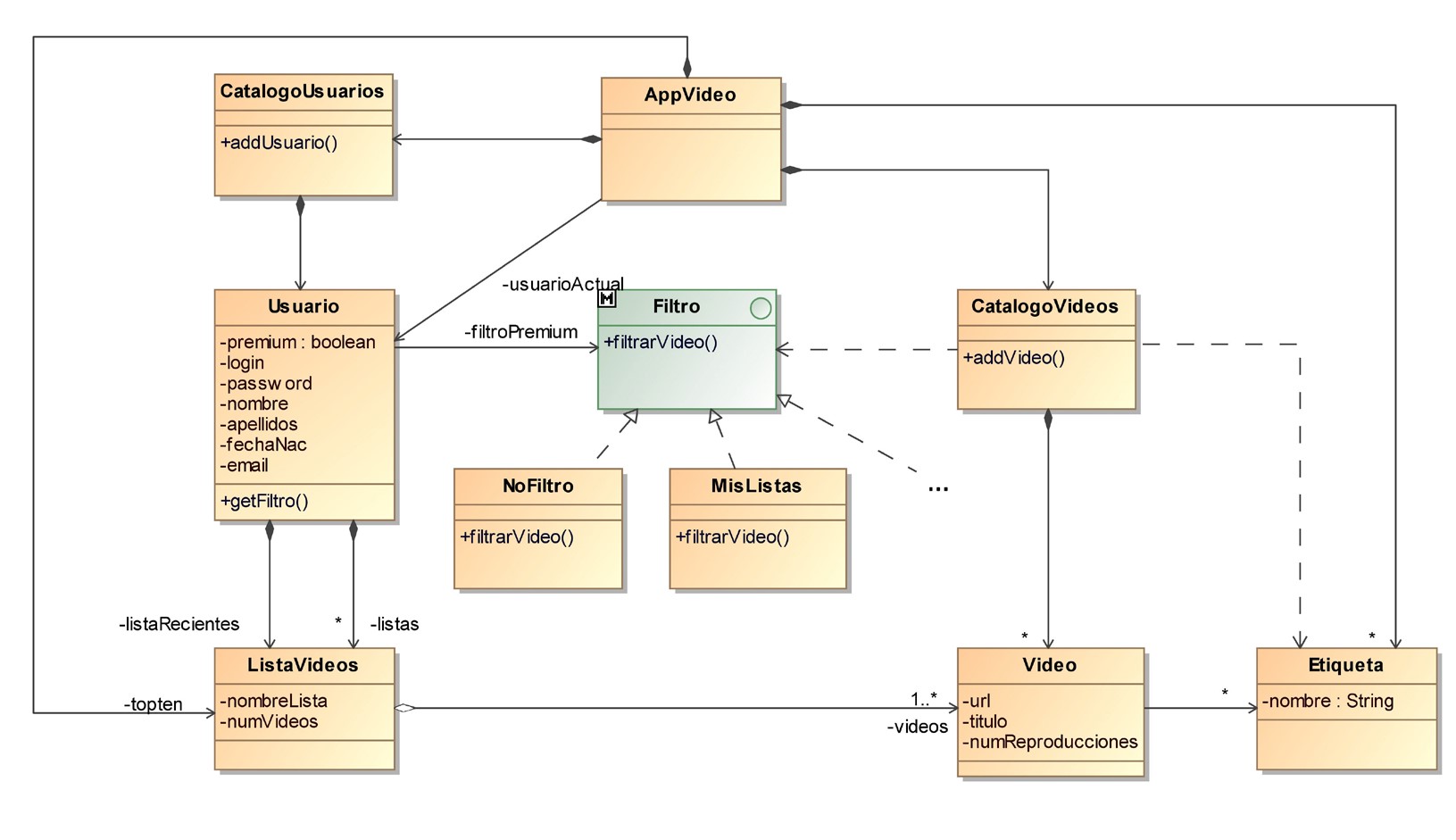
[**1.1 – Títulos** 5](#_Toc534457034)

[VII. Observaciones finales 5](#_Toc534457035)

[**1.1 – Títulos** 5](#_Toc534457036)

# I. Diagrama de clases del dominio

Adjuntamos el diagrama de clases propuesto por el profesorado y utilizado en el desarrollo de nuestra práctica.



