Documentación SMM

Matilde Cabrera González.

1.Estudio del dominio del problema y descripción general del sistema:

1.1 Obtener información sobre el dominio del problema.

Se pretende realizar una aplicación multimedia que permita gestionar diferentes tipos de medios de forma integral. Para ello se desarrollará un entorno basado en escritorio en el que se distinguirán gráficos, imágenes, sonido y vídeo. La aplicación contará con una barra de herramientas de carácter general que incluirá la gestión global en conjunto de todos los medios (nuevo, abrir, guardar), también proporcionará un conjunto de menús y barras de herramientas que permitirán llevar a cabo todas las tareas propias de cada medio.

1.2 Identificar y revisar los objetivos del sistema.

Llevar a cabo la apertura de gráficos, imágenes, sonido y videos. La reproducción de videos y audios, grabar un audio, encender la webcam, capturar imágenes tanto de la webcam como de un video en reproducción, llevar a cabo un conjunto de operaciones sobre cualquier imagen y por último la aplicación permitirá dibujar diferentes formas geométricas (dibujos) con diferentes atributos sobre una imagen o un lienzo en blanco. El sistema tiene que poder dar sus propios atributos a cada figura. Las imágenes y lienzos se podrán guardar con los cambios realizados.

2. Obtención de una lista estructura de requisitos:

Identificar y revisar los requisitos funcionales (RF):

RF-1 Requisitos de carácter general

RF-1.1 Botón nuevo
Permite crear un nuevo lienzo, el usuario deberá indicar el tamaño de la imagen.

RF-1.2 Botón Abrir
Abre un dialogo que permite seleccionar un fichero de imagen, sonido o video. los formatos reconocidos serán los estándares manejados por Java.

RF-1.3 Botón Guardar
Lanza el dialogo "Guardar fichero" y permite guardar la imagen de la ventana que esté seleccionada, incluyendo las figuras. Se podrá almacenar en cualquiera de los formatos reconocidos por Java, obteniendo dicho formato a partir de la extensión indicada por el usuario.

RF-1.4 Botón Ver
Permite ocultar/visualizar las diferentes barras de herramientas.

RF-1.5 Botón Ayuda
Lanzará un diálogo con el nombre del programa, versión y autor.

RF-1.6 Ventana activa de lienzo
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.

RF-2 Requisitos de gráficos (figura)

Se permitirá dibujar diferentes formas geométricas, cada una de ellas con diferentes atributos, sobre una imagen o lienzo.

RF-2.1 Dibujar figuras
Se podrán dibujar las siguientes figuras: línea recta, rectángulo, elipse, curva con un punto de control, trazo libre, una forma personalizada (área) que defina una nueva figura. El lienzo mantendrá todas las figuras que se vayan dibujando. Cuando se dibuje la forma por primera vez, ésta usará los atributos que estén activos en ese momento.

RF-2.2 Seleccionar figuras

Se podrá seleccionar cualquier figura haciendo clic sobre ella. La figura seleccionada deberá identificarse mediante un rectángulo discontinuo.

RF-2.3 Editar figuras

El usuario podrá editar las figuras ya dibujadas. Se podrán editar sus atributos.

RF-2.4 Mover figuras

El usuario podrá mover la figura seleccionada.

RF-2.5 Cambiar ordenación de las figuras

El usuario podrá cambiar la ordenación de las figuras: “enviar al fondo”, “traer al frente”, “enviar atrás”, “traer adelante”.

RF-2.6 Atributo Color

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

RF-2.7 Atributo Trazo

Se podrán modificar el grosor y el tipo de discontinuidad del trazo.

RF-2.8 Atributo Relleno

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 degradado, para la dirección del degradado, se ofrecerá al menos dos posibilidades: horizontal, vertical y opcional inclinado.

