Escuela Técnica Superior de Ingeniería Informática y Telecomunicaciones

Ciencias de la Computación e Inteligencia Artificial

MI APLICACIÓN MULTIMEDIA

Proyecto final de Sistemas Multimedia

Documentación escrita por Jesús Ruiz Castellano.

Profesor:

Jesús Chamorro Martínez.

Análisis sobre el dominio del problema

¿Qué hay actualmente parecido a esta aplicación?

En cuanto a las funciones de edición de imágenes, así como creación de figuras sobre un lienzo estilo *'Paint'* podemos encontrar la clase **Graphics**.

Problema: no hay clases asociados a las formas geométricas, el diseño de ésta es muy limitado, y cualquier funcionalidad que se desvíe un poco de lo que ellos pensaron, tienes que implementarlo tú mismo. Las propiedades modificables son sólo el color y el formato del texto (para los que sí existen clases). Existen diversas aplicaciones con las que se puede editar imagenes, reproducir vídeos o sonidos, como Paint de Windows, VLCPlayer, Reproductor de sonido propio de cada sistema operativo, etcétera.

¿Qué le falta a esas aplicaciones o qué se podría mejorar?

Lo que no existe es una misma aplicación que aúne a todas las aplicaciones citadas en el apartado anterior y las convierta en distintas funcionalidades de una única aplicación, como va a ser este caso.
Lo esencial es un buen diseño, el cuál implicaría cambiar varias cosas de su implementación como el tener que hacer uso de ***instanceof*** para hacer alusión a un tipo de figura determinado.

¿Cuál es mi propuesta para solucionarlo?

Para esto he cambiado el diseño, la manera de ver el concepto de ***Figura*** como una *súperclase* la cuál tiene unos valores que son comunes a todas las figuras y de ahí heredarán cada una de los distintos tipos de figuras aportando, además, a cada una sus distintas propiedades como por ejemplo *contorno* y *relleno*.

Todo esto hace que la implementación de clases y métodos en los que se necesite usar figuras sean más fáciles y sencillos.

Descripción del Sistema

He creado una aplicación multimedia que permite gestionar diferentes tipos de medios: gráficos, imágenes, sonido y vídeo. Sobre cada uno de dichos medios se pueden llevar a cabo diferentes tareas que, en función del medio, abarcan desde la creación y/o captura hasta la edición de imágenes, reproducción y procesamiento de audio, sonido grabado, y vídeo. Todo esto será realizado en diferentes módulos distinguiendo entre imagen, sonido y vídeo.

Para ello, la aplicación contará con un conjunto de menús y barras de herramientas que permitirán llevar a cabo dichas tareas.

Y todo a través de una Interfaz gráfica, para que el usuario pueda realizar cualquier funcionalidad que necesite de la manera más cómoda posible.

Capítulo 1

Análisis de requisitos

1.1. Requisitos de datos

- **RD - 1. Datos asociados a Escritorio y Ventanas :**
  - Escritorio
  - VentanaPrincipal
    - listaFiguras
    - bufferedImage
  - VentanaInterna :
    - VIImagen
    - VICamara
    - VIVideo
- **RD - 2. Datos asociados a los botones de las distintas funcionalidades.**
- **RD - 3. Datos de archivos :**
  - archivoTemporal
  - achuraNuevoLienzo
  - alturaNuevoLienzo
- **RD - 4. Datos de figuras :**
  - forma
  - transparencia
  - esAlisado

- contorno :
  - colorContorno
  - esDiscontinuo
  - discontinuidad
  - grosor
  - espacioDiscont
- relleno :
  - esRelleno
  - colorRelleno

- **RD - 5. Datos para la gestión y manipulación de imágenes :**
  - brillo
  - filtro
  - contraste
  - anguloRotacion
  - escala
  - umbral
  - banda
  - espacioColor
- **RD - 6. Datos de sonido :**
  - listaReproduccion
  - player
    - recorder
  - sonidoPausa
  - sonidoStop
- **RD - 7. Datos de vídeo :**
  - vlcplayer

1.2. Requisitos Funcionales

RF-1 **Funciones sobre medios e interfaz de usuario.**

RF-1.1 *Funciones de carácter general.*

RF-1.1.1 Abrir. Consiste en a nadir alguno de los ficheros a la aplicación, ya sea una imagen, un sonido a la lista de reproducción, o un vídeo.

RF-1.1.2 Guardar. Una vez procesado el ítem que hayamos estado usando, lo guardaremos en el tipo de archivo que corresponda, nuestro propio equipo.

RF-1.1.3 Ver. Con esta opción podemos mostrar/ocultar las diferentes barras de edición y propiedades.

RF-1.1.4 Imagen. En este apartado tenemos un menú para seleccionar cada uno de los diferentes filtros que implementaré en otros botones.

RF-1.1.5 Ayuda. Botón desde el que se proporcionará un diálogo con información acerca del nombre y versión de la aplicación, mi nombre, y año de creación.

RF-1.1.6 Actualizar estado. Con esta función me encargo de que se mantengan constantemente actualizados los valores de los diferentes botones correspondientes a propiedades, independientes para cada *VentanaInterna*.

RF-1.1.7 ActualizarVentana. Proporciona la capacidad de mantener intactos los distintos atributos y propiedades asignados a cada
gura en cada *VentanaInterna*.

RF-1.1.8 Obtener Extensión. Gracias a esta función podemos obtener la extensión del archivo indicado.

RF-2 **Funciones sobre gráficos e imágenes.**

RF-2.1 *Gráficos (figuras).* La gestión debe este módulo debe de hacerse en una misma ventana. Dicha ventana la llamaré *VentanaInternaImagen*.

RF-2.1.1 Nuevo. Abre una nueva *VentanaInternaImagen* con un Lienzo en blanco, dispuesto para añadir cualquier figura y/o imagen.

RF-2.1.2 Dibujar. Ya sea sobre un Lienzo en blanco o sobre un Lienzo que tenga una imagen, se podrán pintar figuras de diferentes propiedades cada una. Cuando finalice este hecho, la nueva figura se añadirá a una lista que contiene todas las figuras dibujadas en un mismo *Lienzo*. Podremos distinguir entre *Punto, Línea, Rectángulo y Elipse*.

RF-2.1.3 Mover. Una vez seleccionada la figura de la lista, que podremos distinguir gracias a q se enmarcará, podremos indicar hacia dónde moverla.

RF-2.1.4 Editar. Con la figura seleccionada, podremos modificar todas las propiedades de ésta que queramos, sin que afecte al resto de figuras, tanto del mismo *Lienzo*, como de otra *VentanaInterna* :

RF-2.1.4.1 Color. Podremos cambiar tanto el color de relleno (en caso de que la figura esté rellena), como el del contorno.

