

A dark blue vertical bar on the left side of the page. A blue arrow points to the right from the bar, containing the date.

12-7-2016

# Paint multimedia

Sistemas multimedia

Several thin, curved lines in dark blue and light grey originate from the bottom left and curve upwards and to the right.

Antonio Alcalá Martínez

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍAS INFORMÁTICA Y  
DE TELECOMUNICACIÓN

<b>1. Introducción</b>	<b>2</b>
1.1 Descripción del problema	2
1.2 Objetivos	2
<b>2. Requisitos</b>	<b>2</b>
2.1 Requisitos funcionales	2
2.2 Requisitos no funcionales	3
<b>3. Análisis</b>	<b>4</b>
3.1 Interfaz de usuario (RF-1)	4
3.2 Gestión Imagen	5
3.2.1 Atributos (RF-2.3 y RF-2.4)	5
3.2.1.1 Clase atributo	5
3.2.1.2 Clases hijas	6
3.2.2 Formas(RF-2.2)	7
3.2.2.1 Clase Forma	7
3.2.2.2 Clases hijas	8
3.2.3 Observaciones atributos y formas	10
3.2.4 Operaciones imagen (RF-2.9)	10
3.2.5 Redimensionar y reescalar imagen (RF-2.5 RF-2.6)	14
3.2.6 Duplicar (RF-2.7)	14
3.2.7 Nueva Imagen(RF-2.1)	14
3.2.8 Editar formas (RF-2.8) y borrar atributos (RF-2.4)	15
3.2.9 Dibujar(RF-2.10)	16
3.3 Gestión audio	18
3.3.1 Reproducción audio (RF-3.1)	18
3.3.2 Grabación de audio (RF-3.2)	19
3.4 Gestión video	19
3.5 Gestión webcam	20
3.5.1 Activar webcam(RF-5.1)	20
3.5.2 Capturar imagen(RF-5.2)	20
<b>4. Diseño</b>	<b>21</b>
4.1 Paint	21
4.2 Menú principal	22
4.3 Ventana Interna sonido	22
<b>5. Bibliografía</b>	<b>23</b>

## 1. Introducción

### 1.1 Descripción del problema

El objetivo es crear un programa de tratamiento de multimedia, donde podrás dibujar, reproducir audio, reproducir video y podrás hacer capturas de imágenes con la webcam.

### 1.2 Objetivos

Los objetivos principales que debe cumplir el software que se pretende desarrollar, en cierto modo, se puede pensar en ellos como requisitos de alto nivel, son los siguientes:

- OBJ-1. El programa permitirá que el usuario pueda dibujar las formas preestablecidas y además podrá jugar con su relleno, su forma y podrá editarla.
- OBJ-2. El programa permitirá que el usuario pueda realizar transformaciones a una imagen.
- OBJ-3. El programa permitirá que el usuario pueda reproducir audio y video.
- OBJ-4. El programa permitirá que el usuario pueda tomar capturas desde la webcam y poder manipular la captura.

## 2. Requisitos

### 2.1 Requisitos funcionales

- **RF-1. Gestión menú**  
El sistema generará un interfaz gráfico con las opciones disponibles y menús disponibles.
  - **RF-1.1. Abrir archivo**  
El sistema permitirá abrir solo aquellos archivos que soporta.
  - **RF-1.2. Guardar archivo**  
El sistema nos permitirá guardar en un archivo el trabajo realizado.
- **RF-2. Gestión imágenes**
  - **RF-2.1. Nuevo lienzo**  
El sistema crea un lienzo para poder dibujar las formas.
  - **RF-2.2. Elegir forma**  
El sistema te deja elegir entre un conjunto de formas, que podrás utilizar para dibujar.
  - **RF-2.3. Añadir atributos a la forma**

El sistema permite añadirle atributos a las formas dibujadas y a las futuras formas que dibujes.

➤ **RF-2.4. Borrar atributos de la forma**

El sistema permite la eliminación de atributos ya asignados a una forma.

➤ **RF-2.5. Redimensionar lienzo**

El sistema permite ensanchar o disminuir el lienzo.

➤ **RF-2.6. Reescalar imagen**

El sistema permite achicar o agrandar una imagen.

➤ **RF-2.7. Duplicar imagen**

El sistema permite hacer una copia exacta de la imagen.

➤ **RF-2.8. Editar forma**

El sistema permite seleccionar una forma y moverla de su posición e incluso cambiarle los atributos.

➤ **RF-2.9. Modificar imagen**

El sistema tiene implementado un conjunto de opciones que permite transformar una imagen, como por ejemplo el negativo el brillo, la umbralización, etc.

➤ **RF-2.10. Dibujar**

El sistema te permite dibujar cualquier forma con los atributos que tu desees.

• **RF-3. Gestión audio**

➤ **RF-3.1. Reproducir audio**

El sistema nos permite reproducir audio, pero solo los formatos wav, au y aif.

➤ **RF-3.2. Grabar audio**

El sistema nos permite a través de un micrófono grabar sonidos.

• **RF-4. Gestión video**

➤ **RF-4.1. Reproducir video**

El sistema nos permite reproducir videos.

• **RF-5. Gestión webcam**

➤ **RF-5.1. Activar webcam**

El sistema nos permite activar una webcam.

➤ **RF-5.2. Capturar imagen**

El sistema nos permite tomar una foto con la webcam y sobre ella poder dibujar y modificarla.

## 2.2 Requisitos no funcionales

• **Facilidad de uso:**

- ✓ **RNF-1.** Es necesario un mínimo de conocimientos previos de informática para manejar la plataforma.

- ✓ **RNF-2.** Se adjuntará una pequeña documentación explicando la funcionalidad de cada operación.
- ✓ **RNF-3.** El interfaz será lo más intuitiva posible.
- **Fiabilidad:**
  - ✓ **RNF-3.** El sistema debe estar libre de fallos.
  - ✓ **RNF-4.** Cuando ocurra una excepción, se reportará a los desarrolladores inmediatamente.
- **Rendimiento:**
  - ✓ **RNF-5.** El sistema debe de ser veloz y con los mínimos tiempos de carga cuando ejecutas una transformación.
- **Soporte:**
  - ✓ **RNF-6.** El sistema será mantenido por el equipo de desarrollo del mismo.
  - ✓ . Habrá portabilidad para diferentes entornos software (Windows, Linux, etc.).
- **Implementación:**
  - ✓ **RNF-7.** Nuestro sistema deberá funcionar tanto en cualquier sistema operativo.
- **Interfaz:**
  - ✓ **RNF-8.** La información será almacenada en memoria de forma temporal, en el caso de que se desee almacenar la información de forma permanente se harán por medio de archivos, guardados en el disco local.
- **Empaquetamiento:**
  - ✓ **RNF-9.** Existirá la opción de instalar por parte del usuario una aplicación totalmente gratis.
- **Legales:**
  - ✓ **RNF-10.** El software tendrá su copyright y será propiedad de la empresa.

