Sistemas Multimedia

Grado en Ingeniería Informática
Universidad de Granada

Curso 2018 - 2019

Práctica final de Evaluación

Tatiana Daniela Pastorini
tatianapastorini@correo.ugr.es

ÍNDICE

1.- Introducción a la aplicación.

Este proyecto tiene como objetivo general el poder trabajar con elementos multimedia como son vídeos, imágenes y sonidos de forma que podamos crearlos, aplicarle transformaciones y visualizarlos con nuestra aplicación.

En los siguientes apartados vamos a ver cómo hemos organizado el proyecto en sus elementos básicos principales, a grandes rasgos, y más adelante en sus respectivos apartados procederemos a detallar cómo se ha realizado cada uno.

1.1. Ventana Principal.

La ventana principal de la aplicación está compuesta por los elementos que mencionamos a continuación, todo ellos representados en la Figura 1.1:

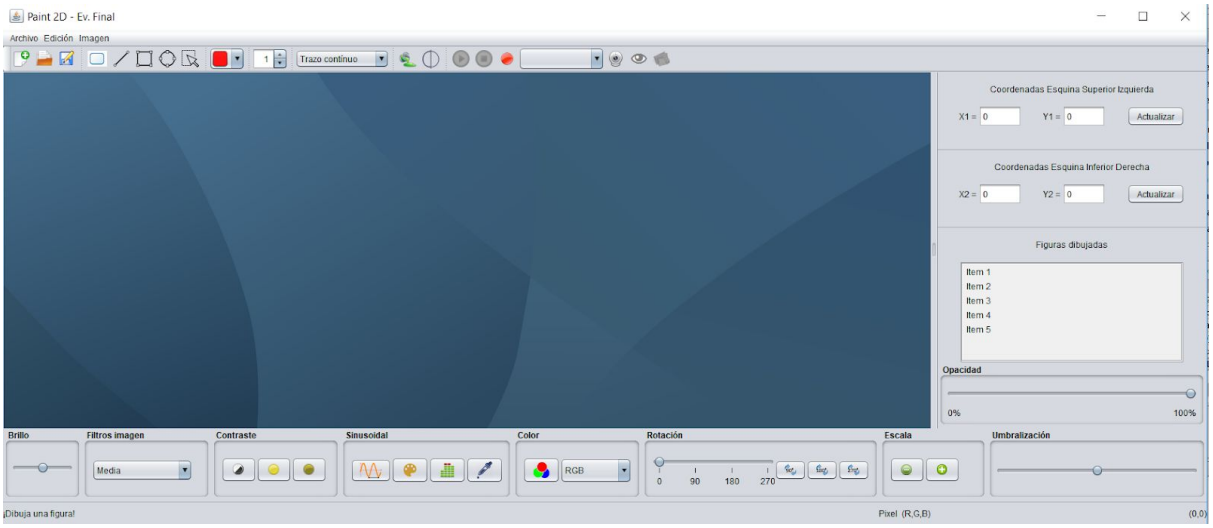


Figura 1.1. Apariencia de la aplicación, Ventana Principal.

En primera instancia nos encontramos con la barra de menú donde tenemos las opciones de Archivo, Edición e Imagen. Debajo del menú tenemos una barra de herramientas donde podemos encontrar acciones asociadas a abrir, guardar y crear ficheros. A continuación, están los botones asociados al dibujo de las

formas, seguido de una lista desplegable de colores predeterminados, la posibilidad de escoger el grosor del trazo de dibujo y una lista de distintos tipos de trazos posibles. Lo siguiente que compone la barra de herramientas es la posibilidad de gestionar si queremos dibujar una figura rellena o no, y si queremos aplicar un alisado de bordes a las mismas.

Como segundo grupo de elementos, dentro de la misma barra de herramientas podemos encontrar una serie de botones asociados a la reproducción de sonidos y vídeos, así como un botón asociado a la grabación de un nuevo sonido, y una lista desplegable de los sonidos que el usuario vaya abriendo desde la aplicación.

Como último bloque, encontramos los mandos asociados a la apertura de la cámara, un detector de movimiento de la cámara y un botón para realizar capturas tanto de la cámara que esté encendida como de cualquier video que se esté reproduciendo desde la aplicación.