RF-2.1.4.2 Trazo. Contorno. Podremos modificar el tipo de continuidad del trazo.

RF-2.1.4.3 Grosor. Además, el grosor del contorno podrá ser modificado.

RF-2.1.4.4 Relleno. Dependiendo de si la figura tiene o no, se coloreará del color seleccionado en la barra de propiedades.

RF-2.1.4.5 Transparencia. Se podrá modificar el grado de transparencia de cualquier figura.

RF-2.1.4.6 Alisado. Además, también se podrá mejorar la calidad del pixelado de los bordes de la figura seleccionada.

RF-2.1.5 Procesar. Editaremos la imagen abierta. Para ello usaremos los distintos datos del apartado **RD-5**.

RF-2.2 *Imágenes.* La aplicación permitirá aplicar un conjunto de operaciones que se podrán llevar a cabo sobre cualquier imagen.

RF-2.2.1 Duplicar. Creará una nueva *VentanaInternaImagen* con una copia de la imagen.

RF-2.2.2 Brillo. Se puede asignar un valor determinado de brillo mediante un deslizador.

RF-2.2.3 Filtros :

- **RF-2.2.3.**1 Emborronamiento.
- **RF-2.2.3.**2 Enfoque.
- **RF-2.2.3.**3 Relieve.

RF-2.2.4 Contraste :

- **RF-2.2.4.**1 Normal.
- **RF-2.2.4.**2 Iluminado.
- **RF-2.2.4.**3 Oscurecido.

RF-2.2.5 Negativo. Función con la que invertiremos los colores de la imagen.

RF-2.2.6 Extracción de bandas. Mostraremos las bandas asociadas a la imagen.

RF-2.2.7 Conversión a los espacios RGB, YCC y GRAY. Se selecciona de la lista que habrá en la *VentanaPrincipal* uno de ellos y tendrá que aparecer en el *Escritorio* una *VentanaInternaImagen* mostrando la nueva imagen en el espacio de color seleccionado.

RF-2.2.8 Giro. Se podrá rotar la imagen a derecha y a izquierda, según los valores de un deslizador y de 3 botones específicos.

RF-2.2.9 Escala. Se podrá aumentar y disminuir el tamaño de la imagen.

RF-2.2.10 Tintado. Al seleccionar esta función, se tintará del color seleccionado toda la imagen.

RF-2.2.11 Ecualización. Incorporaré la posibilidad de ecualizar la imagen seleccionada.

RF-2.2.12 Sepia. Se trata de uno de los efectos más clásicos en los programas de edición de imágenes, en el que se modifica el tono y saturación para darle un aspecto de "fotografía antigua".

RF-2.2.13 Umbralización. Implementaré la umbralización basada en intensidad.

RF-2.2.14 Umbralización inversa. De manera adicional, le he añadido esta funcionalidad, que es muy parecida que la normal. Lo que en el operador normal se ve de color blanco, ahora lo vemos negro, y viceversa.

RF-2.2.15 Umbralización basada en color. Adicionalmente, he añadido la funcionalidad de hacer que el valor de la umbralización aparezca en el color que tengamos seleccionado en el *Combo box de colores*.

RF-2.2.16 MiLookupOp. Un operador "LookupOP"basado en una función propia.

RF-2.2.17 Componente a componente. Una nueva operación de diseño propia aplicada, esta vez, componente a componente.

RF-2.2.18 Pixel a pixel. Una nueva operación de diseño propia aplicada pixel a pixel.

RF-3 **Funciones sobre sonido.** La aplicación permitirá tanto la reproducción como la grabación de audio. Para ello contará con una lista de reproducción, que tendrá una lista desplegable asociada, de forma que al abrir un nuevo fichero de audio éste se incluiría en dicha lista de reproducción :

RF-3.1 Reproducir sonido/audio. Tendrá asociado un botón que se encargue de iniciar la reproducción del archivo de audio seleccionado en la *Lista de reproducción*.

RF-3.2 Parar reproducción (Stop). Al pulsar el botón asociado, detendrá la reproducción, tanto de audio, como de una grabación.

RF-3.3 Pausa. Podremos pausar la reproducción para reanudarla cuando queramos a través del **RF-3.1**.

RF-4 **Funciones de vídeo.** La aplicación permitirá tanto la reproducción como la captura de vídeo, ya sea de un archivo de nuestro equipo, o de la reproducción desde la *cámara* de nuestro ordenador :

RF-4.1 Cámara :

RF-4.1.1 Reproducir. Tendremos un nuevo botón que nos abrirá un nuevo tipo de ventana, *VentanaInternaCamara*, y por la que veremos lo que se está viendo por la cámara de nuestro equipo en directo.

RF-4.2 Captura. Podremos hacer una captura de pantalla de lo que se esté reproduciendo, ya sea vídeo o la cámara. Por supuesto que esta imagen también podrá ser editada con todas las funcionalidades descritas en el apartado **RF-2.2**.

RF-4.3 Vídeo :

RF-4.3.1 Reproducir. Esta función estará incluida en el mismo botón en el que se reproduce el sonido. Su funcionalidad será la misma, pero con el archivo de vídeo que hayamos abierto en la ventana correspondiente, *VentanaInternaVLCPlayer*.

RF-4.3.2 Detener. Esta función también estará incluida en el mismo botón en el que se detiene el sonido. Su funcionalidad será la misma, pero con el archivo de vídeo que estemos reproduciendo en la ventana asociada al vídeo.