### 3. Análisis

#### 3.1 Interfaz de usuario (RF-1)

El interfaz de usuario gira entorno a una clase principal donde estarán todos los iconos y opciones que nos permite realizar el programa. La capa superficial esta implementada en un JFrame que es nuestra clase Paint. Para poder implementar algunas opciones, hemos tenido que crear clases independientes, es el caso de dibujar, de tratamiento de imágenes, de reproducir audio, grabar audio, de reproducir video y de manejo de webcam.

He creado una clase padre llamada VentanaInterna, de la cual heredan un conjunto de clases, como por ejemplo una clase para el tratamiento de imágenes y de dibujo a esa clase la he denominado VentanaInternaImagen, otro ejemplo sería una clase para reproducir video la cual he nombrado VentanaInternaReproduccionVideo

y un último ejemplo sería una clase para reproducir y grabar audio que he denominado `VentanaInternaSonido`. Un aspecto importante de `VentanaInterna` es que tiene un método para saber qué clase hija estamos seleccionando. Tanto la `VentanaInterna`, como las clases que heredan de ella, son de tipo `JInternalFrame`.

Esta implementación nos facilita el camino a la hora de abrir un archivo(RF-1.1), ya que solo tenemos que crear una clase de tipo `VentanaInterna` y según la extensión del archivo hacer el new a la clase hija correspondiente.

En cuanto al grabar archivo(RF-1.2), como solo permitimos grabar imágenes, lo único que tenemos que comprobar es que la `VentanaInterna` sea de tipo imagen y si es así, obtener el `bufferedImage` del lienzo, una vez hecho esto el sistema solicitará el nombre del archivo o que selecciones un archivo, una vez creado el archivo y le asignaremos la extensión.

En la clase `Paint`, implementamos la funcionalidad de todos los botones, barras, menús y slider que la componen.


## 3.2 Gestión Imagen

### 3.2.1 Atributos (RF-2.3 y RF-2.4)

#### 3.2.1.1 Clase atributo

Mi idea original fue crear una clase atributo donde tendríamos diferentes constructores, para según si quisiéramos dar un valor a un atributo u otro, pero vi que me salía una clase muy compleja y de difícil uso. Luego después de resolver unas dudas con el profesor, me di cuenta del error de mi planteamiento, siguiendo el consejo del profesor, pensé de forma más general y entonces es cuando llegue a la conclusión de que mi diseño tendría que tener una clase padre, la cual tendría unas variables generales, como es `RenderingHints` y `Composite`.

La clase `Atributo` es una clase simple donde tendré pocos métodos, tendré un método `aplicar` de tipo abstracto, este método sería el encargado de asignar los atributos y de pintarlos, pero luego me di cuenta de que eso ya lo hacía mi método `draw` de la clase `Forma`, así que al final sólo aplico los atributos. Además, tendré unos `get` y `set` para los atributos generales.

 <b>Atributo</b>
<i>Attributes</i> private RenderingHints render private Composite comp
<i>Operations</i> public void aplicar( Graphics2D g2d ) public void borrarRenderizacion( ) public void borrarTransparencia( ) public RenderingHints getRenderizacion( ) public void setRenderizacion( RenderingHints render ) public Composite getTransparencia( ) public void setTransparencia( Composite comp ) public Object getClaseAtributo( )

### 3.2.1.2 Clases hijas

#### 3.2.1.2.1 Borde

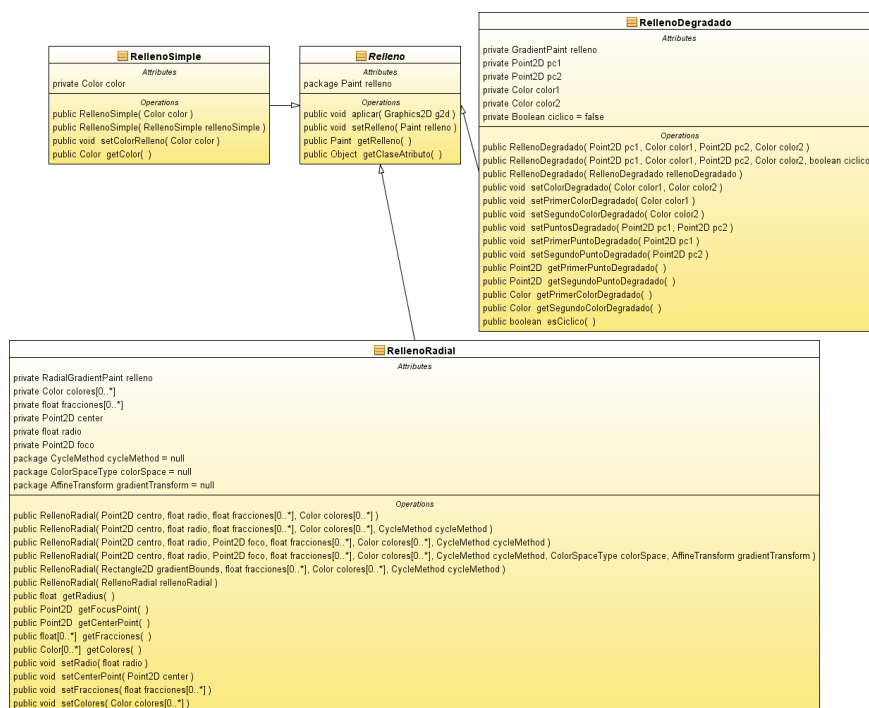
No estaba muy seguro cuales serían mis atributos, después de meditarlo llegué a la conclusión de que todas las figuras tienen un borde y pueden o no tener un relleno. Así que me decante por tener estas dos clases hijas. En cuanto al borde, tenía claro que estaría compuesta por un stroke, así que declaré constructores que permitieran todas las opciones del stroke. El problema del stroke son sus constructores que son poco eficientes y te obliga a meter parámetros que a ti ni te interesan, por eso yo he creado constructores que te permita meter los parámetros que tú desees, para ello establezco unos valores por defecto para los parámetros que no han sido rellenos. Además, he creado métodos que permiten alterar los parámetros del borde sin que tu tengas que construir un nuevo objeto, aunque internamente sí que creas un objeto nuevo. Lo complejo de este planteamiento es que necesito un sistema de variables que guarde todos los valores de los componentes del stroke.

Cada borde tendrá asignado su propio color.

