DECSAI

Universidad de Granada

Sistemas Multimedia

PRÁCTICA DE EVALUACIÓN

Víctor Padilla Cabello | 76066048-B | Curso 2017-18 |vpadi@correo.ugr.es

Introducción

Se requiere la creación de una aplicación que sea capaz de satisfacer los requisitos impuestos por la evaluación de la asignatura Sistemas Multimedia. Estos son la de creación, procesamiento y visualización de medios (imágenes, audio y video).

Así pues, se ha desarrollado una aplicación denominada **MediaTic** que cumpla todos estos objetivos según se ha ido estipulando en las diferentes prácticas que se han desarrollado durante el curso y en la práctica de evaluación. En el siguiente apartado se procederá a especificar lo realizado para solucionar cada requisito y las elecciones de diseño tomadas.

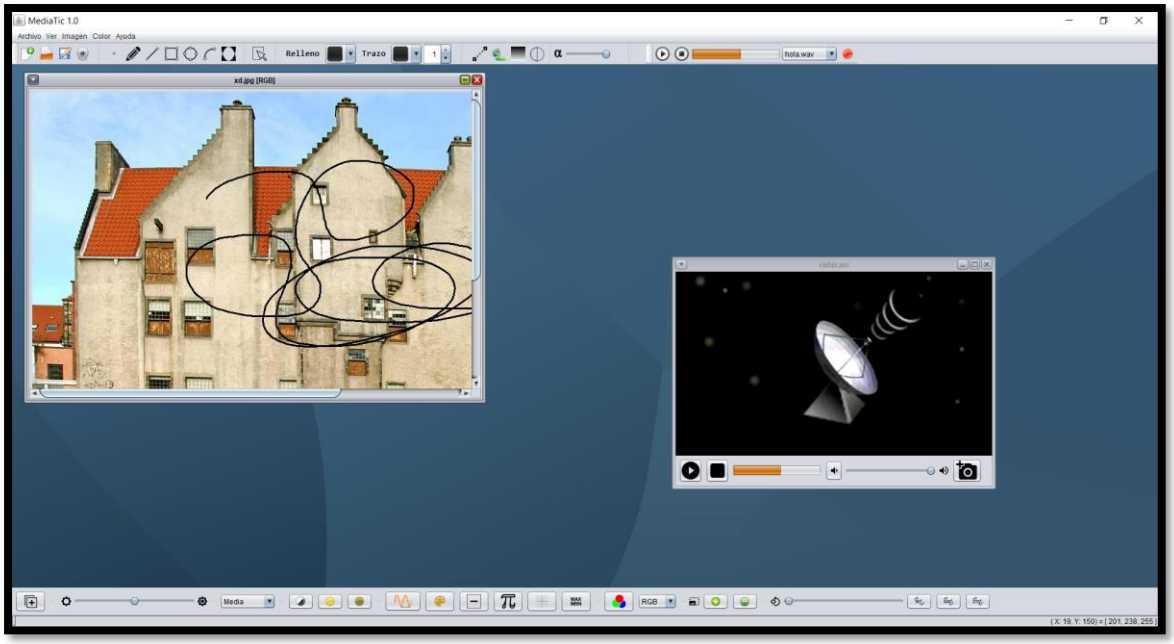


Figura 1: Aspecto de la aplicación básica

Especificaciones del diseño

General

Para realizar la entrada y salida de archivos me he basado en el diseño típico de las aplicaciones que usamos hoy en día: Archivo nuevo, abrir archivo y guardar archivo. Para las opciones de abrir y guardar archivo solo se considerarán las ventanas y el medio de imágenes. Con respecto a abrir, se puede usar sobre cualquiera de los tres medios.

Para la opción de nuevo archivo, la aplicación lanza una nueva ventana de dibujo en blanco que el usuario puede directamente usar para pintar figuras y trastear con ella. Para redimensionar el cuadrante de dibujo solo bastaría irse al

menú Imagen -> Cambiar el tamaño del marco o pulsar Ctrl+T, y se lanzará un dialogo donde se podrán introducir las nuevas dimensiones del lienzo.

Para la opción de abrir archivo se consideran todos los tipos de medios disponibles. Esto incluye Imágenes (que cargará una Ventana Interna del tipo Imagen), Audio (que lo cargará en la lista de reproducción, el comboBox situado arriba a la izquierda) y Video (que cargará una Ventana Interna del tipo reproductor VLC).

