Sistemas Multimedia (2014-2015)
Grado en Ingeniería Informática
Universidad de Granada

Memoria Práctica Final

Francisco Javier Garrido Mellado

8 de septiembre de 2015

Índice

Índice de figuras

1. Introducción

En el presente documento se explican los pasos seguidos para la elaboración de la práctica Final de Sistemas Multimedia. Esta práctica consiste en dar continuidad a las practicas anteriores añadiendole mas funcionalidad, para ello es necesario la elaboración de clases propias. La aplicación cuenta con un entorno multiventa siendo *VentanaPrincipal* la ventana que cuenta con las diferentes barras de herramientas. **Nota: El proyecto UML se ha realizado con Netbeans.**

2. Funcionalidad y manejo de la interfaz

En este apartado se explica la funcionalidad y uso de cada barra de herramientas de la interfaz.

2.1. Menú Archivo

En este menú se encuentra las siguientes opciones en orden respectivo:

- **Nuevo** -Abre un nuevo lienzo con el tamaño especificado.
- **Abrir** -Abre una imagen, video o audio.
- **Guardar** - Guarda la imagen en el formato especificado[5].
- **Abrir Audio** -Abre archivo de audio con formato wav, au o aif.Usa Java Sound API.
- **Guardar Audio** -Guarda archivo de audio.Primero se introduce nombre del archivo y despues se lanza VentanaInternaGrabador.
- **Ventana Reproduccion** -Usa JMF API y permite abrir archivos de audio y video.
- **Ventana Camara** - Usa tambien JMF API y permite visualizar lo que recoge la webcam.
- **Ventana Captura** - Realiza captura de lo que muestra la webcam o la ventana de reproducción.

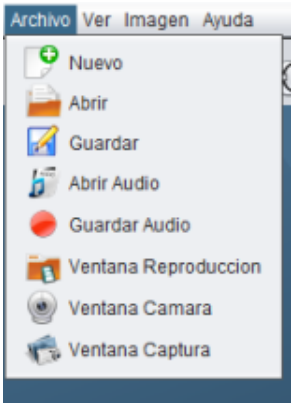

Figura 2.1: Menu Archivo

2.2. Menú Ver

En este menú hay cuatro opciones de visualización:

- **Barra de estados** -Muestra/oculta los textlabel que indican la selección de la forma,de borde y relleno.
- **Barra de formas** -Muestra/oculta los botones para seleccionar forma geométrica y editar.
- **Barra de atributos** - Muestra/oculta la barra para especificar color, grosor,alisado,etc.
- **Barra de atrib. de imagen** -Muestra/oculta la barra de herramientas de imágenes.

2.3. Menú Imagen

En este menú se encuentra las siguientes opciones en orden respectivo:

- **RescaleOp** -A cada componente de un pixel le realiza una transformación.
- **ConvolveOp** -Aplica mascara de convolución.
- **AffineTransformOp** - Realiza una transformación afín.
- **LookupOp** -Realiza una transformacion T sobre los niveles de color.
- **BandCombineOp** -Realiza una combinación lineal de las bandas de una imagen.
- **ColorConvertOp** -Realiza una conversión del espacio de color pixel a pixel.
- **Duplicar** - Realiza un duplicado de una imagen.
- **Negativo** - Realiza negativo de una imagen.

- **ConversionGris** -Transforma una imagen en color a escala de grises.
- **TintadoOp** -Realiza un tintado de la imagen segun el color seleccionado en la paleta de colores.
- **Transf.Logaritmo** -Aplica operación de transformacion logaritmica a una imagen.
- **Raiz n-esima** - Aplica operación de transformacion raiz n-esima a una imagen.
- **Potencia n-esima** - Aplica operación de transformacion de potencia n-esima a una imagen.
- **EliminarMitadBandaCentral** -Se reduce la banda central de RGB a la mitad.

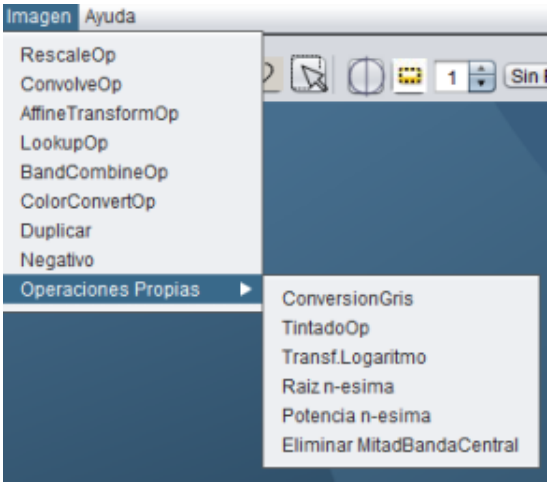

Figura 2.2: Menu Imagen

2.4. Menú Ayuda

En este menú se encuentra la siguiente opción:

- **Acerca de** -Muestra autor, nombre del programa y su version

2.5. Barra archivo

En este menú se encuentra las siguientes opciones en orden respectivo(cumplen la misma función que en menu archivo) :

- **Nuevo** .
- **Abrir**
- **Guardar** .

Figura 2.3: Barra Archivo

2.6. Barra de formas

En este menú se encuentra las siguientes opciones en orden respectivo:

- **Punto** -Selección del punto como forma geométrica.
- **Linea** -Selección de la linea como forma geométrica.
- **Rectángulo** -Selección del rectangulo como forma geométrica.
- **Elipse** -Selección de la elipse como forma geométrica.
- **Rectángulo Redondeado** -Selección del rectángulo redondeado como forma geométrica.
- **Curva** -Selección de la curva como forma geométrica.
- **Editar** - Selección de editado de las formas geométricas, al seleccionar esta opción posteriormente se selecciona los atributos deseados y por último la figura a la que se le aplicará, pudiendose mover a otra posición del lienzo tambien.

Figura 2.4: Barra Formas

2.7. Barra de atributos

En este menú se encuentra las siguientes opciones en orden respectivo:

- **Alisado** -Selección del modo alisado al dibujar la figura geométrica.
- **Discontinuidad** -Establece el trazo como discontinuo.
- **Grosor** -Establece el grosor del trazo.
- **Relleno** -Establece tipo de relleno solido, degradado horizontal, degradado vertical o sin relleno.
- **Transparencia** -Establece transparencia de un 25, 50 o 75 por ciento o sin transparencia.
- **Color** -Permite elegir color, se permite elegir entre una paleta de colores.

Figura 2.5: Barra Atributos

2.8. Barra Multimedia

En este menú se encuentra las siguientes opciones en orden respectivo:

- **Abrir Multimedia** -Permite abrir archivo de audio o video.
- **WebCam** -Permite visualizar en pantalla lo que recoge la webcam.
- **Capturar** -Captura instantanea del reproductor multimedia o de la webcam.
- **Grabar audio** -Permite grabar audio al igual que se explicó en el menu Archivo.
- **Abrir Multimedia con Barra de Progreso** -Permite reproducir archivo de audio adjuntandose una barra de progreso.
- **Reproductor de Audio con Barra de Progreso** -Misma funcionalidad que antes pero sin lanzar una ventana, esta se desarrolla en la VentanaPrincipal.

Figura 2.6: Barra Multimedia

2.9. Barra de Atributos de Imagen

En este menú se encuentra las siguientes opciones en orden respectivo:

- **Brillo** -Permite modificar el brillo de una imagen con un deslizador.
- **Filtro** -Permite aplicar un filtro a la imagen entre un abanico de cinco posiblidades(Media, Binomial, Enfoque, Relieve y Frontera Laplaciana).
- **Contraste** -Permite las opcines de dar contraste, iluminar u oscurecer una imagen.
- **Seno** -Aplica filtro seno a una imagen.
- **Rotación**- Permite rotar una imagen usando un deslizador o botones con rotaciones fijas de 90,180 o 270 grados.
- **Escala**- Permite escalar una imagen usando un factor de 1.25 (aumentar) o 0.75 (disminuir).
- **Umbralizacion Color y Gris**- Permite umbralizar una imagen a escala de grises o al color seleccionado en la paleta de colores.

- **Binarias**- Realiza las operaciones de Blending para mezclar imágenes o de resta para establecer diferencias.

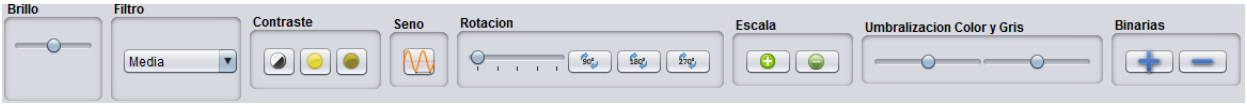

Figura 2.7: Barra Atributo Imagen

3. Entorno multiventana.

Se ha procedido a usar una Ventana Principal como principal interfaz de la practica,dicha clase *VentanaPrincipal* hereda de *JFrame* y contiene las barras de herramientas para asignar forma geométrica, atributos a las formas geométricas, efectuar operaciones sobre imágenes, y las relativas a la apertura de audio y video.

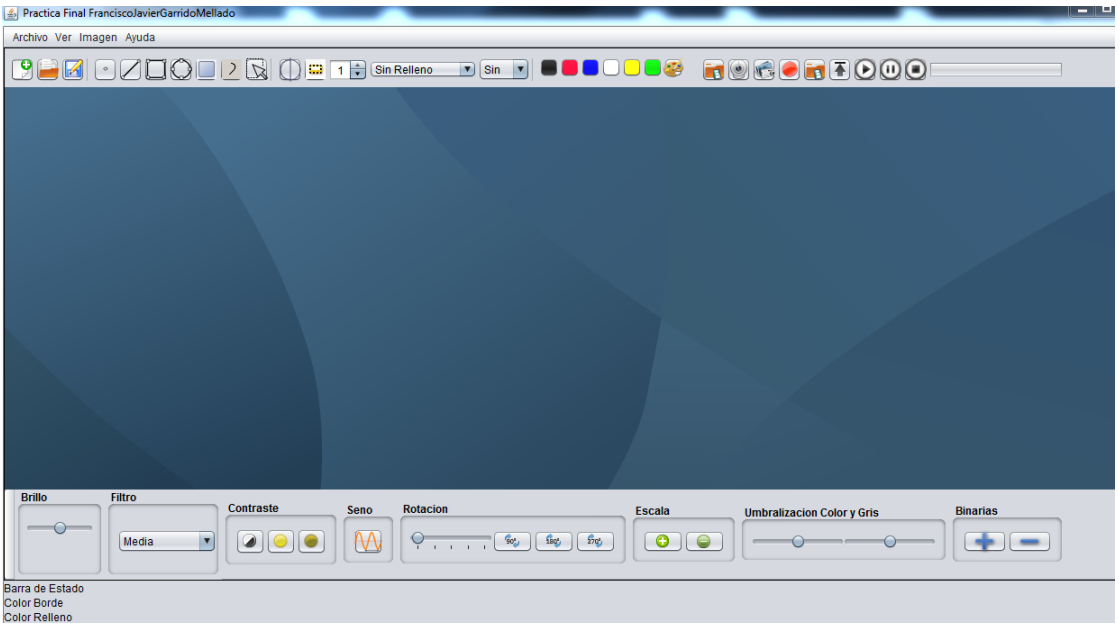

Figura 3.1: Interfaz

Al efectuar una apertura de imagen o bien abrir un nuevo lienzo se lanza la ventana de imagen que no es otra cosa que la clase **VentanaInterna** que 'contiene' la clase **Lienzo2DImagenfinal** y es donde se pinta y se realizan las operaciones de imágenes. Por otra parte, se dispone de otras cuatro ventanas que son usadas para todo lo relacionado con audio o video:

- **VentanaInternaCamara** - Ventana que muestra lo que recoge la webcam.
- **VentanaInternaGrabador** - Ventana para grabar audio.

- **VentanaInternaJMFPlayer** - Ventana para reproducir audio o video.
- **VentanaInternaReproductor** - Ventana para reproducir audio.
- **VentanaInternaReproductorHebra** - Ventana para reproducir audio con el añadido de una barra de progreso lanzada con una hebra.

