Sistemas Multimedia

Práctica final

DANIEL RUIZ MEDINA

2020-2021

Índice

Requerimientos

Lista de requisitos funcionales encontrada en el documento del proyecto, ordenada por bloques:

1. Bloque de gráficos

RF1. Se debe permitir pintar las figuras: líneas, rectángulos, óvalos, rectángulos con bordes redondeados y trazo libre.

RF2. El lienzo o zona de dibujo debe ser capaz de mantener todas las figuras que se vayan dibujando.

RF3. Cada figura tendrá sus propios atributos, independientes del resto de formas.

RF4. Definir una jerarquía de clases asociadas a formas y sus atributos.

RF5. El usuario podrá editar los atributos de cada figura, simplemente clicando sobre ella mientras está pulsado el botón edición (Mover).

RF6. Se podrá elegir el color tanto del borde como del atributo relleno de cada figura.

RF7. El usuario podrá modificar los atributos tipo de discontinuidad (líneas continuas o punteadas) y grosor de la figura.

RF8. El cliente podrá elegir el tipo de atributo relleno (sin color de fondo, con color de fondo o fondo degradado) de cada figura. En el caso de ser fondo degradado se permite elegir la dirección (vertical u horizontal) de este.

RF9. Se podrá activar o desactivar el atributo alisado de bordes de cada figura.

RF10. El usuario podrá establecer el atributo transparencia de las figuras y elegir el grado del mismo.

RF11. El cliente podrá mover la figura seleccionada mediante una operación de arrastrar y soltar el ratón.

2. Bloque de imágenes

RF12. Se podrá duplicar una imagen en una ventana nueva.

RF13. El cliente podrá modificar el brillo de la imagen.

RF14. El usuario podrá aplicar los filtros emborronamiento horizontal, emborronamiento vertical, enfoque y relieve sobre la imagen.

RF15. Se podrá modificar el contraste normal, iluminado y diagonal de la imagen.

RF16. El cliente podrá aplicar un contraste negativo sobre la imagen.

RF17. El usuario usar el operador cuadrático y variar su nivel de la imagen.

RF18. Se podrá extraer las bandas de una imagen.

RF19. El cliente podrá variar el espacio de una imagen entre RGB, YCC y Gray.

RF20. El usuario podrá realizar un giro libre de las imágenes.

RF21. Se podrá aumentar o disminuir el escalado da cada imagen.

RF22. El cliente podrá usar un operador titando con un color seleccionado y variar el grado de mezcla sobre la imagen.

RF23. El usuario dispondrá de un tintado con selección automática de umbral.

RF24. Se podrá aplicar el operador sepia sobre la imagen.

RF25. El cliente podrá hacer uso del operador ecualización a la imagen.

RF26. EL usuario podrá posterizar la imagen y reducir los niveles por banda.

RF27. Se podrá realizar una operación que permita resaltar el rojo de una imagen y variar el umbral de selección del tono rojo.

RF28. El usuario podrá hacer uso de un operado “umbralT” basado en aclarar las zonas oscuras y dejar igual las claras mediante un umbral. Este umbral se podrá variar.

RF29. (**Propio) El usuario podrá sumar un valor determinado a cada pixel de la imagen.

RF30. (**Propio) El cliente dispondrá hacer uso de la operación arco tangente sobre la imagen.

RF31. Se podrá enverdecerla imagen con una operación de combinación de bandas.

3. Bloque de sonido

RF32. Se permitirá reproducir sonidos con los formatos waw y au.

RF33. El usuario podrá grabar sonidos.

RF34. Se podrá reproducir y parar un audio.

4. Bloque de video

RF35. El cliente podrá reproducir videos con formato compatible con vlcplayer.

RF36. El usuario podrá grabar videos haciendo uso de su WebCam.

RF37. Se podrán tomar capturas instantáneas de videos en reproducción o de la WebCam encendida.

5. Interfaz de Usuario

RF38. La aplicación dispondrá de dos barras de herramientas superior e inferior dedicadas, la superior a la parte de dibujo y la inferior a tema de imágenes.

RF39. Dispondrá de una barra de menú con las opciones archivo y ayuda.

RF40. La opción archivo tendrá a su vez las subopciones de nuevo, abrir y guardar.

RF41. Se añadirá a la barra superior la opción de nuevo, abrir y guardar.

RF42. El usuario podrá seleccionar la opción nuevo y creará una nueva ventana interna con un lienzo donde poder dibujar en el escritorio.

RF43. El cliente mediante la opción abrir podrá abrir nuevas imágenes, videos o audios.

RF44. Las imágenes y videos abierta se abrirán en una nueva ventana interna en el escritorio.

RF45. La aplicación contará con una zona central escritorio donde se crearán ventanas internas con lienzos donde se podrán realizar acciones.

RF46. Se permite guarda imágenes en los formatos permitidos.

RF47. Se lanzarán diálogos con las opciones abrir y guardar, donde se podrán aplicar los filtros de formatos seleccionados y elegir un archivo o guardar.

RF48. Cada botón o slider de la interfaz contendrá un tooltip asociado.

RF49. Los botones de atributos de dibujo se actualizarán con los valores de la figura seleccionada.

RF50. El usuario podrá ser consciente de las acciones que esta haciendo con una barra de estado que informará sobre ellas.

Análisis

1. Bloque de gráficos

¿Qué hay para poder abordar este apartado?

Para abordar el apartado de gráficos java dispone de clases tipo shape que a nivel de gráficos nos ayudarían a la representación geométrica de las distintas figuras planteadas. Java también dispone de clases desacopladas para definir y tratar los diferentes atributos de las formas.

