Documentación
Práctica de evaluación
Sistemas Multimedia

Realizado por:

Carlos Doblas Sánchez

Introducción

En esta documentación se abordarán las clases de diseño propio implementadas para la realización de la práctica de evaluación de la asignatura.
Estás serán analizadas siguiendo las pautas propias de la Ingeniería del Software, haciendo especial hincapié en la descripción y justificación de las mismas.

Las clases implementadas se han dividido en dos secciones, una será la parte de *gráficos* en las que se encuentran las clases propias para las distintas formas de dibujo, y la del lienzo. Y por otra parte tendremos la parte de *imagen* en la que se analizarán las clases que implementan el operador de LookupOp y la operación de diseño propio aplicada pixel a pixel.

Gráficos

Esta parte era la más complicada de las dos, debido a que se tenía que rediseñar su funcionamiento ya que se debían de realizar mejoras que no se podían llevar a cabo con el diseño inicial. Para ello tuve que crear una clase para cada forma de dibujo y rediseñar el antiguo lienzo. A todas las clases creadas para poder llevar a cabo esta tarea les he puesto el prefijo *New* seguido del nombre de la clase.
La clase en la que se basa mi diseño de la parte de gráficos es la clase ***NewFigura*** de la que heredan el resto de figuras implementadas y es la clave de su sencillo funcionamiento.

Requisitos

RF-1. Es obligatorio definir clases propias para las distintas formas de dibujo consideradas.
RNF-2. Cada figura tendrá sus propios atributos.
RF-3. El usuario podrá mover la figura seleccionada mediante una operación de “arrastrar y soltar” con el ratón.
RF-4. Se podrá mover la figura seleccionada mediante una operación de “arrastrar y soltar” con el ratón.
RF-5. La definición de una jerarquía de clases asociadas a las formas y sus atributos.
RF-6. El lienzo mantendrá todas las figuras que se vayan dibujando.
RF-7. El usuario podrá editar las figuras ya dibujadas.

RF-8. Al seleccionar una figura, deberán de activarse sus propiedades en la barra de dibujo.
RF-9. El usuario podrá elegir el color del trazo y el de relleno.
RF-10. Se podrán modificar el grosor y el tipo de discontinuidad del trazo.
RF-11. El usuario podrá elegir entre tres opciones a la hora de rellenar: no rellenar, rellenar con un color liso o rellenar con un degradado.

Análisis

Se partía de un diseño en donde el método Paint de la clase Lienzo era el encargado de dibujar todas las figuras con los atributos que estaban activos en este. Esto hacía que todas las figuras se dibujasen con los mismos atributos y tan solo se almacenasen la forma de los mismos. Con este diseño no se podían llevar a cabo el cumplimiento de los requisitos ya que cada figura debía de tener sus propios atributos.

Debido a este gran inconveniente tuve que rediseñar la clase Lienzo2D cambiando en gran medida la manera en la que se debía de operar con las figuras abstrayendo todo el trabajo de las mismas creando clases propias para cada figura.

Para la representación de las figuras, se diferencian dos tipos de figuras.

- Figuras abiertas: Son las figuras que no se encuentran delimitadas, las que no forman un recinto, estas son el punto, la línea, el trazo libre y la curva con un punto de control. Para su representación se ha creado una clase abstracta llamada NewFigAbierta.

- Figuras cerradas: Son las figuras que se encuentran delimitadas, teniendo un área definida. Estas figuras son el rectángulo, y la elipse. Para su representación se ha creado una clase abstracta llamada **NewFigCerrada.**

Estas dos clases se tratarán con profundidad adelante, pero su diseño viene dado a que las figuras cerradas cuentan con una propiedad que las diferencia de la abiertas como es **el relleno**. Esto hace que se tenga que tratar esta propiedad adicionalmente en estas figuras. Además el hecho de que estén delimitadas puede dar a la implementación de diversos métodos geométricos como puede ser el cálculo del área o del perímetro.

NewFigura

Justificación

Está fue la decisión más complicada y que más tuve que reflexionar para la práctica, debido a que la parte de gráficos depende en gran medida de cómo están diseñadas las figuras que se van a dibujar. Para ello tenía dos alternativas para el diseño de la clase NewFigura:

1. Implementar una interfaz Figura que la tuviesen que implementar todas las figuras. Esto haría que cada figura pudiera heredar de su figura en cuestión y a la vez implementar esta interfaz. Pero no me convencía debido a que había que repetir mucho código para cada figura (métodos y atributos) y que no podía haber jerarquía de clases.

