Documentación

Prácticas de Sistemas Multimedia

Antonio Jesús Peláez Priego

Índice

1. Requisitos

Los requisitos de la aplicación serán extraídos del documento de evaluación proporcionado por el profesor.

1.1. Requisitos generales

- La aplicación permitirá trabajar, de forma integrada con: gráficos, imágenes, sonido y vídeo.

- La aplicación tendrá un escritorio central en el que podrán alojarse ventanas internas de diferentes tipos.

- Las ventanas internas se irán creando en función de las acciones que lleve a cabo el usuario (abrir, grabar, capturar, etc.)

- La aplicación contará con una barra de herramientas de carácter general que incluirá, al menos, los botones nuevo, abrir y guardar (todos ellos tendrán que tener asociado un icono y un “ToolTipText”).

- El botón nuevo permitirá crear una nueva imagen (que aparecerá en una nueva ventana). El usuario deberá indicar el tamaño de la imagen (a través de un diálogo que se lance en el momento de la creación o asociado a un menú general de opciones).

- El botón abrir abrirá el diálogo "Abrir fichero" y permitirá: seleccionar un fichero de imagen, sonido o vídeo.
- Dependiendo del tipo de fichero abierto, éste se mostrará en un tipo de ventana u otra.
- Los formatos de ficheros reconocidos serán los estándares manejados por Java.
- Se usará el mismo botón para abrir cualquier medio, teniendo el diálogo abrir asociado filtros para los tipos de ficheros reconocidos.

- Si se produce algún error al abrir el medio (formato desconocido, etc.) se lanzará un diálogo que informe del problema.
- El diálogo abrir solo mostrará extensiones correspondientes a ficheros de formatos admitidos.

- El botón guardar lanzará el diálogo "Guardar fichero" y permitirá guardar la imagen de la ventana que esté seleccionada, incluyendo las figuras dibujadas.
- La opción de guardar estará desactivada para el caso de sonidos y vídeos.
- Se podrá guardar en cualquiera de los formatos reconocidos por Java (JPG, PNG, etc.), obteniendo dicho formato a partir de la extensión indicada por el usuario.
- El diálogo guardar solo mostrará las extensiones correspondientes a formatos admitidos.

- En la barra de menú aparecerá el menú Archivo que incluiría las opciones “Nuevo”, “Abrir” y “Guardar” indicadas anteriormente.

- La barra de menú incluirá la opción Ver que permitirá ocultar/visualizar las diferentes barra de herramientas.

- La barra de menú incluirá la opción Ayuda que tendrá la opción “Acerca de” que lanzará un diálogo con el nombre del programa, versión y autor.

1.2. Requisitos gráficos

- La aplicación permitirá la gestión conjunta de gráficos e imágenes en un mismo tipo de ventana.
- Dada una imagen (ya sea creada nueva, leída o capturada), podremos tanto dibujar sobre ella como procesarla.

- Se requerirá la incorporación de un tipo específico de ventana interna capaz de mostrar imágenes tanto las leídas de un fichero como las creadas nuevas por el usuario o generadas al capturar una imagen instantánea a partir de un vídeo o webcam.

- Cada ventana mostrará una única imagen (existiendo, por tanto, tantas ventanas como imágenes estemos tratando).
- Se incluirá barras de desplazamiento en caso de que la imagen sea mayor que la zona visible.

- El título de la ventana corresponderá al nombre del fichero, si es una imagen abierta o guardada, “nueva”, si ha sido creada por el usuario, o “captura”, si es una instantánea captada de un vídeo o de la webcam.
- El título de la ventana deberá indicar (entre corchetes) el espacio de color en el que está la imagen.
- El título de la ventana deberá indicar si se trata de una imagen asociada a una banda, indicando a qué banda (número o letra) corresponde.

- Al mover el ratón sobre la imagen, se indicará en la barra de estado las coordenadas del pixel sobre el que se está situado.

- Sólo se podrá dibujar sobre el área de la imagen, por lo que estas ventanas tendrán definidas como área de visualización el área rectangular correspondiente a la imagen.

- En el procesamiento de imágenes se aplicará el efecto solo sobre la imágen no sobre las figuras dibujadas.

1.3. Requisitos de dibujo

- El usuario podrá dibujar sobre cualquier imagen utilizando la forma y atributos seleccionados.

- El lienzo mantendrá todas las figuras que se vayan dibujando.

- Cada figura tendrá sus propios atributos, independientes del resto de formas.
- Cuando se dibuje la forma por primera vez, la figura dibujada usará los atributos que estén activos en ese momento.