¿Qué problemas se contemplan?

El problema surge es que buscamos poder modificar y guardar los atributos de cada forma dibujada en el gráfico y puesto que las clases que dispone JAVA están desacopladas y por tanto no podemos resolver los requerimientos RF3, RF4, RF5, RF6, RF7, RF8, RF9, RF10, RF11.

¿Como se piensa solucionar?

En primer lugar, planteé la opción de tener una clase por figura, por ejemplo, que la clase rectángulo heredase de Rectangulo2D.Double para seguir manteniendo las listas y el uso de Shapes. Y que esta clase tenga un atributo, instancia de otra clase, que tiene los atributos y métodos de los atributos. Esta primera opción fue descartada debido a que se incumplía el formato especifico de impresión en el método Paint () de nuestro lienzo.

Por la opción por la que se optó fue crear una clase padre con todos los atributos y métodos de las figuras y crear una clase por cada clase figura planteada que herede de la clase padre. Esta clase a su vez dispone de un atributo forma donde se crear la figura.

2. Bloque de imágenes

¿Qué hay para poder abordar este apartado?

Se ha encontrado que java dispone de un conjunto de clases (RescaleOp, ConvolveOp, …) que heredan de BufferedImageOp que disponen de una serie de operadores y el uso de otras clases (ByteLookupTbale, ColorSpace, BufferedImage, …) con los que podemos resolver los requerimientos RF12, RF13, RF14, RF15, RF16, RF17, RF18, RF19.

Con la clase AffineTransform que también nos proporciona java podemos hacer los requerimientos RF20, RF21.

¿Qué problemas se contemplan?

El principal problema es que existen requerimientos (RF23, RF24, RF25, RF26, RF27, RF28, RF29, RF30, RF31) que no podemos resolver con las clases que nos ofrece java.

¿Como se piensa solucionar?

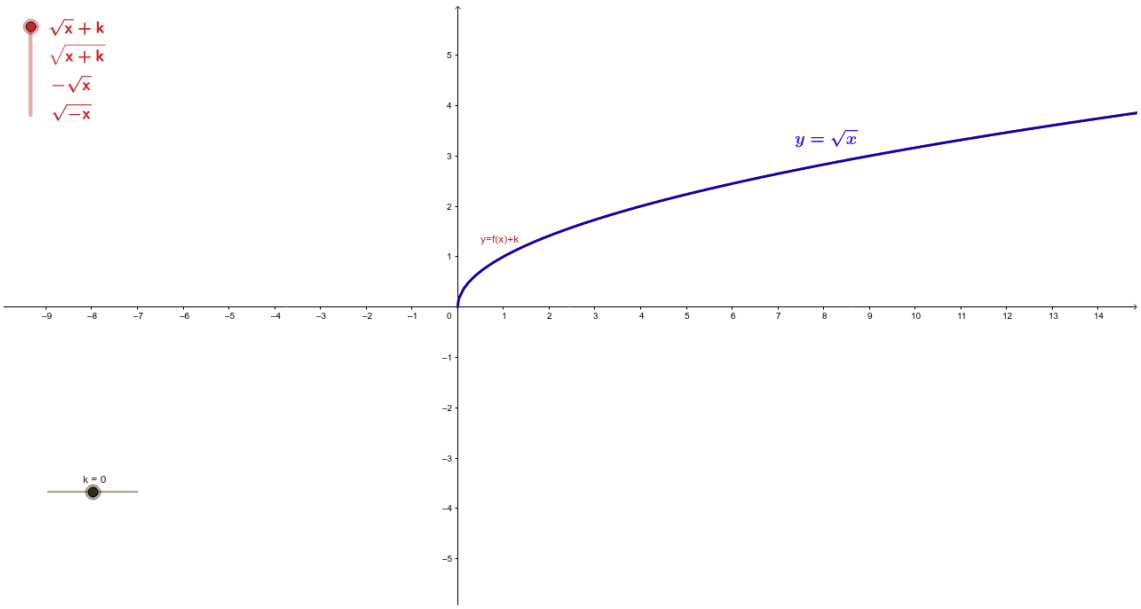
Por lo que además de utilizar el conjunto de clases de java para los requerimiento RF12, RF13, RF14, RF15, RF16, RF17, RF18, RF19, RF20, RF21 se va a utilizar la librería sm.image de la asignatura con la cual podemos abordar los requerimientos RF23, RF24, RF25, RF26, RF27.

Y para los requerimientos RF28, RF29, RF30, RF31 se van a crear clases que hereden de las clases TintOp y BufferedImageOpAdapter (clases de la librería sm.imagen) para realizar los operadores requeridos.

****Explicación de cada operador propio:**

- **RF28. El usuario podrá hacer uso de un operado “umbralT” basado en aclarar las zonas oscuras y dejar igual las claras mediante un umbral. Este umbral se podrá variar.**

  En este operador pasaremos un umbral con el cual controlaremos hasta que valores se debe aumentar la luminosidad y utilizando la función raíz lograremos que a mayor valor de oscuridad conseguir una mayor luminosidad.