2. Crear una clase padre Figura de la que heredasen todas las figuras. Esta alternativa me gustaba más porque se podían hacer jerarquías de clases y simplificaba mucho el código a desarrollar ya que la mayoría de las figuras comparten atributos y métodos. Está ha sido la alternativa elegida finalmente.

Descripción

Es la clase padre de todas las clases propias hechas de figuras. Contiene los métodos, funciones y atributos que deben de poseer mínimamente todas las figuras, como puede ser el color del trazo, el grosor, el método pintar_figura(), y los métodos abstractos que deben de implementar todas las figuras.

Esto permite poder dibujar la figuras individualmente y editarlas como se desee, ya que tan solo hay que cambiar los atributos de las mismas.

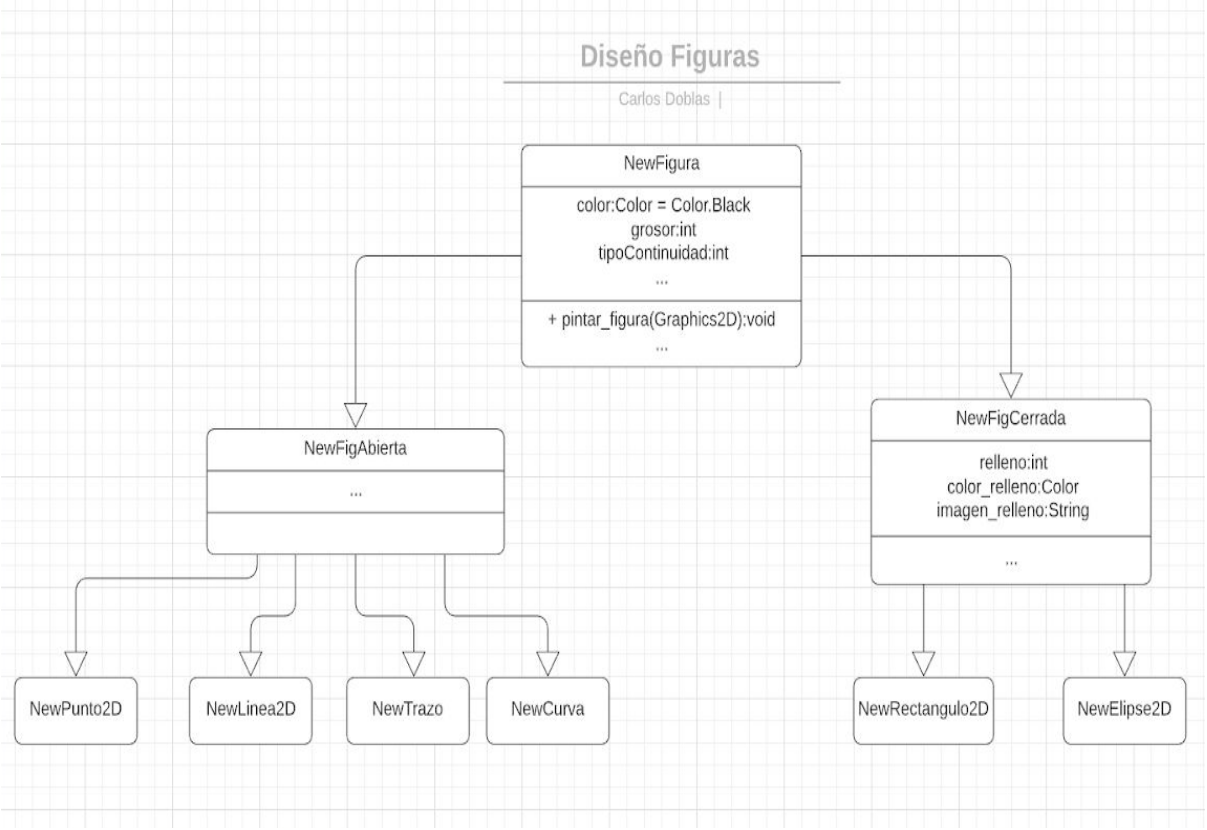
Diseño

Se trata de una clase abstracta que hereda de la clase Shape, por lo tanto debe de implementar los métodos abstractos de esta. Para que esto se adaptase a cualquier figura (línea,elipse...) contiene un atributo de la clase Shape que será el que se asignará en sus descendientes y llamará a los métodos desde esta variable.

