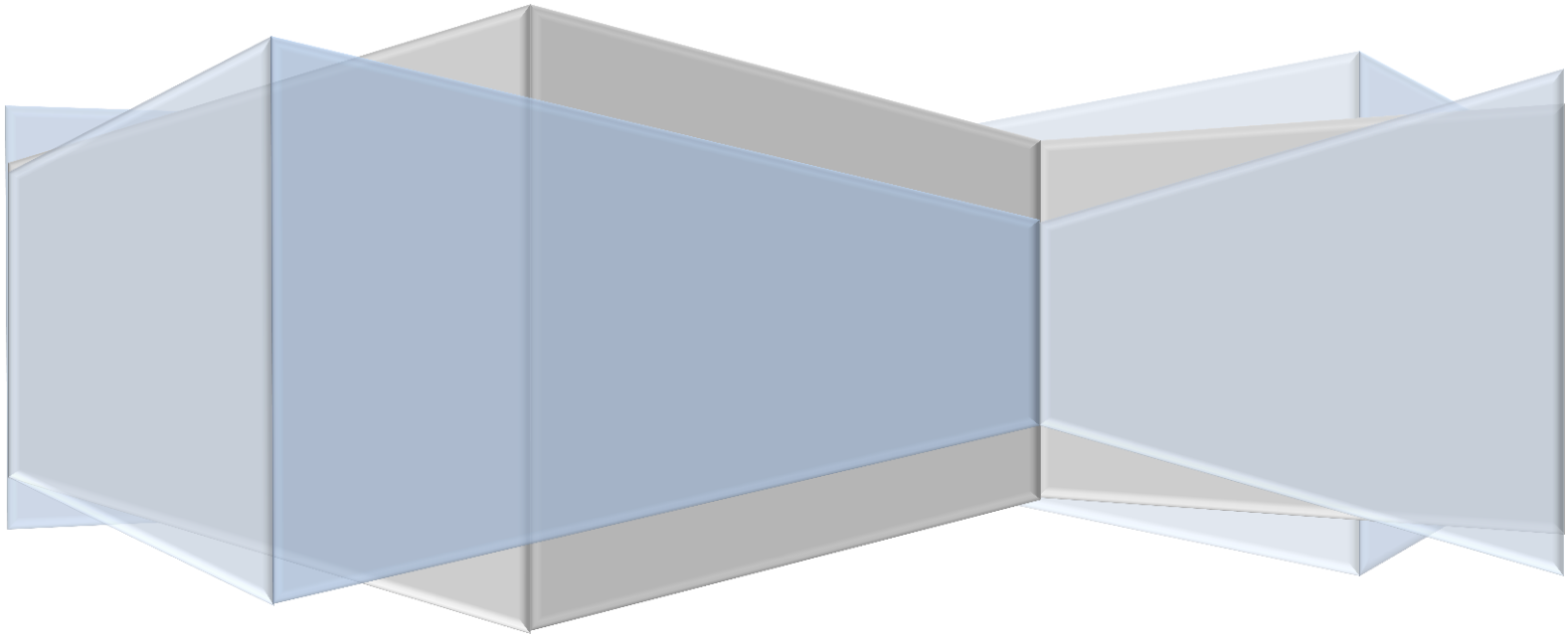
Aplicación Multimedia

Sistemas Multimedia 2013/2014

Rogelio Gil García



ÍNDICE

DESCRIPCIÓN DEL PROBLEMA

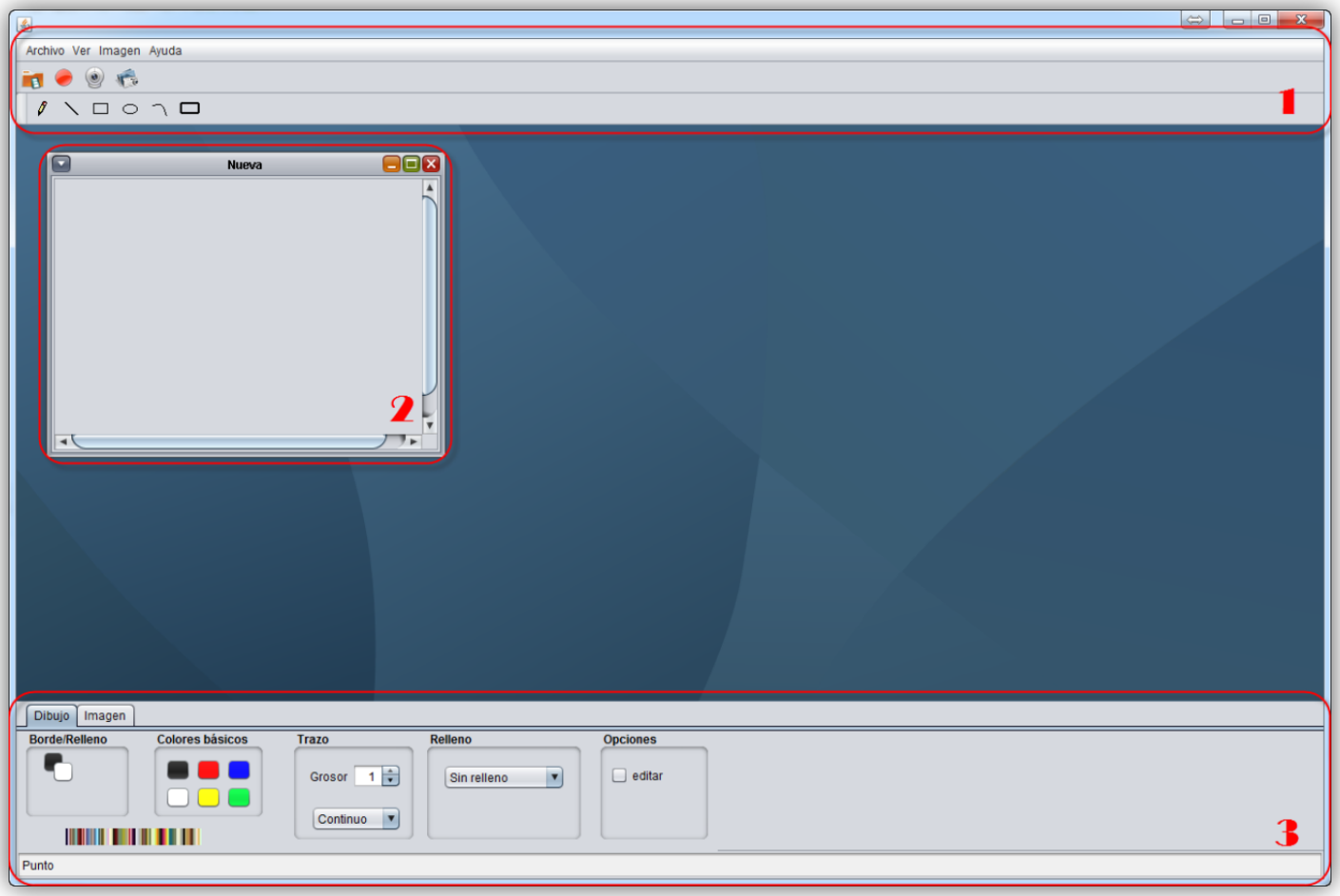
Se necesita una aplicación multimedia que reproduzca y capture audio y vídeo, así como trabajar con imágenes aplicando diferentes filtros y permitiendo además dibujar sobre ellas. Para esto desarrollaremos un entorno multiventana donde encontraremos los siguientes tipos de ventanas:

- Ventana de imágenes.
- Ventana de reproducción de sonido.
- Ventana de reproducción de vídeo.
- Ventana de grabación de audio.
- Ventana para visualizar la webcam.

Para aportar interactividad con el usuario la aplicación contará con un menú y diferentes barras de herramientas relacionadas con las opciones que debe presentar la aplicación sobre todo la parte relacionada con el procesamiento de imágenes y edición de formas ya que ofrecen varias posibilidades al usuario.

INTERFAZ

Para conocer las funcionalidades de nuestra aplicación vamos a hacer una descripción detallada de la interfaz y todos sus elementos.

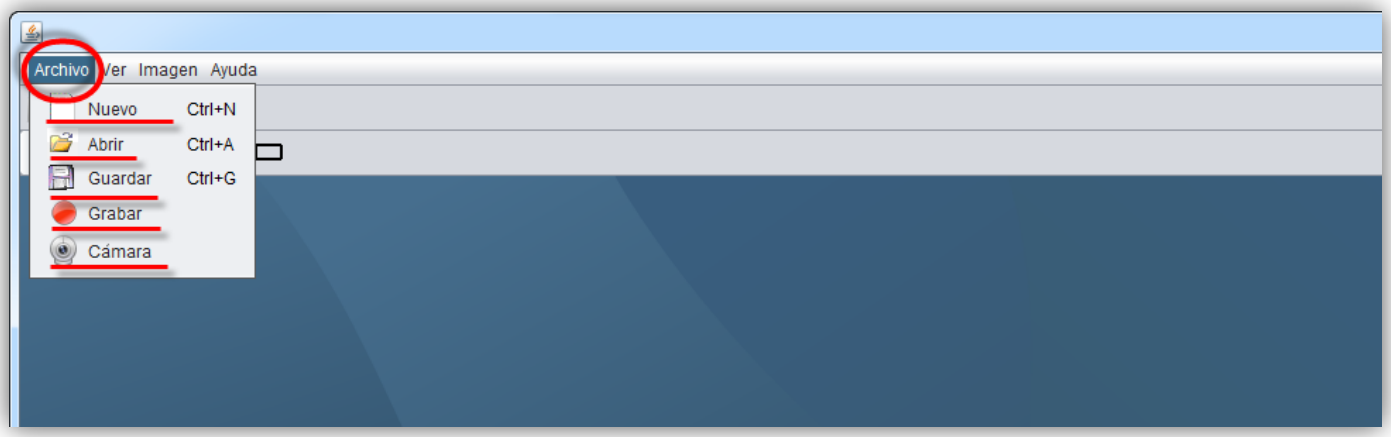


Como podemos ver en la captura anterior, tenemos 3 claros espacios:

- 1. – Barra general y barras de herramientas.
- 2. – Escritorio.
- 3. – Panel con dos barras de herramientas, una para dibujo y otro para imagen.