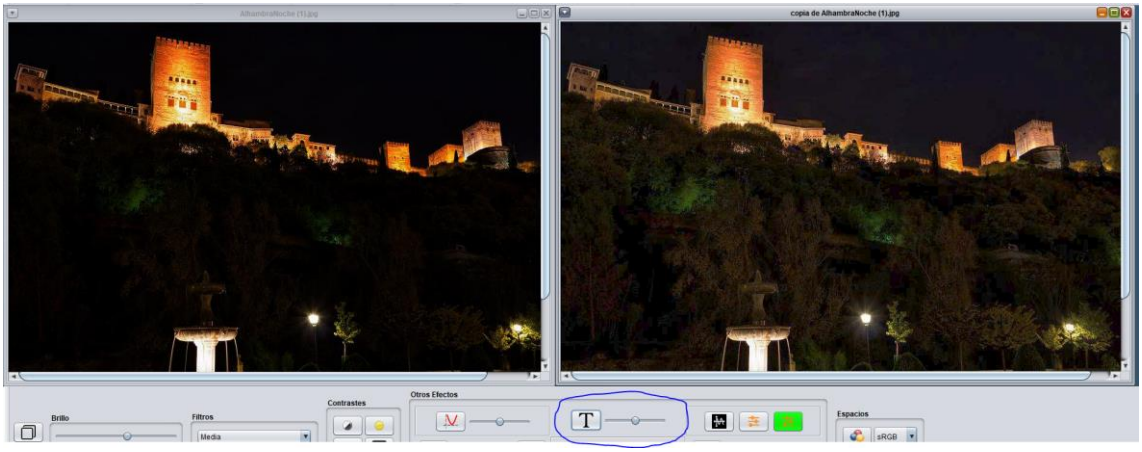


Puesto que nuestros valores deben estar comprendidos entre los valores [0,255] se calcula una constante que se multiplica para en caso de superar dichos valores calcular su modulo.

Ejemplo de código: donde calculamos la constante k1 para realizar el modulo de la operación. Y generando la tabla donde con el umbral ‘m’ realizamos la operación raíz, consiguiendo que dicho valor disminuya de forma proporcionar, para lograr una iluminación suave y el efecto de la imagen se mantenga lo mejor posible.

```
public LookupTable umbralT(double m) {

    double Max = Math.sqrt(m);
    double K1 = m / Max;

    byte lt[] = new byte[256];
    lt[0] = 0;

    for (int l = 1; l < 256; l++) {
       if (l <= m) {
          lt[l] = (byte) (K1 * Math.sqrt(l));
       } else {
          lt[l] = (byte) l;
       }
    }

    ByteLookupTable slt = new ByteLookupTable(0, lt);
    return slt;    }
```

Esta es la imagen donde podemos apreciar que el efecto conseguido es el planteado, aclarar hasta un umbral consiguiendo para esta imagen ver la zona oscura del cielo y arboleda de forma más clara.



RF29. (Propio) El usuario podrá sumar un valor determinado a cada pixel de la imagen.**

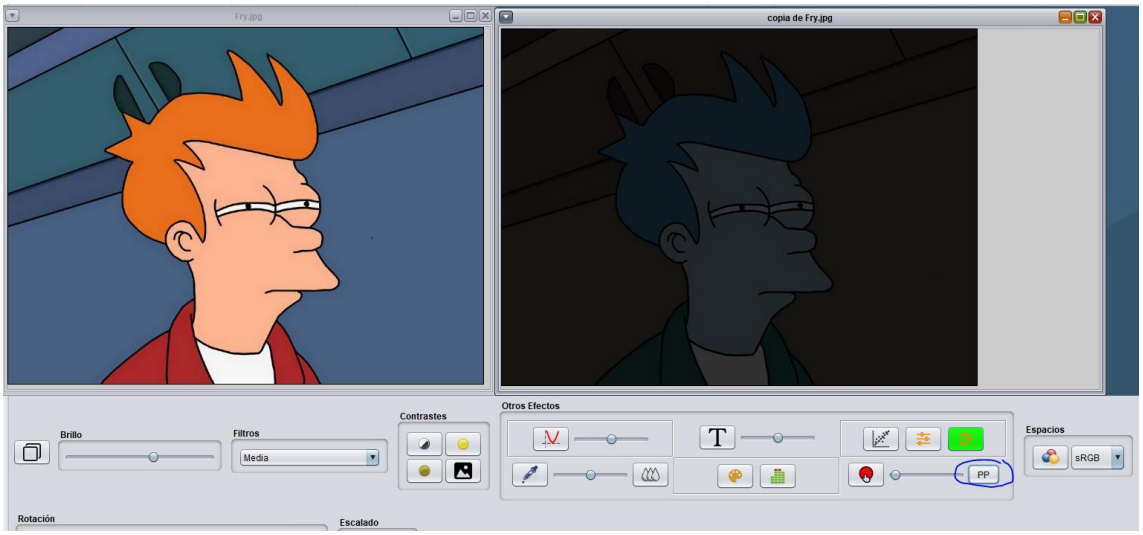
La operación que se plantea para crear un operador donde se trabaje pixel a pixel es para cada pixel hace la suma de los pixeles que no son el entre el parámetro pasado por la aplicación y todo esto modulo 255 para no exceder nunca el valor de 255.

```
public class DOperador extends BufferedImageOpAdapter{
   private int parametro;

   /**
    * Constructor donde definimos el parametro a sumar.
    *
    * @param parametro valor se desea sumar sumar.
    */
   public DOperador(int parametro) {
      this.parametro = parametro;
   }

   /**
    * Método para aplicar un filtro a una imagen origen y se devuelve
    * en una destino.
    *
    * @param src imagen a la que aplicar el filtro
    * @param dest imagen donde se desea aplicar el filtro
    *
    * @return (tipo: BufferedImage) imagen con el filtro aplicado.
    */
   @Override
   public BufferedImage filter(BufferedImage src, BufferedImage dest) {
      if (src == null) {
```

```
            throw new NullPointerException("src image is null");
        }
        if (dest == null) { //si la imagen destino es null se crea
            dest = createCompatibleDestImage(src, null);
        }
        WritableRaster srcRaster = src.getRaster();
        WritableRaster destRaster = dest.getRaster();
        int[] pixelComp = new int[srcRaster.getNumBands()];
        int[] pixelCompDest = new int[srcRaster.getNumBands()];

        for (int x = 0; x < src.getWidth(); x++) {
            for (int y = 0; y < src.getHeight(); y++) {
                srcRaster.getPixel(x, y, pixelComp);

                pixelCompDest[0] = (pixelComp[1]+pixelComp[2])/parametro % 255;
                pixelCompDest[1] = (pixelComp[0]+pixelComp[2])/parametro % 255;
                pixelCompDest[2] = (pixelComp[1]+pixelComp[0])/parametro % 255;

                destRaster.setPixel(x, y, pixelCompDest);
            }
        }
        return dest;
    }
}
```