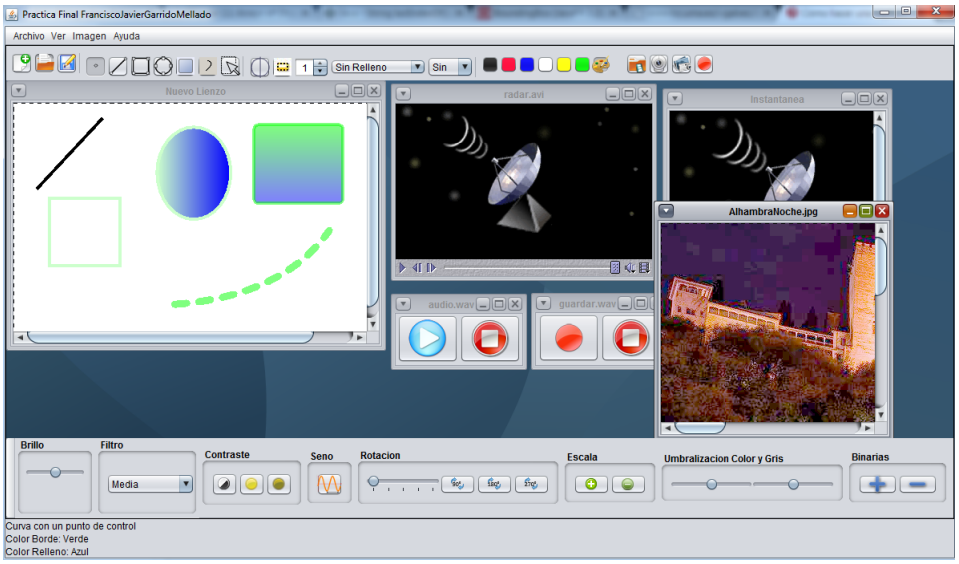

Figura 3.2: Interfaz con todas las ventanas

3.1. Ventana Principal

Como se ha comentado esta ventana recoge las barras de herramientas para seleccionar la forma geómetrica con la que se dibuja, sus atributos como grosor, tipo de transparencia, alisado, etc, operaciones de imágenes como rotar, umbralización por color o niveles de gris, abrir un nuevo video, etc.Además se dispone de cuatro menus:

- **Archivo**- Para abrir un lienzo nuevo, una nueva imagen, el reproductor de audio, grabar audio, una nueva reproducción multimedia o la webcam.
- **Ver**- Para dejar a la vista las diferentes barras de herramientas y los textlabel.
- **Imagen**- Recoge diferentes operaciones sobre imágenes incluyendo las de cosecha propia.
- **Info**-Muestra mi nombre y la versión de la práctica.

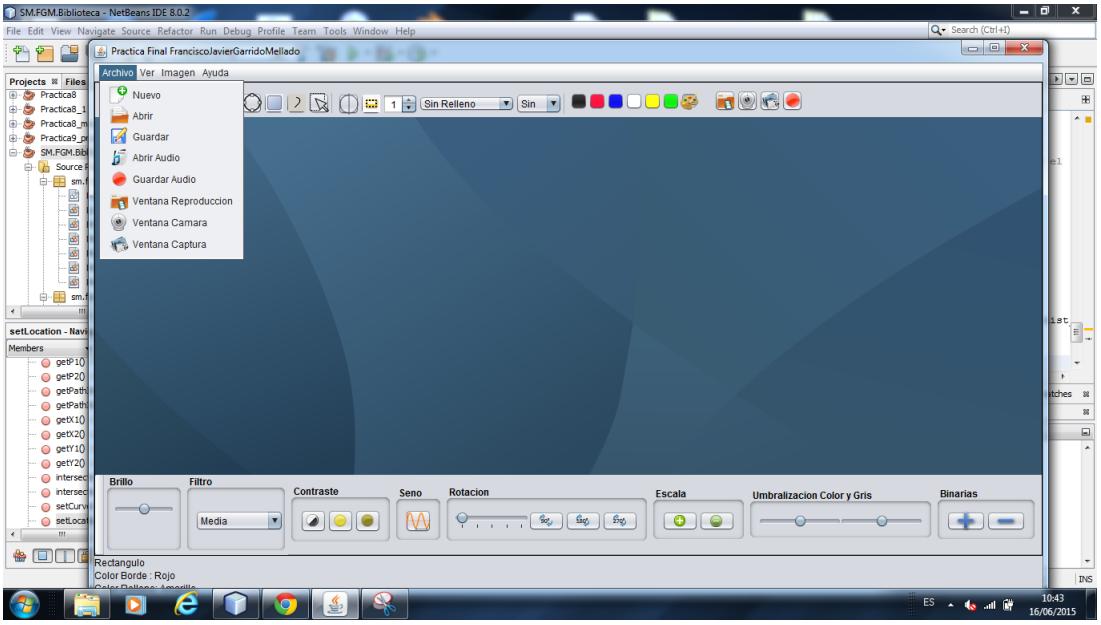

Figura 3.3: Menu de Archivo

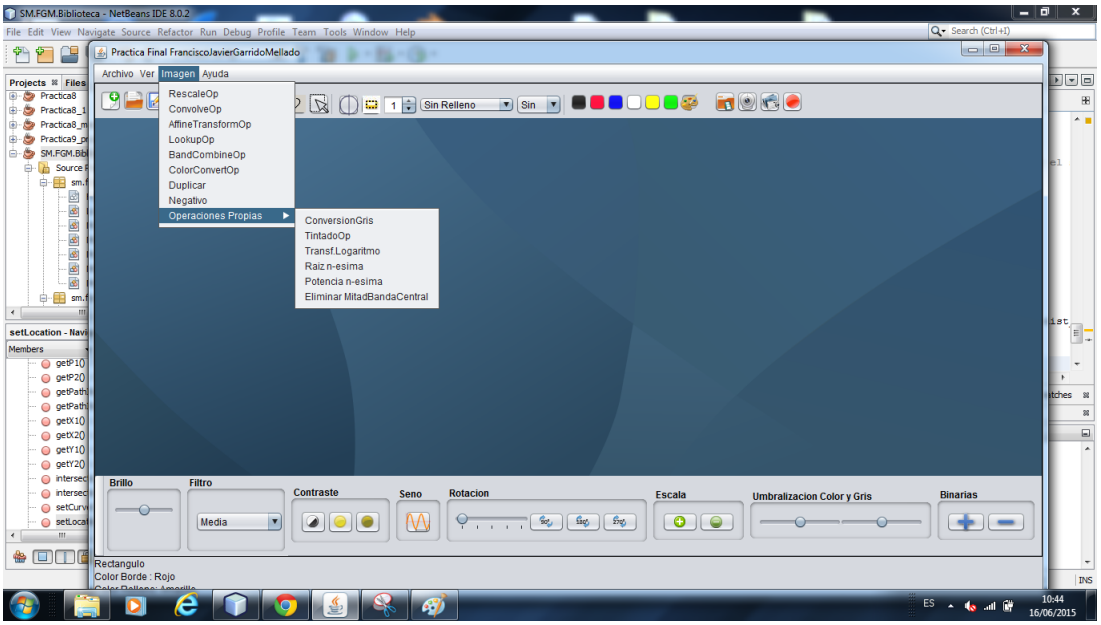

Figura 3.4: Menu de Imagen

Realmente lo que hace esta y el resto de ventanas es recoger un evento y efectuar la operación correspondiente (usando el objeto lienzo correspondiente que es donde se carga el dibujo o imagen).Para su diseño se ha heredado de *javax.swing.JFrame*.(Nota:Debido

a lo amplio que es el UML de VentanaPrincipal se recomienda abrir el proyecto UML con Netbeans o la imagen que esta alojada en la carpeta images de Latex_Final).

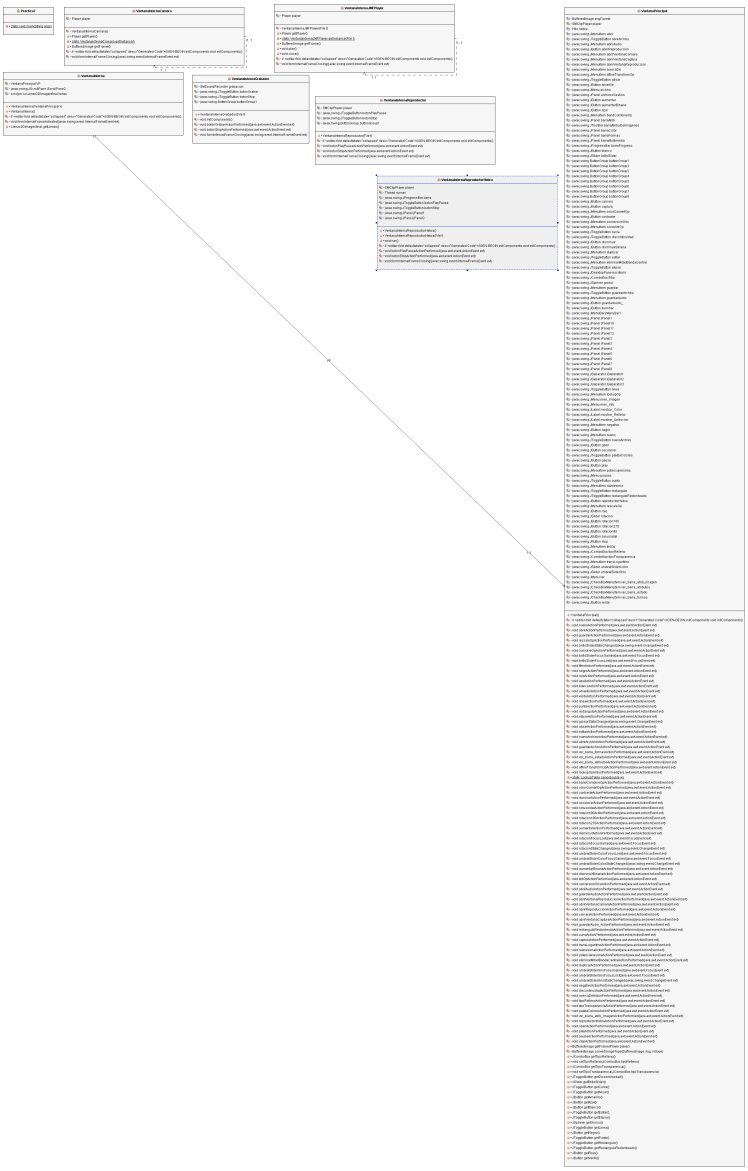

Figura 3.5: UML Ventanas

De los métodos de Ventana Principal destaco los siguientes:

- **private void nuevoArchivoActionPerformed(java.awt.event.ActionEvent evt)**- Para abrir un lienzo nuevo con el tamaño especificado.
- **private void abrirArchivoActionPerformed(java.awt.event.ActionEvent evt)**- Para abrir un fichero, segun la extensión se abre una ventana u otra.

- **private void tintOpActionPerformed(java.awt.event.ActionEvent evt)**- Realiza un tintado de la imagen segun el color seleccionado en la paleta de colores, se usa clase propia para realizar la operación.
- **private void conversionGrisActionPerformed(java.awt.event.ActionEvent evt)**-Convierte a niveles de gris una imagen usando unos factores de conversión.Usa clase propia.
- **private void abrirVentanaCapturaActionPerformed(java.awt.event.ActionEvent evt)**- Realiza una captura de pantalla del video o de la webcam.
- **private void raiznesimaActionPerformed(java.awt.event.ActionEvent evt)**- Aplica una operación de raiz nesima a una imagen, se ha usado clase propia, ademas se dispone de otras operaciones similares.