En el centro podemos encontrar un escritorio donde el usuario podrá ir viendo las ventanas internas que se vayan creando, de forma que puede ver un vídeo mientras dibuja en un lienzo en blanco y/o edita cualquier fotografía.

En la parte derecha tenemos lo relacionado a la gestión de las figuras dibujadas en el lienzo. Cada vez que el usuario dibuje una nueva figura, se añadirá a la lista de figuras y se actualizarán las coordenadas de los puntos de dicha figura en la parte superior de la parte derecha. Además, podrá editar la opacidad de la figura mediante un deslizador que va desde opaco (0%) hasta transparente (100%).

Por último, en la parte inferior de la aplicación podemos encontrar distintas operaciones que podremos aplicar a una imagen abierta.

1.2. Ventanas Internas.

Una ventana interna puede ser de tres tipos: tipo cámara, tipo vídeo y tipo imagen. En la Figura 1.2. podemos ver los tres tipos en funcionamiento, nuevo lienzo e imagen abierta, vídeo y cámara web respectivamente:

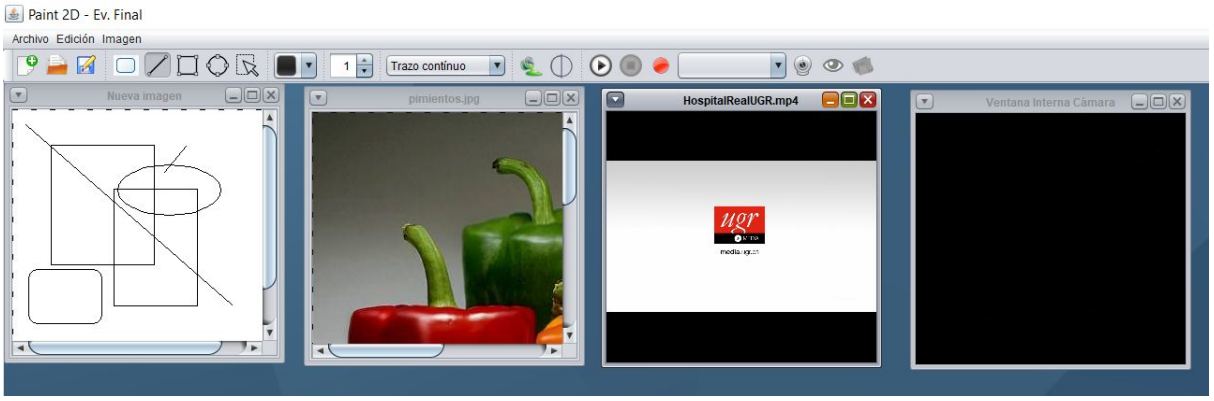


Figura 1.2. Tipos de ventanas internas.

Como se puede observar, todas las ventanas comparten propiedades, por lo que podemos adelantar el hecho de que hemos creado una superclase llamada VentanaInterna de la cual heredarán los 3 tipos de ventana. Esto se detallará más adelante en otro apartado.

1.3. Biblioteca propia.

En una última instancia, el proyecto posee la integración de una biblioteca propia, que en mi caso he llamado *SM.TDP.Biblioteca*, donde se encuentran las clases relacionadas con el tema de gráficos y algunos elementos de la interfaz de usuario. La biblioteca está compuesta por tres paquetes principales que son *graficos, imagen, iu*, donde podemos encontrar, respectivamente, nuestras clases propias relacionadas con el dibujado de las formas junto a sus propiedades; las clases relacionadas con las operaciones que podemos realizar sobre imágenes y por último nuestro elemento lienzo, que es la base del dibujado de formas y la visualización de imágenes.

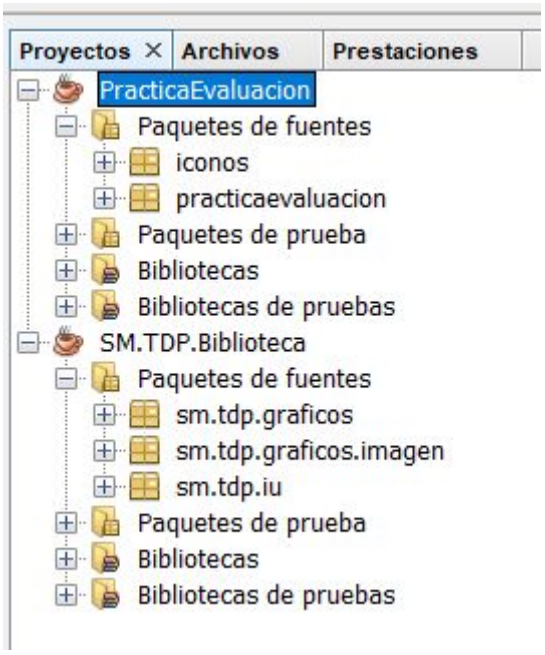


Figura 1.3.1. Organización de paquetes del proyecto principal y la biblioteca.