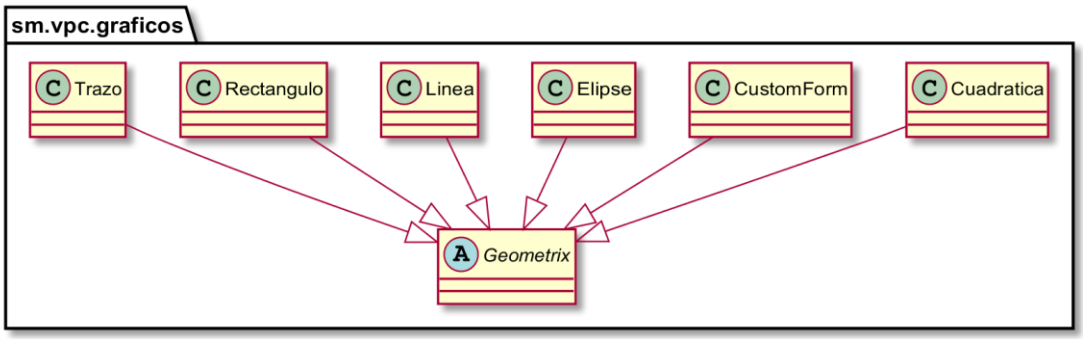
Para el guardado, al igual que el nuevo archivo, se considerará las ventanas del tipo Imagen, y estás se guardará en un archivo (con todas las formas dibujadas) que el usuario especifica a través de una ventana de diálogo (que contendrá un filtro para todos los formatos soportados por JAVA).

Respecto a los menús *Ver* y *Ayuda* se han añadido opciones para cumplir los requisitos de ocultar las diferentes barras y componentes, y mostrar una ventana de información sobre la aplicación.

➢ Dibujo

Para satisfacer los requerimientos de dibujo se ha procedido a realizar lo siguiente:

- Se ha definido una clase nueva de diseño propio **Canvas** que hereda de JPanel. Esta dispone de un vector de formas (*Geometrix)* que guarde todas las figuras que se vayan añadiendo al lienzo.
- Se ha definido una clase abstracta nueva **Geometrix** que hará de superclase para todas las figuras que el usuario quiera definir. Está compuesta por un atributo de la clase Shape *figura*, representación de la figura geométrica en sí, que en cada subclase tomará el valor correspondiente a la forma en sí (*Rectangle, Line2D, Ellipse2D, etc..)*; y todos los atributos de dibujo necesarios para la práctica en forma de objetos de clases que luego se aplicarán al objeto *Graphics2D* (Color, Stroke, transparencia, etc...). El esquema se podría resumir en el siguiente diagrama de clases:



- Para la edición de figuras he elaborado un modo dentro del propio *Canvas* que consiste en un booleano que indica si se está editando o no. Si este es "falso" se pasa a crear nuevas figuras y se ignora la selección de cualquier figura. Si es "verdadero", no se crean nuevas figuras y el cada click cuenta como una posible selección de las figuras. Tal modo se activa con un botón en la barra de tareas.

  Además, he creado un atributo de la clase Rectángulo, *contorno*, que representa a la BoundingBox de las figuras. Este se pinta a parte y con trazo discontinuo llamando a la figura seleccionada y a su método drawEditBox().

  Para cambiar lo atributos de una figura la seleccionamos, y una vez seleccionada, al cambiar cualquier atributo de dibujo de la barra de herramientas, esta llamará a los respectivos métodos "setter" de las figuras de los rasgos cambiados (solo si está en modo editar).

- Para la reordenación de figuras he creado un PopupMenu que es solo accesible si se está en modo editar y se selecciona una figura del lienzo. Es decir, para cambiar la ordenación, hacemos click derecho en cualquier figura del lienzo mientras estamos en modo editar y saldrá un menú emergente con las distintas opciones. Cada será responsable de mover las figuras dentro del Array de *Geometrix*.

Respecto a las formas geométricas posibles (que heredan de *Geometrix)*, he definido 7 que son las formas obligatorias de la práctica, todas ellas asociada a un botón en la barra de herramientas. La correspondencia figura-clase es:

- Línea Recta -> Linea

- Rectángulo -> Rectangulo

- Elipse -> Elipse

- Curva con punto de control -> Cuadratica

- Trazo libre -> Trazo

- Forma personalizada -> CustomForm

En cada clase definida se hereda de *Geometrix* y se le asigna una forma específica de las ya existentes en JAVA, como puede ser *Rectangle, Ellipse2D, Line2D…* Para más información sobre la jerarquía se recomienda ver el diagrama de clases antes mostrado y la API de la biblioteca.