Pueden comentarse muchos mas como la umbralizacion por escala de grises donde se usa una clase propia, la paleta de colores, etc.

```
private void nuevoArchivoActionPerformed(java.awt.event.ActionEvent evt) {
        VentanaInterna vi = new VentanaInterna(this);

        JTextField ancho= new JTextField("300");
        JTextField alto= new JTextField("300");
        JFileChooser dlg = new JFileChooser();

        Object[] ancho_alto ={ "Ancho ",ancho,"Alto",alto };
        JOptionPane.showConfirmDialog(this, ancho_alto,"Introduzca
        anchura y altura",JOptionPane.DEFAULT_OPTION);


        escritorio.add(vi);
        vi.setVisible(true);
        BufferedImage img;
        img = new BufferedImage(Integer.parseInt(ancho.getText()),
        Integer.parseInt(alto.getText()),BufferedImage.TYPE_INT_RGB);

        img.createGraphics().setPaint(Color.WHITE);//permite pintar en el
        img.createGraphics().fill(new Rectangle2D.Float(0,0,img.getWidth(),
        img.getHeight()));
        Lienzo2DImagenfinal li= (Lienzo2DImagenfinal)vi.getLienzo();
        li.setImage(img);
        }

 private void abrirArchivoActionPerformed(java.awt.event.ActionEvent evt) {
JFileChooser dlg = new JFileChooser();
        FileNameExtensionFilter filter = new
```

```
        FileNameExtensionFilter("Imagenes", ImageIO.getWriterFormatNames());

        dlg.addChoosableFileFilter(filter);

        String extension[]={"wav","mp3","ogg","avi","mpg"};
        FileNameExtensionFilter filtro = new
        FileNameExtensionFilter("Videos y Audio",extension);
        dlg.addChoosableFileFilter(filtro);

        dlg.setFileFilter(filter);
        dlg.setFileFilter(filtro);
        dlg.setAcceptAllFileFilterUsed(false);


            int resp = dlg.showOpenDialog(this);
            if( resp == JFileChooser.APPROVE_OPTION) {

                File f = dlg.getSelectedFile();

                if(dlg.getFileFilter()==filter){
                    try{
                      BufferedImage img = ImageIO.read(f);
                      VentanaInterna vi = new VentanaInterna(this);
                      vi.getLienzo().setImage(img);
                      this.escritorio.add(vi);
                      vi.setTitle(f.getName());
                      vi.setVisible(true);
                    } catch(Exception ex){
                    System.err.println("Error al leer la imagen");
                     }
                  }else if(dlg.getFileFilter()==filtro){
                      try{
                    JInternalFrame vi =
                    VentanaInternaJMFPlayer.getInstance(f);
                    this.escritorio.add(vi);
                    vi.setTitle(f.getName());
                    vi.setVisible(true);
                      }catch(Exception ex){
                      System.err.println("Error al abrir video");
                     }
                }


}
```

```
    }

private void tintOpActionPerformed(java.awt.event.ActionEvent evt) {
        JInternalFrame vi = escritorio.getSelectedFrame();
        if ((vi != null)&&(vi instanceof VentanaInterna)) {
            BufferedImage imgSource =
            ((VentanaInterna)vi).getLienzo().getImage();
        if(imgSource!=null){

        try{

            TintadoOp top= new TintadoOp(
            ((VentanaInterna)vi).getLienzo().getColor(),0.5F);

            BufferedImage imgdest = top.filter(imgSource, null);
            ((VentanaInterna)vi).getLienzo().setImage(imgdest);
            ((VentanaInterna)vi).getLienzo().repaint();
          }catch(IllegalArgumentException e){
           System.err.println(e.getLocalizedMessage());
          }
        }
        }
    }

private void conversionGrisActionPerformed(java.awt.event.ActionEvent
evt) {
        JInternalFrame vi = escritorio.getSelectedFrame();
        if ((vi != null)&&(vi instanceof VentanaInterna)) {
            BufferedImage imgSource =
            ((VentanaInterna)vi).getLienzo().getImage();
        if(imgSource!=null){

            try{
              ConversionGrisOp gop = new ConversionGrisOp();

            BufferedImage imgdest = gop.filter(imgSource, null);
            ((VentanaInterna)vi).getLienzo().setImage(imgdest);
            ((VentanaInterna)vi).getLienzo().repaint();
        } catch(IllegalArgumentException e){
            System.err.println(e.getLocalizedMessage());
}
}
}
```

```
 private void abrirVentanaCapturaActionPerformed(
                    java.awt.event.ActionEvent evt) {
         VentanaInterna vi= new VentanaInterna(this);
         JInternalFrame vic = escritorio.getSelectedFrame();
         if(vic instanceof VentanaInternaCamara){
           try{
             vi.getLienzo().setImage(this.getFrame(
             ((VentanaInternaCamara)vic).getPlayer()));
             vi.setTitle("Instantanea");
             vi.setLocation(20+vi.getWidth(),0);
             vi.setVisible(true);
             this.escritorio.add(vi);
             }
           catch (Exception ex)
            {
         System.err.println("Error 1");
            }
         }else if(vic instanceof VentanaInternaJMFPlayer){
              try{
             vi.getLienzo().setImage(getFrame(
             ((VentanaInternaJMFPlayer)vic).getPlayer()));
            // vi.getLienzo().setImage(
            ((VentanaInternaJMFPlayer)vic).getFrame());
             vi.setTitle("Instantanea");
             vi.setLocation(20+vi.getWidth(),0);
             vi.setVisible(true);
             this.escritorio.add(vi);
             }
           catch (Exception ex)
           {
         System.err.println("Error");
       }
         }

private void raiznesimaActionPerformed(java.awt.event.ActionEvent evt) {
   JInternalFrame vi = escritorio.getSelectedFrame();

   if ((vi != null)&&(vi instanceof VentanaInterna)) {
     BufferedImage imgSource =
     ((VentanaInterna)vi).getLienzo().getImage();
     if(imgSource!=null){
       try{
         JTextField raz= new JTextField("2");
         JFileChooser dlg = new JFileChooser();
```

```
        Object[] v_raz ={ "Raiz n-esima ",raz };
        JOptionPane.showConfirmDialog(this, v_raz,
        "Introduzca valor Raiz n-esima",JOptionPane.DEFAULT_OPTION);

        LookupTable lt =RaizNesima.raiz(
        Double.parseDouble(raz.getText()));

        LookupOp lop = new LookupOp(lt, null);
       // Imagen origen y destino iguales
       lop.filter( imgSource , imgSource);
       vi.repaint();

    } catch(Exception e){
      System.err.println(e.getLocalizedMessage());
      }
          }
}
```

3.2. VentanaInterna

Esta clase es la ventana que se abre para dibujar o bien sobre un lienzo o sobre una imagen que se carga.

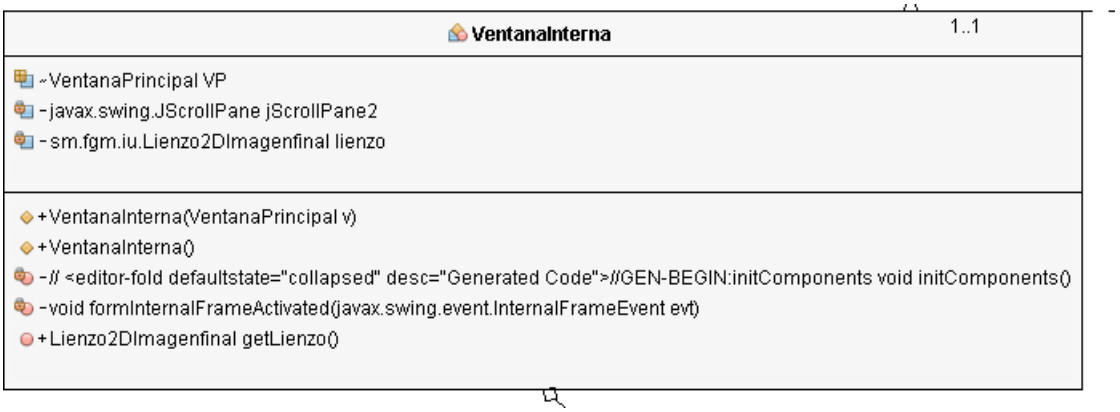

Figura 3.6: UML VentanaInterna

```
private void initComponents() {

        jScrollPane2 = new javax.swing.JScrollPane();
        lienzo = new sm.fgm.iu.Lienzo2DImagenfinal();
        ...
        }
```

Se ha diseñado esta clase haciendola que herede de *javax.swing.JInternalFrame* y con un lienzo de la clase *Lienzo2DImagenfinal* (esta hereda de Lienzo2Dfinal), de esta manera es capaz de recoger figuras geometricas (dibujar) e imágenes. Lo mas destacable de esta clase aparte del lienzo es que recoge mediante el evento *InternalFrameEvent* el estado de este (del lienzo), por tanto cuando se cambie de ventana se carga los atributos que había la ultima vez que se usó.Esto se consigue pasandole por cabecera la VentanaPrincipal cada vez que se crea una VentanaInterna. (ver los *new VentanaInterna(this)* en VentanaPrincipal) .

Figura 3.7: Ventana Interna

3.3. VentanaInternaGrabador

Esta clase (tambien hereda de javax.swing.JInternalFrame) representa a la ventana que usa Java Sound API para grabar audio.Lo mas destacable de ella es el manejador de eventos que se usa para cambiar iconos.

Figura 3.8: Ventana Interna Grabador

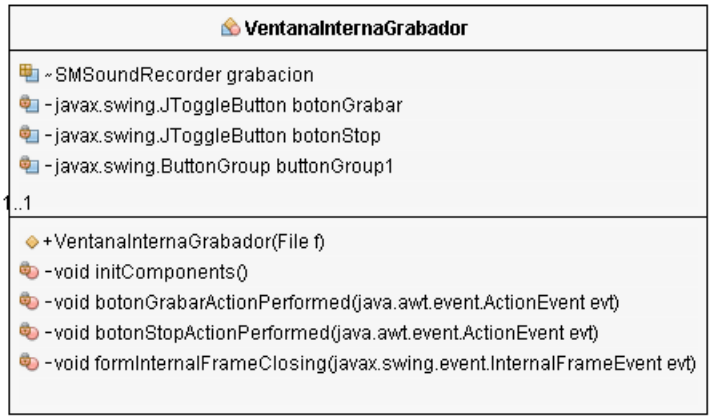

Figura 3.9: UML VentanaInternaGrabador

```
class ManejadorAudio implements LineListener {
@Override

public void update(LineEvent event) {
    if (event.getType() == LineEvent.Type.START) {

    }
    if (event.getType() == LineEvent.Type.STOP) {


        VentanaInternaGrabador.this.botonStop.setSelected(true);
     }
    if (event.getType() == LineEvent.Type.CLOSE) {
       if(grabacion!=null)
       grabacion.stop();
    }
    if(event.getType() == LineEvent.Type.OPEN){

    }
}
```

3.4. VentanaInternaReproductor

Esta clase representa a la ventana que usa Java Sound API para reproducir audio.El código es casi similar al anterior.

Figura 3.10: Ventana Interna Reproductor

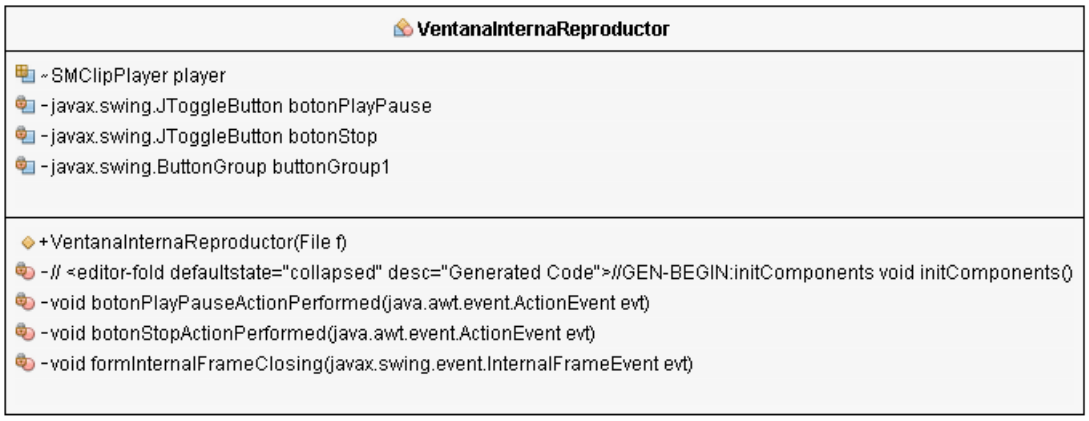

Figura 3.11: UML VentanaInternaReproductor

3.5. VentanaInternaReproductorHebra

Esta clase representa a la ventana que usa Java Sound API para reproducir audio, se añade un elemento *JProgressBar*.Se destaca la creación de la clase *VentanaInternaReproductorHebra* que implementa la interfaz *Runnable* (posteriormente se explica porque se opta por este diseño).

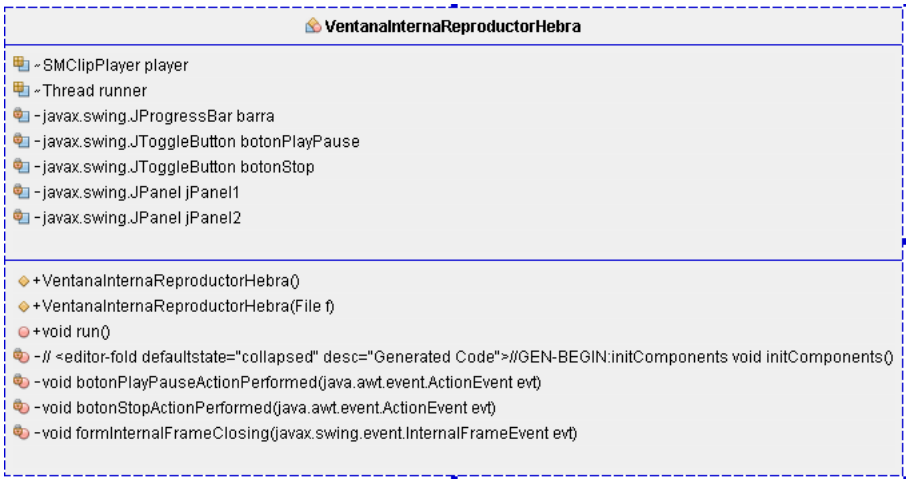

Figura 3.12: UML VentanaInternaReproductorHebra

```
public class VentanaInternaReproductorHebra extends
            javax.swing.JInternalFrame implements Runnable{

    /**
     * Creates new form VentanaInternaReproductorHebra
     */


 public VentanaInternaReproductorHebra() {
  initComponents();
  }

  SMClipPlayer player;
  Thread runner= null;

  /**
   * Constructor
   * @param f File
  */
  public VentanaInternaReproductorHebra(File f) {
     initComponents();
     player = new SMClipPlayer(f);
     player.setLineListener(new
          VentanaInternaReproductorHebra.ManejadorAudio());

    }

 /**
  * Metodo run de la hebra que es invocado cuando se llama al metodo start
  *del objeto hebra declarado
 */
 @Override
 public void run() {
   for(int x=0;x<100;x++){
      barra.setValue(x);
      barra.repaint();

      try{
        Thread.sleep(100);
      }catch(Exception e){
    }
  }
 }
```

```
/**
 * Recoge evento e inicia reproduccion
 * @param evt ActionEvent
 */
private void botonPlayPauseActionPerformed(java.awt.event.ActionEvent evt) {
  if(player!=null){
     player.play();

    if(runner==null){
      runner = new Thread(this);
      runner.start();

    }else{
      runner.resume();

     }

   }

}
/**
  * Recoge evento y para o corta reproduccion
  * @param evt ActionEvent
*/
 private void botonStopActionPerformed(java.awt.event.ActionEvent evt) {
   if(player!=null) player.stop();

     if(runner.isAlive()){

       runner.suspend();
       runner=null;
       barra.setValue(0);
    }

  }
```

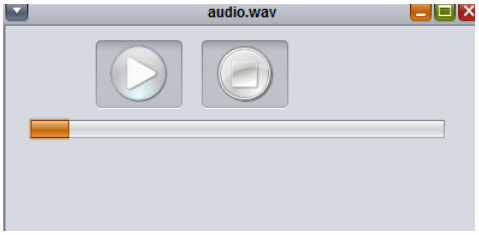

Figura 3.13: UML VentanaInternaReproductorHebra

Figura Nota: Por corregir, hay que adaptar avance barra al tiempo de duración del audio.