- El usuario podrá editar las figuras ya dibujadas.
- El usuario podrá seleccionar cualquier figura haciendo clic sobre ella, para lo cual tendrá que estar seleccionado el botón de “edición” en la barra de herramientas.
- La figura seleccionada deberá identificarse mediante un rectángulo (boundingbox) discontinuo tipo selección.
- Para la figura seleccionada, se podrán editar sus atributos.
- El usuario podrá mover la figura seleccionada.

- El usuario podrá cambiar la ordenación de las figuras.
- La aplicación incluirá la opción “enviar al fondo”, que situará la figura seleccionada detrás del resto de figuras.
- La aplicación incluirá la opción “enviar atrás”, que llevará la figura seleccionada una posición hacia atrás respecto a su ordenación actual, de forma que quede oculta por más figuras.
- La aplicación incluirá la opción “traer adelante”, que traerá la figura seleccionada una posición hacia delante, de forma que quede oculta por menos figuras.
- La aplicación incluirá la opción “traer al frente”, que traerá la figura seleccionada delante del resto de figuras, de forma que ninguna parte quede oculta detrás de otra figura.

- La aplicación permitirá dibujar, al menos, las siguientes formas geométricas: línea recta, rectángulo, elipse, curva, trazo libre, una forma personalizada que defina una nueva figura.

- En la barra superior aparecerá un botón con icono por cada forma de dibujo disponible y se usarán botones de dos posiciones agrupados, de forma que siempre aparezca pulsada la forma seleccionada.

- En la barra superior se incluirá un botón “selección” que, si está pulsado, indicará que estamos en modo edición.

- El usuario podrá elegir los atributos con los que se pintarán las formas.

- El usuario podrá elegir el color del trazo y el de relleno.

- El usuario podrá elegir el grosor y el tipo de discontinuidad del trazo.

- El usuario podrá elegir entre tres opciones a la hora de rellenar: no rellenar, rellenar con un color liso o rellenar con un degradado.

- En el caso del relleno con degradado, éste se aplicará utilizando los dos colores (frente y fondo) seleccionados en ese momento.

- El usuario podrá activar/desactivar la mejora en el proceso de renderizado correspondiente al alisado de bordes.

- Se podrá establecer un grado de transparencia asociado a la forma.

- Cuando se cambie de una ventana interna a otra, los botones de forma y atributos de la barra de herramientas deberán activarse conforme a la forma y atributos de la ventana activa.

1.4. Requisitos de procesamiento de imágenes

- La aplicación permitirá aplicar un conjunto de operaciones que se podrán llevar a cabo sobre cualquier imagen.

- Cada una de esta operaciones se incluirán en una barra de herramientas donde cada elemento tendrán que tener asociado un “ToolTipText”.

- La aplicación deberá incluir la operación duplicar, que creará una nueva “ventana imagen” con una copia de la imagen.

- La aplicación deberá incluir la operación modificar el brillo mediante un deslizador.

- La aplicación deberá incluir filtros de emborronamiento, enfoque y relieve.

- La aplicación deberá incluir las operaciones contraste normal, iluminado y oscurecido.

- La aplicación deberá incluir la operación negativo.

- La aplicación deberá incluir la operación extracción de bandas.

- La aplicación deberá incluir la operación de conversión a los espacios RGB, YCC y GRAY.

- La aplicación deberá incluir la operación de giro libre mediante deslizador.

- La aplicación deberá incluir la operación escalado (aumentar y disminuir).

- La aplicación deberá incluir la operación sepia.

- La aplicación deberá incluir un operador “LookupOp” basado en una función definida por el estudiante.

- La aplicación deberá incluir una nueva operación de diseño propio aplicada componente a componente.

- La aplicación deberá incluir la operación una nueva operación de diseño propio aplicada pixel a pixel.

- Las operaciones se irán aplicando de forma concatenada, es decir, una operación se aplicará sobre el resultado de operaciones aplicadas anteriormente.

- En el caso del brillo, el deslizador permitirá ir variando el brillo sobre la imagen que haya en ese momento, no sobre el resultado del cambio de brillo. Una vez que se elija otra operación, el brillo se aplicará de forma definitiva.

1.5. Requisitos de sonido

- La aplicación permitirá tanto la reproducción como la grabación de audio.

- La aplicación contará con una lista de reproducción, que tendrá una lista desplegable asociada, de forma que al abrir un nuevo fichero de audio éste se incluiría en dicha lista de reproducción.

- La aplicación incluirá un botón para reproducir sonido. Al pulsar el botón, se reproducirá el audio que esté seleccionado en ese momento en la lista de reproducción.