1. BARRA GENERAL Y BARRA DE HERRAMIENTAS.

ARCHIVO



- **Nuevo:** Lanza una nueva ventana para dibujar en ella. Una vez hacemos clic en nuevo o pulsamos Ctrl+N nos aparecerá un diálogo para insertar el tamaño de la nueva ventana (ancho y alto).

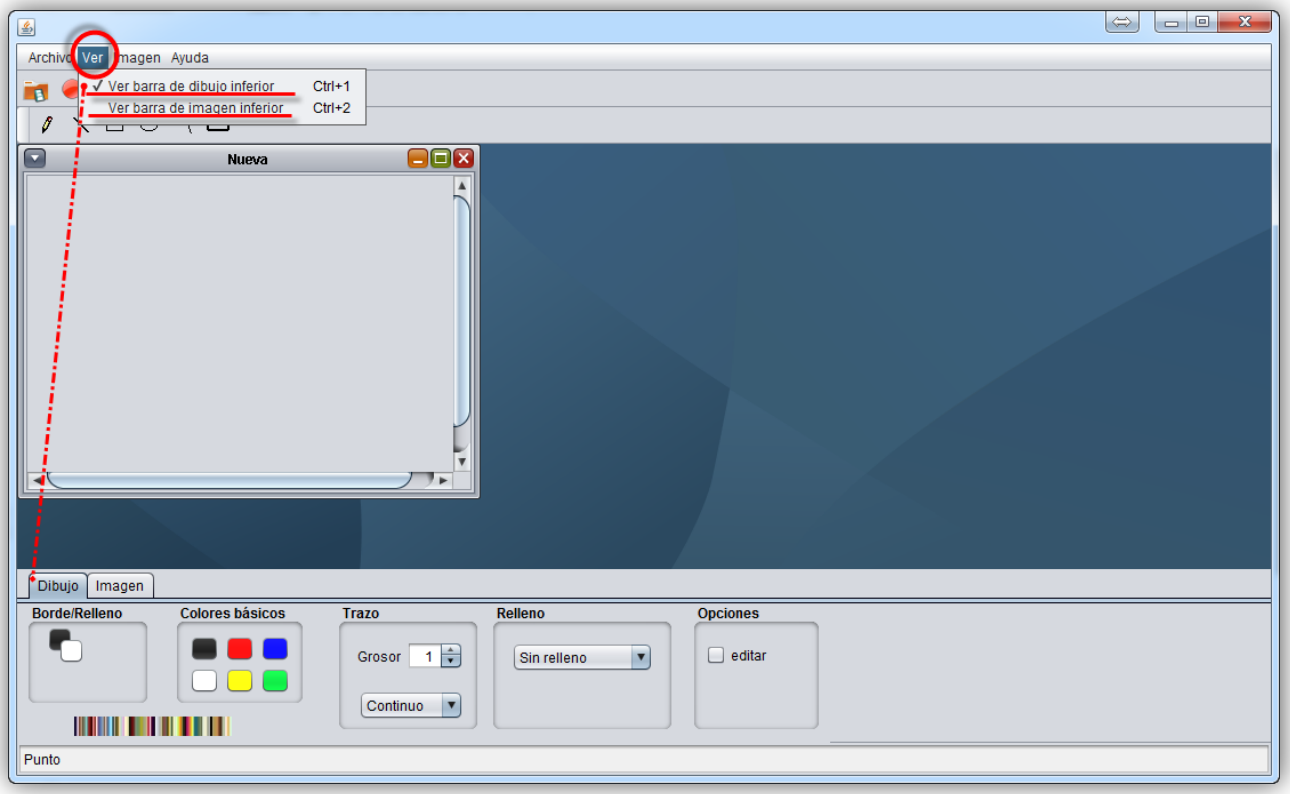
- **Abrir:** Abre un fichero multimedia ya sea una imagen, audio o vídeo. Se lanza una ventana de selección de archivos donde aparecerán los archivos dependiendo del filtro que tengamos seleccionado en la barra inferior. Cuando le demos a abrir se lanzará la ventana correspondiente al fichero seleccionado.

- **Guardar:** Guarda una ventana de dibujo con o sin imagen en una nueva imagen. Se lanza un cuadro de diálogo para seleccionar el lugar donde guardar y el nombre del fichero.

- **Grabar:** Permite grabar un audio mediante el micrófono. Se lanza una ventana para indicar dónde y en qué fichero guardar el audio. Una vez indicado se mostrará una ventana con la opción de iniciar y parar grabación.

- **Cámara:** Abre una ventana donde podremos visualizar lo que capta la webcam.

VER

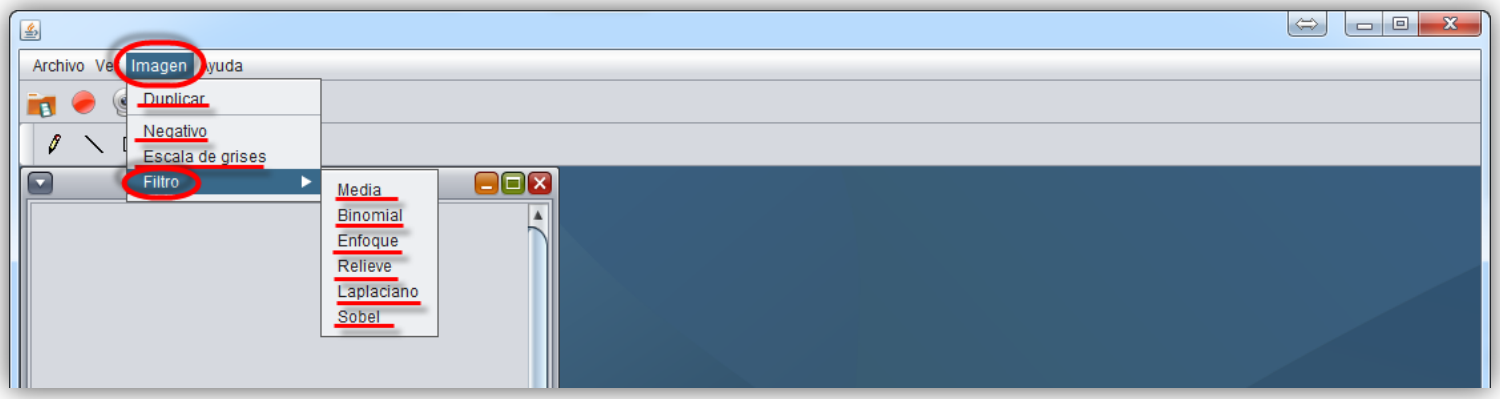


El menú “Ver” servirá para mostrar una barra de herramientas u otra:

- **Ver barra de dibujo inferior:** Muestra la barra de herramientas inferior utilizada para las tareas de dibujo sobre el lienzo o una imagen previamente abierta.

- **Ver barra de imagen inferior:** Muestra la barra de herramientas inferior utilizada para modificar imágenes: tamaño, brillo, contraste,…

IMAGEN



Aquí encontraremos algunas operaciones que podremos hacer con una imagen. La mayoría son filtros u operaciones a aplicar sobre la imagen, sin tener mucha más interacción con el usuario. Las operaciones que necesitan más interactividad con el usuario como slider, selección de colores… se han alojado en la barra imagen inferior.

- **Duplicar:** Crea una nueva ventana de imagen duplicando la imagen de la ventana que tenemos seleccionada. La nueva ventana tendrá como título “Copia”.

- **Negativo:** Se aplica una operación de conversión a negativo a la imagen que tenemos seleccionada.

- **Escala de grises:** Pasa a escala de grises la imagen que hay seleccionada en ese momento.

FILTRO

- **Media:** Filtro de emborronamiento (media). Filtro que da un aspecto de desenfoque suave a la imagen.

- **Binomial:** Filtro de emborronamiento (binomial). Filtro que da otro tipo de desenfoque sobre la imagen.

- **Enfoque:** Filtro para enfocar la imagen.