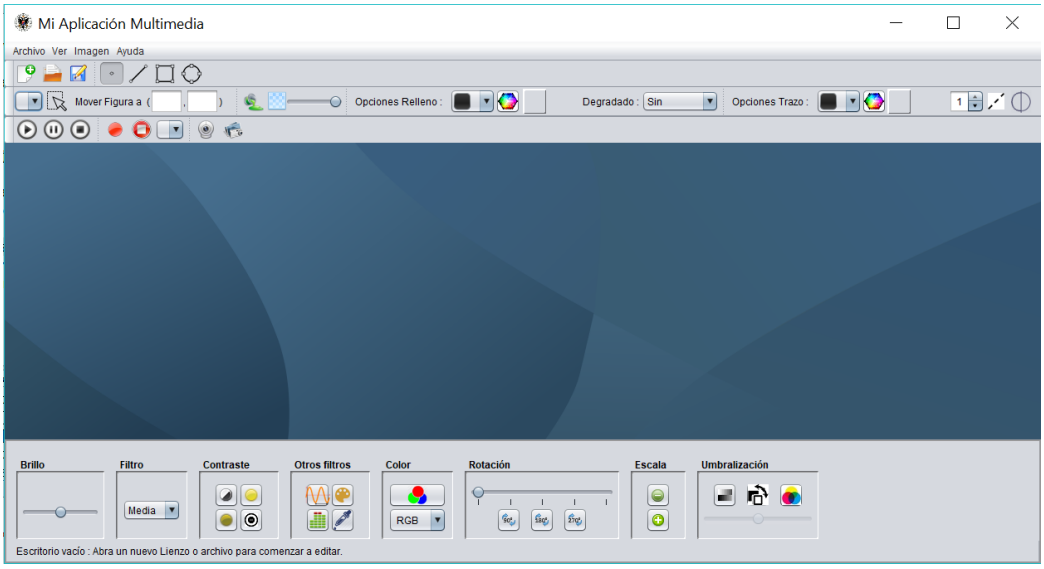
Capítulo 2

Diseño

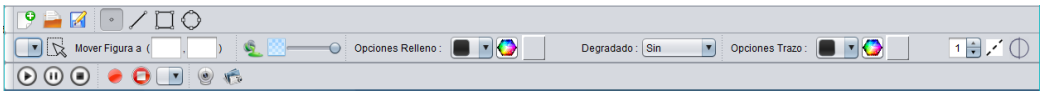
2.1. Interfaz y propósito general

2.1.1. Interfaz

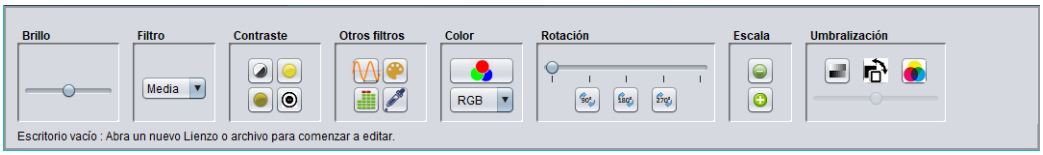
Mi propuesta para solucionar todo lo citado anteriormente consiste en un diseño basado en Jerarquía de clases.
Primero, haré un desglose por Módulos, partiendo por describir el diseño de la *Interfaz gráfica :*



Se trata de una aplicación de Escritorio, definido como **VentanaPrincipalP-Final** en el que están ubicadas todas las barras de herramientas, situadas en la *parte superior* de la ventana,

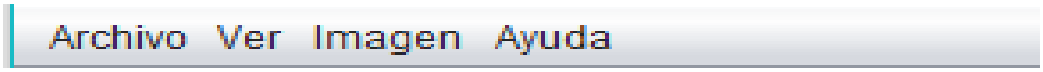


y la barra de propiedades, situada en la *parte inferior* de la ventana.



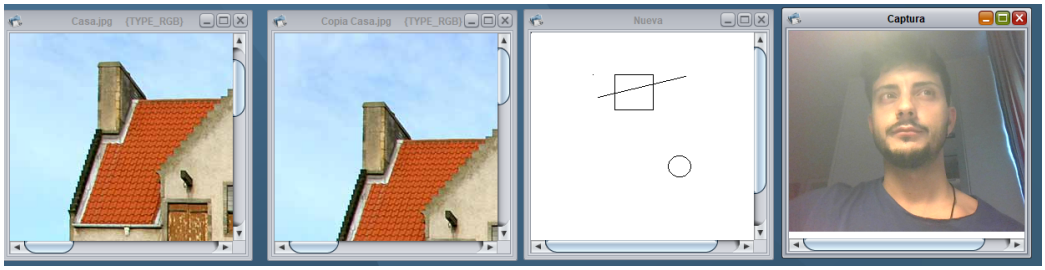
Gracias a las distintas barras de herramientas y a la barra de propiedades, puedo desarrollar las funcionalidades de todos los **Requisitos** del capítulo anterior.

También, y encima de la barra de herramientas, disponemos de un menú de carácter general, como operaciones de archivos, distintas operaciones generales de imágenes y barras de herramientas, y acceso a la información acerca de la creación de la aplicación.

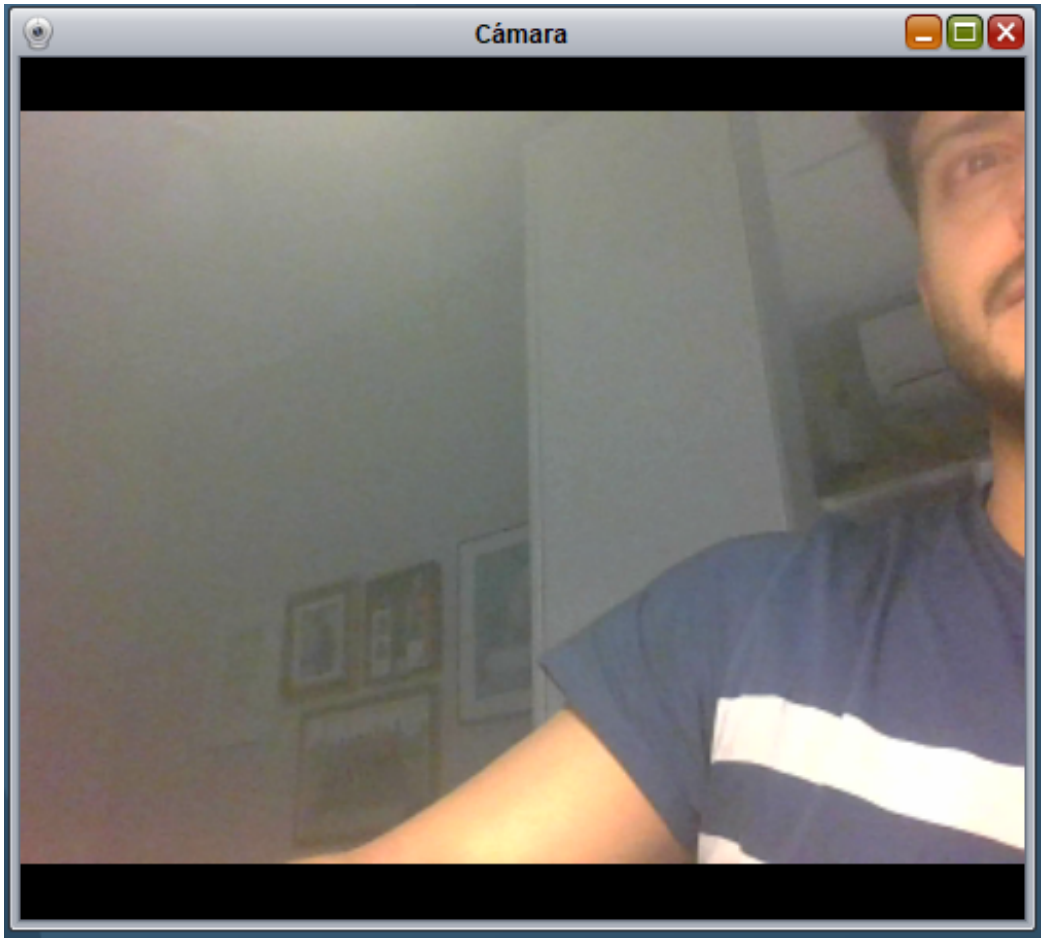


Para esta parte del diseño me he guiado por los consejos dados por el profesor en la clase de prácticas correspondiente, y por el pdf de la práctica final.
Además, el sistema cuenta con otra ventana adicional, múltiple, es decir, que podemos abrir tantas nuevas como queramos, llamada **VentanaInterna**, que será la superclase. Esta superclase podrá ser de 3 tipos :