- La aplicación incluirá un botón para parar la reproducción o la grabación.

- La aplicación incluirá botón para grabar audio, de forma que al pulsarlo se iniciará el proceso de grabación, que terminará cuando se pulse el botón de parada. El sonido se grabará en el fichero indicado por el usuario y se añadirá a la lista de reproducción.

- Tanto la lista de reproducción, como los botones de reproducir, pausar y grabar, estarán dentro de una barra de herramientas.

1.6. Requisitos de vídeo

- La aplicación permitirá tanto la reproducción como la captura de vídeo; para ello, se definirán ventanas internas específicas para cada tarea.

- La reproducción de vídeo requerirá la incorporación de un tipo específico de ventana que dispondrá de una zona de visualización central donde se mostrará el vídeo.

- La aplicación incluirá un conjunto de botones para controlar la reproducción (reproducir, pausar, etc.) Dichos botones se situarán en un panel en la parte inferior de la ventana, de forma que cada ventana tendrá sus propios controles.

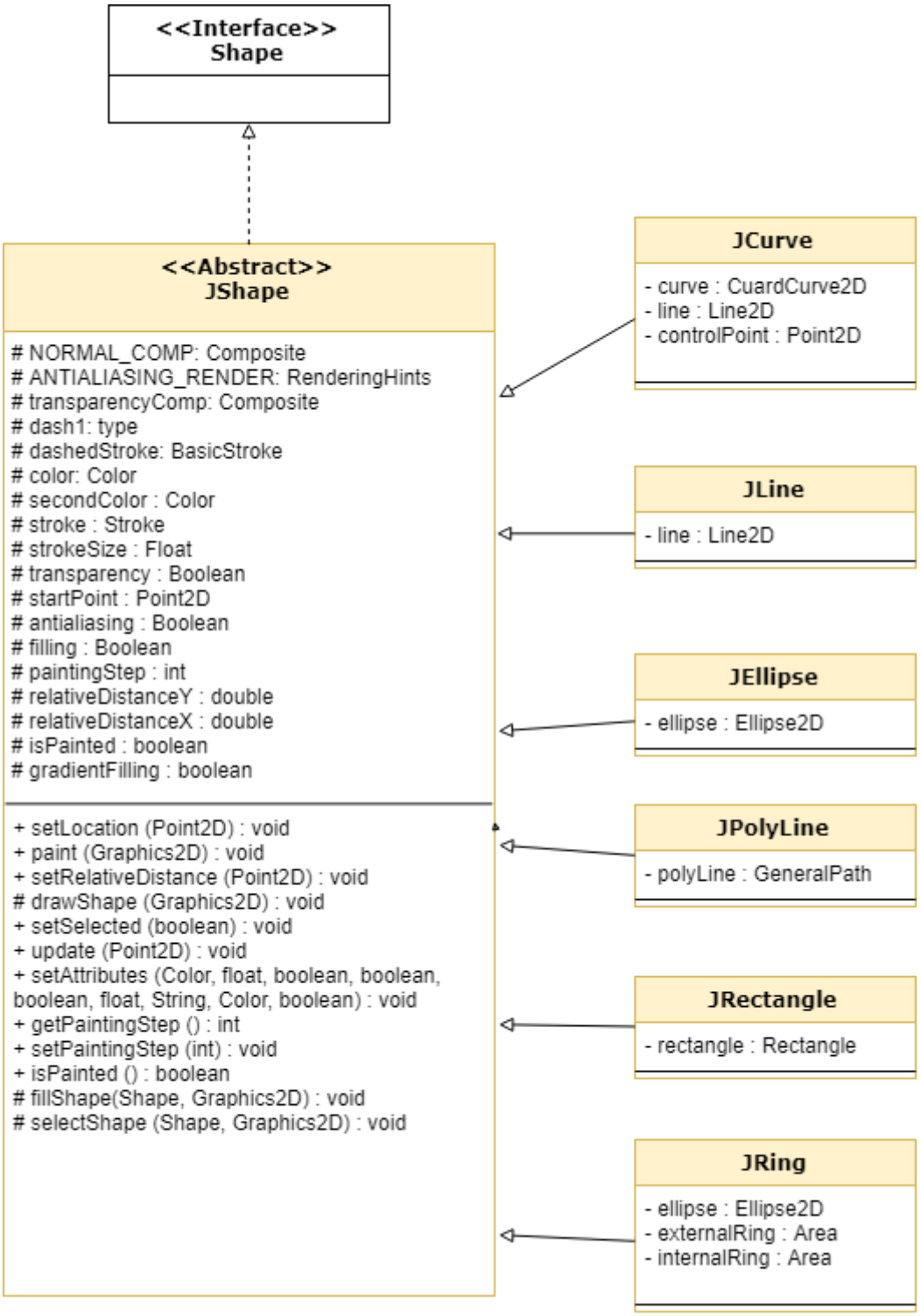
- La ventana tendrá como título el nombre del fichero que se está reproduciendo, existiendo tantas ventanas como vídeos tengamos abiertos.