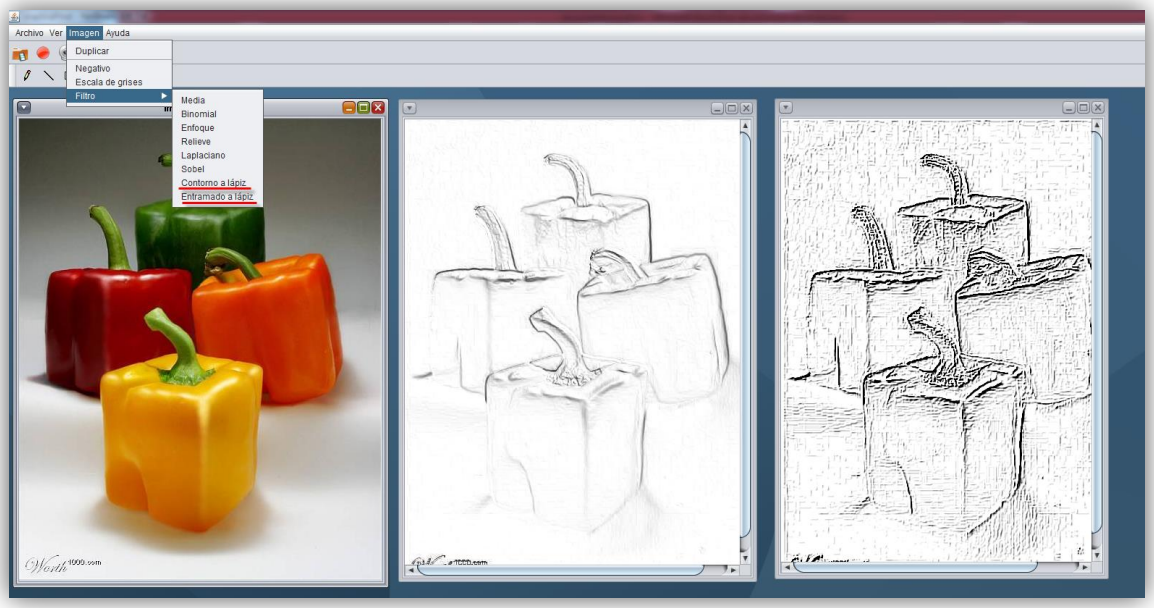
- **Relieve:** Aplica relieve a la imagen. Aumenta el relieve.

- **Laplaciano:** Detección de fronteras laplaciano. Resalta los contornos de las figuras.

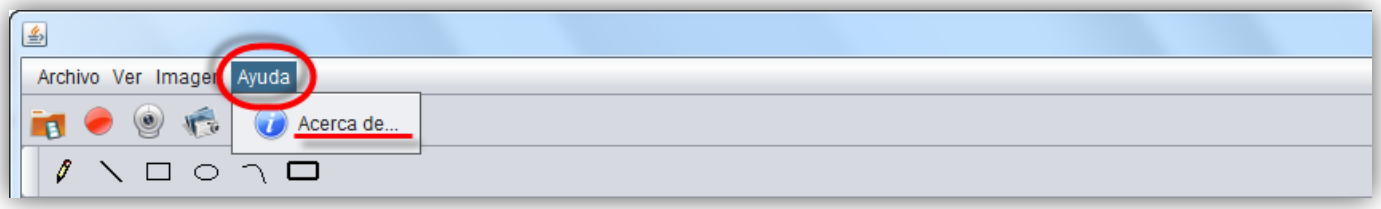
- **Sobel:** Detección de fronteras “sobel”. Sobre un fondo negro, resalta los bordes en blanco.

FILTROS PROPIOS

- **Suave contorno a lápiz:** Operación que simula un contorneado suave con lápiz.
- **Entramado a lápiz:** Un contorneado a lápiz y una trama que “ensucia” un poco la imagen.

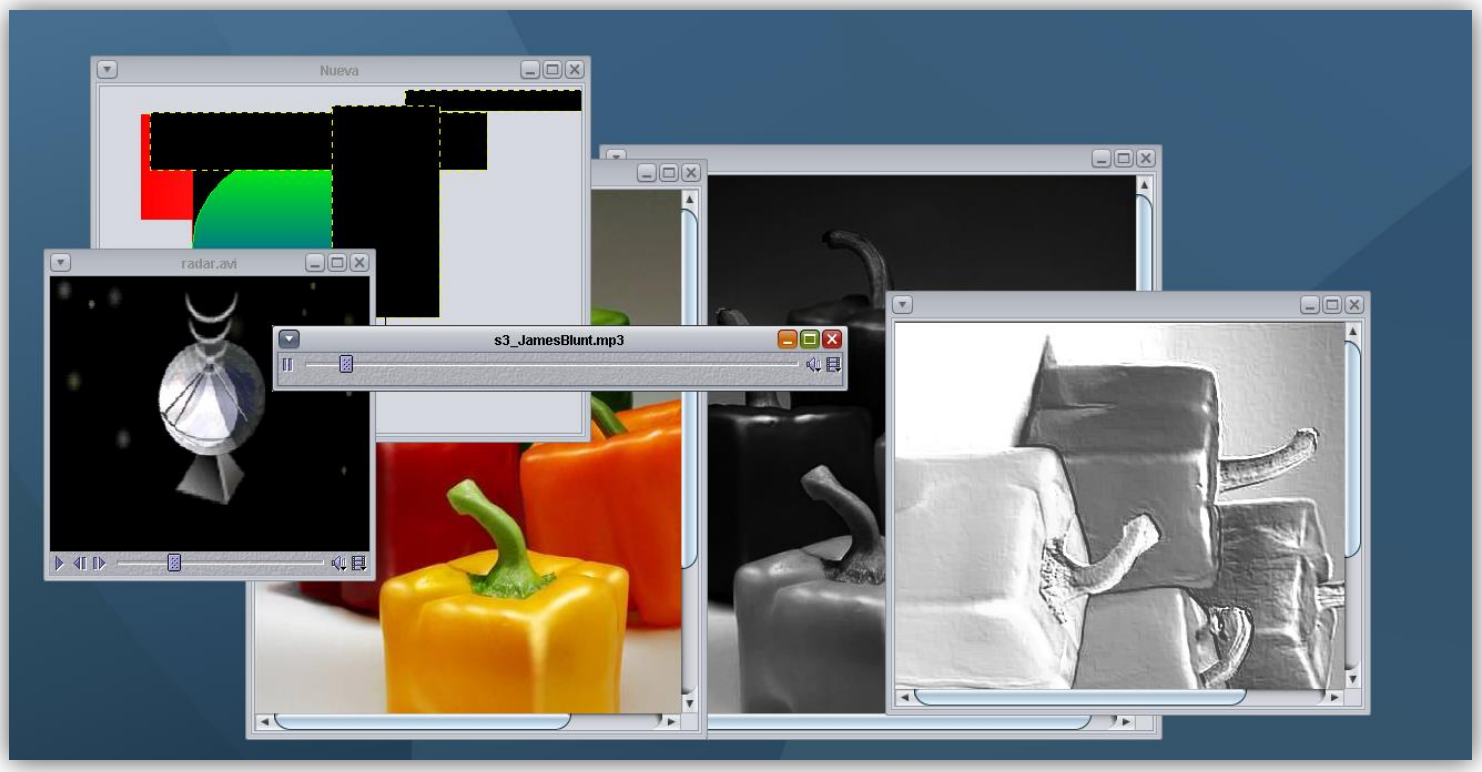


AYUDA



- **Acerca de... :** Muestra un diálogo con el nombre de la aplicación, el autor y la versión.

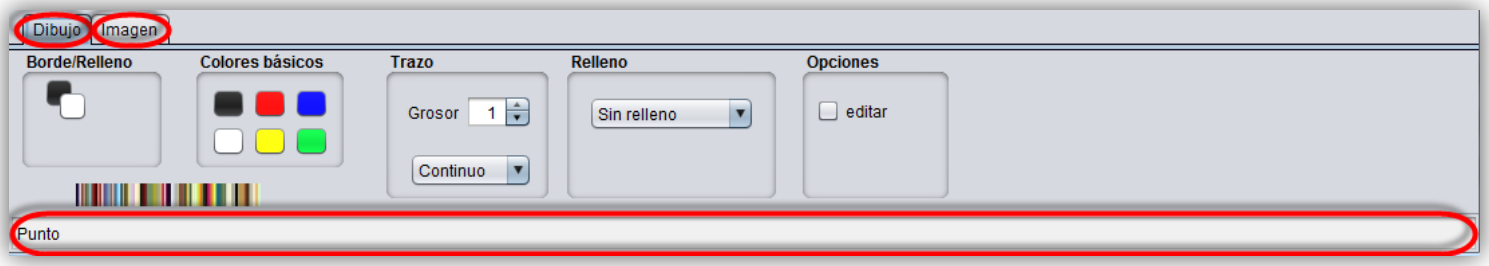
2.- ESCRITORIO



El escritorio de la aplicación es el espacio donde se alojan las diferentes ventanas utilizadas. Los tipos de ventanas como se ha indicado antes son:

- Ventana de imágenes.
- Ventana de reproducción de sonido.
- Ventana de reproducción de vídeo.
- Ventana de grabación de audio.
- Ventana para visualizar la webcam.