- ***VentanaInternaImagen.*** La manipulación de estos medios requerirá la incorporación de un tipo específico de ventana interna, al que reconoceré como Tipo 1, capaz de mostrar imágenes, tanto las leídas de un fichero como las creadas nuevas por el usuario o generadas al capturar una imagen instantánea a partir de un vídeo o la cámara. Cada ventana mostrará una única imagen (existiendo, por tanto, tantas ventanas como imágenes estemos tratando), incluyendo barras de desplazamiento en caso de que la imagen sea mayor que la zona visible. El título de la ventana corresponderá al (1) nombre del fichero, si es una imagen abierta o guardada, (2) "nueva", si ha sido creada por el usuario, o (3) "captura", si es una instantánea captada de un vídeo o de la cámara. Además, deberá indicar (entre corchetes) el espacio de color en el que está la imagen; si se tratase de una imagen asociada a una banda, deberá indicarse a qué banda (número o letra) corresponde.

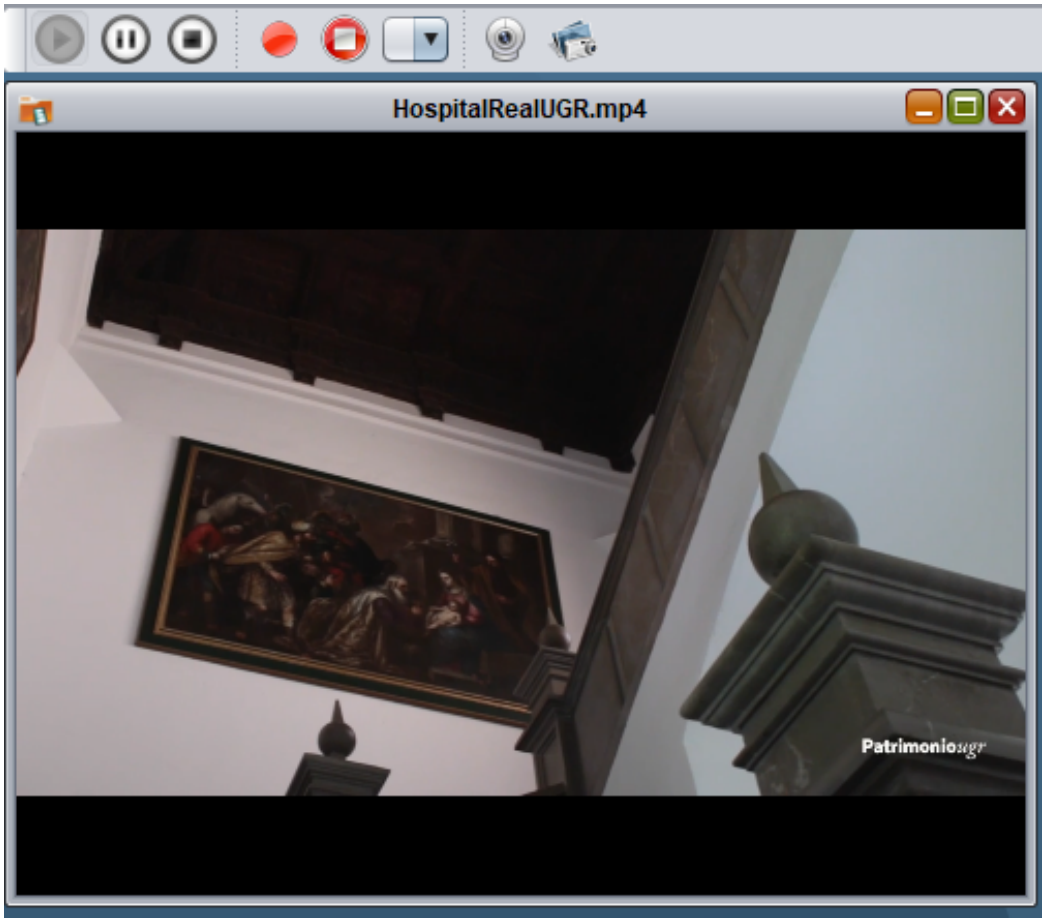


- ***VentanaInternaCamara.*** Le asociaré el Tipo 2. Esta ventana será única y se abrirá exclusivamente para mostrar la secuencia que esté captando la cámara de nuestro equipo.

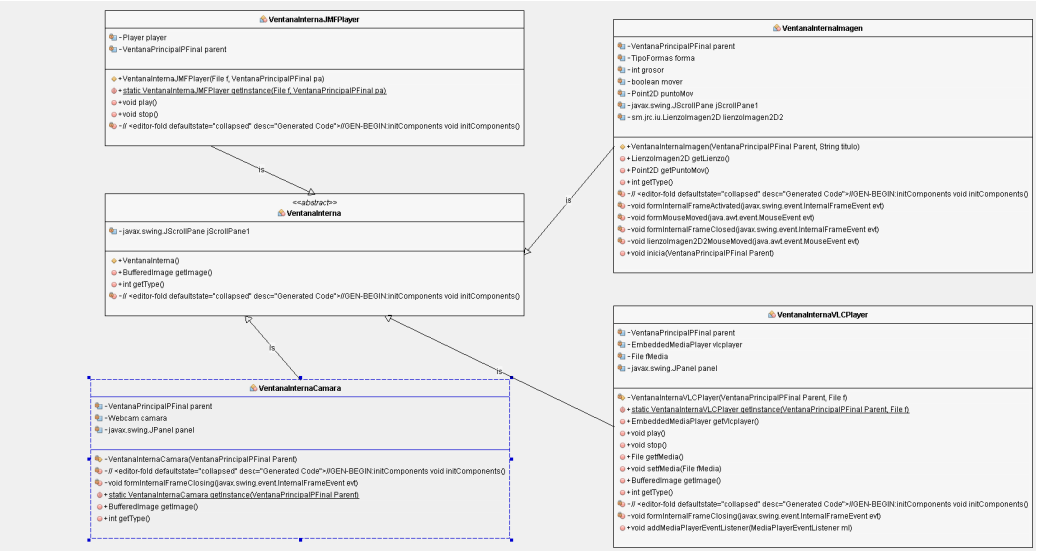


- ***VentanaInternaVLCPlayer.*** A la que le asocio el Tipo 3. La ventana tendrá como título el nombre del fichero que se está reproduciendo, existiendo tantas ventanas como vídeos tengamos abiertos. Dispondrán de una zona de visualización central; además, el usuario podrá controlar la reproducción, con la tercera de las barras de herramientas de la parte superior de la interfaz.