- La aplicación permitirá al usuario capturar imágenes de la cámara o del vídeo que se esté reproduciendo; concretamente, lo hará de la ventana que esté activa, siempre y cuando sea una ventana de tipo “reproducción de video” o “webcam”. La imagen capturada se mostrará en una nueva ventana interna.

2. Análisis y diseño

2.1. Solución jerarquía de clases

Diagrama de clases planteado para la jerarquía de clases requerida en la sección de gráficos.



Como vemos en el diagrama la solución por la que he optado para la implementación de la jerarquía de clases es la siguiente:

JShape : Clase abstracta que implementa la interfaz Shape.

Además JShape contiene los atributos que necesitan nuestras nuevas figuras, tales como color, trazo, transparencia…

La clase JShape también contiene los métodos que utilizarán nuestras nuevas figuras, siendo algunos de estos comunes para todas y otros están implementados como abstractos, por lo que cada figura tendrá su propia implementación de este.

Los métodos abstractos son:

- update : que es usado al pintar la figura, recibe como parámetro un punto que será el siguiente punto con el que se pintará la figura.
- setLocation : método usado para mover la figura
- setRelativeDistance : método usado para calcular la distancia entre la esquina superior izquierda y el punto donde se hace click.
- drawShape: método auxiliar al paint, que se usará para dibujar cada figura.

Los demás métodos tendrán la misma implementación para cada figura.

JCurve : extiende de JShape, además de implementar los métodos abstractos según corresponde tiene 3 atributos propios:

- curve : CuadCurve2D que se usará para pintar la figura curva
- line : Line2D que se usará para pintar la primera fase de la curva
- controlPoint : Point2D que ayudará a fijar el punto de control de la curva

JLine : extiende de JShape, además de implementar los métodos abstractos según corresponde tiene 1 atributo propio:

- line : Line2D que se usará para pintar la figura línea

JEllipse : extiende de JShape, además de implementar los métodos abstractos según corresponde tiene 1 atributo propio:

- ellipse: Ellipse2D que se usará para pintar la figura eclipse

JPolyLine : extiende de JShape, además de implementar los métodos abstractos según corresponde tiene 1 atributo propio:

- polyLine : GeneralPath que se usará para pintar el trazo libre.

JRectangle : extiende de JShape, además de implementar los métodos abstractos según corresponde tiene 1 atributo propio:

- rectangle : Rectangle que se usará para pintar la figura rectángulo.

JRing: extiende de JShape, además de implementar los métodos abstractos según corresponde tiene 3 atributos propios:

- ellipse : ellipse2D que se usará para pintar la figura anillo
- internalRing: Area que se restará del externalRing para formar el anillo
- externalRing: Area total representada por el eclipse dibujado a la que se le restará internalRing para formar el anillo.