RF-2.9 Atributo Alisado de bordes

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

RF-2.10 Atributo Transparencia

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

RF-3 Requisitos de imágenes

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

RF-3.1 Operación duplicar

Creará una nueva “ventana imagen” con una copia de la imagen seleccionada.

RF-3.2 Modificar el brillo

Modificar el brillo mediante un deslizador, el cual permitirá ir variando el brillo sobre la imagen que haya en ese momento, no sobre el resultado del cambio de brillo (es decir, si deslizamos el brillo a su máximo valor –lo que implicaría ver la imagen en blanco-, si después reducimos dicho valor se tiene que volver a ver la imagen inicial). Una vez que se elija otra operación, el brillo se aplicará de forma definitiva (y se concatenará con el resto de las operaciones).

RF-3.3 Filtros

Se modificará la imagen con filtros de emborronamiento, enfoque y relieve.

RF-3.4 Operación contraste

Se realizarán las siguientes operaciones de contraste de la imagen: contraste normal, iluminado y oscurecido.

RF-3.5 Operación negativo

El operador negativo, invertirá los colores de la imagen, de forma que si se aplica dos veces, la imagen volverá al aspecto inicial.

RF-3.6 Extracción de bandas

Incluiremos la opción de mostrar las bandas asociadas a la imagen seleccionada (por ejemplo, para una imagen en RGB, mostraría por separado las bandas correspondientes al rojo, verde y azul).

RF-3.7 Conversión a los espacios RGB, YCC y GRAY

Añadiremos la posibilidad de cambiar de espacio de color. Para ello, incluiremos en nuestra ventana principal una lista desplegable que muestre los distintos espacios de color disponibles (RGB, YCC y GREY).

RF-3.8 Operación Giro

Rotaremos la imagen ofreciendo dos posibilidades:

Mediante deslizador.

Rotaciones predeterminadas de 90º, 180º y 270º.

RF-3.9 Operación Escalado (aumentar y disminuir).

Se incluirán las acciones de aumentar el tamaño de la imagen y la de reducirla.

RF-3.10 Operación Sepia.

Se modifica el tono y saturación para darle un aspecto de "fotografía antigua".

RF-3.11 operador "*LookupOp*"

Un operador "*LookupOp*" basado en una función definida por el estudiante. Se crearán dos operadores propios llamados binarización y posterización.

RF-3.12 Operación de diseño propio componente a componente

Una nueva operación de diseño propio aplicada componente a componente.

RF-3.13 Operación de diseño propio pixel a pixel.

Una nueva operación de diseño propio aplicado pixel a pixel. deberá indicarse claramente en la documentación qué función se aplica, mostrar una representación gráfica de la misma y se explicará qué comportamiento se espera al aplicar esa función.

RF-3.14 Operador Tintado

Tintado rojo de la imagen actual.

RF-3.15 Operador Ecualización

Incorporaremos la posibilidad de ecualizar la imagen seleccionada.

RF-3.16 Operador Umbralización

Umbralización basada en niveles de gris con deslizador para modificar umbral.

RF-4 Requisitos de sonido

Se permitirá gestionar la funcionalidad del medio sonido.

RF-4.1 Reproducir sonido

La aplicación permitirá reproducir un archivo de sonido previamente seleccionado. La reproducción se aplicará sobre los formatos y códecs reconocidos por Java29

RF-4.2 Grabar sonido

Se permitirá grabar sonido, cuando se para la grabación, se añade automáticamente a la lista de reproducción.

RF-4.3 Mostrar evolución de la reproducción

De forma opcional mostrar la evolución de la reproducción en un contador digital.

RF-5 Requisitos de video

Permitirá gestionar la funcionalidad del medio video.

RF-5.1 Reproducir video

La aplicación permitirá reproducir un video previamente seleccionado. La reproducción se aplicará sobre los formatos y códecs reconocidos por Java29

RF-5.2 Mostrar secuencia webcam

La aplicación mostrará la secuencia que esté captando la webcan, para ello c

RF-5.3 Controlar la reproducción

La reproducción se podrá iniciar, parar o pausar.