  En el siguiente ejemplo se puede observar que el botón de Play está deshabilitado, ya que se está reproduciendo un vídeo.



A continuación voy a añadir un diagrama en UML del diseño de estas clases :


VentanaInterna.JMFPlayer
- Player player
- VentanaPrincipalPFinal parent
+ VentanaInterna.JMFPlayer(File f, VentanaPrincipalPFinal pa)
+ static VentanaInterna.JMFPlayer getInstance(File f, VentanaPrincipalPFinal pa)
+ void play()
+ void stop()
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
is
<<abstract>>
VentanaInterna
- javax.swing.JScrollPane jScrollPane1
+ VentanaInterna()
+ BufferedImage getImage()
+ int getType()
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
is
is
is
VentanaInternaCamara
- VentanaPrincipalPFinal parent
- Webcam camara
- javax.swing.JPanel panel
- VentanaInternaCamara(VentanaPrincipalPFinal Parent)
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
- void formInternalFrameClosing(javax.swing.event.InternalFrameEvent evt)
+ static VentanaInternaCamara getInstance(VentanaPrincipalPFinal Parent)
+ BufferedImage getImage()
+ int getType()
VentanaInternaImagen
- VentanaPrincipalPFinal parent
- TipoFormas forma
- int grosor
- boolean mover
- Point2D puntoMov
- javax.swing.JScrollPane jScrollPane1
- sm.jrc.iu.LienzoImagen2D lienzoImagen2D2
+ VentanaInternaImagen(VentanaPrincipalPFinal Parent, String titulo)
+ LienzoImagen2D getLienzo()
+ Point2D getPuntoMov()
+ int getType()
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
- void formInternalFrameActivated(javax.swing.event.InternalFrameEvent evt)
- void formMouseMoved(java.awt.event.MouseEvent evt)
- void formInternalFrameClosed(javax.swing.event.InternalFrameEvent evt)
- void lienzoImagen2D2MouseMoved(java.awt.event.MouseEvent evt)
+ void inicia(VentanaPrincipalPFinal Parent)
VentanaInternaVLCPlayer
- VentanaPrincipalPFinal parent
- EmbeddedMediaPlayer vlcplayer
- File fMedia
- javax.swing.JPanel panel
- VentanaInternaVLCPlayer(VentanaPrincipalPFinal Parent, File f)
+ static VentanaInternaVLCPlayer getInstance(VentanaPrincipalPFinal Parent, File f)
+ EmbeddedMediaPlayer getVlcplayer()
+ void play()
+ void stop()
+ File getMedia()
+ void setMedia(File fMedia)
+ BufferedImage getImage()
+ int getType()
- // <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents void initComponents()
- void formInternalFrameClosing(javax.swing.event.InternalFrameEvent evt)
+ void addMediaPlayerEventListener(MediaPlayerEventListener ml)

2.1.2. Funcionalidades de carácter general

- Nuevo. La funcionalidad de este botón consiste en abrir una nueva *VentanaInternaImagen* indicando, primero, la anchura y altura que queremos asignarle a la ventana (por defecto se asignará las dimensiones 500x300) a través de un diálogo, inicializando la imagen BufferedImage asociada a la de por defecto, y deshabilitando los botones de mover y los cuadros de diálogo de las coordenadas para cambiar la posición de la figura.

- Abrir. Implementada en un único botón para cualquier tipo de archivo. Para ello, le añado unos filtros que mostrarán el tipo de archivo que hayamos elegido en dicho filtro (), abrimos un cuadro de diálogo, para que se puedan seleccionar los archivos de nuestro equipo que estén dentro del conjunto de archivos seleccionables según el filtro de la extensión (Bibliografía, enlace 2).

  Para obtener la extensión de los archivos, implemento en el módulo VentanaPrincipal, el método ***getExtension()****.*

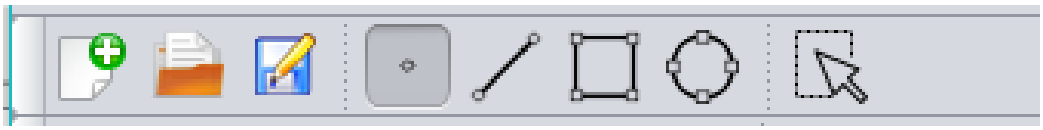
  Una vez sepamos con qué tipo de archivo vamos a trabajar, se le asigna y abre el tipo de ventana Interna correspondiente o se añade a la lista de reproducción en caso de ser un archivo de audio.

- Guardar. A partir de la extensión que le haya asignado al archivo de imagen, lo guardo con una de las extensiones reconocidas para imágenes, en caso de no ser así, se guarda sin extensión.

  Para finalizar, se actualiza el nombre de la ventana con el nombre del archivo guardado.

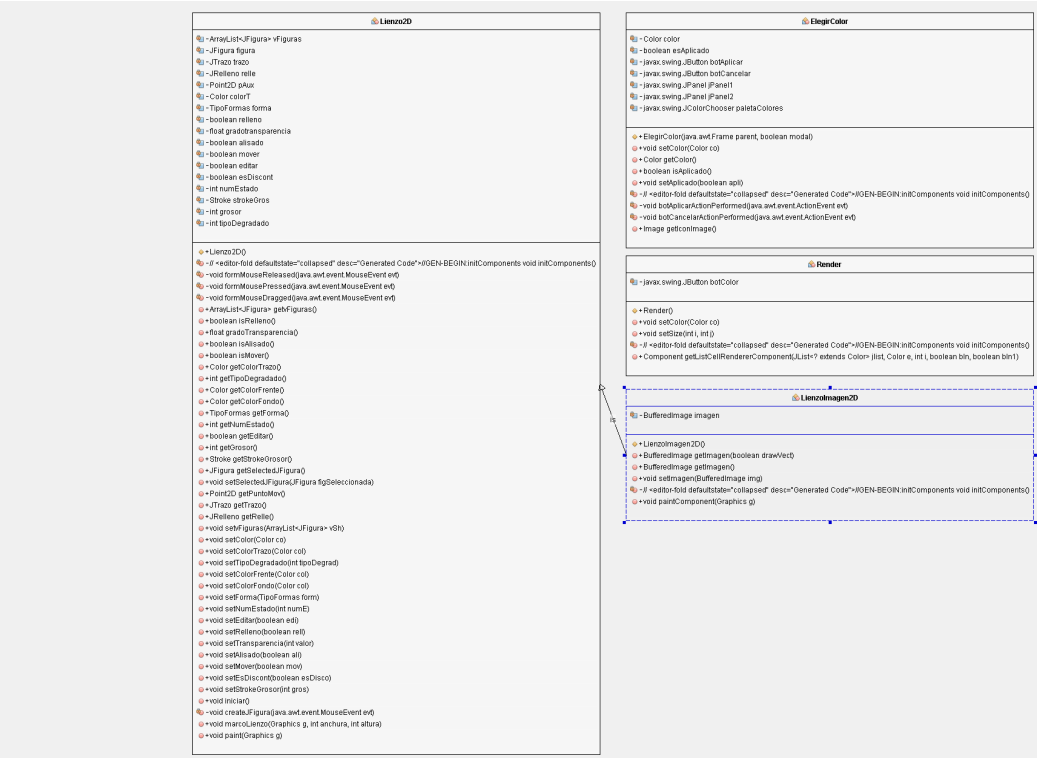
  Para guardar un sonido producido por la grabadora de voz, hago que esta ventana de diálogo me salga cuando pulso el botón **STOP GRABAR**. PAra que esto sea posible, el sonido grabado se almacena en un fichero temporal (Bibliografía, enlace 3), el cuál se renombrará al que se le asigne a través del cuadro de diálogo correspondiente a esta sección.