 <b>Borde</b>
<i>Attributes</i> private Stroke trazo private float grosor private int estiloFinalLinea private int estiloUnionLineas private float corte private float patronDiscontinuidad[0..*] private float comienzoDiscontinuidad private Color color private int Unnamed
<i>Operations</i> public Borde( ) public Borde( float grosor ) public Borde( float grosor, int estiloFinalLinea ) public Borde( float grosor, int estiloFinalLinea, int estiloUnionLineas ) public Borde( float grosor, int estiloFinalLinea, int estiloUnionLineas, float corte, float patronDiscontinuidad[0..*] ) public Borde( float grosor, int estiloFinalLinea, int estiloUnionLineas, float patronDiscontinuidad[0..*] ) public Borde( float grosor, int estiloFinalLinea, int estiloUnionLineas, float corte, float patronDiscontinuidad[0..*], float comienzoDiscontinuidad ) public Borde( Borde borde ) public void setGrosor( float grosor ) public void setEstiloFinalLinea( int estiloFinalLinea ) public void setEstiloUnionLineas( int estiloUnionLineas ) public void setCorte( float corte ) public void setPatronDiscontinuidad( float patronDiscontinuidad[0..*] ) public void setComienzoDiscontinuidad( float comienzoDiscontinuidad ) public float getGrosor( ) public int getEstiloFinalLinea( ) public int getEstiloUnionLineas( ) public float getCorte( ) public float[0..*] getPatronDiscontinuidad( ) public float getComienzoDiscontinuidad( ) public Color getColor( ) public void setColor( Color color ) public void Unnamed( )
<i>Operations Redefined From Atributo</i> public void aplicar( Graphics2D g2d ) public Object getClaseAtributo( )

### 3.2.1.2.2 Relleno

El caso de relleno es un poco peculiar ya que, en un principio, solo tenía una clase que era relleno, en esta clase tenía el degradado y el relleno simple. Pero a la hora de añadir el degradado circular me di cuenta de que podía crear una clase padre que agrupará los métodos generales de los rellenos y degradados. Tanto rellenos y degradados necesitan una variable Paint, una variable Forma y un método aplicar, con solo declararlos en la clase relleno ya podía compartírllos. De esta forma surgió las distintas clases de rellenos, cada clase tienen sus constructores y sus métodos específicos. He intentado que todas las clases sean lo más funcionales posibles, para ello he implementado métodos que alteren los parámetros del relleno o degradado.



## 3.2.2 Formas(RF-2.2)

### 3.2.2.1 Clase Forma

Debo decir para ser honesto que, en este proceso, he recibido la ayuda inestimable del profesor. En un principio tenía claro que tendría una clase padre, de la cual heredarían las figuras que deseaba implementar. No tenía claro si mi clase sería abstracta o sería un interfaz, así que en un principio la hice abstracta.

Al principio pensé en hacerlo a lo bruto, así que mi primera opción fue fijarme en la clase shape y copiar sus métodos, ojo sin heredar de la clase shape, a lo bruto. Luego las clases hijo como rectángulo, línea, etc. las implemente como si fuera una clase nueva, por ejemplo, la clase línea declaré dos puntos como variables e hice



su constructor, además me fijé en la clase Line2D y copié sus métodos. Los métodos que copié los implementé a pelo. Estaba sumamente perdido y equivocado.

Después de ir a un par de clases y preguntar dudas ya el profesor me encaminó, entonces es cuando decidí que mi clase padre se llamaría Forma y heredaría de Shape. Además, también me fijé que entre las clases hijas había métodos en común, por lo cual me dejaba claro que sería una clase abstracta y además tendría métodos implementados. Los métodos que tienen el mismo código en todas sus formas, lo implemento en la clase padre.

Además, me dí cuenta que hacían falta métodos que no estaban en la clase shape o en las clases hijas y eran comunes a todas las formas. Uno de esos métodos es el setLocation ya que no todas las formas tenían un método para situarse en un punto distinto del de creación, otro método importante era el setGeometry, ya que con este método podríamos dándole un solo punto establecer una figura en el lienzo.

Otros métodos que vi importantes eran los de devolver los dos puntos principales de la forma, para ello cree el getP1 y el getP2, sumamente útil a la hora de establecer rellenos con degradados.

El método más importante y que es común para casi todas las formas es el método dibujar o draw, como lo tenía yo al principio sólo pintaba con un draw y no tenía en cuenta si había rellenos o no, luego me di cuenta de mi error y en función del tipo de Atributo ya hacia draw o fill.

### 3.2.2.2 Clases hijas

Para facilitar la herencia cree una variable shape llamada geometría, esto me permite redefinir en cada clase hija la variable geometría y poder declararla del tipo que desee.

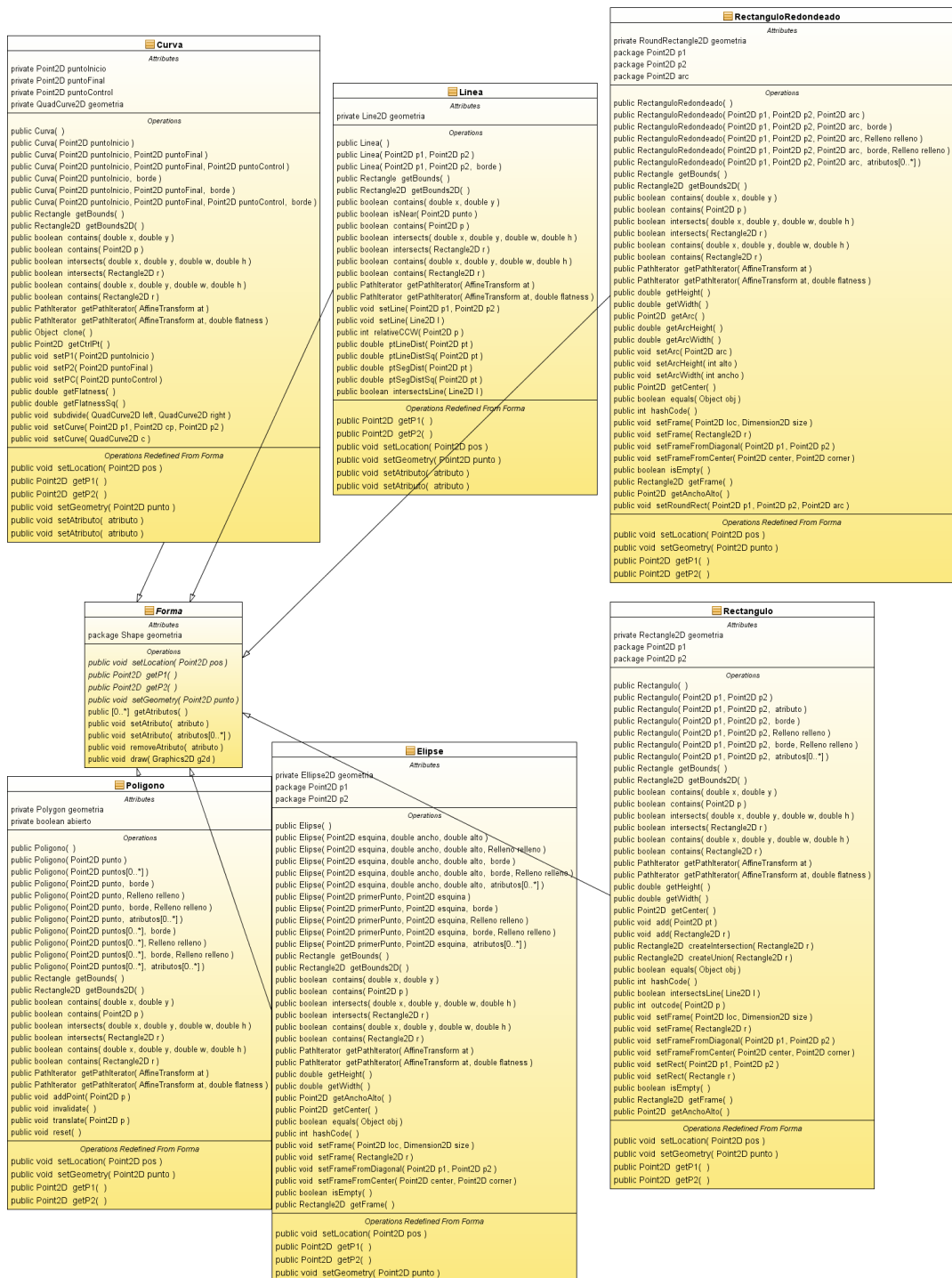
Voy a desarrollar una clase y ese desarrollo es extrapolable a las otras clases. Voy a describir la clase Rectangulo.