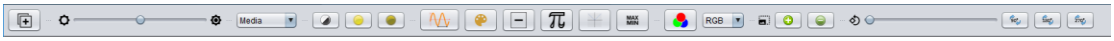
Para los atributos, he añadido objetos de las diferentes clases representantes de atributos disponibles para personalizar un *Graphics2D*. Todos son los

requeridos por la práctica: dos atributos Color para el relleno y el contorno de la figura; otros dos Color para el gradiente si está activado, una variable booleana para saber si está activado el relleno o no; una variable entera que indica el grosor del contorno; una variable booleana que indica si está activado el trazo discontinuo o no; el tipo de discontinuidad; una variable entera que indica el grado de transparencia y una variable stroke que representa el trazo del contorno con todas sus características. Esto conlleva que todos estos atributos tengan su "setter" y "getter" correspondientes.

➢ Imágenes

Para las imágenes se extiende el comportamiento de la clase **Canvas** para que se puede incluir una imagen de fondo en el *lienzo* (será un atributo de la clase y se sobrecargará el método *paintComponent* que tiene los componentes JAVA Swing para usar el *drawImage* de **Graphics**).

En lo que a filtros definidos respecta, hacemos una separación para la explación de estos en la documentación. Por una lado están los que utilizan los propios métodos y clases de las librerías de JAVA, y por otro los que han sido definidos por el mí que vienen incluidos en unas clases propias y por tanto en archivos independientes.



Aspecto de la barra de herramientas asociadas a filtros y cambios en la imagen

En el grupo de los ya definidos nos encontramos con la:

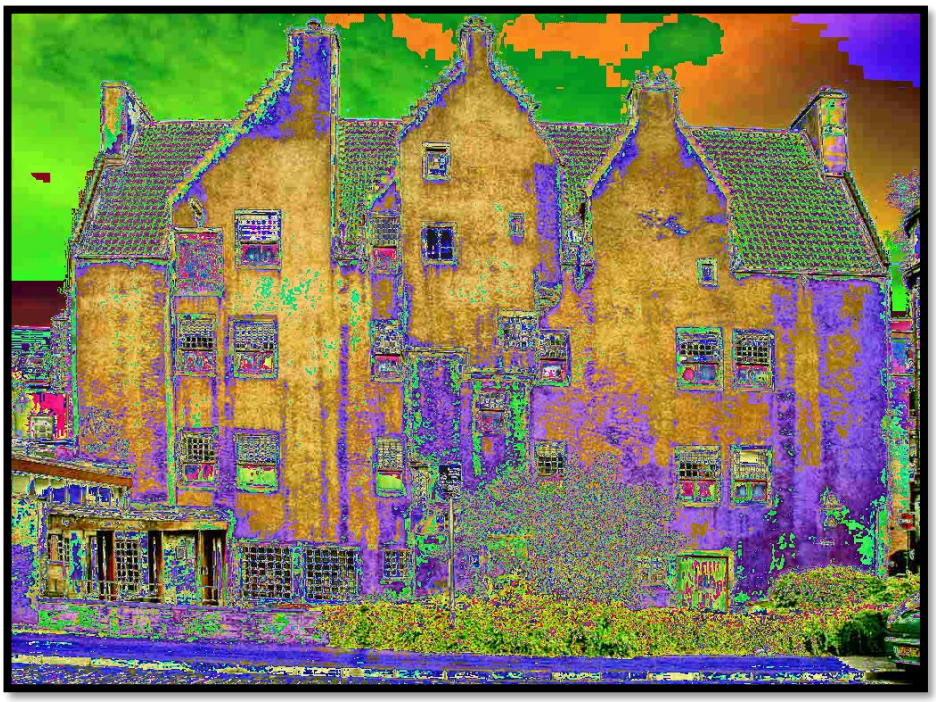
- Duplicación de una imagen (que se basa en crear un WriteableRaster a partir del de la imagen disponible)
- Un selector de brillo, que hace uso de una filtro del tipo **RescaleOp.**
- Un comboBox con diferentes filtro del tipo **ConvolveOp,** entre ellos filtro Media, Binomial, Enfoque, Relieve y Laplaciano (las matrices kernel hacen uso de la clase KernelProducer definida por el profesor).
- Tres botones de ajuste de contraste/brillo, que hacen uso de operaciones del tipo **LookupOp** (las *LookupTable* en este caso la obtenemos de la clase *LookupTableProducer* definida por el profesor).
- Un botón que descompone la imagen en bandas dependiendo del espacio de color y carga cada una en una *VentanaInternaImagen* nueva.
- Una comboBox que permite cambiar de espacio de color entre *RGB, YCC y Grey* (blanco y negro) y carga la imagen con ese espacio de color nuevo en una nueva *VentanaInternaImagen*.
- Dos botones para el reescalado en tamaño de la imagen, que hacen uso de un **AffineTransform.**
- Un Slider y 3 botones que permite modificar la rotación de la imagen, que al igual que la anterior operación hacen uso de ***AffineTransform*** pero esta vez de Rotación.