Consiguiendo que para cada pixel pintar algo parecido a su contrario y como el parámetro lo divide controlamos además la luminosidad de la imagen, esta imagen es para el parámetro 3.

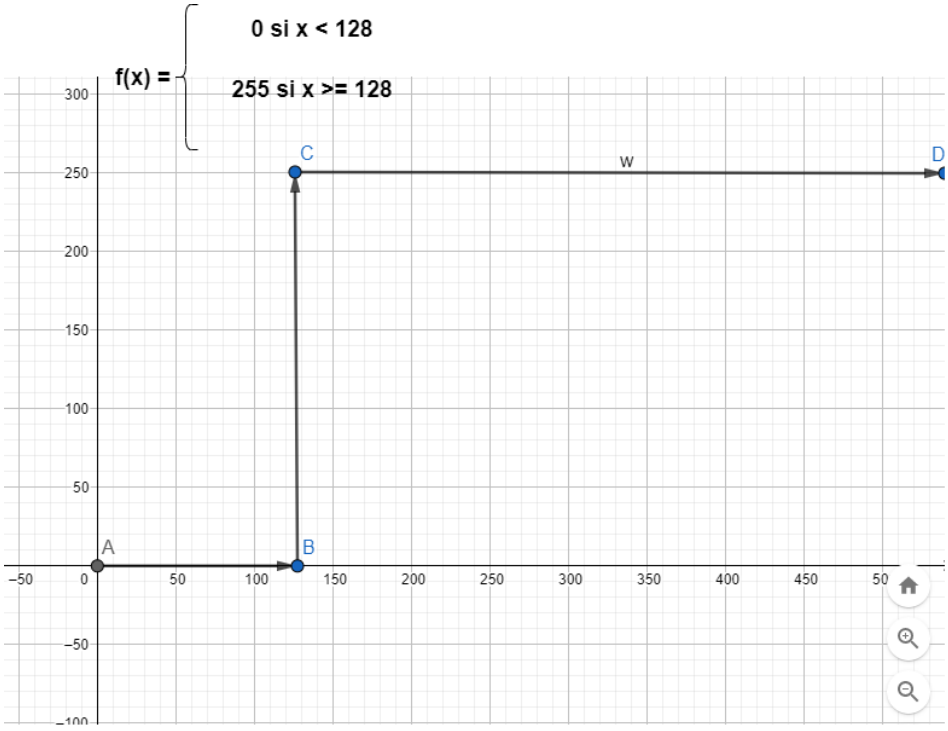
Y esta imagen con el parámetro 10, contemplando que a mayor valor mas oscura se vera la imagen, además de aplicar la operación de los pixeles que se produce como una inversión de los mismos.



RF30. (Propio) El cliente dispondrá hacer uso de la función binarizar sobre la imagen.**

Con esta función lo que hacemos es que los valores por debajo de un valor se le asigna un valor mínimo y los de por encima el valor máximo**.** Por tanto, logramos un efecto binarizado donde solo se mostrarán 8 colores que contengan en sus diferentes bandas RGB valores 0 o 255.

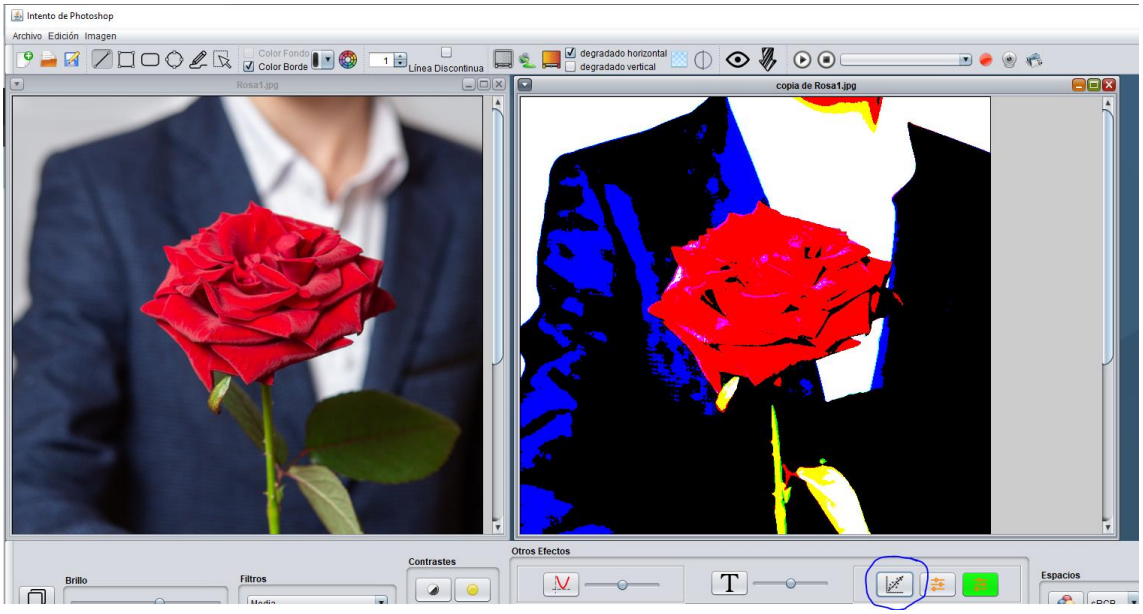
Negro (r=0, g=0, b=0), blanco (r=255, g=255, b=255), rojo (r=255, g=0, b=0), …, cian (r=0, g=255, b=255)