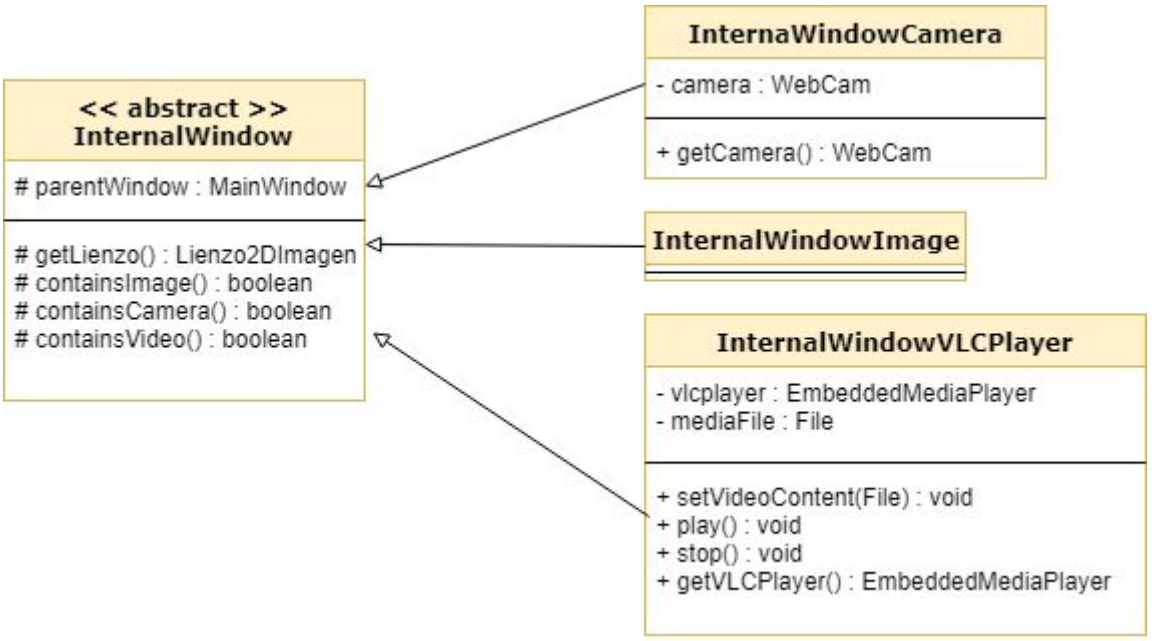
Con esta solución conseguimos que todas las figuras tengan sus propios atributos independientemente de las demás.

Además proporciona una capa de abstracción a la hora de pintar figuras, ya que desde el lienzo nosotros solo tenemos que llamar al constructor para crear la figura que se desea y luego siempre llamaremos al método update para terminar de pintar la figura independientemente de la figura que se haya creado previamente.

También cumplimos el requisito del método paint del lienzo, que solo tendrá que recorrer el vector de figuras y llamar al método paint de cada una, y este se ocupará de pintar la figura según corresponda.

2.2. Solución ventanas internas

Se nos pedía que existieran varios tipos de ventana interna en función del contenido de esta. Para esto hemos implementado la siguiente solución:



Gracias a los métodos containsImage, containsCamera y containsVideo podemos identificar el tipo de ventana y en función del tipo el comportamiento del programa será diferente:

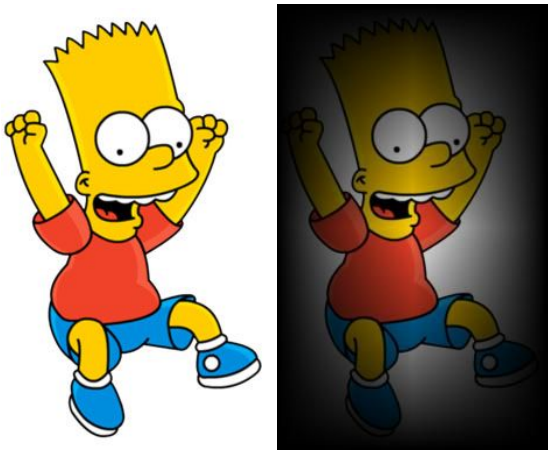
- Se desactivarán los botones de guardar, operaciones de imágenes y dibujo cuando el tipo de ventana no contenga imagen, es decir containsImagen devolverá false.
- Se desactivará el boton de screenshot cuando el tipo de ventana contenga imagen.

En cuanto al método getLienzo() es un método abstracto que solo está habilitado en la clase InternalWindowImage, las demás clases lanzarán una excepción si se intenta llamar a este método pero gracias a esto no tenemos que hacer casting a InternalWindowImage cada vez que queramos llamar al método getLienzo().

2.3. Operación componente a componente

La operación componente a componente que se ha implementado consiste en la operación “viñeta”, es decir, aplicar un efecto de sombreado a los bordes dejando el centro intacto.

Se conseguiría un efecto así:



Este efecto lo he conseguido de la siguiente forma:
- Se calcula el centro de la imagen.
- Se calcula la distancia entre el componente y el centro de la imagen.
- En función de esa distancia se calcula un factor por el que se multiplicará el componente.

La fórmula para calcular el factor es la siguiente:

f = (1-(distanciaX/centroImagenX) * (1-(distanciaY/centroImagenY))

Por lo que un componente que este justo en el centro tendrá una f = 1 y cuanto más alejado este el componente más pequeña será el factor ya que esa (distancia/centro) está restando. (1-(distancia/centro))

Por lo que el componente se irá multiplicando desde 1 (cuando está más cerca del centro) hasta 0 (más alejado del centro).

Al multiplicar los componentes por 0 conseguimos el color negro, y el efecto de sombreado/degradado se consigue gracias a que el factor va disminuyendo poco a poco en función de la distancia hasta el centro.