Todo esto, además de los botones para seleccionar el tipo de forma que tiene la figura que vamos a dibujar, y el de mover figura, los podemos encontrar en la primera barra superior de herramientas.



2.2. Módulo de Gráficos : Figuras

Los requisitos sobre este apartado vienen a resumirse en que la aplicación permitirá dibujar diferentes formas geométricas (líneas, rectángulos, etc.), con diferentes atributos (color, grosor, etc.) sobre una clase propia, a la que hemos llamado ***Lienzo2D***, y sobre otra clase propia, hija de la clase Lienzo, de la que hereda todas las funcionalidades de la propia, llamada ***Lienzo2DImagen***, que contiene además una imagen. El siguiente diagrama de UML nos describe un poco mejor este diseño :



Para esto, hasta esta entrega final nos ha bastado con usar la clase ***Graphics***, incorporándole, eso si, distintas funcionalidades que íbamos necesitando y que ésta no nos podía proporcionar, sobretodo a la hora de pintar las figuras. La principal mejora de esta entrega es que podemos editar cada figura con sus **propios atributos independientes**.

Como solución a este tema, ya que es uno de los requisitos que se indican en el pdf de la convocatoria, he desarrollado una estructura jerárquica de clases :

- **JFigura.** Todo lo que se pinta sobre un Lienzo o imagen es una figura. De ahí que sea la superclase abstracta, de la que heredarán el resto de subclases atributos comunes a todo objeto del mismo tipo, como son los que engloban al *trazo*, subclase de propiedad de la que hablaré un poco más adelante, los *puntos inicial* y *final*, y funcionalidades como la *ubicación* o *pintarla en el Lienzo*.

  Los datos miembro son de tipo protegido, para que solamente las subclases puedan tener acceso a ellos.
  Y los métodos que sean comunes, pero con características diferentes para cada tipo de figura, serán abstractos, para sobrecargarlos donde corresponda y así poder redefinir con más exactitud las distintas funcionalidades de cada objeto.

  Todas las figuras tienen una serie de atributos comunes, a los que englobo en una misma superclase llamada **JAtributos** que dispone de:

  - *gradoTransparencia : int*
  - *composicion : Composite*
  - *esRellena : boolean*

  Además de estos, también hay 2 grupos de atributos, unos correspondientes al diseño del trazo, los cuales he agrupado en una subclase que hereda de JAtributos, llamada **JTrazo**, y que consta de las siguientes propiedades :

  - *color : Color*
  - *grosor : int*
  - *esDiscontinuo : boolean*
  - *strokeGros : Stroke*
  - *esAlisado : boolean*
  - *render : RenderingHints*
  - *patronTrazo : float[]*

  , y otros correspondientes al diseño del relleno de la Figura, que he agrupado en una subclase la cuál hereda de JAtributos, llamada **JRelleno**, y que consta de las siguientes propiedades :

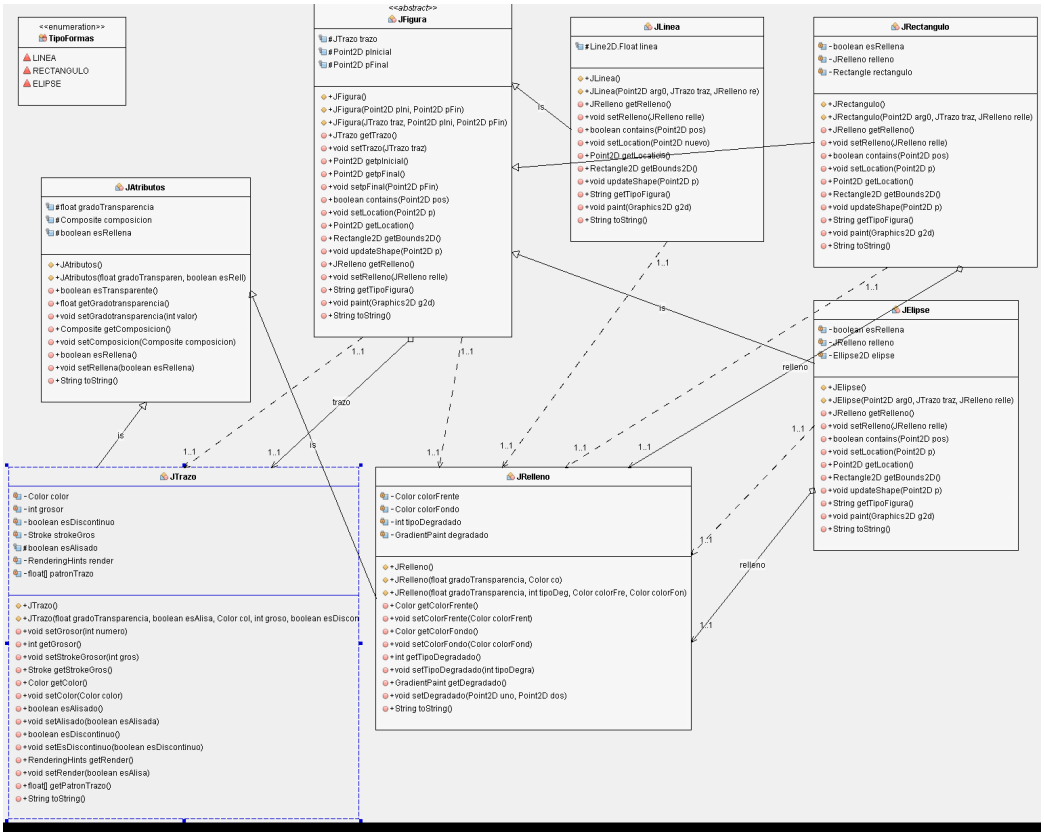
  - *colorFrente : Color*
  - *colorFondo : Color*

- *tipoDegradado : int*
- *degradado : GradientPaint.* (Bibliografía, enlace 1).