Por otro lado, en el grupo que he definido yo, nos encontramos con:

- La operación de filtro sepia que se explicó en clase de prácticas. Esta se define como una clase que hereda de *BufferedImageOpAdapter y* contiene un método *filter* que realiza la operación sobre la imagen.
- La operación de producto por pi modular (Componente a Componente), que su definición es idéntica a la anterior. Está viene definida por:

$$Banda_x = (Banda_x * \pi) \; \boldsymbol{mod} \; 256$$

El resultado de esta es un incremento lineal de las bandas de la imagen (colores más vivaces/saturados), pero que a los colores más vivaces los vuelve oscuros pues se supera el 256 y vuelve a comenzar de 0. Un ejemplo sería este:



La imagen original es la proporcionada en clases: Casa.

- Una operación (Pixel a pixel) que realiza la media del máximo y del mínimo valor del pixel y se la suma al propio valor de la banda ponderado cada uno con 0,5. La formulación podría ser esta:

$$Banda_k = \frac{Max(Pixel_{xy}) + Min(Pixel_{xy})}{2} + \frac{Banda_k}{2}$$

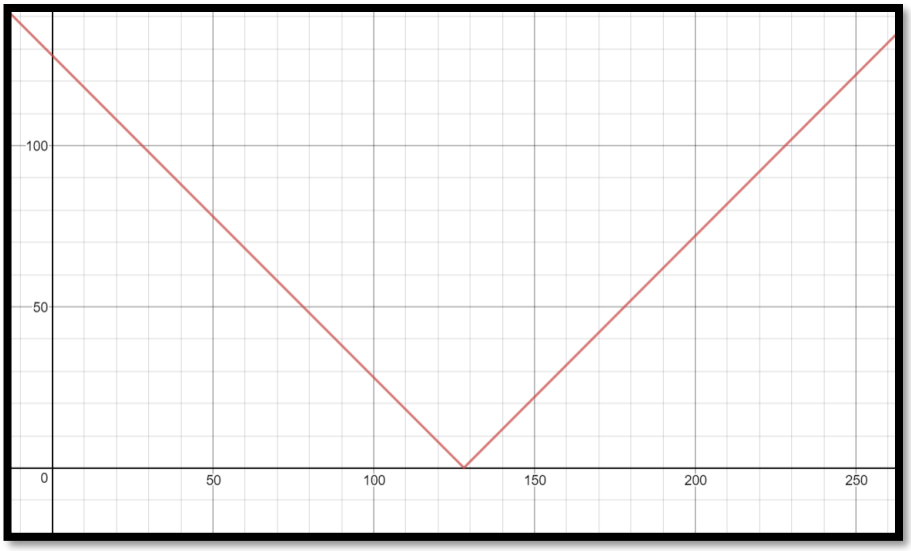
Lo que realiza esta operación es una convergencia de los valores de las bandas a un término medio. Esto es que "suaviza" las diferencias de los valores y cada vez va adquiriendo la imagen un tono más tenue y acercándose a la escala de grises (donde todas las bandas valen igual). Ejemplo de esto:

Aplicándolo una vez

Aplicándolo por segunda vez

- Para la aplicación de filtro seno y de declaración propia, he definido una clase similar a la que nos dio el profesor en prácticas (*LookupTableProducer)* denominada **LookupTableCreator,** donde agrupa todos los métodos de creación de las tablas necesarias para realizar las *LookupOp*.

Con respecto a la operación respecto a la función definida por el estudiante me he inclinado por el uso de la función absoluto de X:



Como se puede observar está ha sido desplazada a 128 siendo la función resultante

$$f(x) = |x - 128|$$

Gracias a esto prescindiendo del factor de corrección *K* pues el máximo siempre es 128.