**Ilustración 1. Diagrama de clases del dominio utilizado**

Nótese que ignoramos el diagrama de clases diseñado para la tarea parcial propuesta sobre este mismo, puesto que no es el utilizado en la práctica.

# II. Diagrama de colaboración para añadir un vídeo

Adjuntamos a continuación el diagrama realizado en *MagicDraw*, mostrando la secuencia de invocación de cada una de las clases involucradas para añadir vídeos a una lista.

# III. Arquitectura de la aplicación

La arquitectura general de la aplicación está basada directamente en la arquitectura de tres capas estudiada en la asignatura *(Modelo-Vista-Controlador)* y el principio de «separación modelo-vista». Excluimos cualquier explicación de dicho patrón arquitectural dado que no es el propósito de esta documentación. No obstante, resaltamos que, en la jerarquía de directorios y paquetes de la aplicación, existen, además de cada una de las tres capas, un paquete para las clases relacionadas con la persistencia para proporcionar mayor legibilidad y separación. Asimismo, incluimos un directorio con todos los recursos visuales utilizados para el desarrollo de la interfaz gráfica.

Sobre las decisiones de diseño, destacamos fundamentalmente el uso de las librerías JavaFX y JFoenix *(explicada en el apartado de componentes de este mismo documento)* de Java para la GUI. Asimismo, a diferencia de Swing, donde los elementos de la interfaz creados son añadidos estrictamente como código Java, el uso de la librería JavaFX y la herramienta visual JavaFX Scene Buildder nos permite generar ficheros «FXML»que contienen la jerarquía de ventanas y elementos de la aplicación en un lenguaje de marcas similar a XML, separando de manera efectiva el control de la vista de su creación.

Este fichero FXML *(en nuestro caso, «Root.fxml»)* se carga en la inicialización de la aplicación junto a la hoja de estilo utilizada, en formato .css, y el controlador de la vista *(especificado dentro del fichero FXML)*.

En cuanto a la estructura de nuestra GUI, destacamos el uso de un único fichero FXML *(bajo recomendación del profesorado)* para representar todas las ventanas indicadas en la especificación de la práctica. Esto facilita su implementación, reduce la complejidad y número de archivos, pero aumenta drásticamente el tamaño del controlador de la vista y no resulta de gran escalabilidad.

De este modo, la aplicación consta, en su nodo raíz *(a nivel de ventanas)*, de un *BorderPane* que, en su región central, contiene un *StackPane* con cada una de las ventanas *(Explorar, Mis Listas, Recientes, etc…)* de la aplicación, inicialmente invisibles. Para “abrir” cualquiera de ellas, simplemente volvemos invisible la ventana que estuviese mostrada, traemos la deseada al frente y la volvemos visible.

Como matices adicionales de elección de diseño, subrayamos la inclusión de la ventana «Nueva Lista», indicada en la especificación, dentro de la ventana de «Mis listas». Consideramos oportuno este cambio puesto que los elementos de cada una de las ventanas son casi idénticos *(en cuanto a la visualización de listas y vídeos)*. En vez de duplicarlos, por la falta de escalabilidad que hemos indicado antes, los fusionamos. Esta modificación queda reflejada en la capacidad de manejo total de las listas del usuario en una única ventana, en vez de separar la creación, edición y borrado de la reproducción.

Asimismo, incluimos una ventana adicional llamada «Mi perfil» que permite la modificación y consulta de la información del usuario actual de manera más extensa que la pedida en la especificación.

# IV. Patrones de diseño utilizados

Detallamos, en diferentes sub-apartados, cada uno de los patrones de diseño utilizados en el desarrollo de la práctica.

## **4.1 – Patrón Factoría Abstracta**

El uso de este patrón reside en la creación de cada uno de los adaptadores DAO utilizados para el manejo de la persistencia. Mediante la clase *TDSFactoriaDAO* conseguimos desacoplar cada uno de los adaptadores específicos, ofreciendo una interfaz genérica de adaptadores en su lugar.

## **4.2 – Patrón Adaptador**

Las clases adaptadoras indicadas en el subapartado anterior son una aplicación directa del patrón estructural adaptador. Su uso nos permite transformar la interfaz del servicio de persistencia utilizado en una interfaz genérica para el programador, ofreciendo mayor compatibilidad entre el servicio y la aplicación.