2.- Requisitos de la aplicación.

Los requisitos que ha de cumplir la aplicación se corresponden con los mencionados a continuación, los cuales hemos ido desarrollando a lo largo de las prácticas de la asignatura así como los requisitos finales de la aplicación detallados en la evaluación.

En primer lugar, la aplicación deberá integrar y gestionar los diferentes tipos de medios que serán los gráficos, imágenes, sonidos y vídeos. Además, para poder realizar esto se necesitará de un conjunto de menús y barras de herramientas que permitan la creación y/o captura, edición, reproducción y procesamiento de los correspondientes medios. Dentro de los requisitos de carácter general, podemos encontrar una barra de menú donde existirán tres opciones: Nuevo, Abrir y Guardar. Estas opciones permitirán, respectivamente, crear una nueva

imagen con un tamaño escogido por el usuario, abrir ficheros de imagen, sonido o vídeo, según los filtros que tenga activados el usuario en ese momento. Y por último guardar ficheros, que permite guardar la imagen de la ventana que esté seleccionada incluyendo las figuras dibujadas si las hay.

2.1. Requisitos de dibujo.

Con respecto a los requisitos del dibujado de formas, la aplicación debe permitir dibujar varias formas geométricas como líneas, rectángulos, elipses y también rectángulos con las esquinas redondeadas. El lienzo donde se esté dibujando deberá mantener todas las figuras dibujadas, y además, cada figura podrá tener atributos propios como el color del trazo, el tipo de trazo y su grosor, si está rellena con un color liso o degradado vertical u horizontal, si tiene alisado de bordes y el grado de transparencia, de forma que cada figura puede tener características diferentes en cuanto al dibujado de las mismas.

Una vez que haya figuras dibujadas en el lienzo, se podrá editar cualquier figura haciendo uso de una lista de figuras dentro de la aplicación, accesible fácilmente en la parte derecha de la ventana principal, de forma que, según cada lienzo activo, se verá el listado de las figuras correspondiente. Cuando cambiemos de ventana de dibujo, se actualizará la lista de figuras y también cuando se dibuje una nueva forma. En cuanto a la edición de los atributos, se usará dicha lista para seleccionar una figura, y entonces se podrán modificar su color de trazo, su grosor de trazo, el tipo de relleno, si tiene o no transparencia o alisado etc. Para saber qué figura estamos editando, se señalará con un recuadro rojo alrededor, para poder identificarla fácilmente.

Por otro lado, cuando editemos una figura, se activarán sus propiedades en la barra de dibujo, de forma que podamos saber cuáles hemos usado para dibujar cada uno. Además, podremos modificar su localización mediante los campos de las coordenadas que estarán junto a la lista de figuras creadas, de forma que

cuando modifiquemos la localización y aceptemos las nuevas coordenadas, se modifique la localización de la misma en el lienzo.

2.2. Requisitos de procesamiento de imágenes.

La aplicación permitirá aplicar operaciones que se podrán realizar sobre cualquier tipo de imagen, ya sea un lienzo con figuras dibujadas o imágenes abiertas. Estas operaciones estarán disponibles en la barra de herramientas de imágenes situada en la barra inferior. El conjunto de operaciones de las que disponemos son las siguientes:

- Duplicar una imagen abierta, ya sea lienzo de dibujo o imagen abierta.
- Modificar el brillo de la imagen mediante un deslizador.
- Filtros de emborronamiento, enfoque y relieve.
- Aplicar contraste normal, iluminado y oscurecido.
- Obtener la imagen en negativo.
- Extraer las bandas de una imagen.
- Conversión a los espacios RGB, YCC y GRAY.
- Rotación libre y estática de una imagen mediante un deslizador.
- Escalar una imagen aumentándola y disminuyéndola.
- Tintar una imagen con el color de la lista desplegable de color activa.
- Ecualizar una imagen abierta.
- Aplicar un efecto sepia.
- Aplicar una umbralización a la imagen.
- Operaciones de imágenes propias con un operador LookupOp, operación componente a componente y operación pixel a pixel.