Esta clase almacena los atributos básicos que comparten todas las figuras, como puede ser el color, el grosor, el trazo, etc. Los atributos más específicos se implementaran en los descendientes.

Para la jerarquía de clases se han implementado las clases NewFigAbierta y NewFigCerrada hijas de esta clase de las que heredarán todas las figuras.

Por lo tanto la jerarquía de clases quedará representada de la siguiente forma:



NewFigCerrada

Descripción y Diseño

La clase NewFigCerrada implementa todo lo necesario para la representación del relleno de las figuras, como es el tipo de relleno, su color o la imagen de relleno. Esta clase implementa todos los métodos necesarios para el manejo de estos atributos además del código correspondiente para el dibujado del mismo.

Lo interesante es que en el método pintar_figura() al que sobrecarga, utiliza el método de la clase padre para dibujar la figura y tan solo le añade el código correspondiente para la gestión del relleno.

```
@Override
public void pintar_figura(Graphics2D g2d){

    super.pintar_figura(g2d);

    Point2D pc1 = new Point2D.Double(shape.getBounds().getCenterX(),shape.getBounds().getCenterY());
    Point2D pc2 = new Point2D.Double(shape.getBounds().getCenterX(),shape.getBounds().getCenterY());

    GradientPaint rellenar;
    switch (relleno){
        case 1: //RELLENO LISO
            g2d.setColor(color_relleno);
            g2d.fill(shape);
        break;

        case 2: //RELLENO DEGRADADO HORIZONTAL

            pc1.setLocation( pc1.getX()  - ( shape.getBounds().width / 2 ), pc1.getY());
            pc2.setLocation( pc2.getX()  + ( shape.getBounds().width / 2 ), pc2.getY());

            rellenar = new GradientPaint(pc1, color, pc2, color_relleno);
            g2d.setPaint(rellenar);
            g2d.fill(shape);
        break;

        case 3: //RELLENO DEGRADADO VERTICAL
            pc1.setLocation( pc1.getX() , pc1.getY() + ( shape.getBounds().height / 2 ));
            pc2.setLocation( pc2.getX() , pc2.getY() - ( shape.getBounds().height / 2 )) ;
```

De esta manera se consigue dibujar el relleno de las figuras cerradas de una manera eficiente.

NewFigAbierta

Esta clase no implementa nada respecto a su clase padre, se utiliza por una cuestión de diseño a la hora de distinguir tipos de figuras y también para tener la posibilidad implementar métodos abstractos para las figuras abiertas en el caso de necesitarlos.

NewLienzo2D

Justificación

Para cumplir con los nuevos requisitos, se debía rediseñar la clase Lienzo, ya que antes todas las figuras tenían los mismos atributos y ahora debían de tener atributos propios y poder ser modificados. Para ello una vez ya sabía el diseño de las figuras, se debe de rediseñar esta clase ampliando su funcionalidad para la creación, modificación y edición de las figuras.

Descripción

Es la nueva clase lienzo, es similar a la de Lienzo2D inicial con la diferencia de que trabaja con figuras de la clase NewFigura y que se debe de poder aplicar los efectos de imagen en las figuras. Se mantienen los atributos que tenía el lienzo además de incluir algunos nuevos. Esta clase también permite mover y editar las figuras creadas anteriormente mediante la opción de editar. Para ello tan solo se debe de clicar en la figura que se desee editar y se identificará la figura seleccionada mediante un rectangulo boundingbox discontinuo. Una vez identificada se puede editar mediante las opciones de la ventana principal.

Para poder aplicar los efectos de imagen a las figuras dibujadas, se implementa el método void volcarFiguras() , cuya función es dibujar las figuras en la imagen del lienzo y vaciar el vector de figuras. Una vez llamas a este método las figuras no se podrán editar.

Diseño

Se trabaja con la variable NewFigura para la creación, modificación, dibujado y edición de figuras volcandoles los atributos del lienzo. Cada figura tiene sus propios atributos y se pinta individualmente llamando a un método propio.

Los principales cambios del diseño vienen dados en el método **createShape**, en donde dependiendo de la herramienta seleccionada se creará la correspondiente figura inicializando con los atributos activos que tiene el lienzo, y estos serán los atributos propios de la figura.

```
public void createShape(Point2D pl) {
    boolean crear = true;
    switch (herramienta) {

        case PUNTO:
            figura = new NewPunto2D(pl, color, grosor, alisado, transparencia);
        break;

        case TRAZO:
            figura = new NewTrazo(pl, color, grosor, alisado, transparencia,tipoContinuidad);
        break;
```

Y por último es añadida al vector de figuras.