Este es el código implementado en el cual no necesitamos poner ninguna constante ya que los valores siempre van a estar entre [0,255].

```
public LookupTable binarizar(double m) {

    byte lt[] = new byte[256];

    for (int i = 0; i < 256; i++) {
       lt[i] = (i < m) ? (byte) 0: (byte)255;
    }

    ByteLookupTable slt = new ByteLookupTable(0, lt);
    return slt;
  }
```

A continuación, se muestra la imagen donde se pude ver como hemos binarizado haciendo que los colores con los que se pinta la imagen sea solo los que contengan en su rgb 0 o 255.



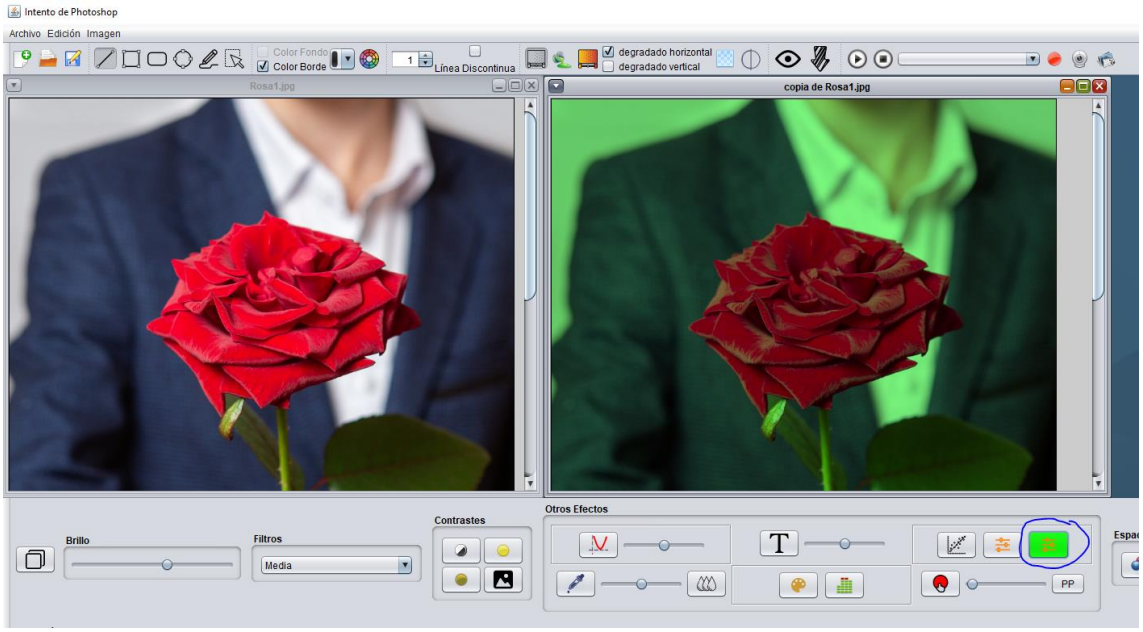
RF31. Se podrá enverdecerla imagen con una operación de combinación de bandas.

Para lograr un efecto de enverdecer con la operación de combinación de banda he creado una matriz donde multiplico por 0,5 red y blue (RGB). Y la banda green la mantengo con su nivel, logrando un efecto de enverdecer cuando hacemos la operación de combinación.

```
private void btnEnverdecerActionPerformed(java.awt.event.ActionEvent evt) {
    VentanaInternaImagen vi = (VentanaInternaImagen) (escritorio.getSelectedFrame());
    if (vi != null) {
       BufferedImage img = vi.getLienzo2D().getImagen();
       if (img != null) {

          try {
```

```
            //definimos matriz de combinacion (verde con azul)
            float[][] matriz = {{0.5F, 0.0F, 0.0F},
            {0.0F, 1.0F, 0.0F},
            {0.0F, 0.0F, 0.5F}};
            BandCombineOp bcop = new BandCombineOp(matriz, null);
            //se aplica sobre el raster
            bcop.filter(img.getRaster(), img.getRaster());
            vi.getLienzo2D().repaint();
          } catch (IllegalArgumentException e) {
            System.err.println(e.getLocalizedMessage());
          }
        }
      }
    }
```

Imagen donde vemos la imagen real a la izquierda y el efecto enverdecido a la derecha.