Lo primero es declarar la variable geometría como una variable Rectangle2D. En mi caso me he quedado con todos los métodos de Rectangle2D, los métodos que están repetidos en la clase Rectangle2D me he quedado con los que me ha interesado, para ello lo que he hecho es definir los métodos con el mismo nombre que están en Rectangle2D a algunos le he cambiado los parámetros, en vez de pedir un x y un y, yo he decidido trabajar con Point2D, luego dentro del método llamo al método correspondiente de Rectangle2D.

Para que mi clase Rectangulo sea más funcional, he añadido métodos propios que desde mi punto de vista le aporta mayor funcionalidad, por ejemplo, un

setLocation que heredo de la clase padre, así como otros métodos heredados de la clase padre.

Por último, nos falta los constructores, he declarado un constructor vacío y un constructor con los parámetros básicos, es decir, los mismos constructores que hay en la clase Rectangle2D. La novedad reside en que he declarado constructores con atributos, es decir cuando construyo un objeto puedo asignarle un atributo preestablecidos, varios atributos preestablecidos o un List de atributos. En el caso de la línea y la curva no tendrán constructores con relleno.



### 3.2.3 Observaciones atributos y formas

En un principio no tenía claro cómo integrar la Forma con el Atributo, así que después de pensarlo y preguntar al profesor, me decidí por un array de atributos, como vemos es la mejor opción, ya que esto nos permite que cada forma tenga n atributos, lo cual, a la hora de pintar la forma, solamente necesitamos un bucle que recorra la lista de atributos, mientras que si fueran atributos sueltos tendría que crear métodos para que vayan aplicando los atributos.

Para ello lo que hice fue declarar en la clase Forma un array de tipo atributo, eso me permite que todas las clases tengan ese array de atributos. El haber declarado el array de atributos en la clase Forma, me permite añadir métodos relacionados con los atributos, como es el caso de setAtributo, removeAtributo, etc.

En el proyecto hay un caso especial que es el caso del TrazoLibre, este como tal no es una forma y en verdad yo pienso que debería ser una clase aparte. Pero por comodidad y facilidad a la hora de ser implementado, así como a la hora de asignarle los atributos, decidí que esta clase heredaré de clase forma y sobrescribir el método draw.

### 3.2.4 Operaciones imagen (RF-2.9)

En mi proyecto he implementado una serie de operaciones básicas que permiten modificar las imágenes, las operaciones implementadas son:

- Brillo, umbralización, tintado y rotación estas operaciones se ha implementado en la clase Paint, son unas operaciones que usa un slider, por ese motivo es importante conocer el uso de los manejadores de eventos de java. Cuando pinchas en el slider ganas foco, cuando ganas foco tú debes cargar la imagen en una variable y cuando pierdas el foco debes poner la variable a null. Mientras mueves el slider vas generando pequeños cambios, que tú debes aprovechar para hacer el cambio necesario en la imagen.
  - ✓ En el caso del brillo usaremos un RescaleOp con el brillo y teniendo en cuenta si la imagen tiene un filtro alpha o no. Una vez creado el RescaleOp, le aplicamos a la imagen el filtro del RescaleOp y con esto nos devuelve la imagen trasformada. El RescaleOp viene definida por defecto en la biblioteca sm.image
  - ✓ En el caso de la umbralización usaremos una clase creada por mí que la he llamado UmbralizacionOP que hereda de sm.image.BufferedImageOpAdapter, en esta clase lo que haremos es mediante el método filter, aplicar una umbralización basada

en intensidad para ello, calcularemos la intensidad de cada pixel con la siguiente fórmula  $I(x,y)=(r(x,y)+g(x,y)+b(x,y))/3$  y si pasa de un umbral, este umbral se le asigna al crear el filtro, se trunca 255 y si es menor a 0. Una vez explicado esto explicare como hacer la umbralización.

Cuando movemos el slider creamos un objeto `UmbralizacionOp` al que le pasamos como parámetro un umbral, una vez creado el objeto le aplicamos el filtro a la imagen y ya tenemos la imagen trasformada.

- ✓ En la rotación guardaremos el valor que te va dando el slider, este valor lo transformamos en radianes, una vez trasformado calculamos el punto central de la imagen y en ese punto haces un `AffineTransform.getRotateInstance` y lo guardas en una variable. Entonces creas un `AffineTransformOp` con la trasformación anterior y con una interpolación. Una vez creada aplicas el filtro.

- ✓ Para el tintado he creado un `JInternalFrame` llamado Tintado, en esa clase he colocado un botón que lanza una paleta de colores, un slider para cambiar el nivel de tintado y un panel donde previsualizar el efecto antes de que sea permanente.

El tintado funciona de la siguiente manera, creo un objeto de tipo `TinOP` pasándole en el constructor el color del tintado y un valor entre 0 y 1. Una vez creado el objeto le aplicamos el filtro. La clase `TintOp` es una clase creada por mí que hereda de `sm.image.BufferedImageOpAdapter`, en la cual he implementado un método `filter`. Este método lo que hace es aplicar a cada pixel la siguiente transformación  $f(x,y)=[r,g,b]$   
 $g(x,y)=\alpha \cdot C+(1-\alpha)f(x,y)$ , donde  $C=[r,g,b]$  es el color con el que se tinta la imagen y  $\alpha \in [0,1]$  el grado ( $\alpha=0$  lo deja igual,  $\alpha=1$  lo pinta todo).

- Sepia, cosrai y sobel, estas tres operaciones tiene clases propias desarrolladas por mí, pero en líneas generales el funcionamiento básico es el mismo: primero se crea el objeto de la clase y una vez creado se aplica el filtro a la imagen. En lo que se diferencia es en el filtro.

- ✓ El filtro del Sepia lo que hace es para cada componente de un pixel aplicar la siguiente fórmula:

$$\text{sepiaR} = \min(255, 0.393 \cdot R + 0.769 \cdot G + 0.189 \cdot B)$$

$$\text{sepiaG} = \min(255, 0.349 \cdot R + 0.686 \cdot G + 0.168 \cdot B)$$

$$\text{sepiaB} = \min(255, 0.272 \cdot R + 0.534 \cdot G + 0.131 \cdot B)$$

En el caso de que el valor sea mayor que 255, truncamos el resultado a 255.