RF-5.4 Capturar imágenes.

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”29. La imagen capturada se mostrará en una nueva ventana interna.

Descripción de una solución:

Vamos a dar solución independiente a cada modulo de requisitos y englobar todo para una sola vista del usuario.

Tenemos una clase llamada “VentanaFinal” que engloba el diseño gráfico e interactúa con el usuario (interfaz de usuario). Hemos creado una clase global llamada “VentanaInternaSM” con los atributos generales de todos los medios y para algunos requisitos tenemos una clase independiente que hereda de nuestra clase global. Es decir, para visualizar videos está “VentanaInternaJMFPlayer”, para gestionar la Webcam tendremos “VentanaInternaCamara”y para Imagen y dibujo tendremos “VentanaInternaImagen” ya que se pide que se pueda pintar

dibujos encima de una imagen, englobamos ambos para darle una buena solución. El sonido lo gestionamos en la propia “VentanaFinal”.

En la gestión de los requisitos de carácter general: nuevo, abrir, guardar

El botón nuevo, habré un nuevo lienzo, hemos pensado en hacerlo de dos formas, o con una clase propia que te pida el tamaño del nuevo lienzo, o en el mismo botón, nos hemos decantado por este último por falta de recursos (tiempo), te pide las dimensiones, primero ancho y después alto, tiene un mínimo de 20 y un máximo, el preferedsize de la ventana. En el caso de las imágenes, se reconocerá el tamaño de la misma y ese será el tamaño de la nueva ventana. El lienzo y las imágenes salen con fondo blanco y un rectángulo de delimita el mismo, es decir, no se podrá pintar fuera del límite marcado.

Hay un solo abrir que dependiendo de la extensión del archivo lo gestiona como sonido, audio o imagen, si no es una extensión reconocida lanza una ventana emergente donde te avisa que no reconoce la extensión del archivo, no podremos trabajar con este, no lo abre.

Guardamos solo imágenes, tanto si incluyen cambios como el brillo o si le añadimos figuras nuevas. No nos dejaba guardar capturas de pantalla sin tratar, para solucionarlo hemos añadido al método “getImage(true)” de la clase “Lienzo2DImagen” y le hemos añadido que, si la imagen es de tipo cero, le cambiamos el tipo y ahora es RGB (BufferedIma.TYPE_INT_ARGB).

Al guardar cualquier imagen quitamos el recuadro que delimita la imagen.

En “editar” podremos tener visibles u ocultos los diferentes menús, el superior que maneja los gráficos, video y audio, el inferior para imágenes, o la etiqueta de barra de menú. Dicha etiqueta muestra los colores R G B cuando pasamos el ratón por encima de una imagen.

Módulo de Gráficos.

Nos plantean una serie de figuras, con atributos independientes cada una de ellas, las cuales se podrán seleccionar en el lienzo, mover por el mismo, cambiar sus atributos una vez seleccionada, y mover en el espacio con las otras figuras. Este requisito implica obligatoriamente la definición de una jerarquía de clases asociadas a las formas y sus atributos.

Vamos a distinguir dos jerarquías de clases, una para los atributos, otro para las diferentes formas de los gráficos.