De la superclase **JFigura** heredarán las siguientes subclases asociadas a las distintas formas geométricas que he creado :

- ***JLinea***
- ***JRectangulo***
- ***JElipse***

Para facilitar la comprensión de las relaciones entre las distintas clases y subclases, tanto del proyecto principal, como de mi biblioteca propia, podemos observar los siguientes diagramas en lenguaje **UML** :



***NOTA : Para entrar en un mayor detalle acerca de las variables y métodos incluidos en cada clase, le recomiendo que lea el siguiente capítulo de este documento, ya que está relacionado con la implementación.**
En este apartado, podrá resolver sus dudas acerca de dónde he obtenido la distinta documentación para resolver los requisitos propuestos, así como una mejor explicación del funcionamiento de cada funcionalidad.

2.3. Módulo de procesamiento de Imágenes

Además de la barra de herramientas superior, ya mencionada anteriormente, disponemos de diferentes botones, deslizadores, y combobox, situados en la parte inferior de la VentanaPrincipal, en la barra de propiedades que también he mencionado más arriba, para realizar todas las diferentes operaciones para abordar esta sección. Las operaciones son aquellas que aparecen descritas en el capítulo anterior, en la parte relacionada con las imágenes **RF-2.2**.

Para el mejor entendimiento de algunas de ellas, puede leer las siguientes explicaciones de funcionamiento :

- Extraccion de bandas. Aparecerán tantas *VentanaInternaImagen* como bandas tenga la imagen; cada una de estas ventanas contendrá una imagen (en niveles de gris) con la correspondiente banda.

- Ecualización. Se ecualizarán todas las bandas, pero primero se pasará al espacio de colores *YCC*, que separa la intensidad de la información cromática.

- Sepia. Para ello previamente he creado mi propia clase operador *SepiaOp*, y en su método filter, guiándome con Bibliografía, enlace 6, he realizado los siguientes pasos (que además están explicados en el código) :

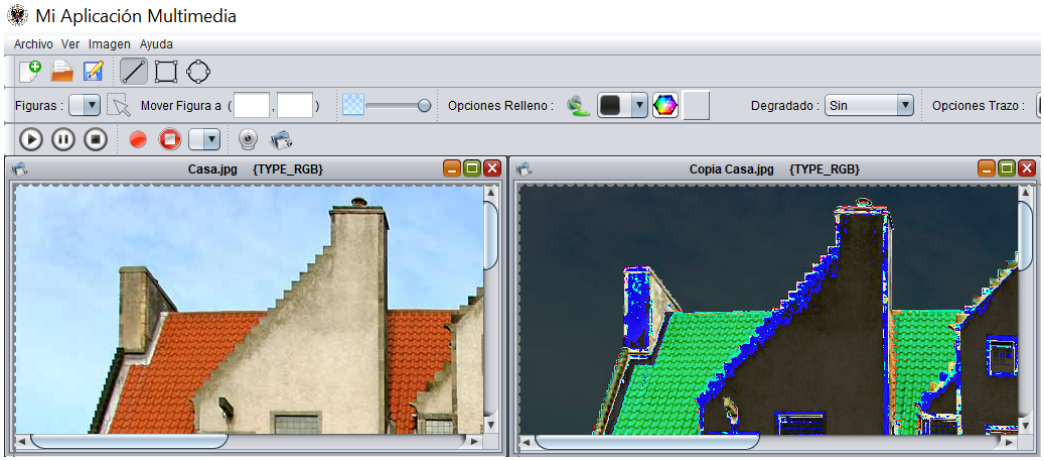
  1. Obtener la cuadrícula de las imágenes fuente y destino.
  2. Recorrer la imagen fuente píxel a píxel con un doble for, y dentro hacemos lo siguiente :
     a) Obtener el píxel en el que nos encontramos.
     b) Se calcula el mínimo del valor de la suma de cada una de las componentes del píxel multiplicadas por un coeficiente.
     c) Y se reemplaza este valor, en lugar del que tenía el píxel anteriormente. No tenemos que cambiar el valor alfa porque solo controla la transparencia del píxel.

- Umbralización. Este operador genera como resultado una imagen binaria donde los píxeles que superan un umbral (obteniendo el valor de su correspondiente deslizador) se le asigna dicho valor, y al resto 0. Igual que en el caso de Sepia, también he creado previamente mi clase operador *UmbralizacionOp*.

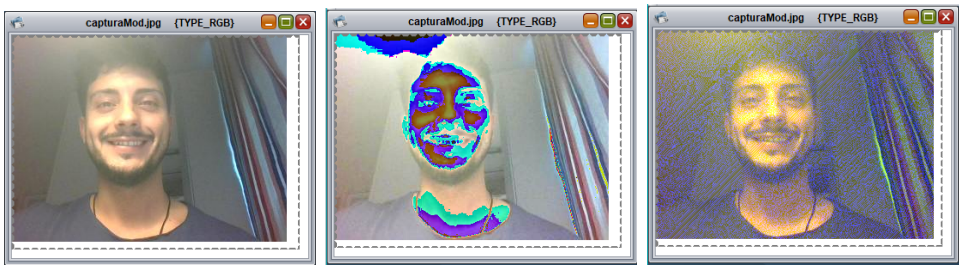
- Umbralización inversa. Igual que la normal, solo que invirtiendo los valores del máximo y mínimo de cada píxel. Se puede entender mejor el funcionamiento observando la documentación de tipo Javadoc, generada para la clase *UmbralizacionOp*, en la que está incluida esta operación.

- Umbralización basada en color. Para ello, dentro de la función *filter* de mi clase *UmbralizacionOp*, a la hora de seleccionar qué valor usar en función del umbral, en lugar del mínimo, utilizaremos el valor Red, Green y Blue del color que hayamos seleccionado. De esta manera, lo que en el operador normal se ve de color negro, ahora lo veremos del color seleccionado.
  (Para hacer la umbralización inversa basada en color, sería igual que este caso, pero invirtiendo los valores del mínimo y máximo).

- MiLookupOp. Muy parecida a la operación Sinusoidal, solo que la función propia elegida es en lugar de aplicarle el Seno al producto del factor por el bit, le aplico la función *arcotangente*, creando la función propia *miLookupOp* que se encarga de realizar la operación matemática.