3.6. VentanaInternaJMFPlayer

Con esta clase(tambien hereda de javax.swing.JInternalFrame) se usa la API JMF para reproducción de audio y video.El constructor es privado, y por tanto se instancia para evitar excepciones que pudieran darse en el constructor.Se incluye el método getFrame para realizar capturas.

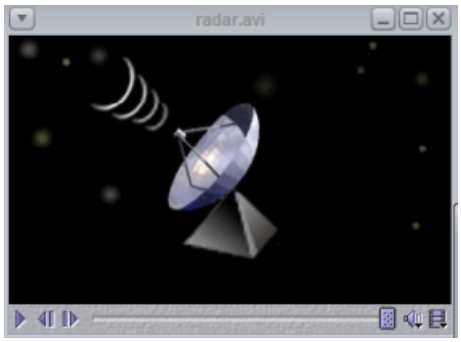

Figura 3.14: Ventana Interna JMFPlayer

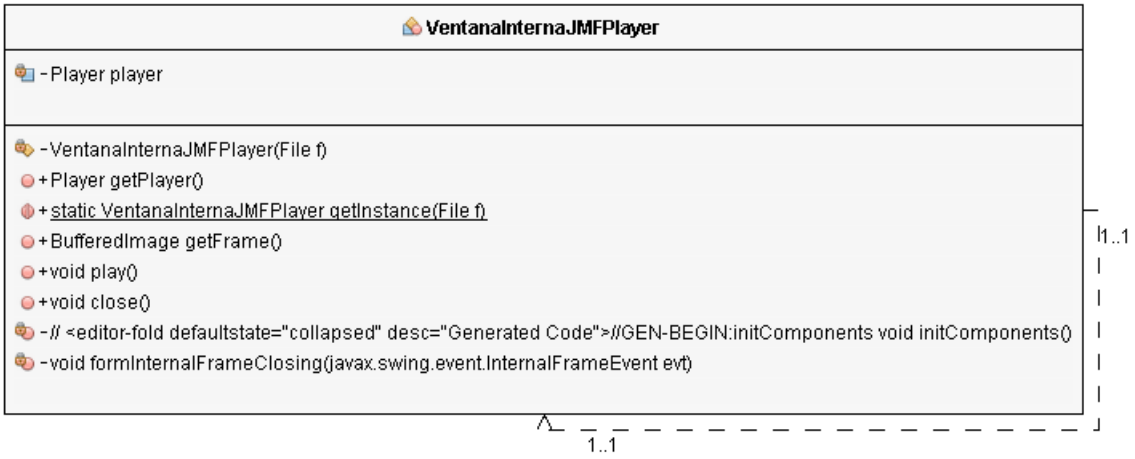

Figura 3.15: UML VentanaInternaJMFPlayer

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

}
```

3.7. VentanaInternaCamara

Al igual que antes se usa la API JMF pero en este caso para reproducir lo que capta la webcam.Por motivos que desconozco mi sistema no la reconoce, pero se ha probado en el ordenador del profesor y funciona.

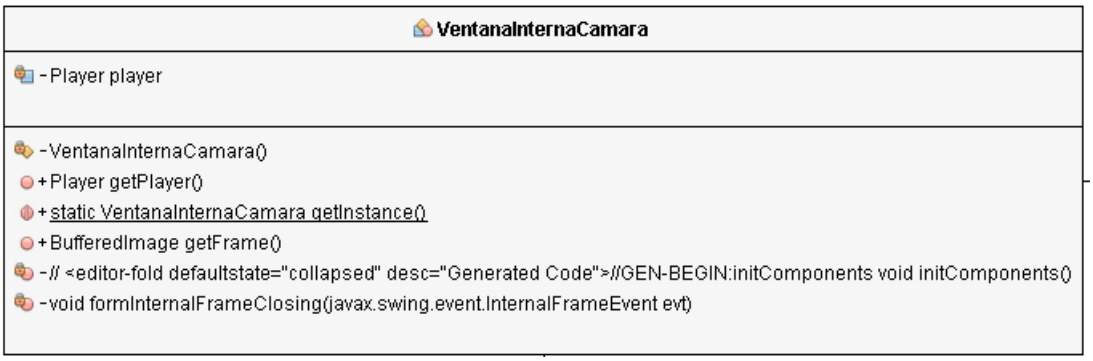

Figura 3.16: UML VentanaInternaCamara

```
private VentanaInternaCamara() {
        initComponents();
         CaptureDeviceInfo deviceInfo;


        String dName="vfw:Microsoft WDM Image Capture (Win32):0";
        deviceInfo = CaptureDeviceManager.getDevice(dName);

        MediaLocator ml = deviceInfo.getLocator();


        try {
            player = Manager.createRealizedPlayer(ml);
            Component vc = player.getVisualComponent();

            if(vc!=null)add(vc, java.awt.BorderLayout.CENTER);
               player.start();

            }catch(Exception e) {
                System.err.println("VentanaInternaJMFPlayer: "+e);
                player = null;
}
    }
```

4. Clases implementadas para las formas geometricas.(sm.fgm.graficos)

Tanto la practica 3 como la 7 presentaban el problema de que todas las formas geométricas tenian los mismos atributos y por tanto limitaban el dibujado[9] de ellas en el lienzo.Esto se ha solucionado creando clases propias de manera que en mi caso he creado una clase

padre abstracta **Figura** que hereda de *Shape* y que contiene los atributos de las diferentes formas geometricas, ademas contiene el metodo pintar dandole una vuelta de tuerca a lo que ocurría en las practicas 3 y 7, ahora si esta orientado a objetos que en este caso es la forma geometrica y por tanto este es uno de sus metodos. Las clases hijas y por tanto que heredan de **Figura** son las diferentes formas geometricas que pueden darse, en mi caso he implementado las clases **MiLinea, MiRectangulo, MiCurva, MiRectanguloRedondeado, MiElipse**, estas implementan los metodos de Shape (no se implementan en Figura) y algunos propios que son necesario para su uso. Una alternativa a esta propuesta es la de crear *Figura* como una interfaz con los métodos comunes que despues tendran cada clase de figura geométrica,es decir, cada clase que hereda tendrá que implementar estos métodos pero ademas hay que declarar los atributos, cosa que no se puede realizar en la interfaz.
Por tanto las ventajas que encuentro al realizar la clase abstracta y heredar de ellas son:

- - No necesito implementar en cada clase metodos iguales (repetir codigo), con implementarlos en la clase abstracta es suficiente.Vease el método *pintar* que es común a todas las figuras.
- Puedo definir los atributos que son similares en las clases de figuras en la clase abstracta.

Entre las desventajas encuentro:

- - Con la interfaz hubiese podido hacer uso de herencia multiple, es decir, la clase de la Figura hubiese podido implementar la interfaz y heredar de la clase Rectangle2D por ejemplo.

Por tanto y tras meditarlo mucho he considerado que la opcion de usar clase abstracta era la idonea (no significa que la otra opcion sea mala), sobre todo porque a la hora de definir los atributos de las figuras y pintar era mucho mas comodo y me ahorraba hacer muchos *instanceof*.

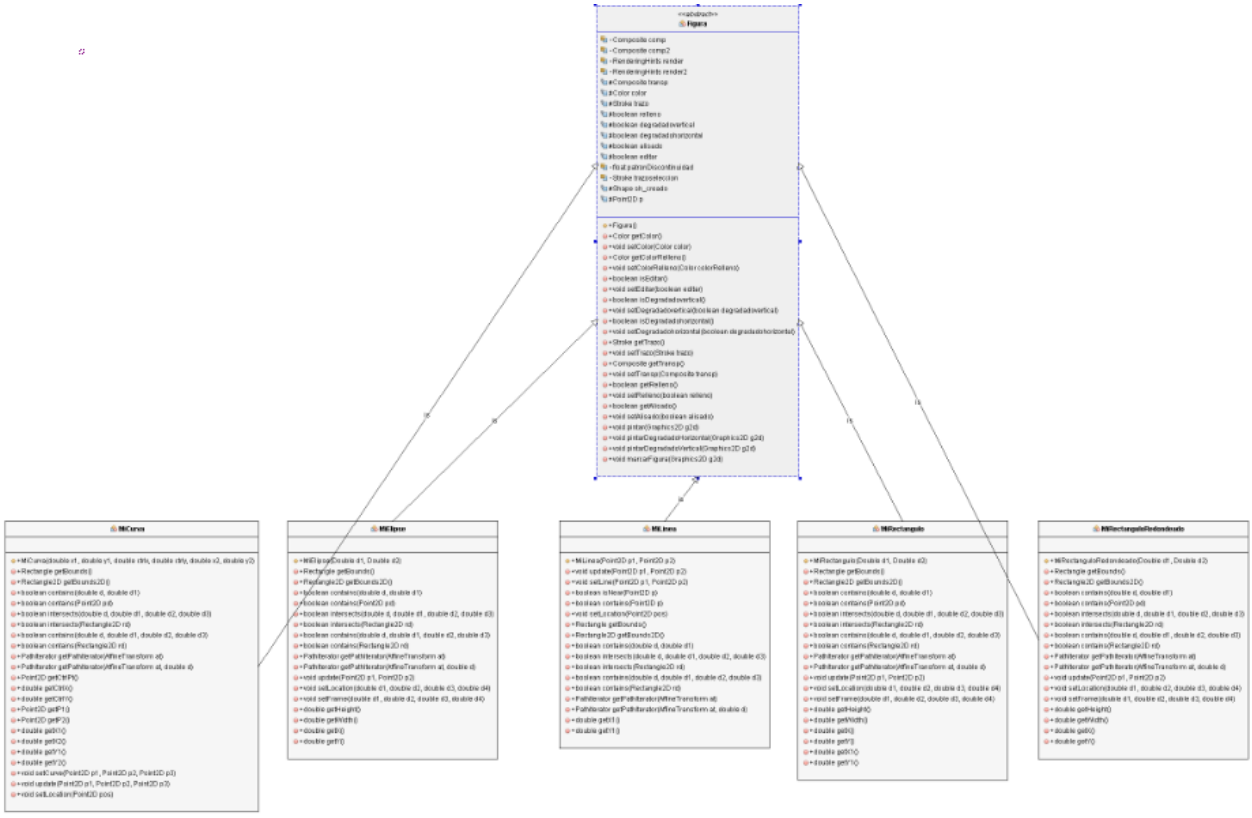

Figura 4.1: Clases Geometricas sm.fgm.graficos

4.1. Figura

Como se comenta en el parrafo anterior, esta es una clase abstracta que hereda de Shape[12] y que define los atributos de cada forma geométrica, es decir los atributos utilizados para el color, relleno, trazo,..., se declaran y por tanto despues son heredados . Su metodo principal es pintar y es usado por cada forma geométrica para pintarse.El modo editar se usa de la siguiente manera en la interfaz gráfica: Al seleccionar esta opción posteriormente se selecciona los atributos deseados y por último la figura a la que se le aplicará, pudiendose mover a otra posición del lienzo tambien.