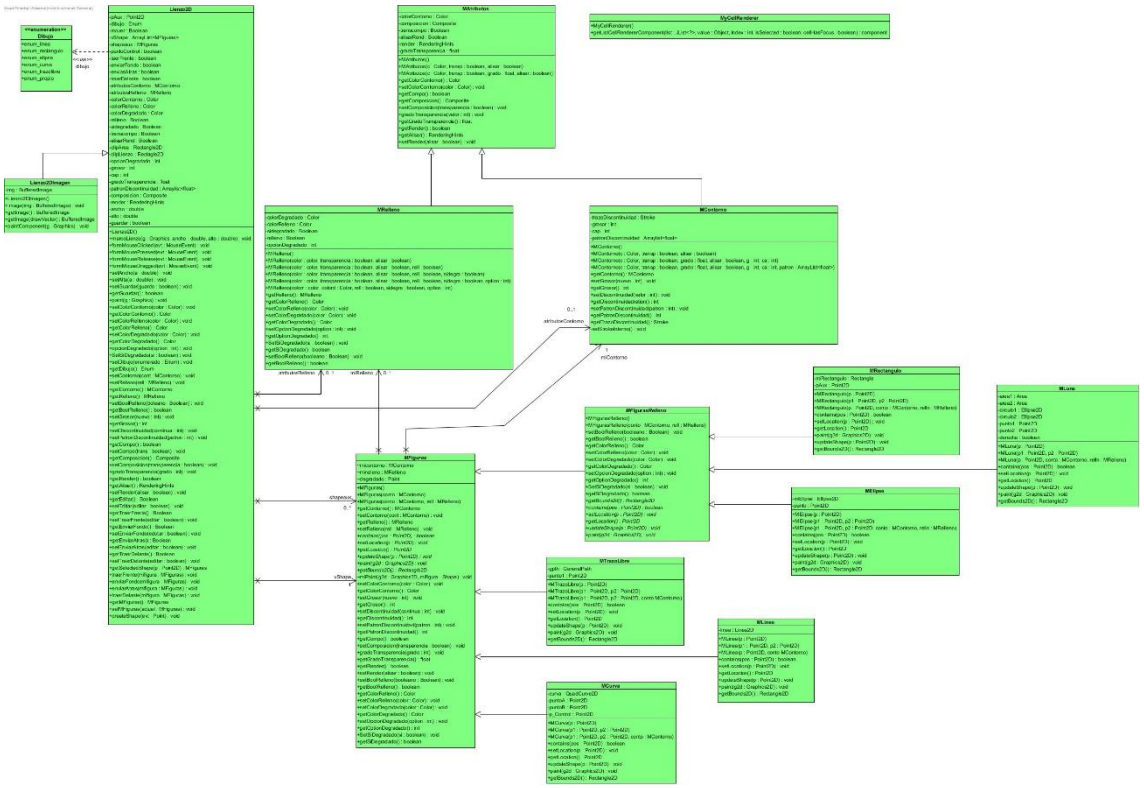
Para los atributos tenemos “MAtributos”, heredaran de esta “MContorno” que tendrá las propiedades especificas de los contornos de los dibujos, o los atributos de la figura en sí, si se trata de una línea o una curva (figuras sin relleno), y “MRelleno” tendrá las propiedades específicas del relleno de las figuras, si es que estas tienen relleno. Hemos decidido tener esta distribución de las propiedades para que las figuras que no tengan relleno no tengan por qué implementar esa funcionalidad. Nos es mas versátil.

Los atributos son usados por el “Lienzo2D” y por “MFiguras”.

La segunda jerarquía de clases para las figuras se llamará “MFiguras”, clase abstracta que contendrá toda la funcionalidad global de cada una de las figuras que se quieran crear, de esta hereda “MFigurasRelleno”, igual que la anterior clase abstracta que engloba la funcionalidad

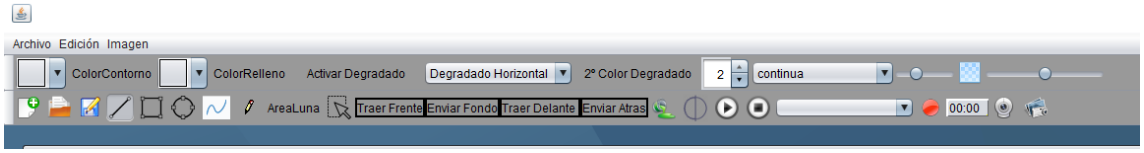
especifica de las figuras que tienen relleno. Después cada figura heredará de su clase superior, es decir, MCurva, MTrasoLibre y MLinea heredan directamente de MFiguras, y las clases MRectangulo, MEllipse y MLuna heredaran de la clase MFigurasRelleno, hemos decidido hacer esta estructura de clases porque es escalable, cualidad que vemos de vital importancia, ya que cuando queramos podemos implementar cualquier figura nueva incluyéndole solo la funcionalidad especifica de la misma. Dentro de las opciones que teníamos y teniendo en cuenta las indicaciones del profesor, nos ha parecido la mejor.