Y en el método **Paint**, donde se irá llamando al método pintar_figura() de cada figura pasandole el graphics del lienzo como parámetro.

```
50      @Override
        public void paint(Graphics g) {
52          super.paint(g);
53          Graphics2D g2d = (Graphics2D) g;
54
55          if (imagen != null) {
56              g2d.drawImage(imagen, 0, 0, this);
57              g2d.clip(new Rectangle(0, 0, imagen.getWidth(), imagen.getHeight()));
58          }
59
            for (NewFigura s : vShape) { //Para cada figura del vector
71              s.pintar_figura(g2d); //Pinta cada figura
72          }
73      }
```

Para editar las figuras desde la ventana principal se ha implementado un manejador para actualizar constantemente la barra de herramientas dependiendo de la figura seleccionada. Y una vez seleccionado cada método Set del lienzo comprueba si se encuentra en modo edición para modificar la figura seleccionada en el caso de haberla.

Ejemplo del método **setTransparencia()** :

```
public void setTransparencia(double transparencia)  {
    this.transparencia = transparencia;
    if (herramienta == TipoHerramienta.EDICION && figura != null){
        figura.setTransparencia(this.transparencia);
        this.repaint();
    }
```

Para analizar la clase con más profundidad consultar Javadoc.

Figuras(NewLinea2D,NewRectangulo2D,NewElipse2D...)

Descripción y Diseño

Todas las formas implementadas heredan de las clases NewFigAbierta o NewFigCerrada como he comentado anteriormente, por lo tanto el diseño de las mismas es el mismo que el de las mismas con el distintivo de que implementan los métodos abstractos y sobrecargan algunos métodos necesarios para poder funcionar adecuadamente.

Al igual que la clase padre de todas ellas, comparten los atributos básicos de la clase NewFigura, y desarrollan los atributos de las figuras cerradas o algunos específicos que pueda tener alguna figura como es en el caso de NewCurva.

Lo más importante del diseño de las figuras se encuentra en el constructor, en donde el atributo shape de la clase Shape se inicializa al respectivo constructor de su figura (Line2D,Rectangle2D...) y será este con el se trabajará de una manera muy sencilla.

Las únicas figuras que voy a comentar individualmente son NewTrazo y NewCurva ya que son las que tienen un factor de diferenciación respecto de las demás.

NewTrazo

Esta figura suponía un reto debido a que no se encuentra implementada para poder operar con ella como el resto de figuras, por lo que suponía un trabajo adicional.

En primera instancia representé este trazo con una lista de puntos (NewPunto2D), que se inicializaban con los atributos del trazo y se dibujaban con un bucle for llamando a su propio método pinta_figura de la clase NewPunto2D. Esto en teoría debía de funcionar, pero a la hora de visualizar el trazo se podían visualizar los puntos y era como una estela en vez un trazo.

Para solventar esto hay que saber diferenciar entre lo que es la representación y como vas a dibujar la figura. Una vez tenido esto en cuenta cambié la representación del trazo siendo dado como una lista de puntos (Point2D).

Con esta lista se consigue almacenar el recorrido del trazo pero no tiene atributos ni se puede dibujar . Para dibujar el trazo se va dibujando cada segmento entre dos puntos de manera consecutiva representado con la clase NewLinea2D volcandole los atributos del trazo a estos segmentos y cumpliendo todos los requisitos. De esta manera se consigue plasmar la forma de manera fluida y se ahorra algo de memoria.

NewCurva

El principal inconveniente que tiene esta figura es su creación, y esto es debido a que se crea en dos pasos, a diferencia del resto de figuras que son en 1 paso. Para solventar este problema la figura cuenta con un atributo adicional que es el estado de la figura. Para crear una curva hay que diferenciar 3 estados para su creación.

1) Cuando se crea la figura, en la que el valor del estado será **0**.
2) Cuando la figura se encuentra creada y queremos definir el segmento de la longitud de los extremos de la curva, dado como una recta. Una vez que el usuario a definido esta recta el valor de la variable estado será **1**
3) Una vez definida la recta, asignar la curvatura de la misma dado el punto de control que será la posición del ratón. Una vez asignada el valor del estado pasará al valor **2,** que significa que la curva a sido terminada.