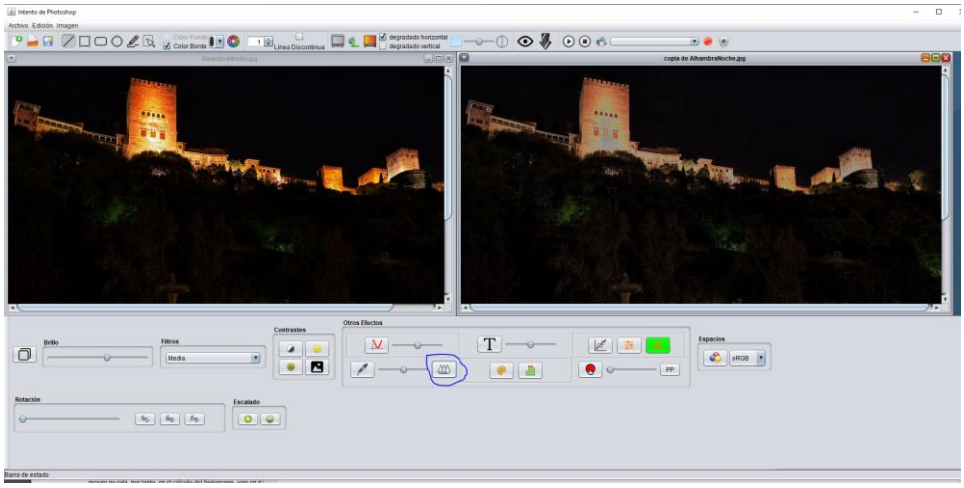
RF23. El usuario dispondrá de un tintado con selección automática de umbral.

Para afrontar este requerimiento se ha creado una clase para el operador llamada TintadoAutoOp, que podemos ver a continuación, donde se ha aplicado la formula de teoría g(x,y) = a * C + (1-a)f(x,y). Siendo C el color gris y a el umbral, este lo calculamos como indica en la práctica 12($\alpha(x, y)$ = $I(x, y)$/255, siendo $I(x, y)$ = ($r(x, y)$ + $g(x, y)$ + $b(x, y)$)/3).

Método filter de nuestro operador:

```
  public BufferedImage filter(BufferedImage src, BufferedImage dest) {
    if (src == null) {
      throw new NullPointerException("src image is null");
    }
    if (dest == null) {
      dest = createCompatibleDestImage(src, null);
    }
    WritableRaster srcRaster = src.getRaster();
    WritableRaster destRaster = dest.getRaster();
    int[] pixelComp = new int[srcRaster.getNumBands()];
    int[] pixelCompDest = new int[srcRaster.getNumBands()];

    for (int x = 0; x < src.getWidth(); x++) {
      for (int y = 0; y < src.getHeight(); y++) {
        srcRaster.getPixel(x, y, pixelComp);

        int media = (pixelComp[0] + pixelComp[1] + pixelComp[2]) / 3;
        double umbral = media / 255.0;

        pixelCompDest[0] = (int)( ( umbral * Color.GRAY.getRGB() ) +(1.0-
umbral)*pixelComp[0]);
        pixelCompDest[1] = (int)( ( umbral * Color.GRAY.getRGB() ) +(1.0-
umbral)*pixelComp[1]);
        pixelCompDest[2] = (int)( ( umbral * Color.GRAY.getRGB() ) +(1.0-
umbral)*pixelComp[2]);

        destRaster.setPixel(x, y, pixelCompDest);

      }
    }

    return dest;
  }
```

Imagen donde se ve el operador aplicado:



3. Bloque de sonido

¿Qué hay para poder abordar este apartado?

Se ha encontrado que java dispone de un conjunto de clases (Clip, AudioSystem, …) del paquete javax.sound.sampled con el cual podemos resolver los requerimiento RF32, RF33, RF34.

¿Qué problemas se contemplan?

No he encontrado ningún problema, solo que las librerías son viejas y puede dar fallo de compatibilidad con muchos formatos de audio actuales.

¿Como se piensa solucionar?

Se ha elegido usar el paquete sm.sound aportado por la asignatura, el cual contiene las clases SMClipPlayer y SMSoundRecorder para cumplir con los requisitos RF32, RF33, RF34.

4. Bloque de video

¿Qué hay para poder abordar este apartado?

Se hay un paquete llamado javafx.scene.media con el cual se podría reproducir videos.

¿Qué problemas se contemplan?

Las librerías que se han encontrado son viejas, no muy claras y no he visto nada de instantáneas o webcam. Incumpliendo los requisitos RF35, RF36, RF37.

¿Como se piensa solucionar?

Se ha elegido usar las clases WebcamPanel y webcam del paquete com.github.sarxos.webcam aportado por la asignatura para poder afrontar los requisitos RF35, RF36, RF37.

5. Interfaz de Usuario

¿Qué hay para poder abordar este apartado?

Java cuenta con clases para ventanas principales, internas, escritorios, botones, sliders, ….

¿Qué problemas se contemplan?

Que las clases o ventanas se deben adaptar para conseguir que las ventanas trabajen de la forma deseada.

¿Como se piensa solucionar?

Por lo tanto, se creará una ventana principal con un escritorio donde podremos abrir ventanas internas e interactuar con ellas con nuestros actuadores (botones, slider) de nuestra barra de herramientas. Cumpliendo los requisitos RF38, RF39, RF40, RF41, RF45, RF48.

Existirá una jerarquía de ventanas internas para poder trabajar con figuras, imágenes y videos, donde cada ventana dispondrá de un lienzo donde podremos realizar acciones de dibujo sobre ella. Completando los requerimientos RF42, RF43, RF44.

Además, se crearán filtros para utilizar a la hora de abrir y guardar documentos facilitándonos el conocimiento de formatos compatibles con nuestra aplicación (RF46, RF47).

También utilizaremos eventos que avisen de acciones a la ventana principal sobre el lienzo para resolver los requerimientos RF49 y RF50.