```
public abstract class Figura implements Shape{

    Composite comp=AlphaComposite.getInstance(AlphaComposite.SRC_OVER,0.5f);
    Composite comp2=AlphaComposite.getInstance(AlphaComposite.SRC);
    RenderingHints render= new RenderingHints(
    RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_ON);
    RenderingHints render2= new RenderingHints(
    RenderingHints.KEY_ANTIALIASING,RenderingHints.VALUE_ANTIALIAS_OFF);
```

```
    protected Composite transp;
    protected Color color,colorRelleno ;
    protected Stroke trazo ;
    protected boolean relleno;
    protected boolean degradadovertical;
    protected boolean degradadohorizontal;
    protected boolean alisado ;

    protected boolean editar;
    float patronDiscontinuidad[] = {15.0f, 15.0f};
    Stroke trazoseleccion = new BasicStroke(
    1.0f,BasicStroke.CAP_ROUND,BasicStroke.JOIN_MITER, 1.0f,
    patronDiscontinuidad, 0.0f);
    ...



public void pintar(Graphics2D g2d){


        if(editar){
            marcarFigura(g2d);
        }

            g2d.setStroke(this.getTrazo());
              g2d.setComposite(transp);

            if(this.getAlisado()){
              g2d.setRenderingHints(render);
              }

            if(this.getRelleno()){
                 g2d.setPaint(this.getColorRelleno());

                 if(degradadohorizontal){
                     pintarDegradadoHorizontal(g2d);

                 }
                 if(degradadovertical){
                     pintarDegradadoVertical(g2d);
                 }

                 g2d.fill(this);
            }
```

```
            g2d.setPaint(this.getColor());
            g2d.draw(this);
        g2d.setRenderingHints(render2);
        g2d.setComposite(comp2);

}
```

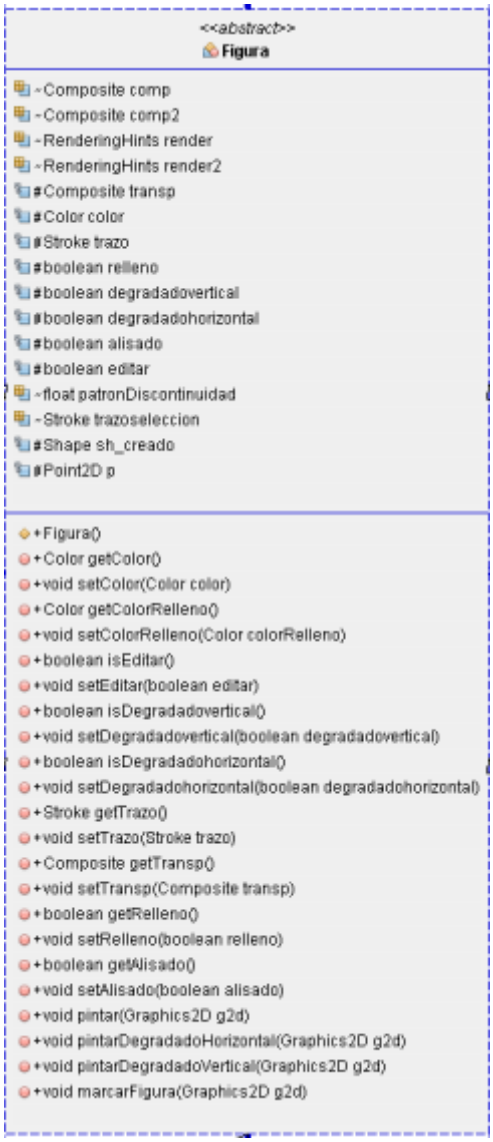

Figura 4.2: Clase Figura

4.2. MiLinea

Esta clase[7] hereda de **Figura** y por tanto sus atributos y metodos implementados. En ella se ha implementado los metodos de Shape que no habian sido implementados en la clase **Figura**.Me apoyo en la clase *Line2D* para su implementación.

```
    public MiLinea(Point2D p1, Point2D p2){
         super();
         sh_creado= new Line2D.Float(p1,p2);
     }

      public boolean isNear(Point2D p){
        if((((Line2D)sh_creado).getX1()!=((Line2D)sh_creado).getX2())||
             (((Line2D)sh_creado).getY1()!=((Line2D)sh_creado).getY2())){
         return ((Line2D)sh_creado).ptLineDist(p) <=2.0;
         }

       return (Math.abs(((Line2D)sh_creado).getX1() - p.getX()))<=3.0;

         }

       @Override
     public boolean contains(Point2D p) {
         return isNear(p);
     }

      public void setLocation(Point2D pos){
       double dx=pos.getX()-((Line2D)sh_creado).getX1();
       double dy=pos.getY()-((Line2D)sh_creado).getY1();
       Point2D newp2 = new Point2D.Double(((Line2D)sh_creado).getX2()+dx,
       ((Line2D)sh_creado).getY2()+dy);

       ((Line2D)sh_creado).setLine(pos,newp2);
}
```

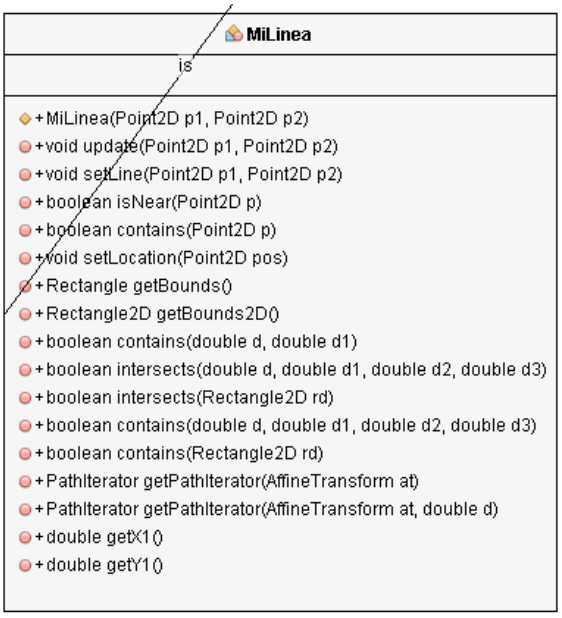

Figura 4.3: Clase MiLinea

Destacar los metodos **public boolean isNear(Point2D p)** y **public void setLocation(Point2D pos)**

4.3. MiRectangulo

Al igual que en el caso anterior hereda de **Figura**.Me apoyo en la clase *Rectangle2D*[10] para su implementación.Los métodos heredados los he implementado de la siguiente forma (en el resto de clases de forma similar a esta):

```
    public MiRectangulo(Double d1, Double d2){
        super();
        sh_creado= new Rectangle2D.Double(d1, d2, 0, 0);

    }

@Override
    public Rectangle getBounds() {
       return ((Rectangle2D)sh_creado).getBounds();
    }

    @Override
    public Rectangle2D getBounds2D() {
        return ((Rectangle2D)sh_creado).getBounds2D();
    }
```

```
@Override
public boolean contains(double d, double d1) {
    return ((Rectangle2D)sh_creado).contains(d, d1);
}

@Override
public boolean contains(Point2D pd) {
    return ((Rectangle2D)sh_creado).contains(pd);
}

@Override
public boolean intersects(double d, double d1, double d2, double d3) {
    return ((Rectangle2D)sh_creado).intersects(d, d1, d2, d3);
}

@Override
public boolean intersects(Rectangle2D rd) {
    return ((Rectangle2D)sh_creado).intersects(rd);
}

@Override
public boolean contains(double d, double d1, double d2, double d3) {
    return ((Rectangle2D)sh_creado).contains(d, d1, d2, d3);
}

@Override
public boolean contains(Rectangle2D rd) {
    return ((Rectangle2D)sh_creado).contains(rd);
}

@Override
public PathIterator getPathIterator(AffineTransform at) {
    return ((Rectangle2D)sh_creado).getPathIterator(at);
}

@Override
public PathIterator getPathIterator(AffineTransform at, double d) {
    return ((Rectangle2D)sh_creado).getPathIterator(at, d);
}
```

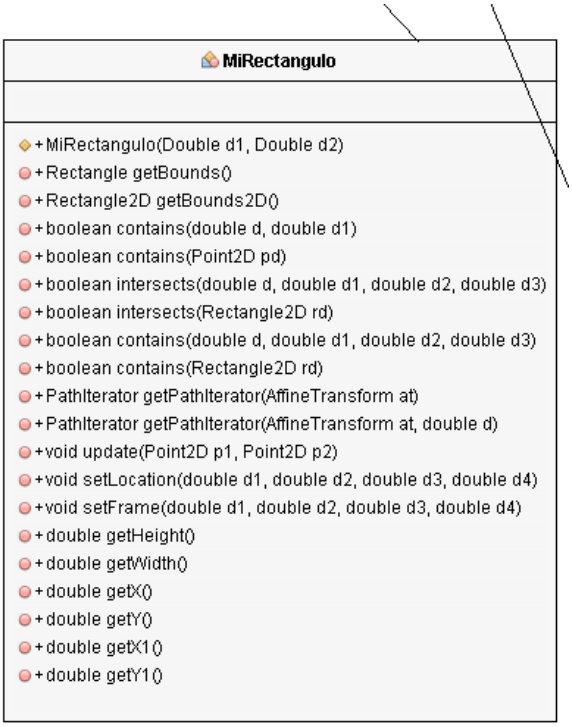

Figura 4.4: Clase MiRectangulo

4.4. MiRectanguloRedondeado

Al igual que en el caso anterior hereda de **Figura**.En cierto modo similar a las clases **MiRectangulo** y **MiElipse**.Nuevamente me apoyo en otra clase para su implementacion[11].

```
 public MiRectanguloRedondeado(Double d1, Double d2){
        super();
        sh_creado= new RoundRectangle2D.Double(d1,d2,0,0,10,10);
}

public void update(Point2D p1, Point2D p2) {
        ((RoundRectangle2D)sh_creado).setFrameFromDiagonal(p1, p2);

    }

    public void setLocation(double d1,double d2,double d3,double d4){
        ((RoundRectangle2D)sh_creado).setFrame(d1, d2, d3, d4);
    }
    public void setFrame(double d1,double d2,double d3,double d4){
        ((RoundRectangle2D)sh_creado).setFrame(d1, d2, d3, d4);
    }
```

```
 public double getHeight(){
     return ((RoundRectangle2D)sh_creado).getHeight();
 }
 public double getWidth(){
     return ((RoundRectangle2D)sh_creado).getWidth();
 }

public double getX(){
return ((RoundRectangle2D)sh_creado).getX();
}

public double getY(){
 return ((RoundRectangle2D)sh_creado).getY();
}
```

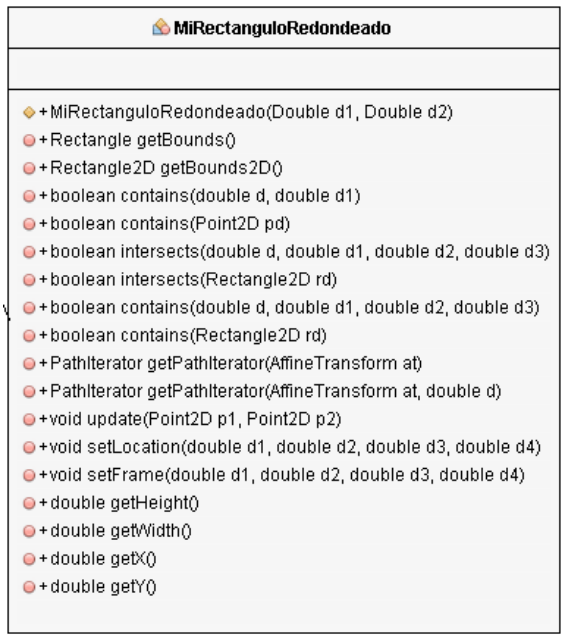

Figura 4.5: Clase MiRectanguloRedondeado

4.5. MiElipse

Al igual que en el caso anterior hereda de **Figura**.Al igual que en las anteriores, me apoyo en *Ellipse2D* para su desarrollo[4]

```
public MiElipse(Double d1, Double d2){
      super();
      sh_creado= new Ellipse2D.Double(d1,d2,0,0);
```

```
 }

      public void update(Point2D p1, Point2D p2) {

     ((Ellipse2D)sh_creado).setFrameFromDiagonal(p1, p2);

 }

   public void setLocation(double d1,double d2,double d3,double d4){
     ((Ellipse2D)sh_creado).setFrame(d1, d2, d3, d4);
 }

  public void setFrame(double d1,double d2,double d3,double d4){
     ((Ellipse2D)sh_creado).setFrame(d1, d2, d3, d4);
 }

 public double getHeight(){
     return ((Ellipse2D)sh_creado).getHeight();
 }

 public double getWidth(){
     return ((Ellipse2D)sh_creado).getWidth();
 }

public double getX(){
 return ((Ellipse2D)sh_creado).getX();
}

public double getY(){
 return ((Ellipse2D)sh_creado).getY();
}
```

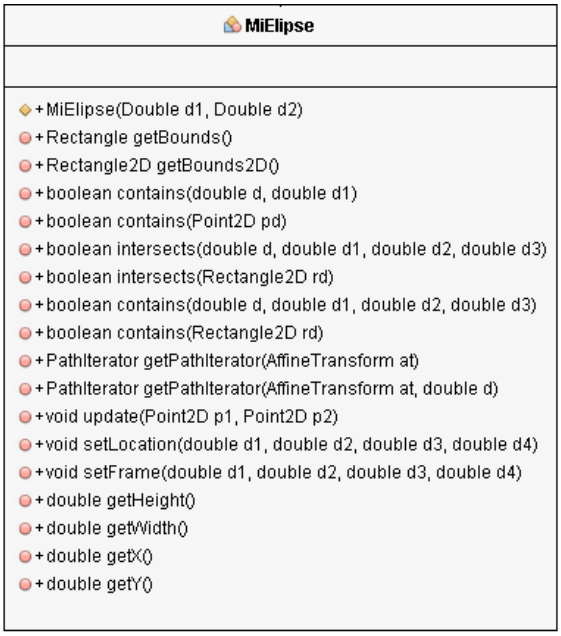

Figura 4.6: Clase MiElipse

4.6. MiCurva

Al igual que en el caso anterior hereda de **Figura**.A tener en cuenta para su manejo el punto de control[3] que será donde se realice la 'curva' y que implica que se dibuje en dos pasos, es decir, primero se dibuja una recta y despues se establece el punto de control como puede observarse en los eventos *pressed, dragged y released* del *lienzo*.