2.3. Requisitos de sonido.

En cuanto a los requisitos de sonido, la aplicación permitirá tanto reproducir como grabar sonidos. Cuando se abra un nuevo sonido, éste se añadirá a una

lista de sonidos disponible en la barra de herramientas, y podrá reproducirlo y parar su reproducción utilizando botones de play y stop. Además, se podrá grabar un sonido utilizando el botón de grabar de forma que, una vez se haya parado la grabación, el usuario escogerá el nombre que desee para el fichero. Si acepta, se guardará el fichero y si cancela, no se guardará ninguno. Los formatos que se pueden utilizar son .au y .wav para la reproducción de sonidos, y para la grabación se creará el fichero temporal de la misma, en un archivo con extensión .au.

2.4. Requisitos de vídeo.

Por último, también se pueden abrir y reproducir vídeos, así como encender la cámara web. Para el caso de un vídeo abierto, se podrá reproducir y parar, así como tomar una instantánea, la cual se abrirá en una nueva imágen para poder aplicarle cualquiera de las operaciones mencionadas en el apartado 2.3.

Además, se podrá abrir la cámara web, de forma que se puedan realizar instantáneas de la misma mientras está activa. De forma opcional, se ha desarrollado la funcionalidad de detección de movimiento de la cámara, de forma que mientras está la ventana de la cámara activa y también activo el sensor de movimiento, cuando se detecte un movimiento se realizará automáticamente una captura del mismo, y se abrirá en una ventana de tipo imagen.

Cualquiera de las instantáneas tomadas podrán procesarse como una imagen cualquiera.

La reproducción de vídeos se podrá realizar con los formatos .mpg y .mp4.

3.- Diseño de las clases utilizadas.

Para ver con un poco más de claridad, se han hecho varios diagramas generales en los que se pueden ver las relaciones entre las clases y sus superclases de las que heredan.

En la figura 3.3. podemos ver, en primera instancia, cómo se ha creado la representación para las figuras que podemos dibujar en el lienzo:

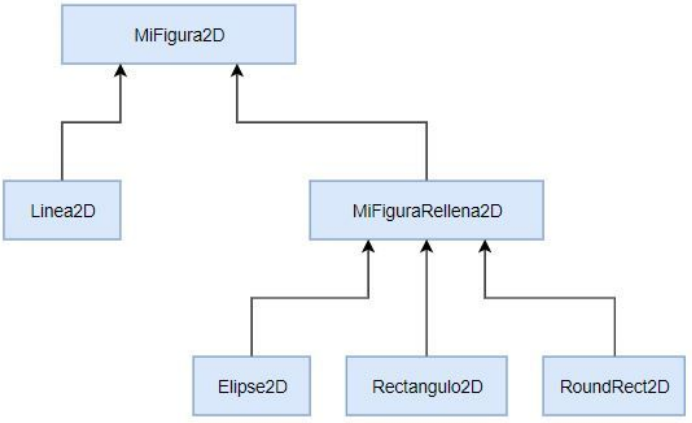


Figura 3.3.: Representación general de las figuras.

Debido a que cada figura debe tener propiedades con sus valores propios, se han creado clases que representen las mismas. Esto se puede ver, de forma genérica, en la figura 3.4.:

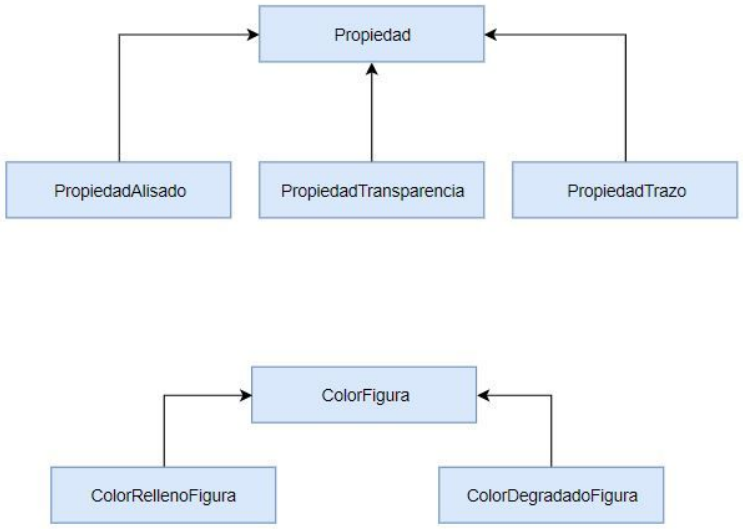


Figura 3.4.: Representación general de las propiedades de una figura.