Diseño

1. Bloque de gráficos

Diagrama de clases de las Figuras

Por lo tanto, se va a crear una clase padre DFormas que va a contener los atributos y métodos comunes de las figuras. Y por cada figura se agregar una clase propia que herede de dicha clase padre.

Cada clase hija dispondrá de un atributo forma que se inicializará con su correspondiente constructor, usando las clases shape de java mencionadas (ejemplo: la clase DRectangulo2D su atributo forma se instancia CON Rectangle2D.Double).

Además, cada clase dispone de su método privado Paint dibujando su correspondiente forma según los atributos heredados del padre.

El método setDragradado()->(degrado) es implementado en cada clase figura por que se calcula la dirección del degradado con una operación de puntos de cada forma.

2. Bloque de imágenes

Este es el diagrama de clase donde vemos los atributos y métodos que se creará por cada operador requerido. Esto heredan de TintOP y BufferedImageOpAdapter de la librería sm.imagen aportada por la asignatura.

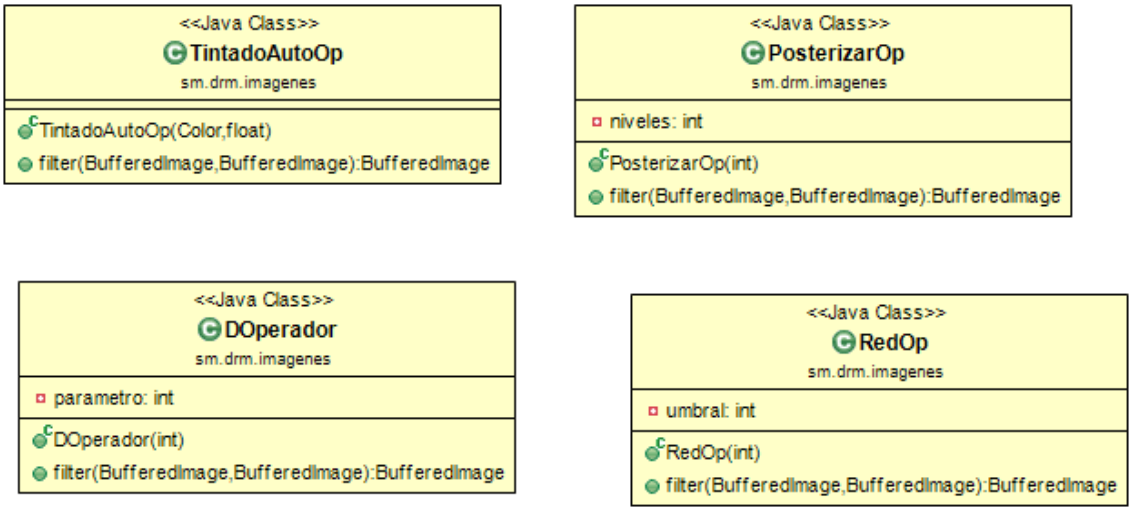


Diagrama de clases de operadores

Además de los constructores, cada clase tiene un método filter donde se efectuarán las diferentes operaciones sobre la imagen.

3. Bloque de sonido

Puesto que las clases aportadas por la asignatura son suficientes, no se ha diseñado ninguna clase extra. Pero si se ha creado una clase privada ManejadorAudio en el archivo de la clase ventanaPrincipal para avisar sobre los distintas acciones que ocurren en el audio (empieza, se detiene, para,..) a la ventana principal.

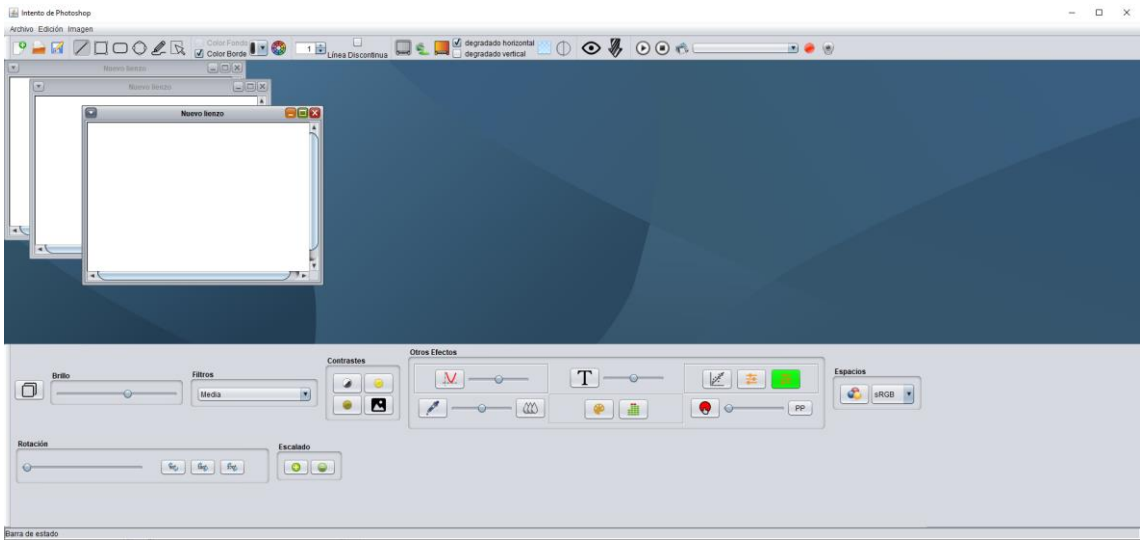
```
private class ManejadorAudio implements LineListener {

    /**
     * Método utlizado para indicar se ha pulsado los botones reproducir,
     * parar o cerrar.
     *
     * @param event evento lanzador.
     */
    @Override
    public void update(LineEvent event) {
       if (event.getType() == LineEvent.Type.START) {
          btnReproducir.setEnabled(false);
       }
       if (event.getType() == LineEvent.Type.STOP) {
          btnReproducir.setEnabled(true);
       }
       if (event.getType() == LineEvent.Type.CLOSE) {
       }

    }
 }
```

4. Bloque de video

Debido a que las clases aportas son suficiente no se ha tenido que diseñar ninguna clase adicional para completar los requerimientos.