3.- PANEL INFERIOR CON DOS BARRAS DE HERRAMIENTAS



En la parte inferior de la aplicación encontramos dos barras de herramientas (dibujo e imagen) y una barra de información, donde aparecerá, por ejemplo, el nombre de la figura seleccionada.

DIBUJO

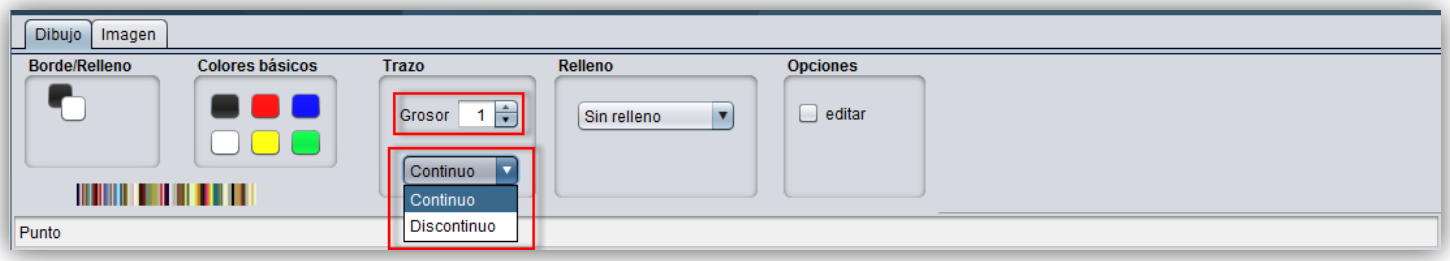
COLOR

- **Borde/Relleno:** Son dos botones superpuestos. El botón que queda detrás y más arriba a la izquierda es el botón con el que seleccionaremos el color del borde. El que queda al frente es el color de relleno de las figuras. Si no estuviese activado el relleno este color no se utiliza (salvo para el degradado). Para cambiar los colores, haremos click en el cuadro de relleno y después en el color a seleccionar.

- **Colores básicos:** Aquí encontramos los 6 colores básicos que frecuentemente van a ser usados.

- **Más colores:** La barra de varios colores abre un diálogo donde podremos seleccionar cualquier color ya sea rgb, cmyk, hsv o hsl en una amplia paleta.

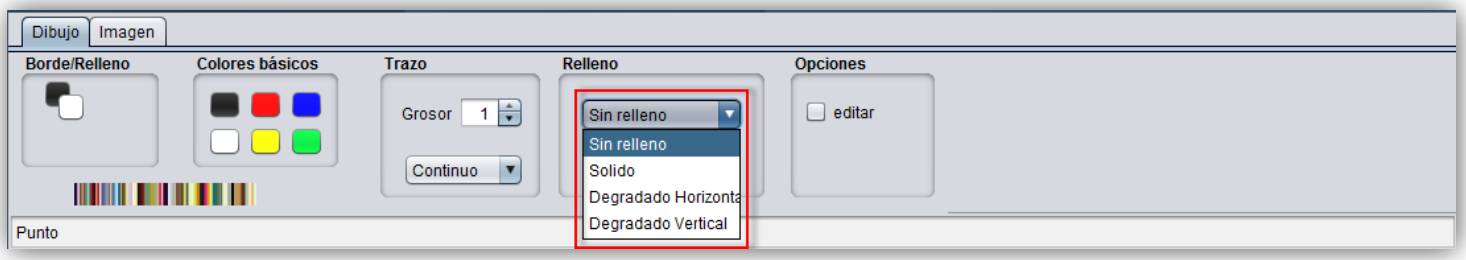
TRAZO



- **<u>Grosor:</u>** Con esta opción podremos aumentar o disminuir el grosor del trazo de la figura. La figura deberá tener como mínimo un grosor de 1 pixel.

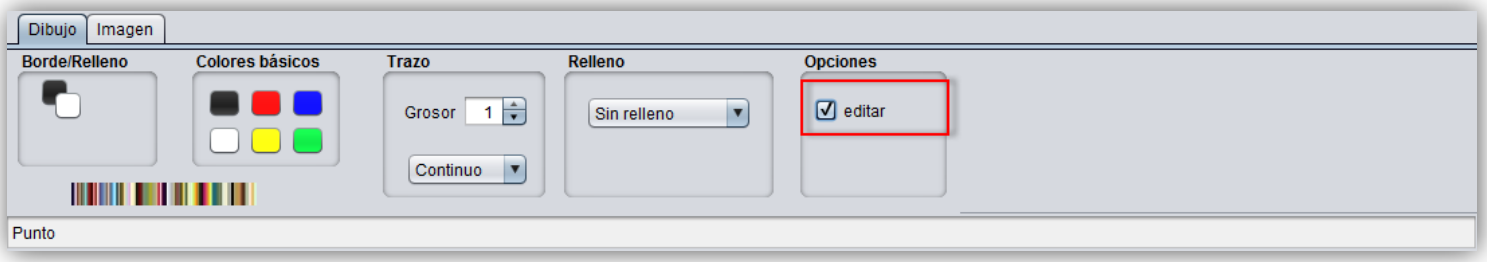
- **<u>Continuo/Discontinuo:</u>** Podremos elegir si nuestro trazo es continuo o discontinuo.

RELLENO



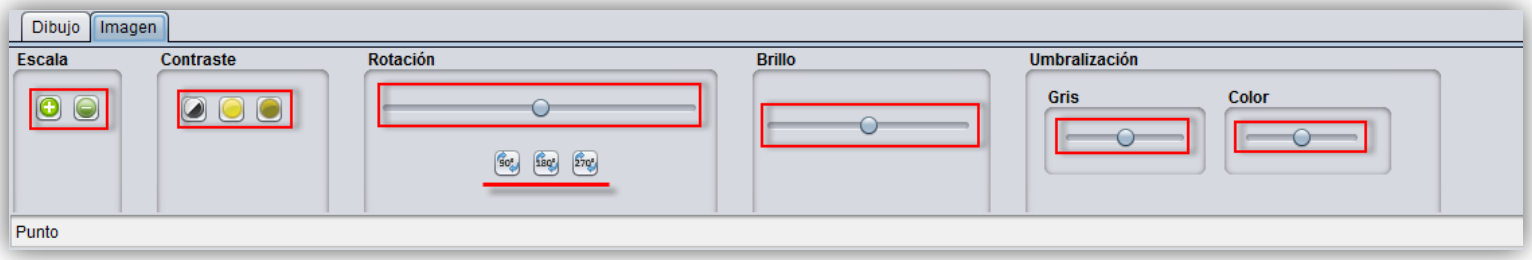
- **Relleno:** Aquí podemos seleccionar El tipo de relleno de la figura:
    - **Sin relleno:** Figura sin relleno (transparente) donde sólo se ve el borde, ya sea continuo o discontinuo y del color que indique la casilla de color de borde.
    - **Sólido:** Relleno liso y opaco del color que indica la casilla de color de relleno.
    - **Degradado horizontal:** Degradado horizontal a partir de los dos colores seleccionados como borde y relleno. El degradado comienza por la izquierda con el color de borde y acaba en la derecha de la figura con el color de relleno.
    - **Degradado vertical:** Degradado vertical a partir de los dos colores seleccionados como borde y relleno. El degradado comienza por arriba con el color de borde y acaba en la parte inferior de la figura con el color de relleno.

OPCIONES - EDITAR



- **Editar:** Al seleccionar la casilla “editar” entramos en el modo de edición de figuras donde no podremos pintar nuevas figuras si no que se destina a cambiar los atributos y posición de las que ya se pintaron.

IMAGEN



ESCALA

Aquí disponemos de dos botones (+, -) para aumentar o disminuir el tamaño de la imagen.

CONTRASTE

- **Contraste normal:** Se aplica una operación de aumento de contraste sobre la imagen seleccionada.

- **Contraste iluminado:** Aumento de contraste por aumento de la luminosidad en la imagen seleccionada.

- **Contraste oscurecer:** Aumento de contraste por la disminución de la luminosidad en la imagen seleccionada.

ROTACIÓN

- **Slider:** Podemos rotar nuestra imagen mediante un slider. Partiendo de la posición central si lo deslizamos a la izquierda la imagen rotará en sentido anti-horario hasta una rotación de 360º. Si deslizamos hacia la derecha la rotación será horaria hasta 360º. La imagen no se guardará hasta que salgamos del slider de rotación.

- **Rotación fija:** También podemos realizar una rotación fija en sentido de las agujas del reloj de 90º,180º y 270º respectivamente con los botones de rotación fija.

BRILLO

Para la modificación del brillo de la imagen tenemos un slider. La imagen tomará su valor inicial en el centro del slider. Cuando lo deslizamos hacia la izquierda disminuiremos el brillo, por el contrario cuando lo deslizamos a la derecha aumentaremos el brillo. Los extremos del slider darán como resultado una imagen negra o blanca respectivamente.

UMBRALIZACIÓN

La umbralización tiene como finalidad segmentar gráficos rasterizados, esto es , separar los objetos dependiendo de un valor umbral de sus píxeles. Se podrá aplicar una umbralización en color o en niveles de gris:

- **Gris:** Se umbraliza la imagen a partir de un valor dado por el slider en escala de grises.