El efecto que resulta de aplicar esta operación podemos deducirlo de la gráfica de la función, hará que los colores muy oscuros se vuelvan más brillantes, y los muy brillantes se vuelvan más oscuros. Sin embargo, lo interesante de esto está cuando se aproxima al término medio (128) donde los colores (bandas) caen su valor en picado hasta el 0, haciendo que los colores intermedios sean más oscuros. Lo interesante de esta operación es que, si se aplica dos veces, da una imagen resultado como si se hubiese aplicado el filtro negativo.

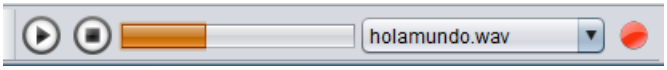
Aquí la ejemplificación:

Ejecutando una vez el filtro

La imagen resultante de ejecutar el filtro de veces.

➢ Audio

Para la reproducción de sonido se ha seguido lo especificado en la práctica 13 de la asignatura. Se ha hecho uso de la clase *SMClipPlayer* para el reproductor de audio y *SMRecorder* para el grabador, por lo que se sigue la filosofía de la JAVA SOUND API.



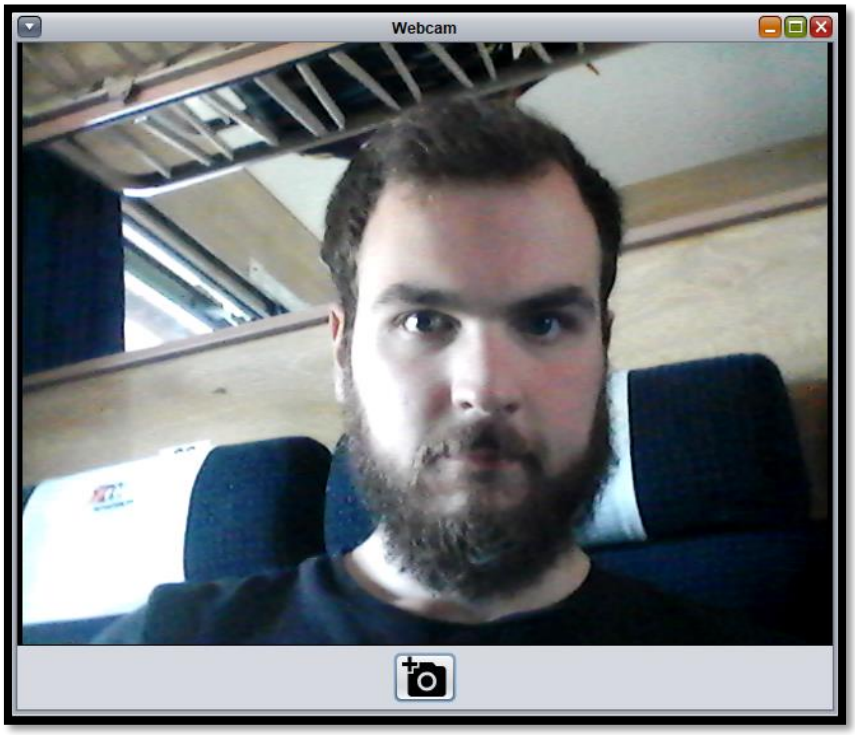
Barra de herramientas de sonido

Se añade una sencilla barra de herramientas con un botón de Play/Pause, que hace uso de los métodos de SMClipPlayer *play, pause y resume*; un botón de stop que detiene la reproducción y la reinicia (método *stop*); un ComboBox con la lista de reproducción de los audios; una ProgressBar que indica el progreso del audio reproduciéndose (Hace uso de una hebra definida por mí para actualizar constantemente el valor de la barra); y finalmente un botón de grabación para comenzar y detener una grabación. Al final de esta, se lanzará un diálogo para seleccionar el archivo donde se guardará el audio grabado.