Para esta operación no es necesario ningún parámetro adicional ya que siempre tendrá el mismo efecto.

2.4. Operación pixel a pixel

Para la operación pixel a pixel se ha implementado una operación que busca el valor más alto de los 3 componentes RGB y aumenta ese valor.

El efecto es el siguiente:

Como apreciamos se consigue una especie de aumento de color, en particular del tono de color predominante.

Este efecto se ha conseguido de la siguiente forma:

- Para cada pixel se busca cual de los componentes RGB tiene más valor.
- Una vez encontrado se multiplica por 1.25 dicho componente, fijando un máximo en 255, por lo que el efecto solo se podrá aplicar X veces en una imagen determinada llegando a un punto en que el efecto será nulo.

Para esta operación no es necesario ningún parámetro de entrada o configuración adicional.

2.5. LookUpOp basado en función propia

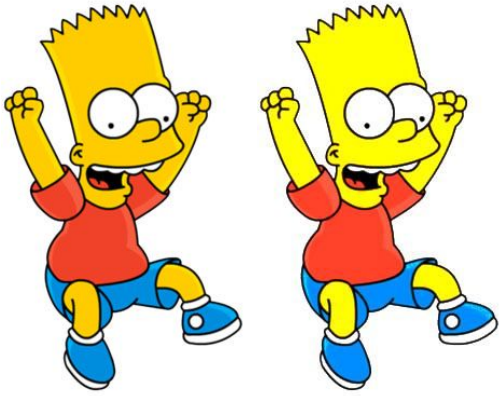
Esta función intensifica el valor de los componentes que se encuentren en el rango especificado.

Para esto usamos un slider desde 40 a 215.
Con ese slider obtenemos un valor p.

Los componentes que estén entre el valor p-40 y p+40 serán los componentes que se intensificarán.

Dichos componentes multiplicarán su valor por 1.25

En este ejemplo p tiene un valor de 215, por lo que la función se aplicará sobre los componentes que tengan valor comprendido entre 175 y 255:



Aunque el efecto depende totalmente del valor que le asignemos en el slider.

La función actúa igual que la función brillo, es decir, veremos los cambios mientras estemos moviendo el slider, pero una vez hagamos otra función los cambios quedan fijos.

3. Codificación

En esta sección explicaré cómo he resuelto a nivel de código algunos de los problemas planteados en la práctica.

- **Figure.java**

Contiene un enum con los distintos tipos de figura, este enum lo usaré a la hora de crear figuras para ver cual de ellas está seleccionada:

```
if (figureToPaint == Figure.LINE) shapeToPaint = new JLine(startPoint);
```

El valor de figureToPaint se fijará desde MainWindow al cambiar al figura que se quiere pintar.

- **Mover figuras**

Esta es una tarea que me costó bastante resolver, al final lo he conseguido de la siguiente forma:

En la clase Lienzo2D el método getSelectedShape al que se le pasa un punto, si encuentra una figura en ese punto, además de seleccionarla fijará su distancia relativa a este punto, esto lo hace llamando al método setRelativeDistance(p) de la clase JShape, y enviándole dicho punto.

El método setRelativeDistance es un método abstracto que se implementará de forma diferente en cada figura.
Su función es dar valor a las variables relativeDistanceY y relativeDistanceX, el valor que les asigna es la distancia entre la esquina superior izquierda de la figura y el punto que recibe como parámetro, que era el punto donde se hizo click.

Estas variables me ayudarán para calcular puntos y distancias en el método setLocation de cada figura, que también es un método que implementará cada figura de forma independiente.

- **Seleccionar figuras**

En el método getSelectedShape de la clase Lienzo2D lo primero que hago es llamar al método unSelectShapes() cuya función es deseleccionar todas las figuras, es decir, que si hacemos click en un espacio blanco donde no haya figuras, deseleccionaremos la figura que teníamos seleccionada previamente si había una.

Luego si encuentra una figura en el punto donde se hizo click, llama al método setSelected() de la clase JShape enviando como parámetro true.
Este método es igual para todas las figuras, lo único que hace es poner un booleano isSelected a true o false.

Este booleano se usará a la hora de pintar, si tiene un valor verdadero se llamará al método selectShape de la clase JShape, que recibe un objeto Shape y otro Graphics2D como parámetros.

Este método se ocupará de obtener las fronteras de la figura, y en función del tamaño de trazo de la figura (para que la línea de selección no quede oculta por el trazo) pintará la caja de selección en línea discontinua.