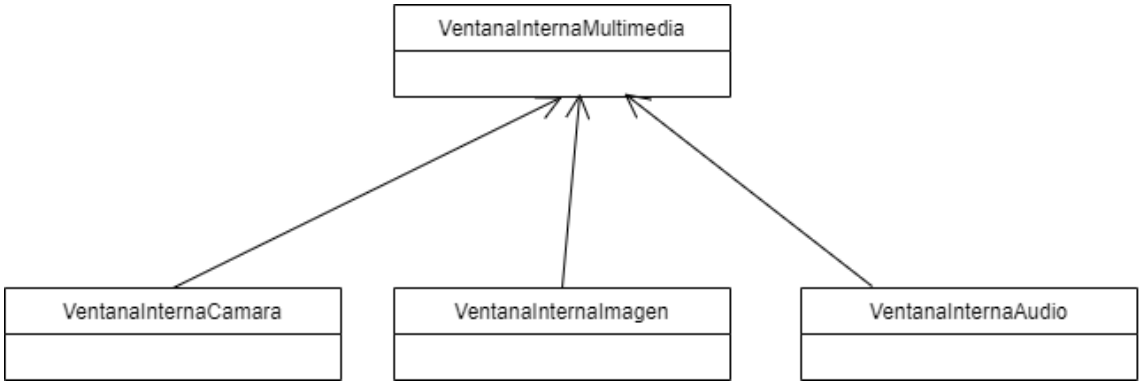
5. Interfaz de Usuario

Crearemos una ventana principal como la que se muestra, en la cual tendremos las diferentes barras de herramientas con sus actuadores, barra de menú con las distintas opciones y un escritorio central donde podremos abrir ventanas internar donde podremos dibujar, procesar imágenes y videos. También contaremos con una barra de estado de pie de ventana para avisar de las acciones al usuario.



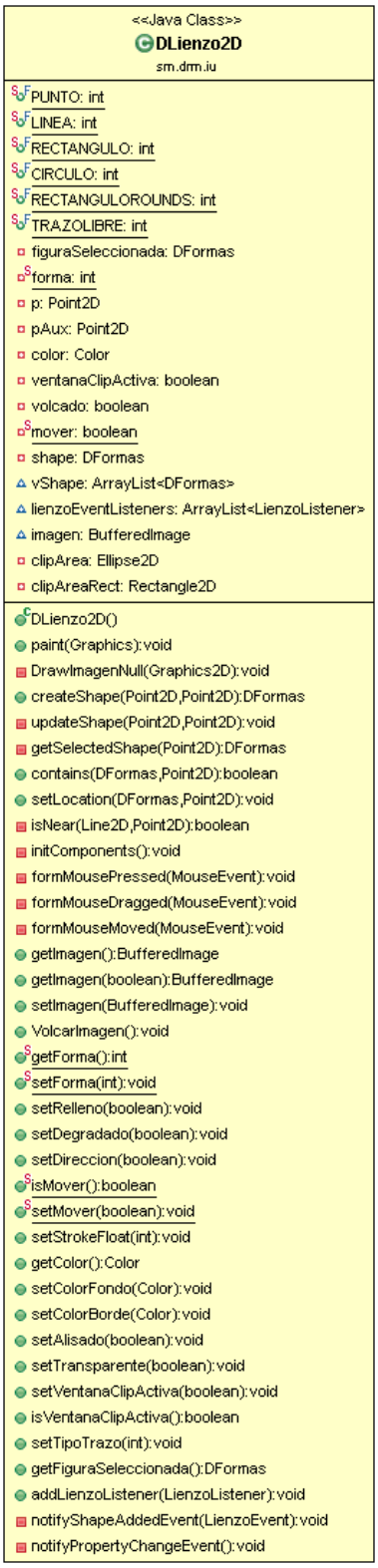
Ventana Principal

Las ventanas internas seguirán la jerarquía que se muestra en el siguiente diagrama, donde tenemos una ventana padre VentanaInternaMultimedia y sus ventanas hijas. Donde cada ventana hija tendrá los métodos propios según su uso (Ejemplo: VentanaInternaCamara podrá realizar instantáneas haciendo getImagen() mientras que en VentanaInternaImagen obtendremos la imagen que con la que se está operando).



jerarquía de ventanas

Cada ventana dispondrá de un lienzo, centrado en la ventana, para la cual se creará la clase DLienzo2D con la cual se podrá realizar las operaciones requeridas sobre imágenes, figuras, videos y demás. La cual contará con los siguientes atributos y métodos para controlar las diferentes operaciones sobre él.



clase DLienzo2D

Además, se crearán filtros para utilizar a la hora de abrir y guardar documentos facilitándonos el conocimiento de formatos compatibles con nuestra aplicación (RF46, RF47).

También utilizaremos eventos que avisen de acciones a la ventana principal sobre el lienzo para resolver los requerimientos RF49 y RF50.

Implementación

Se ha documentado utilizando el generador de documentación de Java(javadoc), las carpetas se encuentran en la carpeta raíz de la entrega.

Documentación de la biblioteca.

Documentación de las ventanas principales e internas.