- ✓ El filtro del cosrai lo que hace es para cada componente de un pixel aplicar la siguiente fórmula,  $(k * \text{Math.abs}(\text{Math.sqrt}(\text{Math.cos}(w * \text{pixel.sample}[i])))$  donde k es 255, esta es una función inventada por mí.
  - ✓ El filtro sobel lo que hace es para cada componente de un pixel aplicar la siguiente fórmula:  $\nabla f = [\partial f / \partial x, \partial f / \partial y] = [\nabla x, \nabla y]$  donde  $\nabla x = 1/4 \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix} \nabla y = 1/4 \begin{pmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 2 & -1 \end{pmatrix}$ . Para una imagen en color, el vector gradiente en un pixel se calculará sumando los vectores gradiente de cada banda. Una vez obtenido el gradiente, la magnitud y la orientación vendrán dados por:  $|\nabla| = \sqrt{\nabla x^2 + \nabla y^2}$ . En nuestro caso, en vez de seguir con la fórmula, truncaremos el valor de la magnitud a 255. Para truncarla usaremos `sm.image.ImageTools.clampRange(magnitud,0,255)`. Para sacar el gradiente de X y el gradiente de Y, usaremos un `ConvolveOp` con unas máscaras ya creadas, con los valores descriptos anteriormente.
- `ConvolveOp` viene definida por defecto en la biblioteca `sm.image`

- Contraste, iluminar, oscurecer, negativo y sinusoidal, estas operaciones tienen en común que usan la clase `LookupOp`, que permite definir una transformación sobre los niveles de color mediante un objeto de la clase `LookupTable`, que se aplicará a cada componente de cada pixel. Todas funcionan de la misma manera: se crea un `LookupTable` al que se le ha asignado un tipo de función y ese `LookupTable` se le pasa como parámetro al constructor de la clase `LookupOp` y una vez creado el objeto se le aplica el filtro a la imagen. Todas estas funciones se encuentran en la biblioteca `sm.image`
  - ✓ Para el contraste se utiliza un método denominado `sFunction`, que lo que hace es aplicar la siguiente fórmula  $(K * (1.0 / (1.0 + \text{Math.pow}(m / (\text{float})l, e))))$ , donde `double K = 255.0 / Max` y `max` es  $(1.0 / (1.0 + \text{Math.pow}(m / 255.0, e)))$ .
  - ✓ Para el ilumina se utiliza un método basado en logaritmos, que ejecuta la siguiente fórmula  $(K * (1.0 + \log((\text{float})l)))$ , donde `double K = 255.0 / Max` y `max` es el logaritmo de 255
  - ✓ Para iluminar se utiliza un método basado en exponentes, que aplica la siguiente fórmula  $(K * (1.0 + \text{pow}(l, n)))$ , donde `double K = 255.0 / Max` y `max` es  $\text{pow}(255, n)$

- ✓ Sinusoidal se utiliza un método basado en el seno, que ejecuta la siguiente fórmula (byte)  $(k * \text{Math.abs}(\text{Math.sin}(w * i)))$ , donde  $k$  es 255.
- ✓ Negativo se utiliza un método llamado negativo, que lo que hace es restar a 255 el valor del pixel y asignárselo de nuevo usando ShortLookupTable.
- La transformación a niveles de grises realiza una conversión de espacio de color pixel a pixel. Para ello primero hay que crear un espacio de color en gris, para este cometido usamos ICC\_Profile, que crea una representación de los datos del perfil del color para espacios independientes. Este perfil lo asignamos a un espacio de color, creamos un ColorConvertOp con este espacio de color y le aplicamos el filtro a la imagen.
- Para la suma y la resta binaria necesitamos dos imágenes, estas operaciones consisten en sumar o restar una imagen respecto a la otra, donde cada pixel se sumará o restará con el otro. Si el resultado es superior a 255 deberá truncarse o escalarse, si una imagen es mayor que la otra, hay dos opciones o descartar lo que sobre o aplicarlo sobre la intersección.
  - ✓ Para sumar debemos hacer el BlendOp de una de las imágenes y aplicar el filtro a la otra imagen. Una vez creada la imagen lo que demos hacer es crear una clase de tipo VentanaInternaImagen y asignarle al lienzo la imagen creada por el filtro.
  - ✓ Para restar debemos hacer el SubtractionOp de una de las imágenes y aplicar el filtro a la otra imagen. Una vez creada la imagen lo que demos hacer es crear una venta de tipo VentanaInternaImagen y asignarle al lienzo la imagen creada por el filtro.
  - ✓ Luego está el caso especial del mezclador de imágenes. Éste se implementa como el sumar, la única diferencia es que el slider cambia el valor del alpha asociado al BlendOp, el valor de alpha tiene que estar entre 0 y 1.
- Para aumentar y disminuir el zoom, hacemos un AffineTransform.getInstance con el valor deseado y lo guardas en una variable. Entonces creas un AffineTransformOp con la transformación anterior y con una interpolación. Una vez creada aplicas el filtro a la imagen.

- Para crear un histograma debemos crear un objeto de tipo Histogram y pasarle como parámetro en el constructor la imagen. La clase Histogram viene definida en la biblioteca sm.image.

Para poder mostrar los valores del histograma he creado una clase propia que hereda de JInternalFrame, esta clase la he denominado Histograma. Esta clase se compone de 4 JPanel donde mostrará las gráficas. Para poder crear las gráficas he tenido que usar la biblioteca jfreechart y la jcommon. Estas bibliotecas nos permiten de forma más o menos fácil crear los gráficos, para ello sólo debemos ir grabando los datos de cada banda en una variable DefaultCategoryDataset, una por cada banda. Una vez que los datos han sido almacenado creamos el gráfico con ChartFactory. Hay multitud de tipos de gráficos, decantándome por uno de barras y por último añadimos el gráfico al JPanel.

### 3.2.5 Redimensionar y reescalar imagen (RF-2.5 RF-2.6)

Para redimensionar y reescalar una imagen he creado un JInternalframe, donde tendré dos campos: uno para meter el ancho y otro para el alto, además de dos botones uno para reescalar y otro para redimensionar.

Cuando pulsas sobre redimensionar llama a un método asociado al lienzo que encontraremos en Lienzo2DImagen. Este método lo que hace es crear una nueva imagen con el tamaño deseado, una vez creada se graba la imagen antigua en la imagen nueva; en el caso de que la imagen nueva sea más grande, la parte que sobra aparecerá en blanco, en caso de que sea menor la imagen será recortada.

Cuando pulsas sobre reescalar llamas a un método asociado al lienzo que encontraremos en Lienzo2DImagen, este método lo que hace es crear una nueva imagen con el tamaño deseado, una vez creada se graba la imagen antigua en la imagen nueva, en el caso de que la imagen nueva sea más grande, la imagen se ensanchará hasta ocupar todo el espacio, en caso de que sea menor la imagen será achicará para que entre en el espacio.