Una vez conociendo estos posibles estados, la forma de saber cuando un usuario quiere pasar de un estado a otro será al principio a través del evento del lienzo **formMousePressed** para crearla como todas las figuras y luego *del estado 2 al estado 3* y del *estado 3 a finalizada* ocurrirá con el evento **formMouseReleased.**

Para cambiar de estado se ha implementado el método **cambiarEstado()** que es llamado desde el manejador del evento **formMouseReleased.**

Imágenes

En esta parte voy a tratar las dos clases propias implementadas para poder llevar a cabo operador de LookupOp y la operación de diseño propio aplicada pixel a pixel.

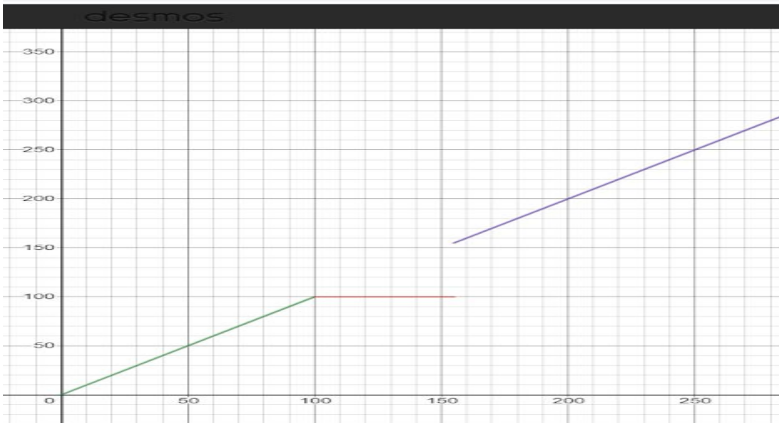
Requisitos

RF-1. Un operador "LookupOp" basado en una función definida por el estudiante; dicha función deberá de tener, al menos, un parámetro
RNF-2. Una nueva operación de diseño propio aplicada pixel a pixel; dicha función deberá de tener, al menos, un parámetro.

Operador LookupOp (NewLookUp)

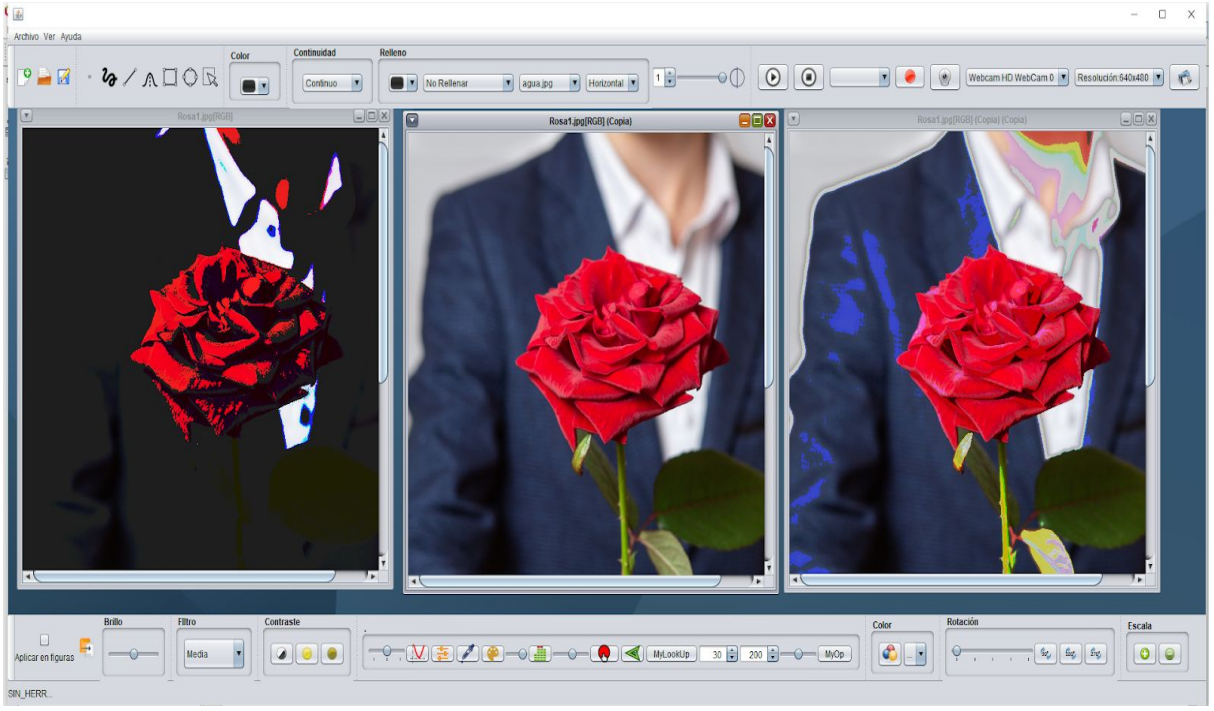
El operador LookupOp implementado se basa en una **función a trozos** con dos **parámetros, x** e **y.** Está función se encuentra formada por tres trozos cuyos valores son la función identidad y luego un trozo formado por una función constante que dependerá de los parámetros introducidos y por último la continuación de la función identidad. La función constante se encuentra en la mitad del eje de abscisas y su longitud dependerá del ***parámetro x***, este parámetro tendrá un valor entre **0 y 127**. Por lo tanto si este parámetro vale 0 no existirá la función constante y la función obtenida será la función identidad. Y por otro lado si vale 127, representará la función constante al completo. Por lo tanto lo interesante es ir jugando con este valor para ajustar el recorrido de la función a la medida que desee el usuario.
Por último, **el valor de esta función** constante vendrá dado por el ***parámetro y***, por lo tanto este parámetro deberá de tener un valor entre **0 y 255.**