## **4.3 – Patrón Singleton**

Encontramos el uso de este patrón en la creación de diversos elementos de la aplicación, como los catálogos de usuarios y vídeos, los adaptadores previamente explicados, la factoría DAO, y la clase controladora de la aplicación *(AppVideo)*. Con este patrón garantizamos el acceso de una única instancia para cada una de estas clases y evitamos el uso de variables globales.

## **4.4 – Patrón Fachada**

El controlador de nuestra aplicación es un ejemplo claro del uso de este patrón. Con él, conseguimos proporcionar una única interfaz que combina el manejo de la capa de negocio y la capa de persistencia. De este modo, el usuario no necesita, por ejemplo, para operaciones como crear una lista de vídeos, trabajar con el catálogo y los adaptadores involucrados. En su lugar, proveemos un único método, *crearListaVideos*, abstrayéndonos a un nivel significativamente más alto.

# V. Componentes utilizados

Destacamos el uso de los siguientes componentes en el proceso de desarrollo de la aplicación:

## **5.1 – Componentes de la librería JFoenix**

Tal y como hemos indicado en el apartado de la arquitectura de la aplicación, la mayoría de los elementos visuales de nuestra aplicación son de la librería JFoenix. Puesto que, de manera general, estos elementos son en esencia los mismos que los de JavaFX pero con un aspecto visual minimalista y animado, únicamente puntualizamos los de mayor interés o peculiaridad. Estos son:

* **JFXDatePicker**
  + Este componente es el de nuestra elección para la introducción de la fecha de nacimiento en la ventana de registro. Aparte de proporcionar un panel bastante sofisticado para elegir una fecha, permite fácilmente obtener un objeto del tipo *LocalDate* con el método *getValue*.
* **JFXMasonryPane**
  + Utilizamos este tipo de panel para la visualización de las miniaturas de los vídeos en las ventanas de «Explorar» y «Mis Listas». Su funcionamiento es similar a un *FlowPane*, con el *bonus* añadido de ofrecer una animación cada vez que se añade un elemento o se redimensiona la ventana.
* **JFXNodesList**
  + Este componente proporciona un menú desplegable animado en forma de botones *(generalmente circulares)*. Lo utilizamos en la ventana de «Mis Listas» para organizar cada una de las funcionalidades disponibles sobre una lista *(borrar, modificar, añadir, etc…)*.

## **5.2 – Componente Buscador de Vídeos**

Este componente es el que hemos desarrollado siguiendo las instrucciones y código fuente proporcionado para la creación de un componente JavaBean. Puesto que las propias instrucciones contienen una explicación extensa sobre el componente, nos limitaremos únicamente a indicar su uso en nuestra aplicación.

La clase que importa el componente es *ViewController*, la clase controladora de la GUI. Esta, almacena la clase controladora, *AppVideo*, como oyente. Cuando se presiona el botón que hemos proporcionado para cargar vídeos *(dentro de la ventana de «Mi perfil»)*, se pide al usuario un fichero y se notifica al oyente, y este se encarga de procesar y añadir los vídeos cargados por el componente.

## **5.3 – Biblioteca iTextPDF**

Para la implementación de la funcionalidad Premium de generar un archivo .pdf con las listas de vídeos del usuario, utilizamos el componente *PdfWriter*, dentro de la clase controladora. No indagamos en profundidad en las diferentes opciones de esta herramienta, limitándonos únicamente a crear el PDF en un formato sencillo.

## **5.4 – Componente VideoWeb**

Para la reproducción de los vídeos de la aplicación utilizamos el componente proporcionado en los recursos de la asignatura. Como su explicación y forma de uso ya está explicada, únicamente matizamos su implementación en nuestro código.

Todos los vídeos se muestran dentro de una ventana emergente del tipo JFXDialog. Esta ventana contiene todos los elementos de la especificación, como las etiquetas y las reproducciones del vídeo. Parametrizamos la creación de este diálogo en un método auxiliar llamado *showVideoDialog* para la reutilización del código en las diversas ventanas donde se puede reproducir un vídeo.

# VI. Tests unitarios

Adjuntamos en la entrega los tests unitarios bla bla luego cuando esté hecho se matiza

# VI. Manual de usuario

Horrible necesito paciencia para hacer esto

## **1.1 – Títulos**

Texto

# VII. Observaciones finales

Texto

## **1.1 – Títulos**

Texto