- **Color:** Se umbraliza la imagen a partir de un valor dado por el slider correspondiente a las tres bandas de colores rgb.

DISEÑO DE CLASES

La aplicación cuenta con 16 clases que vamos a agrupar para facilitar la descripción de las mismas. De esta descripción se omiten las clases proporcionadas externamente para aportar funcionalidades a la aplicación, describiendo únicamente el diseño de las clases propias.

DIAGRAMA DE CLASES

***NOTA**: Para simplificar un poco los diagramas se han omitido algunos métodos o atributos que no son relevantes para el estudio y solución del problema.*

***NOTA2:** Adjunto al proyecto se incluye una carpeta (practicaFinal / UML) con imágenes más completas sobre estos diagramas que por razones de espacio no se incluyen en este pdf.*

FIGURAS.

Para cumplir el requisito de la aplicación que exponía que cada figura tendría que tener sus atributos propios se ha implementado una clase por cada figura. Como primer paso se optó por plantear las diferentes posibilidades que nos permitía la herencia de clases y la implementación de interfaces llegando a la conclusión de que teníamos dos opciones:

1.- Crear una clase RShape que herede de la clase abstracta Shape y cada clase de nuestras figuras heredara de nuestra clase RShape.

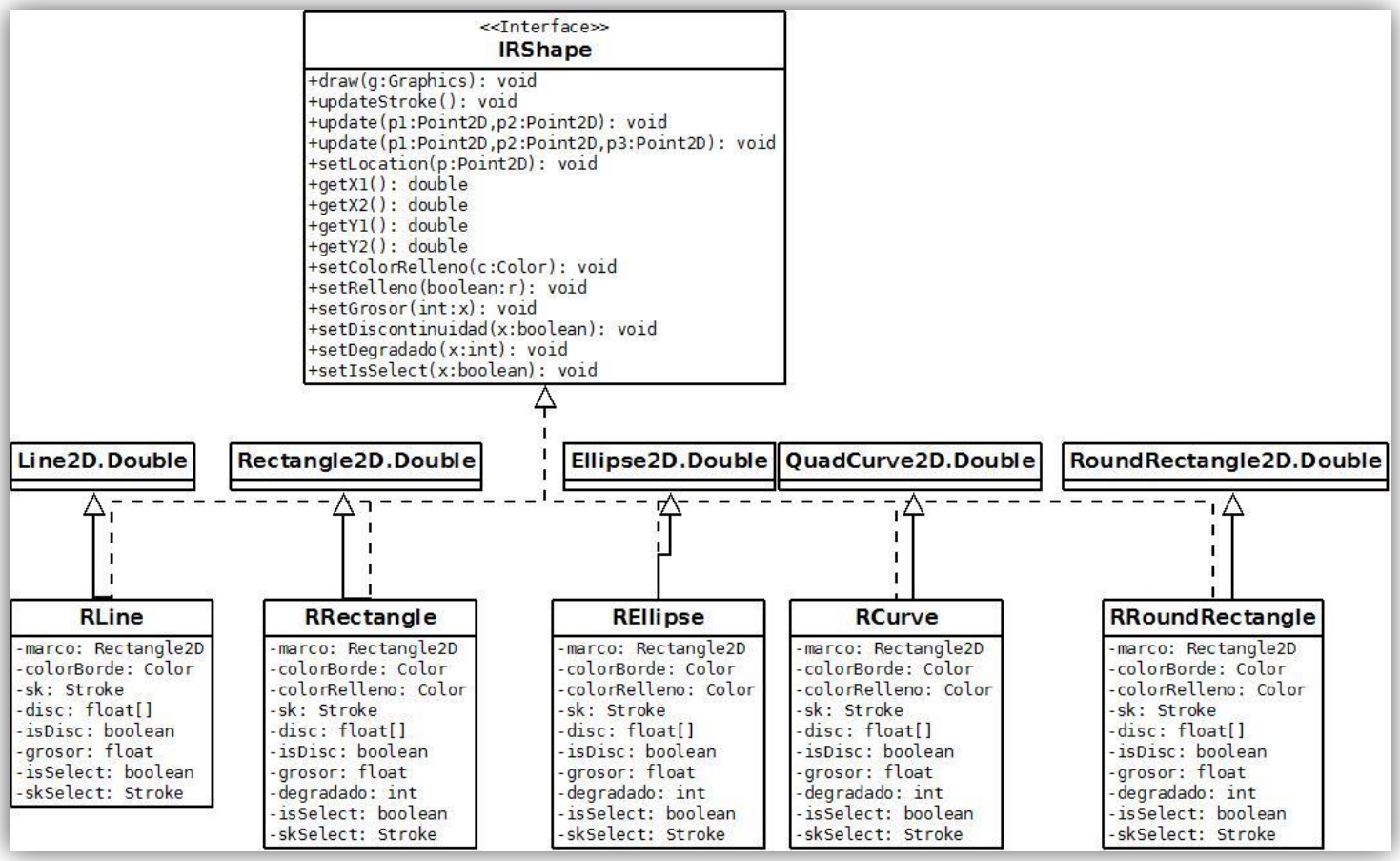
2.- Crear una interfaz IRShape con los métodos comunes y que cada clase de nuestras figuras implemente dicha interfaz.

Llegamos entonces a la siguiente solución:

Ya que todas las figuras van a compartir la mayoría de las funcionalidades se ha optado por crear una interfaz IRShape que nos obligará a incluir e implementar los métodos en cada una de las clases que implementen dicha interfaz, pero sólo aquellos que necesitemos para cumplir las especificaciones de nuestro problema.

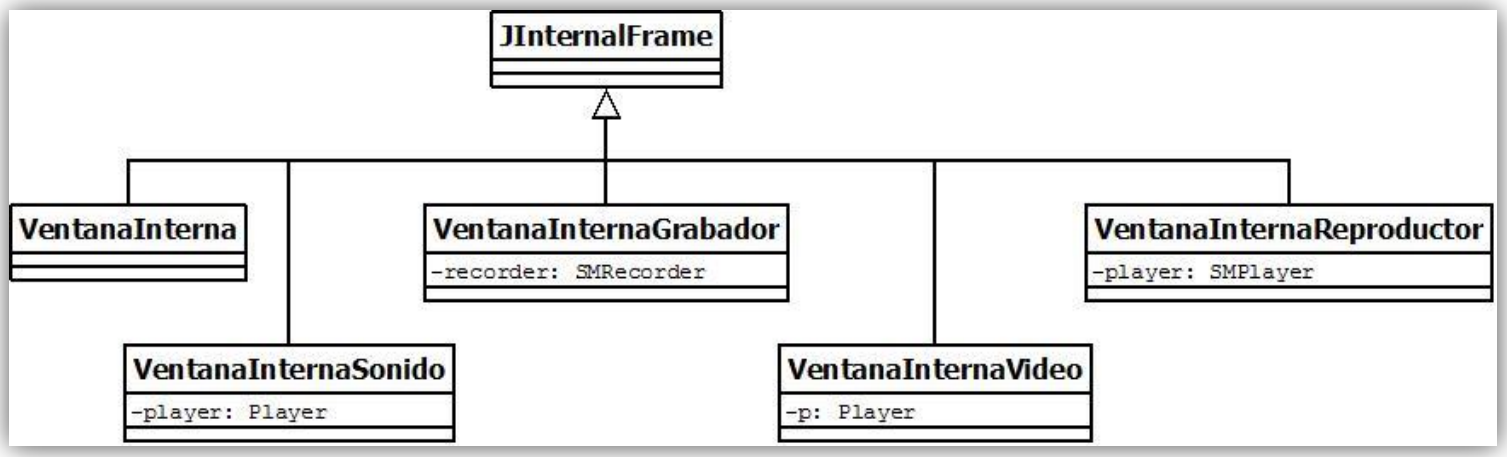
Esta forma de diseñar las clases tiene sus ventajas y desventajas frente al uso de clases abstractas. La principal desventaja que hemos encontrado ha sido que, como podemos ver en el siguiente diagrama de clases, los atributos deben implementarse privados en la clase de cada figura aun siendo comunes a todas las figuras ya que la interfaz no permite tener atributos. Este inconveniente se solventaría usando la opción de clases abstractas ya que sí

nos permite tener unos atributos en la clase padre, comunes a todas las figuras. Por otra parte, al usar esta interfaz no estamos obligados a implementar todos los métodos de la clase abstracta Shape, que quizá no sean necesarios para la resolución de nuestro problema. Podemos ver en la siguiente captura un diagrama de clases de nuestra solución.



VENTANAS.

Las ventanas que se usan para pintar, visualizar imágenes, reproducir vídeos, grabar sonidos y ver la webcam se han implementado como clases que heredan de JInternalFrame, como ya vimos con la ventana interna en las prácticas de la asignatura.