A continuación muestro ejemplos de mis operaciones propias :



(a) Original (b) Comp a Comp (c) Píxel a Píxel

Figura 2.1: Operadores clases propias

- Operación componente a componente. Para este caso, he tenido que crearme la clase MiOpCC, y en su método Filter, realizo la operación que consiste en que se multiplica el valor de cada banda por el valor absoluto de la arcotangente del valor de cada banda. : Band = Band * abs(arc.tg(Band))

- Operación píxel a píxel. Para realizarlo, también me he tenido que crear una clase propia, llamada MiOpPP, y realizar la operación matemática de cada píxel en su método sobreescrito filter. Dicha operación consiste asignarle a la componente blue de cada píxel el valor de la propia componente blue de cada píxel por la suma de las coordenadas del propio píxel : B = B * (x+y)

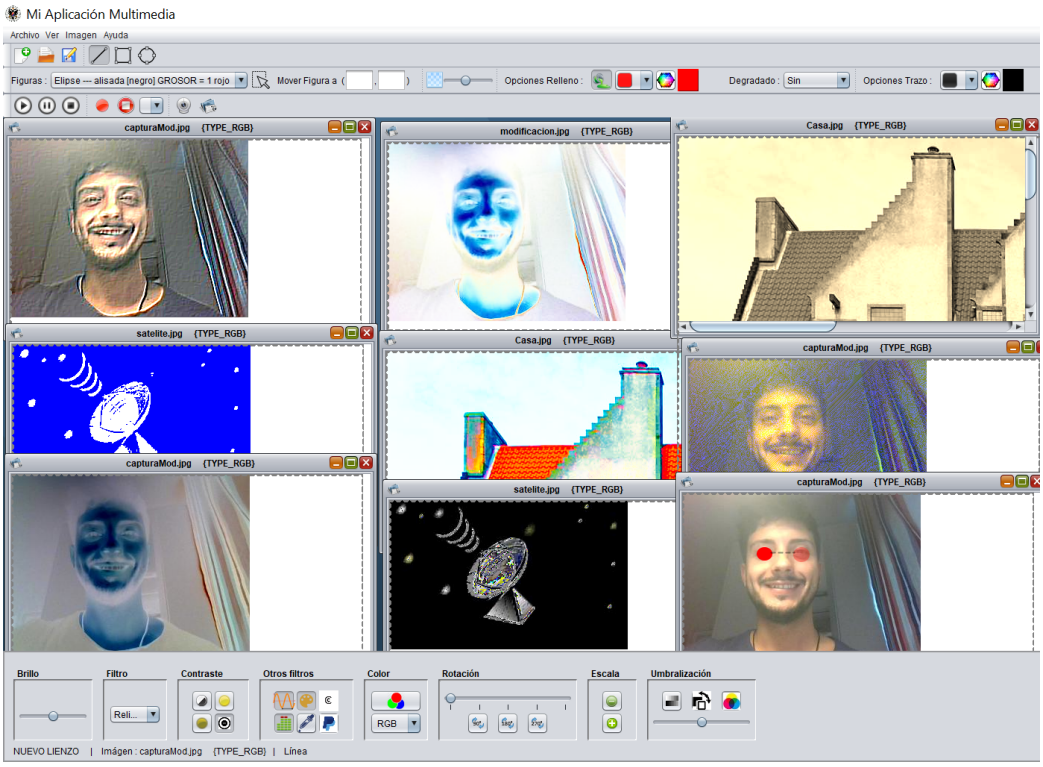
Capítulo 3

Manual de uso y ejemplos

3.1. Manual

Para empezar a usar la aplicación, primero debe de seleccionar, arriba a la izquierda, tanto en la barra de menú, como en los correspondientes botones, si crear un nuevo Lienzo (en cuyo caso se abrirá una nueva VentanaInterna de tipo Imagen, preguntando por la dimensión de dicha ventana), o si abrir algún tipo determinado de archivo, que puede ser una imagen, un vídeo o un sonido, todos en los formatos que reconoce Java.

Una vez que ya hayamos seleccionado el fichero podremos proceder o bien a la edición de imágenes, dando como resultado alguno de estos ejemplos :



Para el caso de haber abierto un fichero de sonido, éste se nos añade a la lista de reproducción que hay en la barra superior correspondiente al sonido y vídeo. Además estaría seleccionado y listo para ser reproducido. Los botones de Play y Stop distinguen automáticamente si estamos reproduciendo un vídeo o un sonido, sin ningún tipo de problema. El botón de Pausa funciona exclusivamente si estamos reproduciendo un archivo de sonido.
En el caso de una grabación de sonido, lo único que hay que hacer es pulsar el botón rojo para que de comienzo la grabación. Cuando queramos parar, pulsaremos el botón de Stop en rojo y, seguidamente seleccionaremos dónde queremos guardar nuestra grabación gracias a la ventana con el diálogo para guardar.

Si lo que queremos reproducir es un archivo con formato soportado por VLC-Player, **además de tener instalada la versión del reproductor VLC de arquitectura igual a la de nuestro equipo**, basta con abrir un archivo de este tipo, y pulsar el botón del Play. Para pararlo, pulsaremos Stop y el vídeo se volverá a poner listo para reproducir desde el inicio.

Bibliografía

[1] DEGRADADO, *Enlace de la web donde he obtenido la técnica* http://swing-facil.blogspot.com/2012/03/gradientpaint-demo-ejemplos-del-uso-de.html

[2] FILTROS DE BÚSQUEDA, *Enlace de la web donde he obtenido la forma de cómo filtrar para que en los diálogos de abrir y guardar sólo aparezcan los tipo de ficheros que correspondan* https://www.programcreek.com/java-api-examples/javax.swing.filechooser.FileNameExtensionFilter

[3] ARCHIVOS TEMPORALES, *Web donde he encontrado cómo se crea un archivo temporal* http://lineadecodigo.com/java/crear-un-fichero-temporal-con-java/

[4] PASAR STRING A INT, *Web donde he encontrado cómo hacer el casting de un string a un int* https://www.mkyong.com/java/java-convert-string-to-int/

[5] CAMBIAR LOGO PRINCIPAL, *Web donde he encontrado cómo cambiar el logo de la Ventana Principal* http://wiki.netbeans.org/TaT_CambiarIconoJFrame

[6] FILTRO SEPIA, *Página web donde encontré cómo implementar el método filter de la clase SepiaOp* https://www.dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-sepia-image

[7] FUNCIÓN SENO, *PDF Práctica 10, página 2. Usado como guía para crear mi propia función LookupOp*

[8] DECSAI, *Página web del departamento de Ciencias de la computación y de la asignatura* https://decsai.ugr.es