Por último, podemos ver las clases para el lienzo tanto de dibujo como de imagen. Una hereda de la otra dado que, para una imagen concreta que tengamos abierta, podemos aplicar todo lo mencionado para el lienzo en blanco, es decir, para dibujar. Esto implica que, por tanto, podemos dibujar sobre la imagen y luego guardarla haciendo el dibujo, persistente.

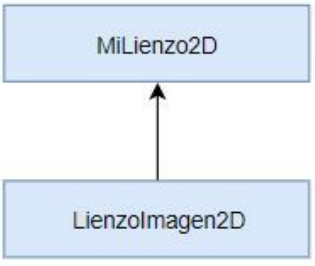


Figura 3.5.: Representación general del lienzo.

Por último, vemos en la figura 3.6. la representación de los tipos de ventanas internas que pueden utilizarse en nuestra aplicación.

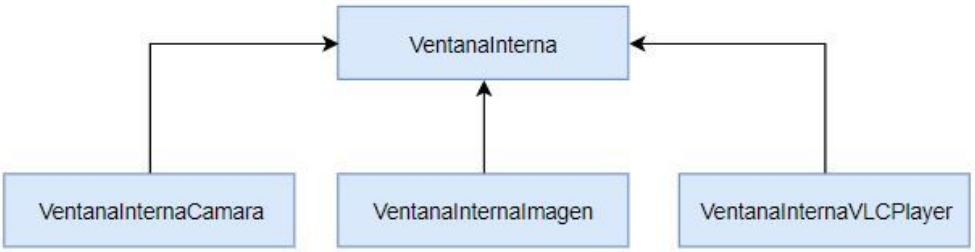


Figura 3.6.: Representación general de las ventanas internas.

3.1. Clases del proyecto principal.

Las clases que conforman el proyecto principal son las que se definen a continuación.

3.1.1. Clase Ventana Principal.

Esta clase es la que contiene todo el área de visualización que hemos visto anteriormente en la figura de la interfaz. Permite gestionar un entorno de ventanas internas, así como aplicar operaciones sobre las mismas.

3.1.2. Clase Ventana Interna.

Esta clase es la que contiene la definición de una ventana interna. Puede ser de varios tipos: Ventana Interna Cámara, Ventana Interna Vídeo y Ventana Interna Imagen. Dentro de la ventana interna de la cámara, podemos visualizar la cámara activa y capturar instantáneas, así como activar el detector de movimiento que también genera una instantántea cada vez que se detecta un movimiento.

La ventana interna de tipo vídeo, permite visualizar un vídeo así como reproducirlo y pararlo/pausarlo como hemos mencionado en los requisitos de vídeo.

Por último, la ventana interna de imagen contiene un panel de tipo LienzoImagen2D que es la que permite visualizar una imagen abierta, así como dibujar y procesar sobre ella.

Los tres tipos de clase heredan de su superclase VentanaInterna.

3.1.3. Clase Establecer Tam Lienzo.

Esta clase permite que el usuario pueda establecer un ancho y alto fijo a su área de dibujo así como dejar el por defecto si desea. Esto determinará el tamaño con el que se crea el área de dibujo en una ventana interna de tipo imagen.

3.1.4. Clase Panel Relleno.

Establece el tipo de relleno que puede tener una figura, a la hora de dibujarla, de forma que podemos aplicar un dibujado sin relleno, con relleno liso o relleno degradado vertical u horizontal.

3.1.5. Clase Redimensión Lienzo.

Establece los nuevos valores al lienzo que se desee de forma que se aumente o disminuya el área de dibujo según quiera el usuario.

3.2. Clases de la biblioteca SM.TDP.Biblioteca.

Las clases que conforman el la biblioteca propia son las que se definen a continuación.