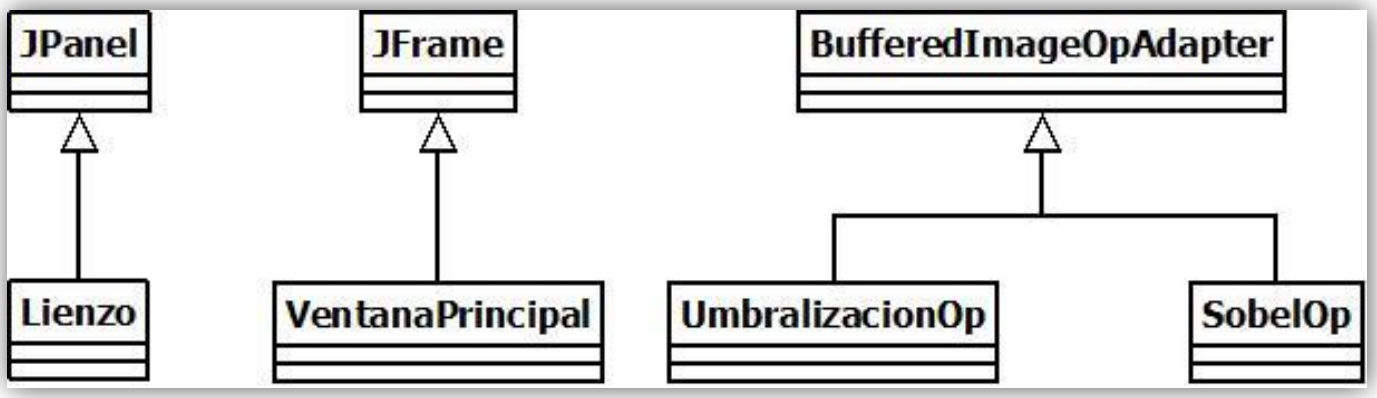


LIENZO, VENTANA PRINCIPAL Y OPERACIONES

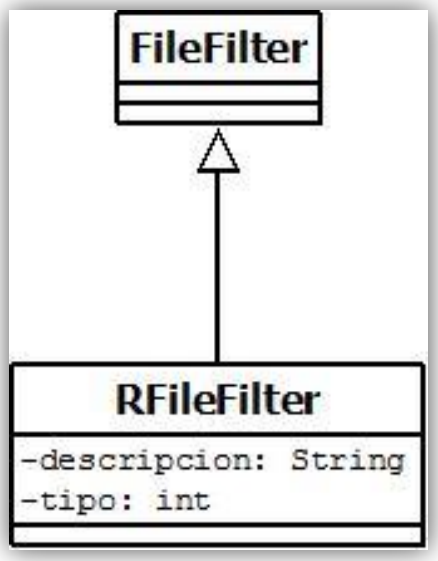
Se han implementado dos clases necesarias para operaciones como la umbralización o detección de bordes (sobel). Estas dos clases heredan de BufferedImageOpAdapter.

La ventana principal, donde podemos visualizar toda la interfaz, botones menús, el escritorio… es una clase que hereda de JFrame.

El lienzo que será nuestro espacio de dibujo hereda de JPanel.



FILTROS



La clase RFileFilter hereda de FileFilter y nos permite implementar los filtros de archivos que le pasaremos a JFileChooser para que sólo nos muestren los archivos que necesitamos, ya sean audio, video o imagen.

DESCRIPCIÓN DE LAS OPERACIONES.

Con el fin de aclarar el funcionamiento de alguna clase o método de clase así como la declaración de atributos se va a proceder a una descripción detallada de cómo se resuelven los requisitos del enunciado de la aplicación. Vamos a agrupar las funcionalidades de nuestra aplicación en:

- Funcionalidades de carácter general.
- Funcionalidades relacionadas con el dibujo.
- Funcionalidades relacionadas con las imágenes.
- Funcionalidades relacionadas con la grabación y reproducción de audio.
- Funcionalidades relacionadas con la captura de frames en vídeo y webcam.

FUNCIONALIDADES DE CARÁCTER GENERAL.

ABRIR (CTRL+A)

El botón abrir lanza un cuadro de diálogo donde podremos seleccionar los ficheros que nuestra aplicación es capaz de visualizar. Para ello hacemos uso de la clase RFileFilter.

Lo primero que haremos será crear tantos filtros como sean necesarios y aplicarlos sobre el cuadro de diálogo que vamos a lanzar.:

```
JFileChooser dlg = new JFileChooser();
RFileFilter filtroImagen= new RFileFilter("Archivos de imagen");
RFileFilter filtroSonido= new RFileFilter("Archivos de audio");
RFileFilter filtroVideo= new RFileFilter("Archivos de vídeo");
dlg.addChoosableFileFilter(filtroImagen);
dlg.addChoosableFileFilter(filtroSonido);
dlg.addChoosableFileFilter(filtroVideo);
dlg.setAcceptAllFileFilterUsed(false);
```

Si el usuario ha seleccionado un fichero se generará la ventana correspondiente (interna, sonido o vídeo) a dicho fichero:

```
if(dlg.getFileFilter()==filtroImagen){
        BufferedImage img = ImageIO.read(f);
        VentanaInterna vi = new VentanaInterna();
        …
      //Ventana Sonido
}else if(dlg.getFileFilter()==filtroSonido){
      VentanaInternaSonido vi;
      vi = VentanaInternaSonido.getInstance(f);
        …
      //Ventana Vídeo
}else if(dlg.getFileFilter()==filtroVideo){
        VentanaInternaVideo vi;
        vi = VentanaInternaVideo.getInstance(f);
        …
      }
```

NUEVO

Lanza una ventana de diálogo donde pide al usuario el tamaño de la ventanaInterna (ventana de dibujo) que se va a crear:

```
Object[] componentes ={
        "Indique el ancho (pixel).",w,
        "Indique el alto (pixel).",h
    };
JOptionPane.showConfirmDialog(this, componentes,"Introduzca el tamaño",
            JOptionPane.DEFAULT_OPTION);
```

Una vez obtenido el tamaño se crea la ventana interna:

```
VentanaInterna vi = new VentanaInterna();
    vi.setTitle("Nueva");
```

GUARDAR

Permite guardar al usuario su imagen retocada o su lienzo pintado con las figuras:

```
image = vi.getLienzo().guardarShape(image);
ImageIO.write(image, getExtension(f), f);
```

Como podemos ver se llama al método guardarShape de Lienzo. En este método volcaremos todas las figuras para poder guardarlas en la imagen:

```
 public BufferedImage guardarShape(BufferedImage image){
    Graphics2D g2d = image.createGraphics();
    for(IRShape s:vShape) {
       s.draw(g2d);
    }
    return image;
 }
```

GRABAR

Lanza una ventana para grabar audio pidiendo antes dónde guardar el audio que vamos a grabar. Para la grabación de audio se ha usado la biblioteca SM.

```
JFileChooser dlg = new JFileChooser();
      int resp = dlg.showSaveDialog(this);
      if( resp == JFileChooser.APPROVE_OPTION) {
           File f = dlg.getSelectedFile();
            VentanaInternaGrabador vi = new VentanaInternaGrabador(f);
```

Como podemos ver, la forma de trabajar lanzando cuadros de diálogo es similiar en todas las ocasiones.

1. Crear cuadro de diálogo (dlg).
2. Mostrar cuadro de diálogo (dlg).
3. Obtener fichero del cuadro de diálogo.
4. Procesar fichero.

Una vez creada la ventana interna tendremos la opción de empezar a grabar y parar la grabación. Los botones que no se puedan usar se desactivarán en este momento.

```
SMRecorder recorder;
private void recordActionPerformed(java.awt.event.ActionEvent evt) {
      if(recorder!=null) recorder.record();
   }
   private void stopActionPerformed(java.awt.event.ActionEvent evt) {
      if(recorder!=null) recorder.stop();
   }
```

Como podemos ver la biblioteca nos da opción a grabar y parar con el micrófono con un simple p.record(); y p.stop();

CÁMARA

Muestra una ventanaCamara con las imágenes captadas por la webcam. Debido a los diversos problemas con la compatibilidad de webcams con la JMF de java no se ha podido probar este módulo como se debería. Aunque se incluye, no se asegura su correcto funcionamiento en todos los equipos.

```
VentanaInternaCamara vi;
vi = VentanaInternaCamara.getInstance();
```

Como podemos intuir se usa un método público para acceder al constructor privado de la ventana.

```
 private Player p;
private VentanaInternaCamara() {
      initComponents();
      try {
          CaptureDeviceInfo deviceInfo;
          String dName="vfw:Microsoft WDM Image Capture (Win32):0";
          deviceInfo = CaptureDeviceManager.getDevice(dName);

         MediaLocator ml = deviceInfo.getLocator();
         p = Manager.createRealizedPlayer(ml);
         Component  areaVisual = p.getVisualComponent();
         if(areaVisual!=null) this.add(areaVisual,
               java.awt.BorderLayout.CENTER);
         Component  panelControl = p.getControlPanelComponent();
         if(panelControl!=null) this.add(panelControl,
               java.awt.BorderLayout.CENTER);
         p.start();
      }catch(Exception e){System.err.println("Error al crear la ventana de webcam");}

   }

   public static VentanaInternaCamara getInstance() {
      VentanaInternaCamara v= new VentanaInternaCamara();
      return v.p!=null?v:null;
   }
```

La creación de Player puede generar excepciones y por tanto no debería crearse la ventana. Si usamos un constructor estándar implicaría la creación de la ventana aun habiéndose generado la excepción. Es por esto que usamos el método getInstance(File f) que llame al constructor privado y en caso de error, devolverá null.