```
public void setCurve(Point2D p1,Point2D p2,Point2D p3){
        //((QuadCurve2D)sh_creado).setCurve(x1, y1, ctrlx, ctrly, x2, y2);
        ((QuadCurve2D)sh_creado).setCurve(p1, p2, p3);
    }

   public void setCurve(Point2D p1,Point2D p2,Point2D p3){
        //((QuadCurve2D)sh_creado).setCurve(x1, y1, ctrlx, ctrly, x2, y2);
        ((QuadCurve2D)sh_creado).setCurve(p1, p2, p3);
    }

    public void update(Point2D p1,Point2D p2,Point2D p3){
        //((QuadCurve2D)sh_creado).setCurve(x1, y1, ctrlx, ctrly, x2, y2);
        ((QuadCurve2D)sh_creado).setCurve(p1, p2, p3);
    }


    public void setLocation(Point2D pos){
```

```
        Point2D p_aux=null;
        Point2D p_aux_control=null;
        double dist_x,dist_y;

         dist_x = pos.getX() - ((QuadCurve2D)sh_creado).getX1();
         dist_y = pos.getY() - ((QuadCurve2D)sh_creado).getY1();
         p_aux = new Point2D.Double( ((QuadCurve2D)sh_creado).getX2()+dist_x,
               ((QuadCurve2D)sh_creado).getY2()+dist_y);
         p_aux_control = new Point2D.Double(((QuadCurve2D)sh_creado).getCtrlX(
              +dist_x,((QuadCurve2D)sh_creado).getCtrlY()+dist_y);

         setCurve(pos,p_aux_control, p_aux);

    }
```

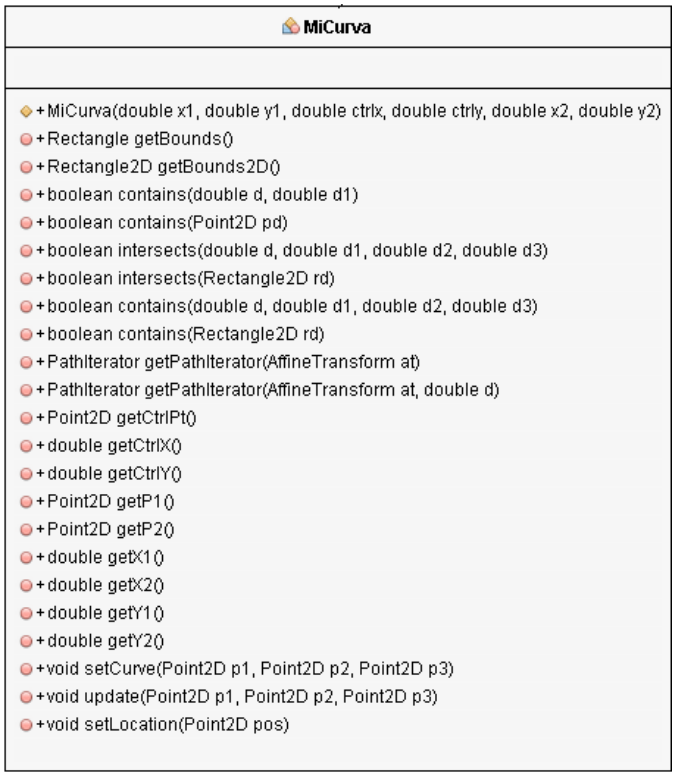

Figura 4.7: Clase MiCurva

5. Clases utilizadas para el dibujado de formas geométricas e imágenes.(sm.fgm.iu)

En esta sección se explica las dos clases utilizadas para el manejo(utilizacion de los metodos) de las clases geometricas e imagenes (realmente las imagenes se manejan desde la VentanaPrincipal utilizando el objeto lienzo y sus metodos).

5.1. Lienzo2Dfinal

Esta clase es la que recoge y muestra las figura geométricas en pantalla a traves del metodo *public void paint(Graphics g)* , esto se consigue recogiendo los eventos(seleccion de color, trazo,..) de *VentanaPrincipal* y vinculandolos al objeto que se desea dibujar (estos objetos tienen los atributos color, trazo, transparencia,...), a su vez se guarda en un array de figuras que será lo que se recorra en dicho metodo.

Figura 5.1: UML Lienzo2Dfinal

Destacar los tres eventos utilizados para la creación de la figura geometrica:

- **formMousePressed** - Corresponde a pulsar el botón izquierdo del ratón.
- **formMouseDragged** - Corresponde a arrastrar con el botón pulsado.
- **formMouseReleased** - Corresponde a soltar el botón del ratón.

public void paint(Graphics g)

En este método se recorre el array de figuras geómetricas y se llama al metodo *pintar* de cada uno de ellos pasandole un objeto *Graphics2D* como argumento.

```
public void paint(Graphics g){
        super.paint(g);
        Graphics2D g2d = (Graphics2D)g;

        if (this.clip != null) {
             g2d.clip(this.clip);
         }
        for(Shape s:vShape) {
            ((Figura)s).pintar(g2d);

        }
}
```

private Shape createShape(Point2D p_a)

Crea el objeto seleccionado en *VentanaPrincipal* con los atributos correspondientes.

```
 private Shape createShape(Point2D p_a){

        if(p_a==null){
            return null;
        }

        if(modo==0){
             sh_creado= new MiLinea(p_a,p_a);
              //punto con las mismas coordenadas, asi crea punto en vez linea
         }else if(modo==1){
             sh_creado= new MiLinea(p_a,p_a); // linea
         }else if(modo==2){
            sh_creado= new MiRectangulo(p_a.getX(),p_a.getY());
         }else if(modo==3){
             sh_creado= new MiElipse(p_a.getX(),p_a.getY());
         }else if(modo==4){
            sh_creado= new MiRectanguloRedondeado(p_a.getX(),p_a.getY());
         }else if(modo==5){
            sh_creado= new MiCurva(p_a.getX(),p_a.getY(),p_a.getX(),p_a.getY()
            p3=p_a;
         }else{
             sh_creado=null;
         }
```

```
if(sh_creado!=null){
   ((Figura)sh_creado).setColor(this.color);
   ((Figura)sh_creado).setTrazo(this.stroke);
   ((Figura)sh_creado).setAlisado(this.alisado);
   ((Figura)sh_creado).setRelleno(this.relleno);
   ((Figura)sh_creado).setColorRelleno(this.colorRelleno);
   ((Figura)sh_creado).setDegradadohorizontal(this.degradadohorizontal);
   ((Figura)sh_creado).setDegradadovertical(this.degradadovertical);
   ((Figura)sh_creado).setTransp(this.comp);
   ((Figura)sh_creado).setEditar(this.editar);
     }
     return sh_creado;
 }
```

private void updateShape(Point2D p_a, Point2D p_b)

Usado para actualizar las formas geométricas al llamar al *dragged*.

```
private void updateShape(Point2D p_a, Point2D p_b){
    if (sh_creado instanceof MiLinea) {
      ((MiLinea)sh_creado).update(p_a, p_b);
    }else if(sh_creado instanceof MiRectangulo) {
      ((MiRectangulo)sh_creado).update(p_a, p_b);
    }else if(sh_creado instanceof MiElipse) {
      ((MiElipse)sh_creado).update(p_a, p_b);
    }else if(sh_creado instanceof MiRectanguloRedondeado) {
      ((MiRectanguloRedondeado)sh_creado).update(p_a, p_b);
    }else if(sh_creado instanceof MiCurva) {
      ((MiCurva)sh_creado).update(p_a,p_b, p2);
    }else if(sh_creado instanceof RectangularShape) {
      ((RectangularShape)sh_creado).setFrameFromDiagonal(p_a, p_b);
    }

    }
```

private void formMousePressed(java.awt.event.MouseEvent evt)

Corresponde al evento que se produce cuando se pulsa el boton izquierdo del raton, en este método se distingue si esta editando o no. En el modo que no edita se diferencia dos casos, un primer caso donde puede corresponder a la curva ya pintada donde solo queda establecer el punto de control o un segundo caso donde se crea la figura geometrica mediante la llamada a *createShape* (notar que se definen los atributos). El segundo

caso corresponde al modo editar donde se calcula el sitio geográfico del punto p4 con el objetivo de evitar el 'efecto ancla' al editar.

```
private void formMousePressed(java.awt.event.MouseEvent evt) {
  p4=new Point2D.Double();
  p = evt.getPoint();


  if(editar){
    this.sh_creado =  getSelectedShape(p);

    if(sh_creado!=null){
       double x=0,y=0;
       if (sh_creado instanceof MiLinea){
         x=((MiLinea)sh_creado).getX1();
         y=((MiLinea)sh_creado).getY1();

       }else
         if (sh_creado instanceof MiRectangulo){
          x=((MiRectangulo)sh_creado).getX();
          y=((MiRectangulo)sh_creado).getY();

       }else if (sh_creado instanceof MiElipse){
          x=((MiElipse)sh_creado).getX();
          y=((MiElipse)sh_creado).getY();

       }else if (sh_creado instanceof MiRectanguloRedondeado){
          x=((MiRectanguloRedondeado)sh_creado).getX();
          y=((MiRectanguloRedondeado)sh_creado).getY();

       }else if (sh_creado instanceof MiCurva){
          x= ((MiCurva)sh_creado).getX1();
          y= ((MiCurva)sh_creado).getY1();
       }

       p4.setLocation(x - p.getX(), y- p.getY());
      }

     if(sh_creado!=null){
       ((Figura)sh_creado).setColor(this.color);
       ((Figura)sh_creado).setTrazo(this.stroke);
       ((Figura)sh_creado).setAlisado(this.alisado);
       ((Figura)sh_creado).setRelleno(this.relleno);
       ((Figura)sh_creado).setColorRelleno(this.colorRelleno);
```

```
        ((Figura)sh_creado).setDegradadohorizontal(this.degradadohorizontal);
        ((Figura)sh_creado).setDegradadovertical(this.degradadovertical);
        ((Figura)sh_creado).setTransp(this.comp);
        ((Figura)sh_creado).setEditar(this.editar);
       }
} else{
    if((modo==FORMA_CURVA)&&(puntoControlfalta)){
      if((Figura)vShape.get(vShape.size()-1) instanceof MiCurva){

      ((MiCurva)vShape.get(vShape.size()-1)).update(p3, evt.getPoint(), p2);
        }
     }else{
         Shape nuevo_s= createShape(p);
         vShape.add(nuevo_s);
      }
  }
}
```

private void formMouseReleased(java.awt.event.MouseEvent evt)

Corresponde al evento de soltar el boton izquierdo,en el nuevamente se distinguen dos casos.En el modo editar se establece el cursor por defecto y se pone con valor a false la booleana editar de la figura geometrica con el objetivo de quitar el boundingbox (el recuadro desaparece cuando dejemos de editar).En el modo de no editar o bien se termina de dibujar la curva o se termina las otras figuras salvo que sea la curva donde se usa una booleana para indicar que falta establecer el punto de control.

```
private void formMouseReleased(java.awt.event.MouseEvent evt) {

  if (editar) {
     setCursor(Cursor.getDefaultCursor());
     if(sh_creado!=null){
       ((Figura)sh_creado).setEditar(false);
     }
 }else{
    if((modo==FORMA_CURVA)&&(puntoControlfalta)){
      ((MiCurva)sh_creado).update(p3, evt.getPoint(), p2);
       puntoControlfalta=false;
       repaint();
    }else{
       puntoControlfalta=true;
       this. formMouseDragged(evt);
    }
  }
```

```
}
```

private void formMouseDragged(java.awt.event.MouseEvent evt

Evento que corresponde a arrastrar el raton con el boton pulsado.Nuevamente se distinguen dos casos, el modo edicion y el modo donde no se edita. En el modo edición se mueve la figura al lugar donde se desee mientras que en el modo de no edicion se distinguen tres casos:

- Un primer caso donde solo falta establecer el punto de control de la curva
- Un segundo caso donde todavia no es necesario el punto de control sino acabar de definir el primer paso de construccion de la curva que es realizar la recta.
- El tercer caso es el resto de las opciones, un caso generico para la linea, rectangulo, elipse,..., donde se actualiza la figura.

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

    if(editar){

        if(sh_creado!=null){

            setLocation(sh_creado,
              new Point2D.Double(evt.getX()+p4.getX(),evt.getY()+p4.getY()));
        }
   } else if (modo != 0) {//el punto no se actualiza{
     if((modo==FORMA_CURVA)&&(puntoControlfalta)
        &&(sh_creado instanceof MiCurva)){

        ((MiCurva)sh_creado).update(p3,punto, p2);
     }else if((modo==FORMA_CURVA)&&!(puntoControlfalta)){
           p2=evt.getPoint();
           updateShape(p, punto);
     }else{
          puntoControlfalta = false;
          updateShape(p, punto);
     }
 }


       repaint();
   }
```

private Shape getSelectedShape(Point2D p)

Este metodo es usado en el editar del Pressed para saber que figura se esta editando.

```
 private Shape getSelectedShape(Point2D p){
    for(Shape s:vShape)
       if(s.contains(p)) return s;
          return null;
}
```

private void setLocation(Shape s, Point2D pos)

Sirve para actualizar la figura cuando se realiza el dragged.

```
private void setLocation(Shape s, Point2D pos){
  if (s instanceof MiLinea){
     ((MiLinea)s).setLocation(pos);
  }else if (s instanceof MiRectangulo){

    ((MiRectangulo)s).setLocation(pos.getX(), pos.getY(),
       ((MiRectangulo)s).getWidth(), ((MiRectangulo)s).getHeight());
  }else if (s instanceof MiElipse){

   ((MiElipse)s).setLocation(pos.getX(), pos.getY(),
      ((MiElipse)s).getWidth(),((MiElipse)s).getHeight());
  }else if (s instanceof MiRectanguloRedondeado)
     ((MiRectanguloRedondeado)s).setLocation(pos.getX(), pos.getY(),
        ((MiRectanguloRedondeado)s).getWidth(),
        ((MiRectanguloRedondeado)s).getHeight());
  }else if (s instanceof MiCurva){

    ((MiCurva)s).setLocation(pos);

  }else if(s instanceof RectangularShape){
    ((RectangularShape)s).setFrame(pos.getX(), pos.getY(),
       ((RectangularShape)s).getWidth(),((RectangularShape)s).getHeight());

        }
}
```

5.2. Lienzo2DImagenfinal

Clase que hereda de *Lienzo2Dfinal* y es la que se usa en *VentanaInterna*, por tanto hereda los metodos de estas permitiendo dibujar sobre el lienzo o bien sobre una imagen cargada

(Lienzo2Dfinal no permite cargar una imagen).Sobre esta imagen se pueden efectuar operaciones de rotacion, umbralización, etc.