### 3.2.6 Duplicar (RF-2.7)

Para duplicar lo que hago es coger la ventana seleccionada y crear una ventana nueva, pasándole al lienzo la imagen que quiero copiar.

### 3.2.7 Nueva Imagen(RF-2.1)

Crear un nuevo lienzo en sí no es complicado, es crear una ventanaInternalImagen y asignarle un BufferedImage con el tamaño establecido.

### 3.2.8 Editar formas (RF-2.8) y borrar atributos (RF-2.4)

Desde Paint podemos asignar atributos a una Forma que va a ser pintada o modificar los atributos de una forma ya pintada, para ello tenemos dos grandes atributos, uno es el relleno y otro es el borde. Además, de asignar atributos y borrarlos podemos mover las formas, la parte mover formas la explicaré en el apartado dibujar(3.2.9).

- En cuanto a las opciones del borde, podemos seleccionar el tipo de trazo, para ello hemos creado un menú desplegable, donde aparece un icono con el trazo a elegir. Para poder poner iconos en un menú desplegable he tenido que crear una clase llamada Iconos que herede de `ListCellRenderer<Icon>` y de `javax.swing.JPanel`, e implementar un método que admita una lista de objetos `Icon` y los vaya asignando a un botón que tiene esa clase. En el `comboBox` que he utilizado para mostrar los tipos de trazo, he tenido que pasarle al constructor como parámetro un array de iconos.

Otro de las opciones que se le puede asignar a una forma es el color, que también está implementada como una lista, sólo que en este caso en vez de `Icon` es `Color`; la clase se llama `IconoColor`. Y la última opción que se puede modificar es el grosor.

- En cuanto a las opciones del relleno hay tres posibilidades:
  - ✓ Sin relleno: es la opción por defecto, no se aplica relleno a la forma además tiene la opción de eliminar el relleno de una forma.
  - ✓ Relleno simple: para el relleno simple he creado una clase que hereda de `JInternalFrame`, que he denominado `Rellenos`. En esta clase he añadido una paleta de colores para poder elegir el color del relleno que desees, una vez que hemos dado a aceptar, creamos el relleno con los parámetros seleccionados y mediante una función implementada en `Lienzo2D` asignamos el relleno a la forma
  - ✓ Relleno degradado: para el relleno degradado he creado una clase que hereda de `JInternalFrame`, que he denominado `RellenosDegradado`. En esta clase he implementado unos colores por defecto para el fondo y el frente, pero por si alguien deseaba poner otros colores, he implementado unos iconos que lanzan una ventana con una paleta de colores. En la parte de la derecha tenemos un menú desplegable con los tipos de rellenos, este menú se ha implementado de la misma manera que el menú de los trazos. Además, he añadido dos casillas para que puedan saber que colores han elegido.



Este relleno sólo se puede crear una vez dibujada la forma, ya que nos basamos en las dimensiones de la forma para adaptar el relleno.

Internamente esto está implementado con un sistema de if/else, en el cual según el tipo de relleno que hayas seleccionado crea un objeto relleno u otro.

### 3.2.9 Dibujar(RF-2.10)

Para poder dibujar en mi programa he creado una clase Lienzo2D que heredará Lienzo2DImagen.

- Empezaré hablando de Lienzo2DImagen. Esta clase, como hemos dicho antes, hereda de Lienzo2D. La creé para satisfacer la necesidad de trabajar con los BufferedImage, ya que con Lienzo2D podíamos dibujar, pero no podíamos pintar ni tratar imágenes. En esta clase todo lo relacionado con dibujar formas y asignar atributos lo hereda de Lienzo2D, aquí básicamente tenemos métodos para obtener y poner imágenes en el lienzo, así como para reescalar y redimensionar imágenes. Pero el método clave en esta clase es el paintComponent, ya que nos permite dibujar primero los bordes y luego dibujar el resto del contenido de la imagen. Un ejemplo sería imaginar que tenemos una imagen de flores y sin este método te dibujaría un lienzo en blanco. A la hora de redimensionar, reescalar y obtener la imagen, es muy importante crear la imagen del tipo correspondiente a la imagen original.
- Ahora hablaré de Lienzo2D. Es la clase más importante de todo el proyecto ya que sin esta clase no podríamos hacer casi nada. En esta clase se crean y dibujan las formas y atributos seleccionados. En esta clase es especialmente importante el manejo de eventos de ratón, ya que según el evento que se produzca estarás comenzando a dibujar una forma o estarás terminando de dibujar dicha forma.
  - ✓ Voy a explicar con más detalle esta parte, tenemos figuras que son pinchar, arrastrar y soltar, como es el caso del rectángulo, la elipse, la línea, la línea con punto de control y el rectángulo redondeado. En estos casos cuando pinchas en el lienzo estas llamando a un método que se llama formMousePressed, que a su vez llama a createShape donde creas la forma con los atributos que tengas seleccionado en ese momento. Ese método devuelve una forma que agregarás a un vector de formas, que nos permite tener varias formas en el lienzo. Una vez que has pinchado, lo que haces a continuación es arrastrar el ratón para darle forma a la figura, en este caso lo que hace el lienzo es llamar al evento dragged, donde llamamos un método de la clase forma llamado setGeometry que nos permite

ir actualizando la forma de la figura. En este evento tenemos en cuenta la Forma a dibujar, ya que con la línea con punto de control hemos tenido que implementar un sistema de etapas, debido a que la línea de control funciona de la siguiente manera: en la primera etapa dibujas la línea y en la segunda etapa estableces la curva de la línea. Por eso cuando una vez hemos soltado el ratón el resto de figuras se quedan dibujadas de forma definitiva, pero en el caso de la curva tendrá un segundo ciclo de pinchar, arrastrar y soltar. En este caso sólo afectará a la curva, no al inicio ni final de la misma.

Luego están las Formas que en vez de pinchar es clickear, para estas Formas editamos el manejador `formMouseClicked`, siendo las formas polígono y polígono abierto. Cada vez que cliqueamos añadimos un punto al polígono, para poder dar por finalizado la Forma hemos tenido que utilizar el doble clic del botón derecho del ratón, que por fortuna está implementado como una máscara en el evento del ratón

Otro caso especial es el punto, ya que este sólo necesita del `pressed`, por lo tanto, cuando haces el `pressed` ya lo creas de forma definitiva.

- ✓ Otra parte importante es el tratamiento de los Atributos, para ello tenemos métodos para asignar y obtener los atributos. A continuación, voy a describir los métodos para asignarlos:
  - ❖ `setColorTrazo`. Este método nos permite asignarle un color al borde de la Forma que se va a dibujar o que se está editando. Para ello usamos una variable `Color` que almacenará el color, en el caso de que sea una figura que se está editando, obtendremos los atributos de esa figura y buscaremos el Atributo Borde y una vez encontrado le asignaremos un nuevo color.
  - ❖ `setGrosor`. Este método nos permite asignarle un trazo a una forma, ya sea una Forma que se va a dibujar o una Forma que se está editando. Para ello grabamos el grosor en un variable de tipo `int` y en el caso que estemos editando, obtendremos los atributos de esa figura y buscaremos el Atributo Borde. Cuando lo encontremos le asignaremos un nuevo trazo.