CAPTURA

Con la captura de imágenes podremos capturar un frame durante la reproducción de un vídeo o la visualización de la webcam. El proceso es muy simple ya que podemos obtener la imagen de un JInternalFrame mediante el método getSelectedFrame();

```
vv = (VentanaInternaVideo) escritorio.getSelectedFrame();
vi.getLienzo().setImageSource(vv.getFrame());
```

Igual para la captura de la webcam.

El método getFrame() de la clase VentanaInternaVideo / VentanaInternaCamara se encargará de obtener el frame que se está reproduciendo en ese momento y pasarlo a un formato imagen capaz de asignarse a la imagen de nuestro Lienzo.

```
public BufferedImage getFrame(){
    FrameGrabbingControl fgc;
    String claseCtr = "javax.media.control.FrameGrabbingControl";
    fgc = (FrameGrabbingControl)p.getControl(claseCtr );
    Buffer bufferFrame = fgc.grabFrame();
    BufferToImage bti;
    bti=new BufferToImage((VideoFormat)bufferFrame.getFormat());
    Image img = bti.createImage(bufferFrame);
    return (BufferedImage)img;
  }
```

VER

Tenemos la opción de seleccionar los paneles que queremos ver en la parte inferior de nuestra aplicación, ya sea el de dibujo como el destinado a imagen.

Simplemente tendremos que seleccionar el índice del panel inferior que corresponda.

```
  private void verDibujoActionPerformed(java.awt.event.ActionEvent evt) {
    if(verDibujo.isSelected()){
      barraHerramientas.setSelectedIndex(0);
      verImagen.setSelected(false);
    }
  }
```

```
private void verImagenActionPerformed(java.awt.event.ActionEvent evt) {
    if(verImagen.isSelected()){
        barraHerramientas.setSelectedIndex(1);
        verDibujo.setSelected(false);
    }
}
```

FUNCIONALIDADES RELACIONADAS CON EL DIBUJO.

Gracias al diseño de clases que se ha llevado a cabo la forma en que las figuras se dibujan es muy similar por lo que no vamos a entrar en detallar todas.

LÍNEA

Es una línea recta que únicamente necesita como parámetros 2 puntos. Sobre el lienzo tendremos 3 eventos de ratón a usar: **pressed, dragged y released.**

- Pressed -> presionado. Corresponde a cuando presionamos el botón izquierdo del ratón.
- Dragged -> arrastrado. Corresponde a cuando movemos el ratón con el botón izquierdo aún pulsado.
- Released -> liberado. Corresponde a cuando soltamos el botón izquierdo del ratón.

Haremos click en el lienzo y **crearemos la figura** con un único punto como punto inicial y final. Esta figura no será visible pues aún no tiene los dos puntos necesarios (aún no llamamos a “repaint();”). También añadiremos la figura a un array de IRShape donde almacenaremos todas las figuras.

Cuando sin soltar el ratón arrastramos, actualizaremos la forma creada en el Pressed, aportándole ahora el **segundo punto a la figura** y llamando a la función **repaint().**

Para crear y pintar esta figura no es necesario más que llamar al evento MouseDragged al soltar el botón izquierdo del ratón.

Nota: No se incluye el código completo para comprender y visualizar mejor la forma en que pintamos las figuras. Podemos consultar los ficheros fuentes para ver el código completo.

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
      Point2D point = evt.getPoint();
           …
           p = point;
           this.s = createShape(this.p,this.p);
           this.vShape.add(this.s);

 }
 private void formMouseDragged(java.awt.event.MouseEvent evt) {
           ….
         this.p2 = evt.getPoint();
         s.update(p,p2);
         repaint();
 }
 private void formMouseReleased(java.awt.event.MouseEvent evt) {
           …
           formMouseDragged(evt);
}
```

Ya tenemos creada una figura pero **¿cómo se ha pintado?** Nuestra clase Lienzo tiene un método público **paint(Graphics g).** En este método recorremos nuestro array de IRShape llamando al método draw propio de cada clase de figuras. No nos olvidemos de que nuestra aplicación es capaz de abrir una imagen y pintar sobre ella, por lo que tendremos que pintar la imagen, si la hubiese, antes de pintar las figuras, si no éstas quedarían tras la imagen.

```
public void paint(Graphics g){
     super.paint(g);
     Graphics2D g2d = (Graphics2D)g;
     /*Pintamos imagen*/
     if(img!=null)
        g2d.drawImage(img,0,0,this);

     /*Pintamos figuras*/
     for(IRShape s:vShape) {
        s.draw(g);
     }
  }
```

¿Y el método draw de cada figura?

El **draw** de cada figura se encargará de actualizar las propiedades de los trazos que tenemos seleccionadas en nuestra interfaz de la aplicación como el grosor, estilo de trazo, etc. Establecerá los colores de borde y relleno así como

el estilo de relleno(degradados) y por último pintará la figura llamando al método draw de Graphics2d.

```
@Override
public void draw(Graphics g){
    Graphics2D g2d = (Graphics2D)g;
    updateStroke();
    …
    g2d.setPaint(colorBorde);
    g2d.setStroke(sk);
    g2d.draw(this);
}
```

Como hemos dicho la mayoría de las demás figuras se utilizan igual. La única diferencia entre una línea y un rectángulo a la hora de construir el objeto será que los dos puntos que se envían a la clase RLine serán el inicio y fin de la línea, mientras que para construir el rectángulo le pasaremos los dos puntos que conforman la diagonal, de la misma forma para la elipse pero hay una forma que es un poco peculiar y es **la curva con punto** de control RCurve.

CURVA CON PUNTO DE CONTROL

La gran diferencia con nuestras otras figuras es que en esta figura necesitamos tres puntos en lugar de dos. Para la implementación de la clase RCurve (curva con punto de control) no conlleva grandes cambios, a excepción de los tres puntos en lugar de dos, sin embargo si tendremos que tener en cuenta algunas situaciones en los tres eventos de ratón de la clase Linezo.

Esta vez, deberemos controlar qué figura estamos pintando y si es el momento en que necesitamos el punto de control que creará la curva.

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
    Point2D point = evt.getPoint();

      if(this.s != null){
          s.setIsSelect(false);
      }
      if(getSelect()==CURVA && needPControl){
        if(vShape.get(vShape.size()-1) instanceof RCurve){
            vShape.get(vShape.size()-1).update(p, evt.getPoint(), p2);
        }
      }
```

```
        else{
            p = point;
            this.s = createShape(this.p,this.p);
            this.vShape.add(this.s);
        }
    }
```

```
private void formMouseDragged(java.awt.event.MouseEvent evt) {
        if(this.s != null){
            if(getSelect()==CURVA && needPControl){
                s.update(p,evt.getPoint(),p2);
            }else if(getSelect()==CURVA && !needPControl){
                this.p2 = evt.getPoint();
                s.update(p,p2,p2);
            }else{
                this.p2 = evt.getPoint();
                s.update(p,p2);
                needPControl = false;
            }
        }
        repaint();

    }
```

```
private void formMouseReleased(java.awt.event.MouseEvent evt) {
        if(getSelect()==CURVA && needPControl){
            if(s!=null){
                s.update(p,evt.getPoint(),p2);
                needPControl = false;
                repaint();
            }
        }else{
            needPControl = true;
            formMouseDragged(evt);
        }

    }
```

La primera vez “needPControl” que es la variable que controla si necesitamos un punto de control se encontrará a false, y crearemos unalínea

recta, ya que en la clase RCurve, si pasamos dos puntos, el punto de control se iguala al segundo punto, creando así la línea recta.

Una vez hayamos creado la línea con el Pressed y el Dragged es hora de pasarle el punto de control, ahora la variable “needPControl = true”, por lo que en los tres eventos del ratón le pasaremos a la clase RCurve el punto de control con:

```
update(p, evt.getPoint(), p2);
```

ATRIBUTOS Y EFECTOS DE LAS FIGURAS.

Como ya dijimos antes los atributos serán propios de cada objeto, y la forma de modificarlos es para todas las clases similar. Veamos por ejemplo como modificamos **el color y el trazo** de las figuras.

La mayoría de los atributos de nuestra clase Lienzo son “static” ya que todas las ventanas y figuras van a compartir el mismo espacio de interfaz. Esto se traduce en que si tenemos seleccionado el color rojo, las figuras las pintaremos de color rojo, y cuando cambiemos de figura seguiremos pintando rojo hasta que cambiemos de color, lo esperado.