- **Rellenar figuras**

Cada figura contiene un booleano “filling” que en el caso de ser verdadero llamará al método fillShape() enviando un objeto Shape y otro Graphics2D como parámetro, este método se ocupará de rellenar la figura con degradado si está activado el booleano de degradado, o normal si no lo esta.

- **Dibujar figuras**

El dibujado de figuras lo gestionamos desde la clase Lienzo2D al recibir eventos de ratón, al recibir un mousePressed revisamos:

- Si se está editando figura, entonces llamamos a getSelectedShape()
- Si la figura aún se está pintando, es decir, es una figura que se pinta en varios pasos, como la curva, entonces llamamos a updateShape()
- Sino, llamamos a createShape()

createShape() se encarga de crear la figura en función de la que haya seleccionada, darle atributos a esta, en función de los atributos que estén fijados, y añadir la figura al vector de figuras.

updateShape() simplemente llama al método update() de la clase JShape enviando un punto como parámetro.

update() este método es abstracto y se implementará de forma diferente en cada figura, en función de la forma en que se tenga que dibujar.

- **Cambiar ordenación figuras**

Para mover una figura hacia adelante o hacia atrás uso el método moveShapePosition que recibirá como parámetro 1, o -1 en función de en qué sentido queramos mover al figura.

Se calcula la posición actual de la figura y se hace un swap usando el método Collections.swap de esa posición con la nueva posición deseada que será la misma +-1.

Para mover una figura a la última posición simplemente la remuevo del vector con un remove y luego la agrego en al primera posición con un add(0, shape)

Para mover la figura a la primera posición simplemente remuevo la figura del vector con un remove y la vuelvo a agregar con un add que por defecto la incluye en la última posición.

Estos dos últimos puntos parecen que estan al reves, pero cuando pintamos se pintan primero las primeras figuras y luego las últimas haciendo que la última figura se superponga sobre las anteriores, es decir, que si queremos mover una figura a la primera posición para que no tenga otra figura encima suya deberíamos poner la figura en la última posición del vector.

- **Mostrar coordenadas del ratón**

Desde la clase InternalWindowImage que representa una ventana interna que contiene una imagen, cuando recibe un evento de ratón mouseMoved llamamos al método setMouseCoords() de la clase MainWindow, esto podemos hacerlo gracias a que cuando construimos la clase InternalWindow desde MainWindow le enviamos ella misma como parámetro para el constructor.

El método setMouseCoords() simplemente cambia el valor del labe situado en la esquina inferior derecha, mostrando así las coordenadas del ratón dentro de la ventana interna.

- **ComboBox colores**

Esta fué también una tarea algo compleja que conseguí resolver gracias a diferentes fuentes por stackoverflow.

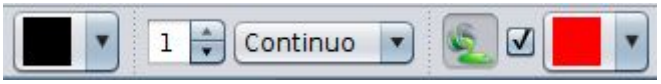
Para conseguir un comboBox que muestre varios colores lo hice de la siguiente forma:
“colors” vector de colores que contiene los que necesito.
“imagenesColores” vector de Object[] que contiene ImageIcons.

Este vector imagenesColores se rellena al crear la ventana principal llamando al método createColorImage y enviando por parámetro un color, así por cada color que tengamos en “colors”.

El método createColorImage crea un ImageIcon a partir de un color, para esto crea una imagen de dimensiones 24x24 y la rellena con un rectángulo del color recibido como parámetro.