Ejemplo de representación de la función :



En la figura anterior se puede observar la representación del operador descrito anteriormente.
Analizando la función, podemos deducir que el tramo de color verde y morado corresponde con la función identidad y el tramo en color rojo con la función constante.
En este caso, el **parámetro x** tendría un valor entorno a 30, y el **parámetro y** tendría un valor de 100 coincidiendo con el final del primer tramo, aunque esto no tiene por qué suceder, podría tener cualquier valor.
Tras probarlo, el efecto obtenido es el esperado.

Ejemplo de uso



En esta primera instantánea se puede ver la aplicación del operador con dos imágenes, de izquierda a derecha tenemos la imagen con el operador aplicado con:

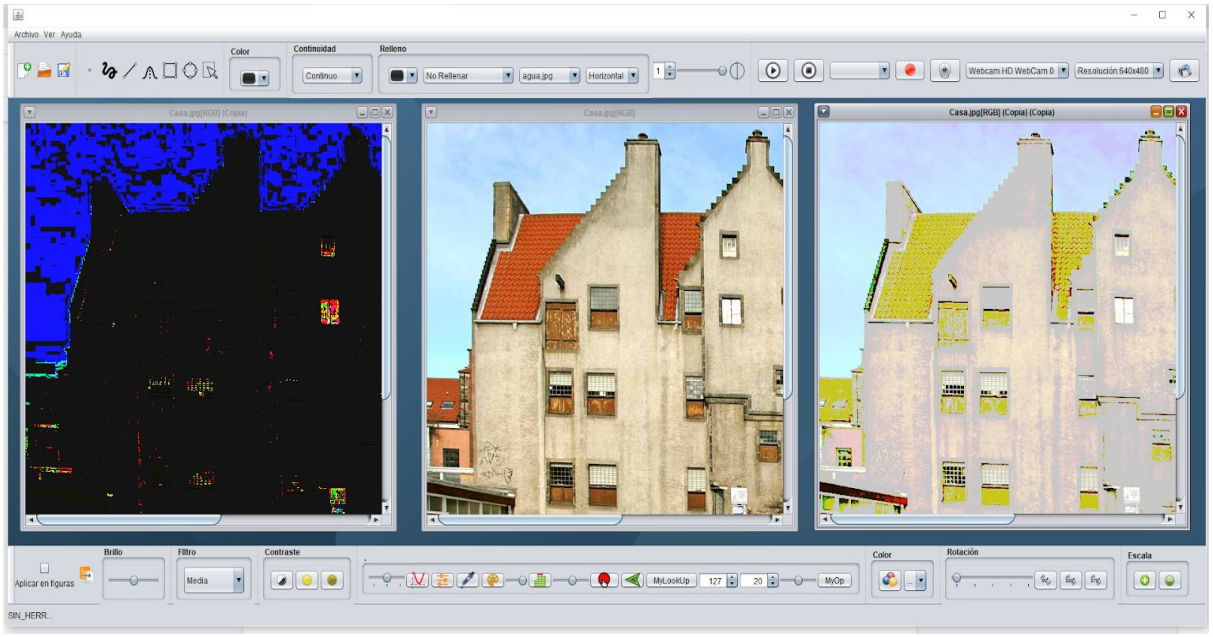
Param x = 100 - Param y = 30

Como se puede observar en la primera imagen, el operador es muy intrusivo ya que el **parámetro x** está a un **valor muy alto** afectando a la gran mayoría de pixeles y su efecto final es de oscurecer la imagen debido a que el valor del **parámetro y** es **muy bajo.** En este caso aplica un efecto similar al del operador de posterización cuando se disminuye el número de colores.