- ❖ `setAlisar`. Este método nos permite alisar la Forma, para ello obtendremos los atributos de esa Forma y le asignaremos un `RenderingHints`
- ❖ `setStroke`. Este método nos permite crear un trazo nuevo o modificar el trazo de una Forma, para ello según el tipo de trazo elegido crea un Borde de un tipo o de otro. Una vez creado el borde, si no se está editando se almacenará en una variable global, en el caso de que se esté editando una forma, se busca si hay un borde en la lista de atributos o no, en el caso de que lo halla, se saca los valores del alisado y de la transparencia, y se proceda a borrarse ese borde y añadir el nuevo borde con los valores del alisado y la transparencia
- ❖ `setRelleno`. Este método nos permite crear un relleno nuevo o modificar el relleno de una Forma, para ello tenemos tres opciones explicadas en el apartado 3.2.8. Si no se está editando se almacenará el relleno en una variable global; en el caso de que se esté editando una forma, se busca si hay un borde en la lista de atributos o no; si lo halla, se saca los valores del alisado y de la transparencia, y se proceda a borrarse ese borde y añadir el nuevo borde con los valores del alisado y la transparencia.
- ❖ `getSelectedShape`. Nos permite seleccionar las figuras

- ✓ `Paint` es la función más importante es la encargada de dibujar todas las formas en el lienzo.

### 3.3 Gestión audio

#### 3.3.1 Reproducción audio (RF-3.1)

Para la reproducción de audio uso la clase `VentanaInternaSonido`, en ella para poder reproducir audio usando la biblioteca `sm.sound`, para ello debo crearme una variable `SMPlayer` e inicializarla con un archivo, una vez creada la

variable para reproducir o para parar el sonido basta con usar los métodos implementados en la clase.

En esta ventana he implementado un contador de tiempo y una barra de progreso, para ello he tenido que crear un manejador de eventos propios. Cuando el sonido esta reproduciéndose lo que hago es que en el método update del manejador (el if correspondiente a play) le asigno a la barra de progreso como valor máximo la duración en microsegundos de la canción y conforme se va reproduciendo le asigno como valor a la barra los microsegundos que lleva reproducido el sonido.

En este if también establezco el tiempo máximo de duración del audio. Para establecer la duración máxima de la canción lo que he hecho es obtener la duración máxima en microsegundos y pasarla a minutos. Para establecer el tiempo de reproducción del audio, obtengo los microsegundos de la canción en ese momento dado y los paso a minutos. Estos dos datos los visualizo en pantalla.

Tanto para la barra de reproducción como para el contador debo obtener el clip del archivo de audio y mediante un while(clip.running) voy actualizando los valores del contador y de la barra de progreso.

### 3.3.2 Grabación de audio (RF-3.2)

Para grabar sonido, como la barra de reproducción la he integrado en la clase Paint, debo crear una variable SMSoundRecorder. Esta clase la encontraremos en la biblioteca sm.sound, además de un archivo temporal donde almacenaremos el sonido. Para grabar basta con pulsar el botón de grabar, internamente lo que hace es inicializar la variable SMSoundRecorder con el fichero temporal y ejecutar la función record(). En el botón de grabar he añadido una hebra que hará de cronómetro para saber cuánto tiempo llevo de grabación. En este botón también he añadido una variable booleana que indica si estas grabando o no

Para parar y grabar el audio basta con darle a stop, internamente lo que hace es comprobar que la variable SMSoundRecorder no esté a null y la variable booleana esté a true, entonces abriremos un diálogo para que el usuario señale un archivo donde grabar el audio o ingrese el nombre del archivo. Si no le añade extensión le añadimos una por defecto. Por último, grabamos el contenido del archivo temporal en el archivo definitivo.

## 3.4 Gestión video

Para la reproducción de video (RF-4.1) uso la clase VentanaInternaReproduccionVideo, en ella he usado las bibliotecas jmf de java, las de multiplayer y mediaplayer. En esta clase no he añadido nada propia, lo único que hay que hacer es crear una variable MediaLocator e inicializarla con el archivo de

video, creamos el reproductor, inicializando la pantalla y la barra de reproducción para cuando le demos a play se inicia éste.

## 3.5 Gestión webcam

### 3.5.1 Activar webcam(RF-5.1)

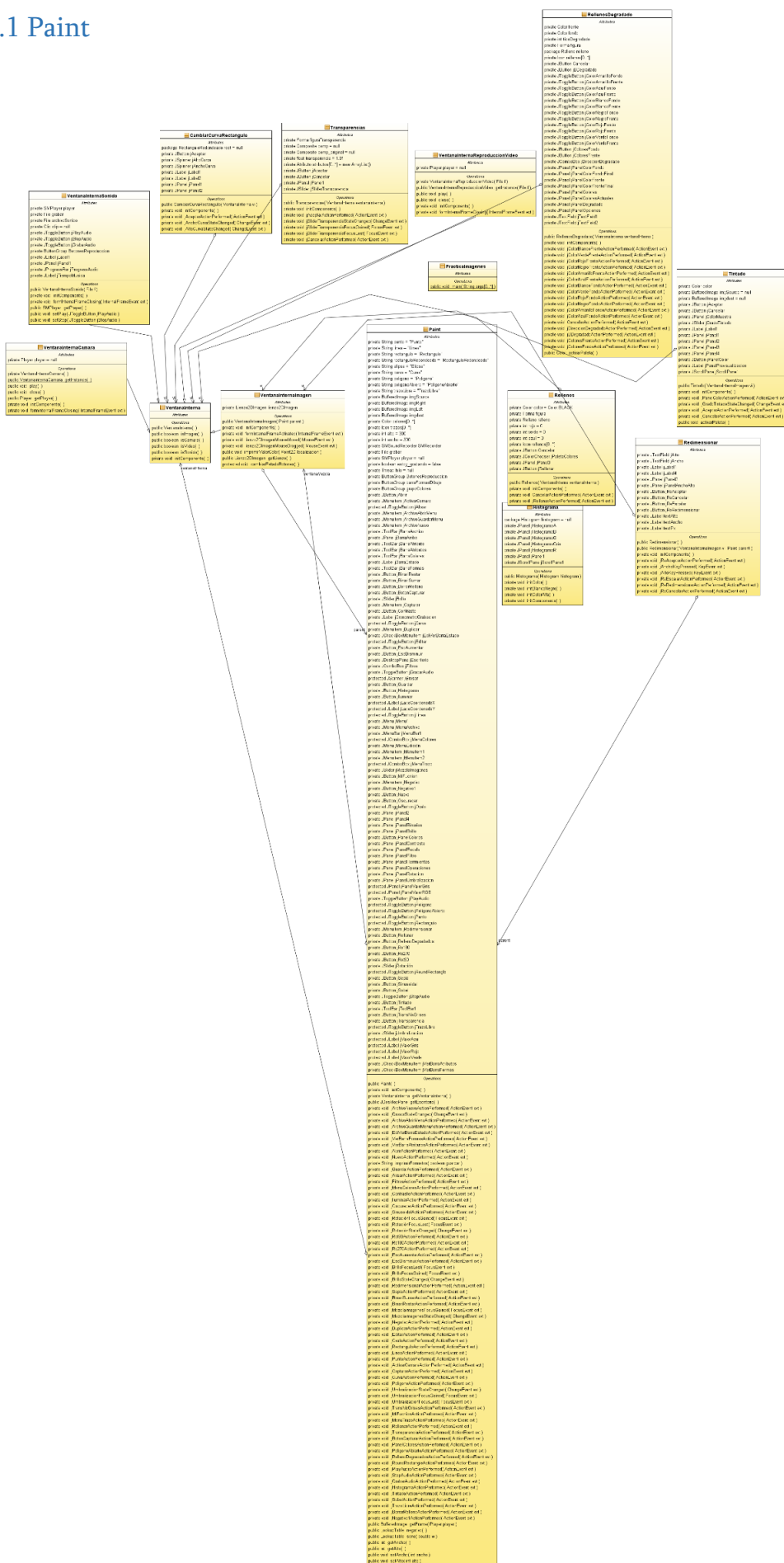
Para la activación de la webcam (RF-4.1) uso la clase `VentanaInternaCamara`, en ella he usado las bibliotecas `jmf` de java, las de `multiplayer` y `mediaplayer`. Aquí cambia sustancialmente ya que primero debemos obtener las cámaras disponibles en el sistema. Usamos `CaptureDeviceInfo`, usando `getDeviceList` y pasándole como formato `YUVFormat`, obtenemos una lista de dispositivos. Nos quedamos con el primero y se lo asignamos a `MediaLocator`, después creamos el reproductor de vídeo, inicializando la pantalla y la barra de reproducción. En mi caso he puesto que se active de forma automática la cámara.