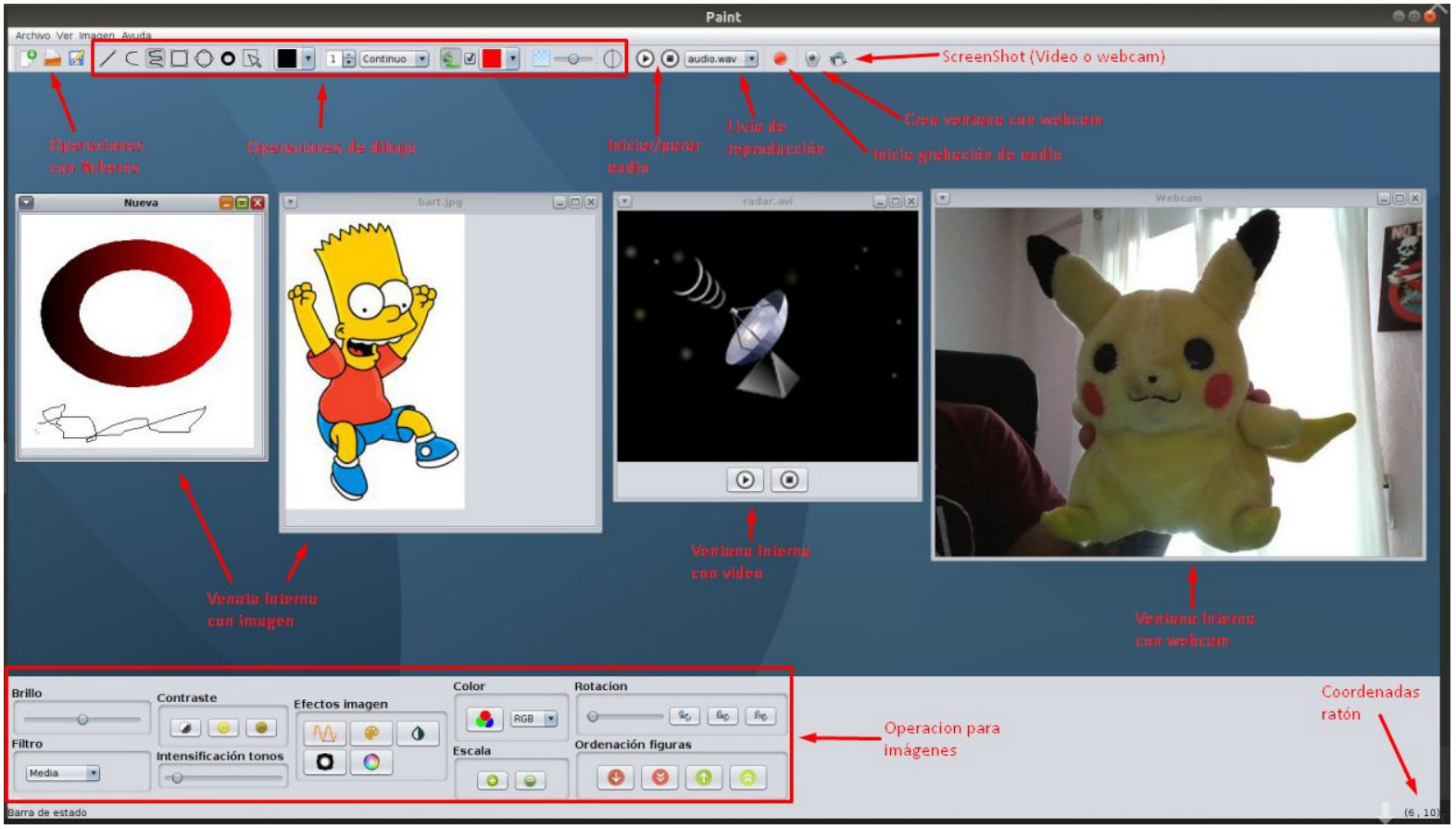
En función del index del comboBox podemos saber el color para enviarlo al lienzo ya que los colores están ordenados en el vector.

- **Relleno con degradado**



Para el degradado simplemente he introducido una checkbox y un segundo comboBox de colores.
En el caso de que el botón de relleno y la checkbox estén activos, se rellenará la figura con un degradado que empezará por negro y terminará por rojo en este ejemplo.

4. Manual


Paint
Archivo Ver Imagen Ayuda
Continuo
audio.wav
ScreenShot (Video o webcam)
Operaciones con ficheros
Operaciones de dibujo
Iniciar/parar audio
Lista de reproducción
Crea ventana con webcam
Inicia grabación de audio
Nueva
bart.jpg
radar.avi
Webcam
Ventana interna con imagen
Ventana interna con video
Ventana interna con webcam
Brillo
Contraste
Efectos imagen
Color
Rotacion
Filtro
Media
Intensificación tonos
Escala
RGB
Ordenación figuras
Operacion para imágenes
Coordenadas ratón
Barra de estado
(6, 10)

5. Bibliografía

Creo que entre un 70 y un 80% del contenido se ha sacado de las transparencias de teoría y prácticas de la asignatura.

El resto de bibliografía usada:

https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html

https://stackoverflow.com/questions/3514158/how-do-you-clone-a-bufferedimage

https://stackoverflow.com/questions/20186681/how-to-move-specific-item-in-array-list-to-the-first-item/20186749

https://stackoverflow.com/questions/5321463/how-to-move-the-selected-item-to-move-to-the-top-of-the-list

https://stackoverflow.com/questions/10839131/implements-vs-extends-when-to-use-whats-the-difference

https://stackoverflow.com/questions/8065571/change-state-of-toggle-button-from-another-button

https://stackoverflow.com/questions/47137636/swing-new-imageicon-from-color

https://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/index.html