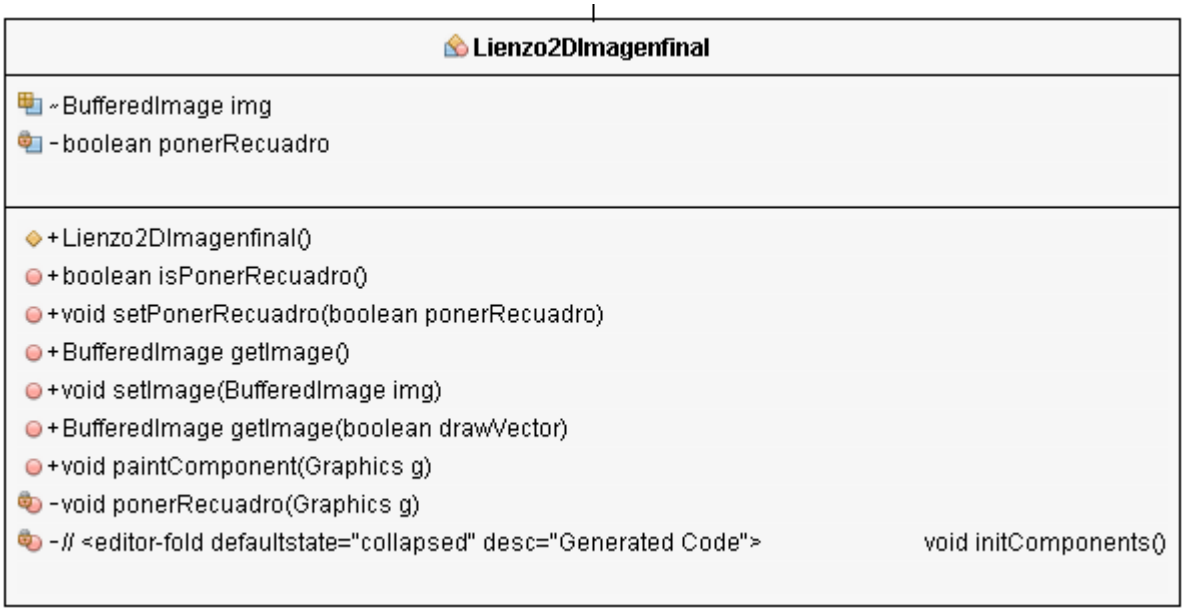

Figura 5.2: UML Lienzo2DImagenfinal

public void setImage(BufferedImage img)

Con este método se carga la imagen con su correspondiente clip (rectangulo).

```
public void setImage(BufferedImage img) {
    this.img = img;
    if(img!=null) {
        setPreferredSize(new Dimension(img.getWidth(),
          img.getHeight()));
        Rectangle r = new Rectangle(0, 0, img.getWidth(), img.getHeight());
        setClip(r);
}
    }
```

public void paintComponent(Graphics g)

La implementación por defecto del método paint de la clase JComponent llama, en este orden, a los métodos paintComponent, paintBorder y paintChildren. Este método contiene el código que determina cómo se pinta el componente, con ello se consigue dibujar encima de la imagen.

```
public void paintComponent(Graphics g){
```

```
    super.paintComponent(g);//llama al paint de lienzo2Dfinal que es el padre
    if(img!=null)
      g.drawImage(img,0,0,this);
    if ((this.clip != null) && (this.ponerRecuadro)) {
                    ponerRecuadro(g);
            }
     }
```

6. Operaciones Propias

En esta sección se explican las diferentes operaciones propias[8] (desarrolladas en clases) que se han utilizado para la parte de imágenes, algunas de ellas puede observarse que heredan de la clase abstracta *BufferedImageOpAdapter* donde solo es necesario implementar el método *filter*.

public class ConversionGrisOp extends BufferedImageOpAdapter

En esta clase se realiza una conversión de la imagen pasandola de color a tonalidad de grises.Esto se consigue multiplicando cada banda RGB por los factores 0.299, 0.587 y 0.114 respectivamente.

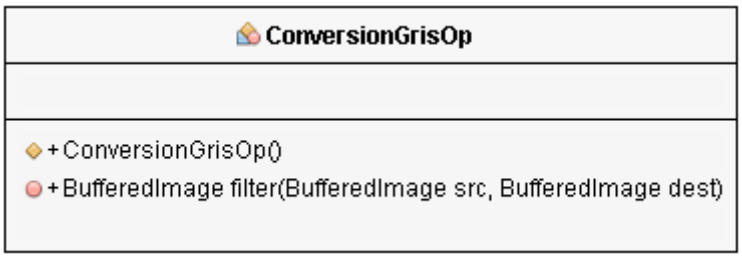

Figura 6.1: UML ConversionGrisOp

```
 @Override
public BufferedImage filter(BufferedImage src, BufferedImage dest){
  if (src == null) {
    throw new NullPointerException("Imagen fuente a null");
   }

  if (dest == null) {
    dest = createCompatibleDestImage(src, null);
   }

  double[] color={255.0,255.0,255.0};
```

```
  WritableRaster srcRaster = src.getRaster();
  WritableRaster destRaster = dest.getRaster();

for (BufferedImagePixelIterator it = new BufferedImagePixelIterator(src);
        it.hasNext();) {
    BufferedImagePixelIterator.PixelData pixel = it.next();
    //a las 3 bandas le asigno mismo valor
   color[0]=color[1]=color[2]=
          (int) ((0.299*pixel.sample[0])+(0.587*pixel.sample[1])+
                (0.114*pixel.sample[2]));
   destRaster.setPixel(pixel.col, pixel.row, color);
   }
      return dest;
  }
       }
```

public class EliminacionMitadBandaCentralOp extends BufferedImageOpAdapter

En esta clase propia lo que se ha realizado es la reducción de la banda central a la mitad.Por tanto en las bandas RGB se reduce G a 0.5 de su valor.

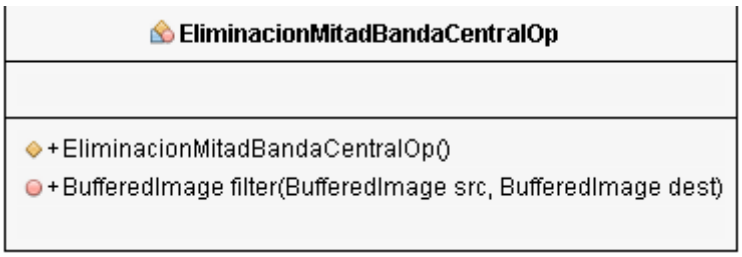

Figura 6.2: UML EliminarMitadBandaCentralOp

```
  @Override
public BufferedImage filter(BufferedImage src, BufferedImage dest) {
  if (src == null) {
     throw new NullPointerException("Imagen fuente a null");
  }

  if (dest == null) {
     dest = createCompatibleDestImage(src, null);
   }

  double[] color={255.0,255.0,255.0};
```

```
  WritableRaster srcRaster = src.getRaster();
  WritableRaster destRaster = dest.getRaster();

  for (BufferedImagePixelIterator it = new BufferedImagePixelIterator(src);
      it.hasNext();) {
    BufferedImagePixelIterator.PixelData pixel = it.next();
    color[0]=(int) (pixel.sample[0]);
    color[2]=(int) (pixel.sample[2]);
    color[1]=(int) (0.5*pixel.sample[1]);
    destRaster.setPixel(pixel.col, pixel.row, color);

 }

    return dest;
  }
```

public class FiltroSeno

Consiste en aplicar un filtro seno[2] a una imagen.

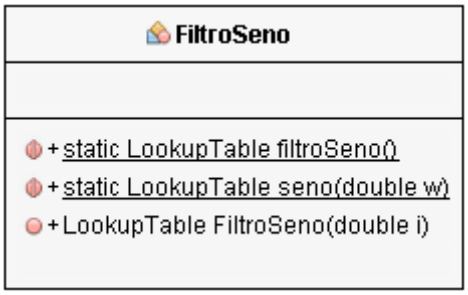

Figura 6.3: UML FiltroSeno

```
 public static LookupTable seno(double w)
  {
double K = 255.0;
byte[] lt = new byte[256];

for (int i = 0; i <= 255; i++) {
  lt[i] = ((byte)(int)(K * Math.abs(Math.sin(Math.toRadians(i * w)))));
   }
 ByteLookupTable lookdevuelto = new ByteLookupTable(0, lt);

return lookdevuelto;

  }
```

public class PotenciaNesima

Consiste en aplicar un filtro potencia a una imagen segun un valor N introducido.

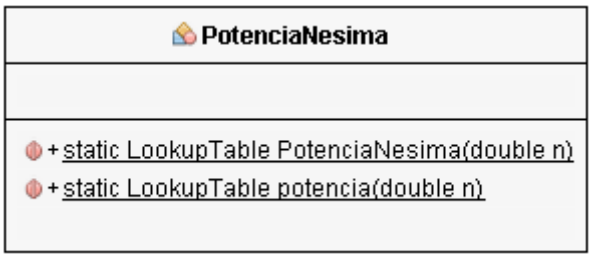

Figura 6.4: UML PotenciaNesima

```
public static LookupTable potencia(double n)
 {
 double K = 255.0/(Math.pow(255,n));
 byte[] lt = new byte[256];

 for (int i = 0; i <= 255; i++) {
   lt[i] = ((byte)(int)(K*Math.pow(i,n)));
 }
ByteLookupTable lookdevuelto = new ByteLookupTable(0, lt);

 return lookdevuelto;

}
```

public class RaizNesima

Se aplica filtro Raiz a una imagen segun un valor N introducido.

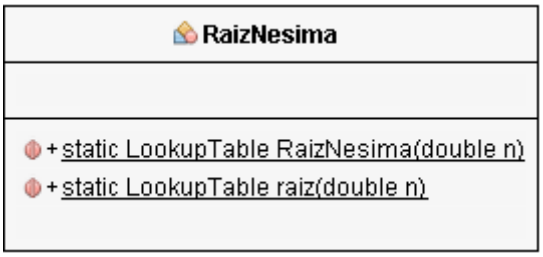

Figura 6.5: UML RaizNesima

```
public static LookupTable raiz(double n)
 {
```

```
double K = 255.0/(Math.pow(255,n));
byte[] lt = new byte[256];

for (int i = 0; i <= 255; i++) {
  lt[i] = ((byte)(int)(Math.pow((i/K),(1/n))));
 }
ByteLookupTable lookdevuelto = new ByteLookupTable(0, lt);

 return lookdevuelto;

 }
```

public class RestaOp extends BinaryOp

Operación binaria para visualizar diferencias entre dos imágenes.Lo realmente interesante es la implementacion del método heredado *binaryOp* (método abstracto de la clase BinaryOp).

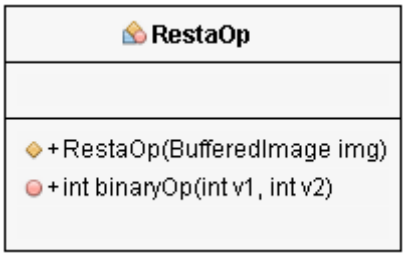

Figura 6.6: UML RestaOp

```
public int binaryOp(int v1, int v2){
    int rdo = v1-v2;
    if(rdo<=0) rdo=0;
    else if(rdo>=255) rdo=255;
    return rdo;
}
```

public class SumaOp extends BinaryOp

Operación de blending que permite mezclar imágenes.

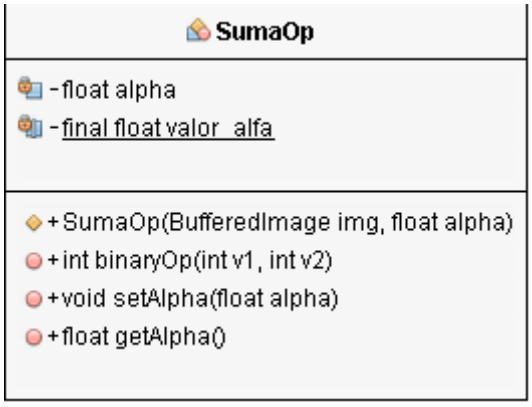

Figura 6.7: UML SumaOp

```
 public int binaryOp(int v1, int v2){
    int rdo = (int)((alpha*v1)+((1-alpha)*v2));
    if(rdo<=0) rdo=0;
    else if(rdo>=255) rdo=255;
    return rdo;
}
```

public class TintadoOp extends BufferedImageOpAdapter

Clase que hereda de *BufferedImageOpAdapter* y que entre otros implementa el método abstracto filter.En esta operación se efectua una transformacion en cada pixel tintandolo de un color determinado.