3.1.4. Clase MiFigura2D.

Esta clase define la estructura que va a tener un objeto que representa a cada figura que vamos a poder dibujar. Se ha diseñado de forma que es una clase abstracta, y va a contener los puntos inicial y final de la figura, un objeto de tipo Shape, las propiedades de la figura, como son el tipo de alisado, el tipo y grosor de trazo, la transparencia y el color. Además, define los distintos tipos de figuras que podemos crear.

Está formada, además de con los métodos get y set de las propiedades, por dos métodos abstractos, que cada figura implementará en su respectiva clase. Estos métodos son el de pintar la figura y además el de la modificación de la localización.

Por otro lado, tenemos un método que pintará el recuadro de edición de una figura, de forma que cuando se seleccione en la lista de figuras, se dibuje un recuadro rojo a su alrededor para saber cuál estamos editando.

Clase Linea2D.

Esta clase extiende de la clase MiFigura2D. Posee un constructor que recibe todos los parámetros necesarios para construir una figura. Estos son los mencionados en la clase MiFigura2D.

A la hora de dibujar una figura, implementa el método de forma que primero se establecen sus atributos de dibujo activos en el lienzo, y por último se dibuja la figura utilizando el método graw, dado que una línea no tiene relleno.

El método de localización, establece las coordenadas de la figura, de forma que se determinan los puntos inicial y final de la misma para formar la línea.

Clases Rectangulo2D, Elipse2D y RountRect2D.

Estas clases son muy parecidas entre ellas porque todas tienen las mismas propiedades que la línea, y además poseen relleno. Todas se construyen de igual forma que la línea, pero necesitan de un par de parámetros más. Estos son si la figura se va a pintar con o sin relleno y el tipo, de forma que podamos pintar con relleno liso o degradado, y sus respectivos colores de trazo, frente y fondo.

Al igual que la línea, implementan los métodos de la superclase MiFigura2D pero heredan de la clase MiFiguraRellena2D (la cual hereda de MiFigura2D, por lo que pueden realizar la sobrecarga de los métodos , que son el pintarShape y setLocalizacionShape). Las funcionalidades de estos métodos son las mismas que para la línea de forma que no se entrará en detalle. La única diferencia entre los método de una línea y una figura con relleno se puede ver en la siguiente figura:

```java
/**
 * Dibujado de forma de tipo Linea2D
 * @param g objeto Graphics
 */
@Override
public void pintarShape(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;

    g2d.setRenderingHints(this.prop_alisado.getRender_alisado());
    g2d.setComposite(this.prop_transparencia.getAlfa_transparencia());
    g2d.setStroke(this.prop_trazo.getTipo_trazo());
    g2d.setPaint(this.color_figura.getColor_trazo());
    g2d.draw(this.s);
}

/**
 * Establece la localización de la figura Linea2D
 * @param ini punto nuevo inicial
 * @param fin punto nuevo final
 */
@Override
public void setLocalizacionShape(Point2D ini, Point2D fin) {
    this.punto_ini = ini;
    this.punto_fin = fin;

    ((Line2D)this.s).setLine(this.punto_ini, this.punto_fin);
}
```

```java
/**
 * Dibujado de forma de tipo Elipse2D
 * @param g objeto Graphics
 */
@Override
public void pintarShape(Graphics g) {
    Graphics2D g2d = (Graphics2D)g;

    g2d.setRenderingHints(this.prop_alisado.getRender_alisado());
    g2d.setComposite(this.prop_transparencia.getAlfa_transparencia());

    if(this.si_relleno_liso || this.si_degradado){
        if(this.si_degradado){
            g2d.setPaint( ((ColorDegradadoFigura)this.color_figura).getDegradado() );
        }
        else
            g2d.setPaint( ((ColorRellenoFigura)this.color_figura).getColor_liso() );

        g2d.fill(s);
    }

    g2d.setStroke(this.prop_trazo.getTipo_trazo());
    g2d.setPaint( ((ColorFigura)this.color_figura).getColor_trazo());
    g2d.draw(this.s);

}

/**
 * Establece la localización de la figura Elipse2D
 * @param ini punto nuevo inicial
 * @param fin punto nuevo final
 */
@Override
public void setLocalizacionShape(Point2D ini, Point2D fin) {
    this.punto_ini = ini;
    this.punto_fin = fin;
    if(this.si_degradado){
            ((ColorDegradadoFigura)this.color_figura).setPuntoFin(fin);
        }
    ((Ellipse2D)s).setFrameFromDiagonal(ini, fin);
}
```

Todas las figuras sobrecargan un método toString que se utiliza para poder imprimir el nombre de la figura en la lista de figuras, de forma que podamos reconocer qué figura es cada una.