Para un mejor entendimiento del desarrollo de la jerarquía de clases de las figuras y sus atributos exponemos un diagrama de clases solo de ese paquete en concreto.



Cada figura tiene su botón, podrá estar seleccionado solo una figura a la vez, cuando esté seleccionado el botón de editar, ninguna figura estará seleccionada, con esta última opción, cuando cliquemos encima de una figura saldrá un recuadro discontinuo indicando que esa es la figura, podremos cambiar los atributos de la misma, así como cambiar el orden dentro de superposición con el resto de figuras, cuando cambiamos de figura seleccionada, los botones o diferentes formas de elegir los atributos se ponen como en la figura seleccionada de forma automática, para ello, en ventanaInternaImagen devolvemos la información de la figura activa en cada click de ratón, esa información se trata en la clase “VentanaFinal” en el método “setMiVentana”.

He visto conveniente que cuando se cambie de selección de una figura, por ejemplo, estoy en línea y clico sobre rectángulo, a partir de ese momento cuando cambie los atributos (ejemplo de color grosor) no afecte a la figura recién pintada, sino a la siguiente. la figura seleccionada no sería la actual, será la que vamos a pintar.



Como podemos ver en la imagen, tenemos los siguientes atributos para cada figura del lienzo, por supuesto si la figura seleccionada es de tipo relleno y se cambia un atributo de relleno esta selección se queda guardada y si seguido se dibuja una figura de relleno, se aplica.

Tenemos tanto para el contorno como para el relleno un ComboBox de colores y un dialogo para elegir el color, cuando elegimos el color con el dialogo, si el color existe entre los colores del ComboBox, el aspecto visual del mismo cambiará a dicho color.

Si tenemos activo el degradado, al inicio saldrá negro, hasta que elijamos otro color en un dialogo especifico para esto, no es la implementación esperada, pero por falta de recursos hemos decidido no cambiarlo por ahora. El degradado podrá ser horizontal, vertical o inclinado, podremos elegir la opción de una lista. Podemos decidir el grosor del contorno de las figuras, el tipo de discontinuidad de la misma, siendo las opciones: continua por defecto, punteada traza cuadrada, punteada traza redonda, punteada traza a tope. Con un slider contiguo podemos darle más separación a las líneas discontinuas.

Si activamos el botón de transparencia por defecto sale al 50%, con un slider contiguo podemos darle el grado de transparencia que queremos en cada figura.

En la barra de herramientas de abajo tenemos los botones de relleno y renderizado.

Módulo de imágenes.

Se creará un menú independiente en la parte de abajo, solo para imágenes, que permitirá hacer todas las modificaciones. Este podrá estar visible u oculto según tengamos activa la opción de la barra superior, “edición”.

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

Para duplicar (RF-3.1) añadimos la opción al menú superior “archivo”.

Modificaremos el brillo (RF-3.2) de la imagen aplicando el operador “RescaleOp”, se establece el valor definido por el usuario mediante un deslizador.

Podremos aplicar diferentes filtros (RF-3.3) basados en el operador de convolución. probaremos máscaras definidas en el paquete “sm.igame”, tenemos emborronamiento media, emborronamiento binomial, enfoque, relieve, fronteras laplaciano, y uno de diseño propio llamado experimento para probar una matriz 5x5.