La segunda imagen sería la imagen original.
Y por último la tercera imagen sería el resultado de aplicar el operador con:

Param x = 30 - Param y = 200

En este caso ocurre lo contrario que en el primer caso, como el valor del **parámetro x** se encuentra cerca del mínimo, el operador será menos intrusivo. Y el contraste disminuye ya que el valor del **parámetro y** es alto.



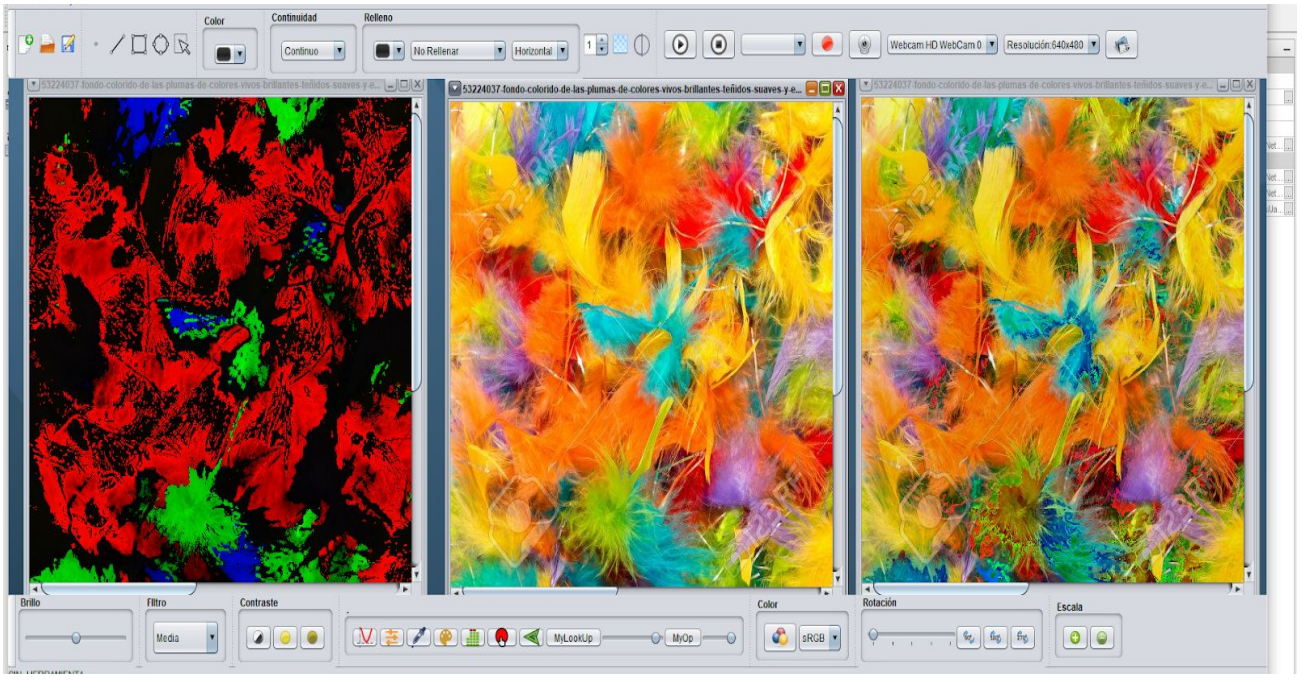
En esta segunda instantánea, se muestran otras posibles combinaciones de los parámetros de manera más intrusiva. Como se puede observar el resultado no es muy estético.
De izquierda a derecha el valor de los parámetros es el siguiente:

Imagen 1: **Param x = 80 - Param y = 200**
Imagen 2: **Por defecto**
Imagen 3: **Param x = 127 - Param y = 20**

Operador de diseño propio (NewOp)