➢ Video

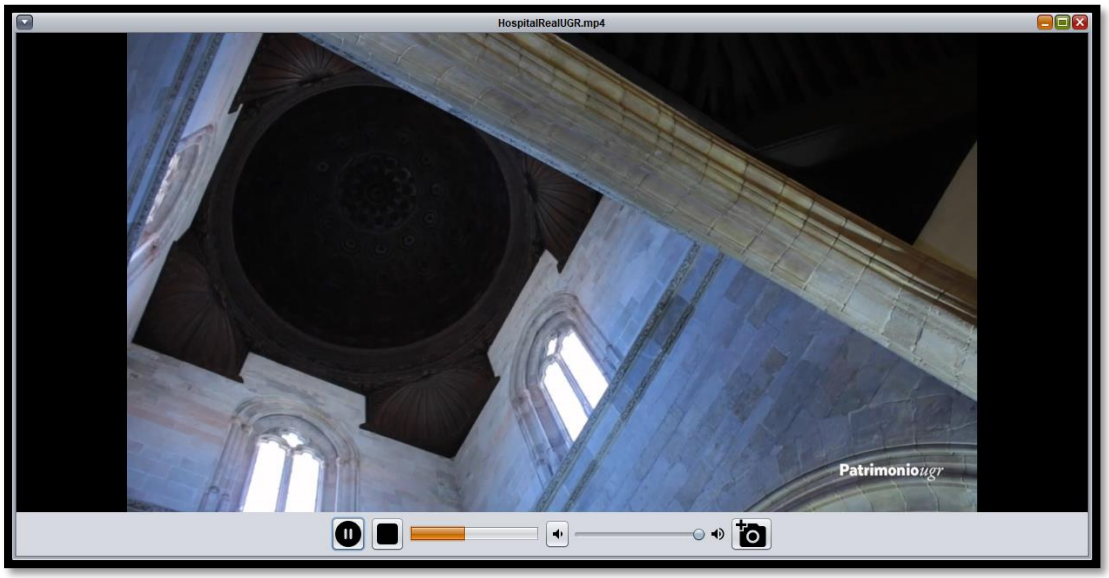
Para satisfacer los requisitos correspondientes a los medios de video (reproducción y captura de webcam) he definido dos clases nuevas que heredan de la otra clase de definición propia *VentanaInterna*: **VentanaInternaCamara** y **VentanaInternaVLC** (también está definida otra en la que se utiliza la *JMF* pero como no se ha hecho uso de ella en la aplicación final no se describirá).

- Respecto a **VentanaInternaCamara** se ha seguido estrictamente lo estipulado en la última práctica de la asignatura, haciendo uso de la librería de tercerlos *WebCam* que facilita mucho la reproducción en el entorno visual de JAVA. Cuenta con la adición de un botón en la barra inferior que realiza una captura del "frame" actual y crea una VentanaInternaImagen con el resultado de la operación donde se podrá manipular al igual que se hace con cualquier otra imagen.



Aspecto de la ventana de webcam

- Respecto a **VentanaInternaVLC** se ha seguido la filosofía del uso de bibliotecas de terceros (*VLCj)* para la reproducción de video. El Internal Frame cuenta con un panel en el medio que genera el *vlcplayer* de la clase **EmbeddedMediaPlayer.** Este dispone abre un medio (tanto audio como video) y se puede obtener un panel donde se reproducirá la secuencia en el caso que se trate de un video. Este panel lo colocamos como hemos mencionado en el centro de una VentanaInterna con *BorderLayout*.



Ventana interna para la reproducción de video.

En lo que a los controles del video refiere, he añadido los básicos de un reproductor: un botón de play/pause (que llama internamente a los métodos *play y stop);* un botón de stop que reinicia el video; un barra de progreso que muestra el avance del video; un botón de silenciar/desilenciar el video; una barra de control de volumen (en forma de JSlider) y un botón para realizar la captura de un frame del video reproduciéndose y al igual que pasaba con la webcam, lo lanza como una imagen dentro de una **VentanaInternaImagen**.

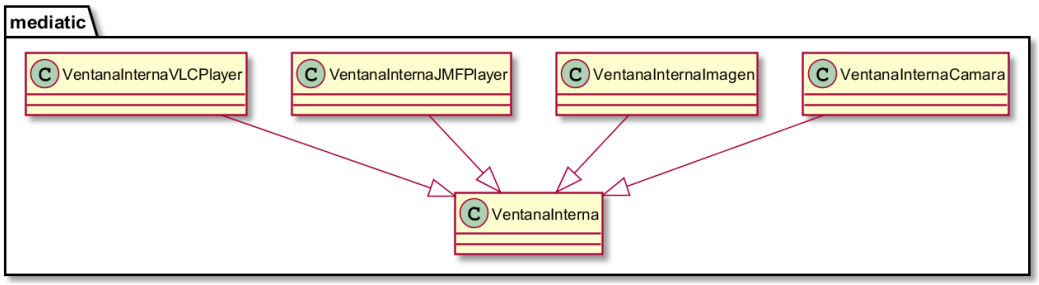


Diagrama de clases de las ventanas internas