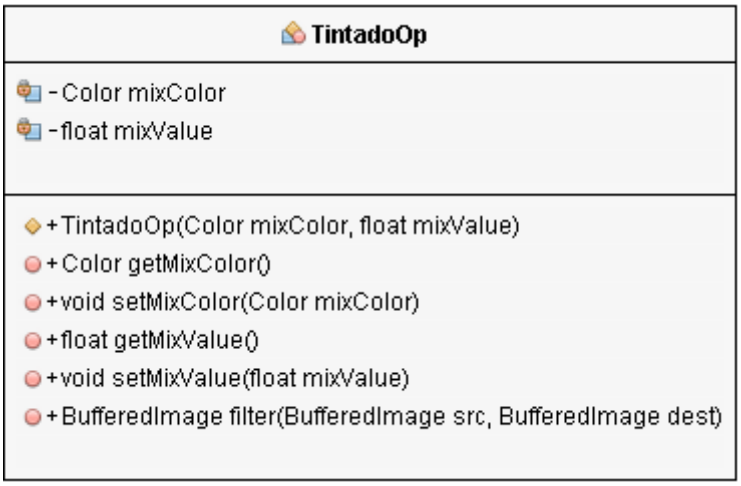

Figura 6.8: UML TintadoOp

```
public BufferedImage filter(BufferedImage src, BufferedImage dest){
  if (dest == null) {
    dest = createCompatibleDestImage(src, null);
  }
  if (src == null) {
    throw new NullPointerException("Imagen fuente a null");
  }
 WritableRaster destRaster = dest.getRaster();
 float mixColorComp[] = mixColor.getColorComponents(null);

for (BufferedImageSampleIterator it =
     new BufferedImageSampleIterator(src); it.hasNext();) {

   BufferedImageSampleIterator.SampleData sample = it.next();
   float colorBand= 255 * mixColorComp[sample.band];
   sample.value = ((int) (sample.value * (1.0f - mixValue) + colorBand * mixV
   destRaster.setSample(sample.col, sample.row, sample.band, sample.value);

 }

 return dest;
   }
```

public class TransformacionLogaritmica

Se aplica un filtro logaritmico a la imagen.

Figura 6.9: UML TransformacionLogaritmica

```
public static LookupTable logaritmo()
 {
 double c = (255.0/Math.log(256));
 byte[] lt = new byte[256];

 for (int i = 0; i <= 255; i++) {
```

```
    lt[i] = ((byte)(int)(c * Math.log(i+1)));
  }
  ByteLookupTable lookdevuelto = new ByteLookupTable(0, lt);

  return lookdevuelto;

  }
```

public class UmbralizacionOp extends BufferedImageOpAdapter

Clase que hereda de *BufferedImageOpAdapter* y realiza una umbralización en escala de grises(para color se ha usado la aportada en la asignatura).

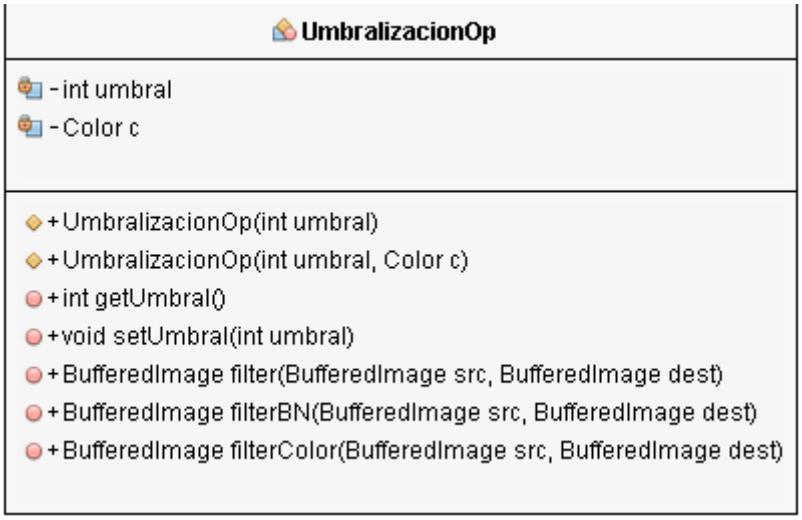

Figura 6.10: UML UmbralizacionOp

```
/*metodo usado en filter*/
public BufferedImage filterBN(BufferedImage src, BufferedImage dest){

  WritableRaster srcRaster = src.getRaster();
  WritableRaster destRaster = dest.getRaster();

 for (BufferedImagePixelIterator it =
    new BufferedImagePixelIterator(src); it.hasNext();) {
    BufferedImagePixelIterator.PixelData pixel = it.next();

   int[] blanco=new int[pixel.numSamples];
   Arrays.fill(blanco,255);
```

```
    int[] negro=new int[pixel.numSamples]; //creo array con esa dimension
    Arrays.fill(negro,0);//rellena el array de valor 0

    int suma=0;
    for (int i = 0;i<pixel.numSamples;i++){
       suma+= pixel.sample[i];
    }
      suma/=pixel.numSamples;

     if(suma>=umbral){
       destRaster.setPixel(pixel.col, pixel.row, blanco);

     }else{
        destRaster.setPixel(pixel.col, pixel.row, negro);
     }
}

     return dest;
}
```

7. Uso de Hebras

En esta sección se explica el uso de hebras para el reproductor de audio, haciendose hincapie en explicar el porqué se ha usado de una manera determinada. La finalidad de usar *hebras* es permitir ejecutar de manera independiente un conjunto de instrucciones desde principio a fin, es lo que se llama *programación paralela.*Un caso muy recomendado es cuando es necesario la realización de animaciones(en este caso el avance continuo de una barra de progreso).
Una hebra no es mas que un objeto **Thread** y hay dos maneras de crearlas [1]:

- Definiendo una subclase de *Thread* [6] (heredando de esta) y sobrecargando el método *run()*
- Definiendo una clase que implemente la interfaz *Runnable* donde se declara una instancia de *Thread* por cada hebra a ejecutar. Esto permitirá mayor flexibilidad.

Por esta segunda opción es la que me he decantado debido a que permite el uso de herencia multiple, en el caso de *VentanaPrincipal* se podia haber hecho tal que así *public class VentanaPrincipal extends javax.swing.JFrame implements Runnable* pero hay que tener en cuenta que solo habría un metodo *run()* en ella y que todas las hebras que se usen tendrían que invocarse con ese único método(limitaría el uso de hebras en la aplicación), por ello he decidido crear una clase que implementa *Runnable* dentro de la clase *VentanaPrincipal* (en *VentanaInternaReproductorHebra* se ha optado por heredar de *JInternalFrame* e implementar la interfaz *Runnable*, esto hace que las hebras que se

declaren solo puedan invocarse con un único metodo *run()*). En definitiva se ha optado por permitir herencia multiple y poder incorporar otras hebras en la aplicación en un futuro si fuese necesario en *VentanaPrincipal*. A destacar en el caso de que se usen diferentes hebras la recomendación de sincronizarlas mediante el uso de cerrojos con el cualificador *synchronized* o con una solución de alto nivel como son los *monitores de Hoare*

7.1. Reproductor de la VentanaPrincipal

He implementado los botones de *Open*,*Play*,*Pause* y *Stop* en la *VentanaPrincipal*, donde :

- El boton de *Play* es el encargado de reproducir audio e inicializar el avance de la barra de progreso (inicializa el objeto de la clase *Hilo* sino lo estuviese).
- El boton de *Pause* es el encargado de pausar la reproduccion de audio y el avance de la barra de progreso.
- El boton de *Stop* es el encargado de parar la reproduccion de audio y vuelve la barra de progreso a su estado inicial(establece el objeto de la clase *Hilo* a null).

Figura 7.1: Barra de Progreso

```
public class VentanaPrincipal extends javax.swing.JFrame {
    private BufferedImage imgFuente;
    SMClipPlayer player;
    Hilo hebra=null;

    ....

  private void openActionPerformed(java.awt.event.ActionEvent evt) {
    JFileChooser dlg = new JFileChooser();
    String extension[]={"wav","au","aif"};
    FileNameExtensionFilter filtro = new
                    FileNameExtensionFilter("Audio[wav,au,aif]",extension);
    dlg.setFileFilter(filtro);


    int resp = dlg.showOpenDialog(this);
    if (resp == JFileChooser.APPROVE_OPTION) {

      try
```

```
      {
       File f = dlg.getSelectedFile();
       player = new SMClipPlayer(f);
       if((hebra!=null)||(barraProgreso.getValue()!=0)){
         hebra.thr.suspend();
         barraProgreso.setValue(0);
         hebra=null;
       }
      }
      catch (Exception ex)
      {
      System.err.println("Error");
      }
   }
    }

private void playActionPerformed(java.awt.event.ActionEvent evt) {
   if(player!=null){
     player.play();

     if(hebra==null){
       hebra=new Hilo();
       hebra.thr.start();
     }else{
       hebra.thr.resume();
     }

   }
}

 private void pauseActionPerformed(java.awt.event.ActionEvent evt) {
  if(player!=null){
    player.pause();

    if(hebra.thr.isAlive()){
        hebra.thr.suspend();
    }

 }
}

private void stopActionPerformed(java.awt.event.ActionEvent evt) {
  if(player!=null) player.stop();
     if(hebra.thr.isAlive()){
```

```
            hebra.thr.suspend();

    }
     hebra=null;
     barraProgreso.setValue(0);
   }



class Hilo implements Runnable{
    protected Thread thr;

    public Hilo(){
      thr = new Thread( this ) ;
    }
    @Override
    public void run() {
      for(int x=0;x<100;x++){
        barraProgreso.setValue(x);
        barraProgreso.repaint();

      try{
        Thread.sleep(100);
      }catch(Exception e){
       }
     }
}

}
```

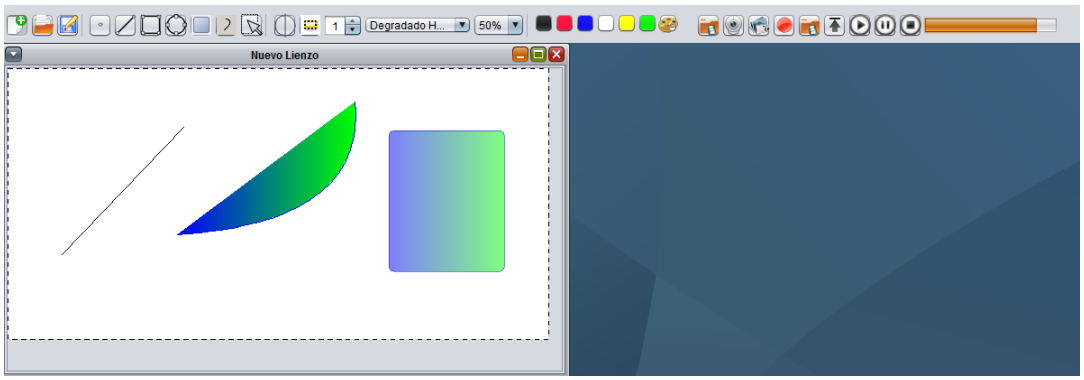

Figura 7.2: Barra de Progreso 2

Nota:

Se ha generado tambien *JavaDoc* del proyecto realizado.

Referencias

[1] Pedro Villar Castro. Sistemas concurrentes distribuidos. 2015.

[2] Jesús Chamorro. Sistemas multimedia.decsai. 2015.

[3] http://docs.oracle.com/javase/7/docs/api/java/awt/geom/quadcurve2d.double.html, consultado el 12 de Julio de 2015.

[4] http://docs.oracle.com/javase/7/docs/api/java/awt/geom/ellipse2d.double.html, consultado el 12 de Julio de 2015.

[5] http://www.tutorialspoint.com/java/java_string_lastindexof.htm, consultado el 12 de Julio de 2015.

[6] http://recursosformacion.com/wordpress/2013/05/java-para-programadores-6-4hebras-y-animacion/, consultado el 5 de Septiembre de 2015.

[7] http://docs.oracle.com/javase/7/docs/api/java/awt/geom/line2d.float.html, consultado el 01 de Junio de 2015.

[8] Rafael Molina. Tratamientos de imágenes digitales.decsai. 2015.

[9] http://www.lcc.uma.es/ galvez/ftp/libros/java2d.pdf, consultado el 12 de Julio de 2015.

[10] http://docs.oracle.com/javase/7/docs/api/java/awt/geom/rectangle2d.double.html, consultado el 12 de Julio de 2015.

[11] http://docs.oracle.com/javase/7/docs/api/java/awt/geom/roundrectangle2d.double.html, consultado el 12 de Julio de 2015.

[12] http://docs.oracle.com/javase/7/docs/api/java/awt/shape.html, consultado el 12 de Julio de 2015.