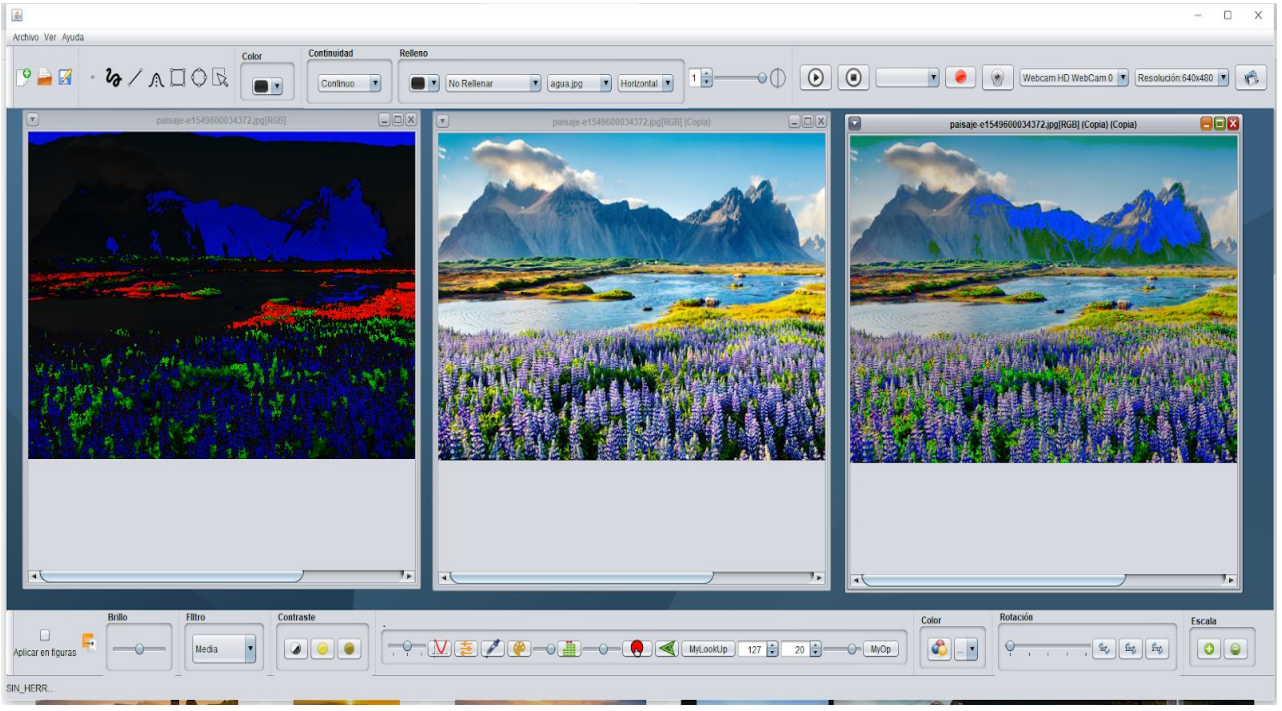
Tras reflexionar acerca de cómo iba a hacer el operador, llegue a la conclusión de que quería implementar un operador que premiase las bandas de los pixeles con mayor valor y que penalizará el resto. Y así poder ver las zonas de una imagen que más presencia tiene una banda respecto a las otras. La cantidad que se premia o se penaliza vendrá dada por un parámetro k que deberá tener un valor entre 0 y 1, cuánto más cerca se encuentre del valor 0, más premiará o penalizará las bandas siendo mayor el impacto del filtro en la imagen, y al contrario, si se encuentra próximo al valor 1 ocurrirá lo opuesto, premiará y penalizará en menor medida, siendo menor su impacto en la imagen.

Ejemplo de uso



Partiendo de izquierda a derecha, en primer lugar tenemos la imagen una vez aplicado el filtro con el *parámetro k* próximo a 0, como se puede ver se penaliza en gran medida las bandas con valores más bajos y premia las bandas con mayor valor. En este caso predomina la banda roja.
En segundo lugar nos encontramos ante la imagen sin aplicar el efecto.
Y por último lugar, se observa la imagen con el parámetro k muy próximo a 1, siendo casi despreciable la penalización o incremento de las bandas.

La función implementada para desarrollar este operador es una **función condicional** que comprueba si el valor de una banda es mayor que la suma de los otros dos, este es premiado y el resto es penalizado. Si no hay una banda que predomine se penaliza igualmente. Como se puede observar en la imagen, el efecto es el esperado.



Otro ejemplo del uso del operador, el resultado es el esperado.

Anexo:

La funcionalidad del relleno con imagen no funciona desde el .jar, desde el proyecto en Netbeans si funciona correctamente.