Modificaremos el contraste (RF-3.4) de la imagen aplicando el operador “LookupOp”. Para ello, incluiremos tres botones correspondientes a tres posibles situaciones, contraste normal, contraste con iluminación y oscurecimiento. Todas estas funciones se encuentran implementadas en la clase “LokupTableProducer” del paquete “am.image”, basta con usar los parámetros por defecto que ofrece el método createLookupTable. De la misma forma hacemos el negativo de la imagen (RF-3.5).

Incluiremos la opción de mostrar las bandas asociadas a la imagen seleccionada (RF-3.6), incluiremos un botón que, al pulsarlo, hará que aparezcan en el escritorio tantas ventanas internas como bandas tenga la imagen, cada una de estas ventanas contendrá una imagen (en niveles de gris) con la correspondiente banda. No existe un método que nos devuelva una banda concreta así que crear nosotros mismos la imagen banda deseada, recorreremos el numero de bandas y para cada una de ellas llamaremos al método “getBanda” de la clase utilidades.

Añadiremos la posibilidad de cambiar de espacio de color (RF-3.7). Para ello, incluiremos en nuestra ventana principal una lista desplegable que muestre los distintos espacios de color disponibles RGB, YCC y GREY.

Rotaremos (RF-3.8) la imagen ofreciendo dos posibilidades, con giro libre, donde el usuario podrá girar 360 la imagen usando un deslizador, o rotaciones predeterminadas de 90, 180, 270. Todas las rotaciones llaman a la misma función que realiza la acción con el uso del operador “AffineTransformOp”. El escalado (RF-3.9) se hace con esta misma función, tendremos dos botones, uno para aumentar la imagen, otro para disminuirla de tamaño (en proporción al 25%, tanto para aumentar como para disminuir).

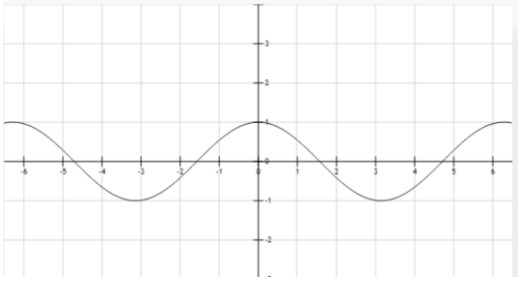
Aumentar, disminuir y rotación, da error cuando capturas una imagen, así que le paso un filtro propio que hace que al cambiar y tener el alpha activo siga haciendo la función esperada.

Incorporaremos el operador “sepia” (RF-3.10), hay varias transformaciones que producen este tipo de efecto, nos decantamos por realizar una clase propia SepiaOp que hereda de “BufferedImageOpAdapter”, en ella crearemos un filtro donde cambia el aspecto de la imagen pixel a pixel.

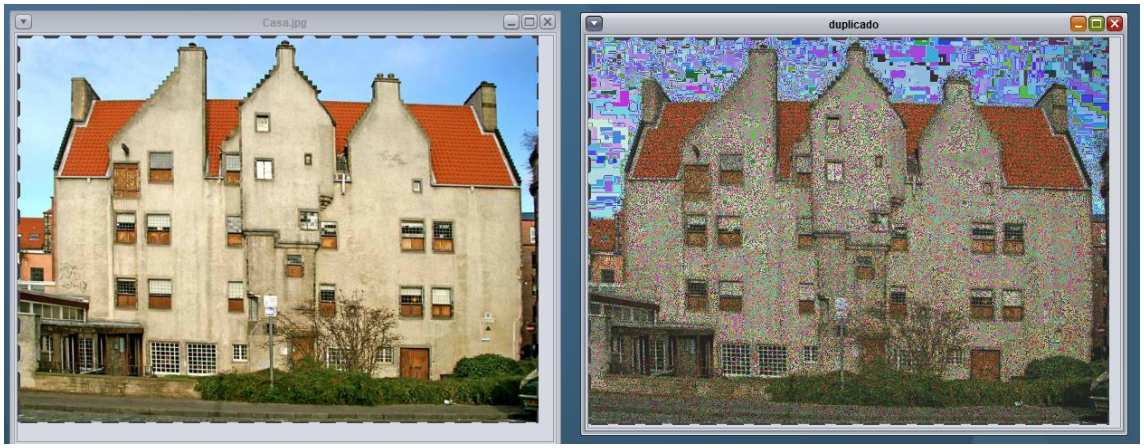
Respecto al operador basado en una función definida por el estudiante, definimos una clase llamada MlookupOp, ponemos dos botones, binariza y posteriza, asociamos cada botón a un filtro creado en la mencionada clase, donde se crean dos “LookupTable”, en binariza marcamos la mitad del espacio de color en 128 todo lo que esté por debajo lo dejaremos en 0 y por encima lo pondremos a 255 (data[i] = (i < 128)? (short) 0: (short) 255). En posteriza hacemos el %32 al recorrer el LookupTable de la imagen.