### 3.5.2 Capturar imagen(RF-5.2)

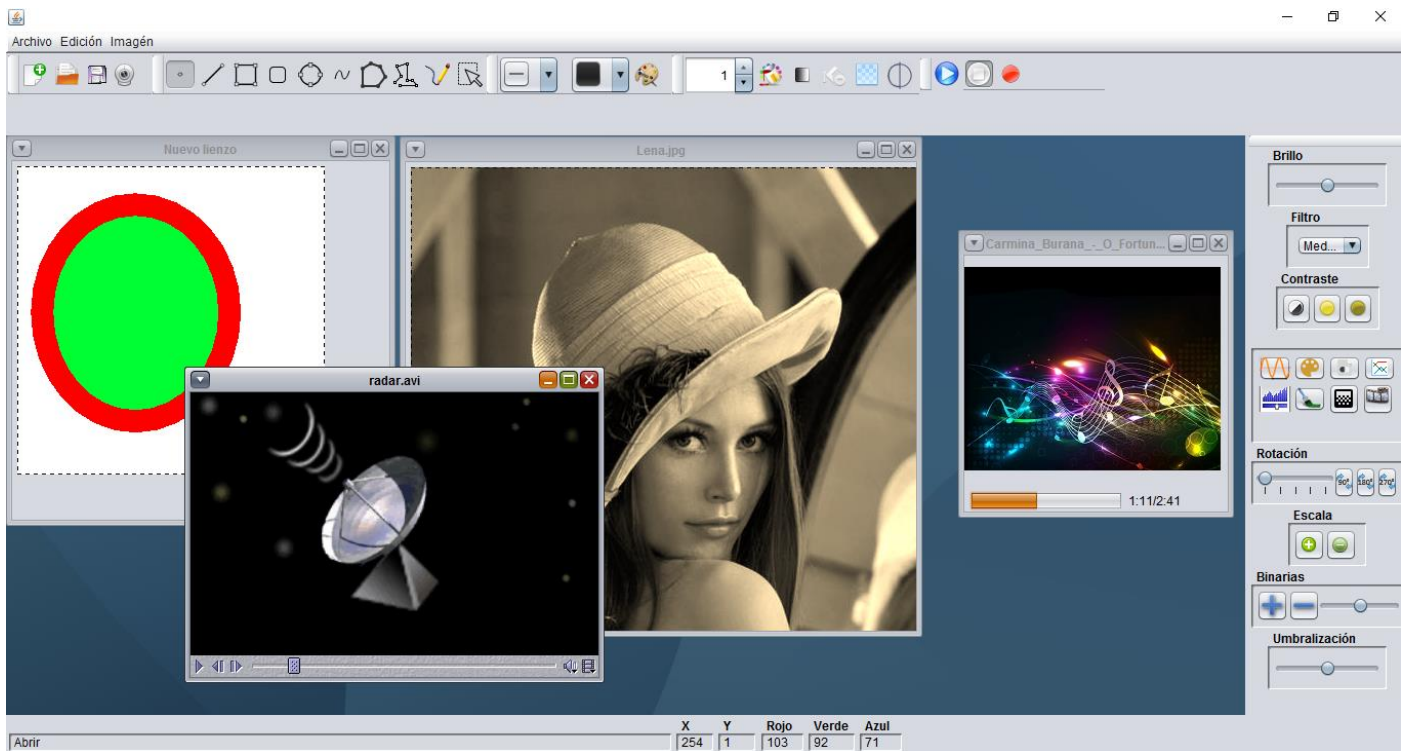
Para capturar una imagen desde la webcam, primero debemos activar la webcam(RF-5.1), una vez activa en la clase `paint`, en el botón `capturar`, debemos obtener el player de la `VentanaInternaCamara` y una vez que lo tengamos, creamos una `VentanaInternaImagen` asignándole un frame obtenido a través del método `this.getFrame(p)`.

## 4. Diseño

## 4.1 Paint



## 4.2 Menú principal



## 4.3 Ventana Interna sonido



## 5. Bibliografía

<http://swing-facil.blogspot.com.es/2012/03/gradientpaint-demo-ejemplos-del-uso-de.html>

<http://ces2601.blogspot.com.es/2010/08/cronometro-en-java-usando-netbeans.html>

<http://algoimagen.blogspot.com.es/2013/09/java-crear-histograma-acumulado.html>

[http://aprenderaprogramar.com/index.php?option=com\\_content&view=article&id=646:documentar-proyectos-java-con-javadoc-comentarios-simbolos-tags-deprecated-param-etc-cu00680b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188](http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=646:documentar-proyectos-java-con-javadoc-comentarios-simbolos-tags-deprecated-param-etc-cu00680b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188)

<https://docs.oracle.com/javase/8/docs/api/index.html?overview-tree.html>

Apuntes de la asignatura sistemas multimedia de la universidad de Granada