Clases Propiedad, PropiedadAlisado, PropiedadTrazo y PropiedadTransparencia.

Estas clases representan, de forma genérica mediante Propiedad, las propiedades trazo, alisado y transparencia de una figura. Las clases PropiedadAlisado, PropiedadTrazo y PropiedadTransparencia heredan de Propiedad dado que comparten atributos.

Clases ColorFigura, ColorDegradadoFigura y ColorRellenoFigura.

Estas clases representan, de forma genérica, el color y el relleno de una figura con área. En la clase ColorFigura se define el color de trazo, y las demás clases heredan de ella dado que las figuras con relleno también poseen trazo, de forma que, como comparten esa propiedad se ha realizado de esta forma.

3.1.5. Clase MiOperacionCC.

Esta clase define mi operación propia componente a componente. Debido a que hereda de BufferedImageOpAdapter, he tenido que implementar el método filter para realizar mi filtro propio.

El filtro realiza una operación componente a componente, de forma que, para cada “sample”, cuando el valor de la banda es igual a 2, es decir, la banda azul. se realiza la siguiente operación:

```
// Componente a componente, porque cada cálculo de componente es
// independiente de las demás
for (int x = 0; x < srcRaster.getWidth(); x++) {
    for (int y = 0; y < srcRaster.getHeight(); y++) {
        for (int band = 0; band < srcRaster.getNumBands(); band++){
            int sample = srcRaster.getSample(x, y, band);

            if(band == 2)
                sample = (int) (sample/3 + 255*Math.random());

            destRaster.setSample(x, y, band, sample);
        }
    }
}
```

El resultado de aplicar esta operación se puede ver en la siguiente figura:

3.1.6. Clase MiOperacionPP.

Esta clase define mi operación propia pixel a pixel. Debido a que hereda de BufferedImageOpAdapter, he tenido que implementar el método filter para realizar mi filtro propio.

El filtro realiza una operación pixel a pixel como se puede ver en la siguiente figura:

```
// Pixel a pixel, porque cada cálculo de componente es
// independiente de las demás
for (int x = 0; x < srcRaster.getWidth(); x++) {
    for (int y = 0; y < srcRaster.getHeight(); y++) {
        int sample = srcRaster.getSample(x, y, 1);
        sample = (int) (Math.log((sample*20 - 125)) + (255 * Math.random()));
        destRaster.setSample(x, y, 0, sample);
    }
}
```

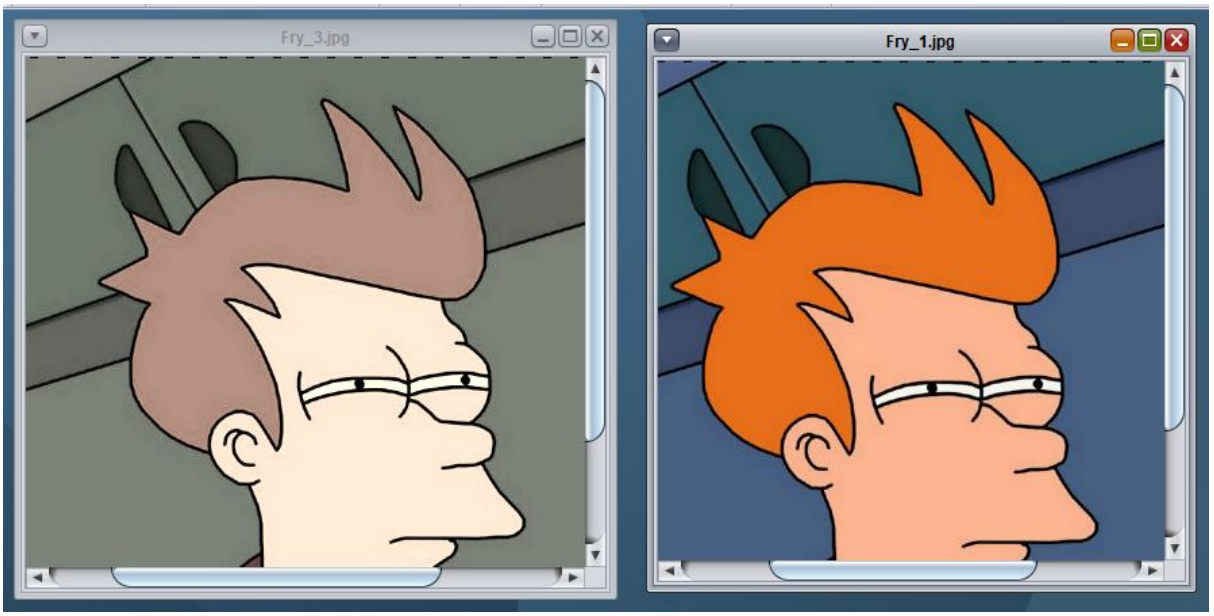
El resultado de aplicar esta operación se puede ver en la siguiente figura:

3.1.6. Clase MiSepiaOp.

Esta clase implementa el filtro sepia para una imagen. Se ha implementado como se ha visto en las prácticas desarrolladas durante el curso, y el método filter para realizar esta operación se puede ver en la siguiente Figura:

```
// Operación pixel a pixel
for (int x = 0; x < src.getWidth(); x++) {
    for (int y = 0; y < src.getHeight(); y++) {

        int[] pixelComp = null;
        pixelComp = srcRaster.getPixel(x, y, pixelComp);

        //sepiaR = min(255, 0.393·R + 0.769·G + 0.189·B)
        //sepiaG = min(255, 0.349·R + 0.686·G + 0.168·B)
        //sepiaB = min(255, 0.272·R + 0.534·G + 0.131·B)

        pixelComp[0] = (int) Math.min( 255, 0.393*pixelComp[0] + 0.769*pixelComp[1] + 0.189*pixelComp[2] );
        pixelComp[1] = (int) Math.min( 255, 0.349*pixelComp[0] + 0.686*pixelComp[1] + 0.168*pixelComp[2] );
        pixelComp[2] = (int) Math.min( 255, 0.272*pixelComp[0] + 0.534*pixelComp[1] + 0.131*pixelComp[2] );

        destRaster.setPixel(x, y, pixelComp);
    }
}
```

Y el resultado de aplicar el efecto sepia a mi imagen es el que se ve en la siguiente figura:



3.1.7. Clase MiUmbralizacionOp.

Esta funcionalidad no me funciona...

3.1.8. Clase LienzoImagen2D.

Esta clase representa la imagen que podemos abrir en nuestra aplicación y como extiene de MiLienzo2D, ofrece, además de poder dibujar sobre las mismas, la aplicación de filtros y transformaciones sobre imágenes.

El procesamiento que podemos aplicarle a una imagen puede ser uno de los siguientes, aunque también es posible concatenar tantos como se desee:

- Redimensionar imagen.
- Aplicar función seno.
- Girar una imagen con ángulo fijo o personalizado.
- Escalar una imagen.
- Obtener la imagen negativo.
- Aplicar contraste iluminado, normal y oscurecido.
- Extracción de bandas.
- Aplicar el tintado en base al color seleccionado en la lista desplegable.
- Ecualizar una imagen.
- Obtener un efecto sepia.
- Aplicar las operaciones propias px a px y componente a componente.
- La umbralización no funciona, pero el código está creado.

3.1.9. Clase MiLienzo2D.

Esta clase es la que gestiona la creación de un objeto MiFigura2D, así como la edición de cada una de las propiedades de la figura seleccionada para editarla. Además de tener los get y set para cada propiedad del lienzo, (que son las que se le pasan a las figuras creadas) gestiona tres eventos para la creación de las figuras:

- **formMousePressed** : Para la creación de una figura, se llama al método createShape que se encarga de crear los objetos de tipo MiFigura2D

según la figura que se está dibujando. En caso de estar en modo edición, no se podrá dibujar dado que la lista se encargará de gestionar la edición de una figura seleccionada.

- **formMouseDragged**: Se encarga de actualizar el último punto de cada figura en el momento en que la estamos creando, de forma que cuando el usuario suelte el ratón y pare de dibujar la figura, ya se ha terminado de crear del todo.

4.- Bibliografía.

[1] Guiones de las prácticas realizadas durante el curso académico.
[2] Temario de teoría de la asignatura.
[3] Guión de la práctica de Evaluación.