Desde nuestra clase VentanaPrincipal (que recordamos es nuestra interfaz) indicamos a lienzo que vamos a pintar en rojo, y actualizamos el color seleccionado de la interfaz.

```
private void rojoActionPerformed(java.awt.event.ActionEvent evt) {
    Lienzo.setColor(Color.RED);
    updateColors(Color.RED);
    repaint();
  }
```

El método updateColors(Color c) comprueba qué zona tenemos seleccionada, si borde o relleno, y pinta el recuadro de ese color para hacer constancia al usuario qué color ha seleccionado.

```
private void updateColors(Color c){
    if(Lienzo.getSelectColor()==BORDE)
      colorBorde.setBackground(c);
    else
      colorRelleno.setBackground(c);
     }
```

Una vez hemos cambiado y personalizado nuestra figura vamos a dibujarla, entonces en nuestra clase Lienzo() crearemos la figura (CreateShape(p,p2)) y le pasaremos estos atributos a la clase RRectangle(), por ejemplo. Lo vemos aquí:

```
private IRShape createShape(Point2D p1, Point2D p2){
        ...
  this.s = new RRectangle(p1, p2);break;
        ....
  if(s!=null){
        s.setColorRelleno(colorRelleno);
       s.setColorBorde(colorBorde);
       s.setRelleno(relleno);
       s.setGrosor(grosor);
       s.setDiscontinuidad(disc);
       s.setDegradado(degradado);
  }
  return this.s;
  }
```

Ya solo queda utilizar el método draw() de cada figura como ya vimos antes, y es aquí en cada clase donde aplicamos estos atributos a las formas:

```
@Override
  public void draw(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;
    updateStroke();
...
    if (relleno){
      g2d.setPaint(colorRelleno);
      if (degradado==1){
        Point2D pg1= new Point2D.Double(this.getX(),0);
        Point2D pg2= new Point2D.Double(this.getX()+getWidth(),0);
        GradientPaint gradient = new GradientPaint(pg1, colorBorde,
                                  pg2, colorRelleno);
        g2d.setPaint(gradient);
      }else if(degradado == 2){
        Point2D pg1= new Point2D.Double(0,this.getY());
        Point2D pg2= new Point2D.Double(0,this.getY()+getHeight());
        GradientPaint gradient = new GradientPaint(pg1, colorBorde,
                                  pg2, colorRelleno);
        g2d.setPaint(gradient);
      }
      g2d.fill(this);
    }
```

```
        g2d.setPaint(colorBorde);
        g2d.setStroke(sk);
        g2d.draw(this);
    }
```

Aquí vemos cómo comprobamos el tipo de relleno que tenemos, y utilizamos el método updateStroke() para actualizar la brocha del borde antes de dibujar.

Las opciones para el borde son continuo o discontinuo:

```
@Override
    public void updateStroke(){
        if(isDisc){
        sk = new BasicStroke(
                    grosor,                 // grosor
                    BasicStroke.CAP_BUTT,     // terminación: recta
                    BasicStroke.JOIN_ROUND,   // unión: redondeada
                    1f,                     // ángulo: 1 grado
                    disc,         // línea, espacio, línea , espacio
                    2                       // fase
            );
        }else
            sk = new BasicStroke(grosor);
    }
```

Si es continuo creamos un BasicStroke solamente con el grosor. En cambio si es un borde discontinuo tenemos un stroke más elaborado.

FUNCIONALIDADES RELACIONADAS CON LAS IMÁGENES.

Para las imágenes en los lienzos no se ha implementado ninguna clase propia, ya que no se les han dotado de atributos propios a cada objeto. Las imágenes estarán ligadas al lienzo, teniendo este un objeto privado BufferedImage img; que consultaremos y modificaremos desde la interfaz, Lienzo(), con sus métodos get() y set().

Las **operaciones sobre imágenes** se aplican de forma muy similar y se aplican desde VentanaPrincipal() directamente en la mayoría de los casos:

1. Obtenemos la imagen de Lienzo (por ejemplo mediante getImage();)
2. Definimos el tipo de filtro para realizar la operación sobre la imange.
3. Aplicamos el filtro, devolviéndonos la imagen ya modificada.
4. Actualizar la imagen de Lienzo mediante el método setImage(image);

Para aplicar los filtros mediante un slider, como por ejemplo es el caso del brillo, es necesario realizar los cambios sobre una imagen fuente, por lo que en la clase lienzo vamos a definir dos imágenes, una imagen fuente y una imagen actual:

```
private BufferedImage img;
private BufferedImage imgSource;
```

Vamos a ver por ejemplo la operación de aumento y disminución de brillo con un slider.

```
private void brilloStateChanged(javax.swing.event.ChangeEvent evt) {
    VentanaInterna vi = (VentanaInterna) (escritorio.getSelectedFrame());
    if (vi != null) {
      if(vi.getLienzo().getImageSource()!=null){
        try{
          RescaleOp rop = new RescaleOp(1.0F, brillo.getValue(), null);
          BufferedImage imgdest = rop.filter(vi.getLienzo().getImageSource(), null);
          vi.getLienzo().setImage(imgdest);
          vi.getLienzo().repaint();
        } catch(IllegalArgumentException e){
          System.err.println(e.getLocalizedMessage());
        }
      }
    }
  }
```

En esta operación podemos ver claramente cómo aplicamos los pasos que hemos descrito antes.

```
1.      VentanaInterna vi = (VentanaInterna) (escritorio.getSelectedFrame());
2.          RescaleOp rop = new RescaleOp(1.0F, brillo.getValue(), null);
3.          BufferedImage imgdest = rop.filter(vi.getLienzo().getImageSource(), null);
4.  vi.getLienzo().setImage(imgdest);
```

Vemos cómo en el paso 2 obtenemos el valor del slider con brillo.getValue().

En los ficheros fuente podemos consultar las diferentes operaciones que permite realizar la práctica, pero por simplificar la documentación y dada la similitud entre ellas, no se van a describir detalladamente todas.

FUNCIONALIDADES RELACIONADAS CON LA GRABACIÓN Y REPRODUCCIÓN DE AUDIO/VÍDEO.

Para la reproducción de video y audio se usa la biblioteca JMF, que nos proporciona un nivel de abstracción muy alto hasta el punto que con solo crear un objeto podremos crear un panel de reproducción (Component) o un área para visualizar el vídeo.

Para abrir una ventana de reproducción le pasamos como parámetro un fichero al constructor, fichero que vamos a cargar en el reproductor.

Para crear la barra de reproducción añadiremos un objeto Component, y lo colocaremos en el sur del InternalFrame, el área de visualización lo crearemos de la misma manera y lo colocaremos en el centro. Esta orientación a los componentes se la damos ya que la ventana interna tiene como Layout un Border Layout.

```
private Player p = new MediaPlayer();
private VentanaInternaVideo(File f) {
    initComponents();
    try{
      MediaLocator ml = new MediaLocator("file:"+f.getAbsolutePath());
      p = Manager.createRealizedPlayer(ml);
      Component  areaVisual = p.getVisualComponent();
      if(areaVisual!=null) this.add(areaVisual,java.awt.BorderLayout.CENTER);
      Component  panelControl = p.getControlPanelComponent();
      if(panelControl!=null) this.add(panelControl,java.awt.BorderLayout.SOUTH);
      this.pack();
    }catch(Exception e){System.err.println("Error al crear la ventana de vídeo");}
 }
```

Una vez creada la ventana ya podemos reproducir, y tan fácil como darle al play y stop.

Podemos asociar eventos a esta ventana, como parar la reproducción cuando se cierre la ventana o iniciarla al abrirla. Hay que tener en cuenta que si no lo indicamos explícitamente, la reproducción seguirá su curso al cerrar la ventana.

Para ello usaremos los siguientes eventos relacionados con el InternalFrame:

```
private void formInternalFrameClosing(javax.swing.event.InternalFrameEvent evt) {
        p.stop();
}
```

```
private void formInternalFrameOpened(javax.swing.event.InternalFrameEvent evt) {
        p.start();
}
```

Esta descrita es la ventana para vídeo. Como en la reproducción de audio también usamos la JMF la diferencia será que no tendremos el componente areaVisual.

```
Component  areaVisual = p.getVisualComponent();
        if(areaVisual!=null) this.add(areaVisual,java.awt.BorderLayout.CENTER);
```

No es así para la ventana de **grabación de audio**, ya que para esta hemos usado SM. Aunque no es la misma biblioteca si que sigue el mismo nivel de abstracción aquí podemos ver el constructor:

```
SMRecorder recorder;
  public VentanaInternaGrabador(File f) {
    initComponents();
    recorder = new SMSoundPlayerRecorder(f);
  }
```

Aquí los componentes hay que implementarlos, para nuestra aplicación únicamente hemos necesitado un botón para iniciar a grabar y otro para parar. A estos botones les hemos asignado los **eventos** necesarios:

```
private void recordActionPerformed(java.awt.event.ActionEvent evt) {
    if(recorder!=null) recorder.record();
    record.setEnabled(false);
    stop.setEnabled(true);
  }//GEN-LAST:event_recordActionPerformed
```

```
  private void stopActionPerformed(java.awt.event.ActionEvent evt) {//GEN-
FIRST:event_stopActionPerformed
    if(recorder!=null) recorder.stop();
    stop.setEnabled(false);
    record.setEnabled(true);
  }//GEN-LAST:event_stopActionPerformed
```

BIBLIOGRAFÍA

- http://javapiola.blogspot.com.es/2009/11/tutorial-de-jfilechooser.html
- http://stackoverflow.com/questions/5603966/how-to-make-filefilter-in-java
- http://www.um.es/geograf/sigmur/teledet/tema06.pdf
- http://www.oracle.com/technetwork/java/javase/documentation/index-137868.html
- http://www.adictosaltrabajo.com/tutoriales/tutoriales.php?pagina=javadoc