A la derecha la imagen original, en el centro la imagen binarizada, y a la derecha posterizada.

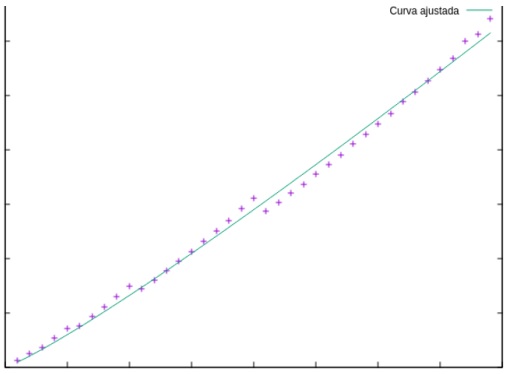
Para el operador de diseño propio componente a componente (RF-3.12), hacemos una clase llamada “PropioCC”, el cual hará la transformación de la imagen, recorre la imagen componente a componente aplicándole a cada componente el coseno de sí mismo (sample *= abs(cos(sample));).



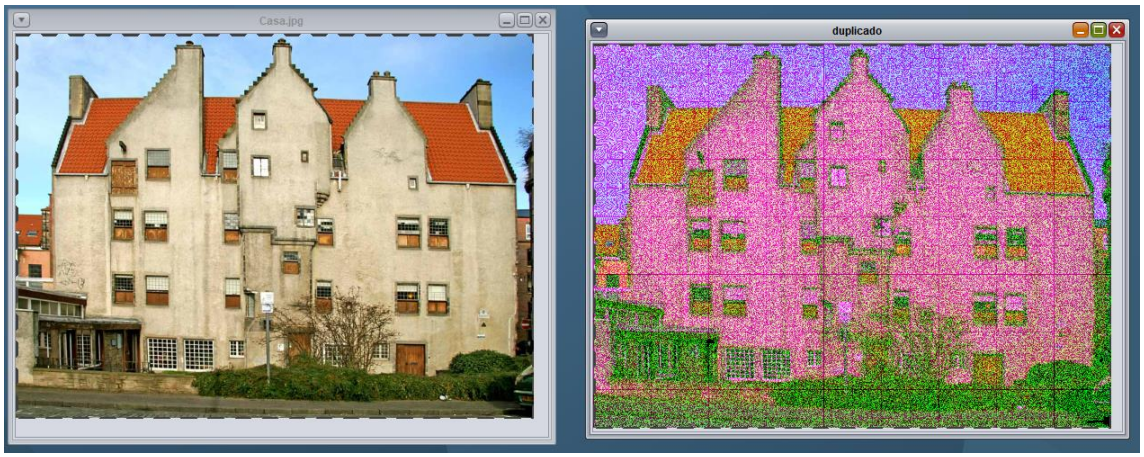
Vemos como queda visualmente la función definida.



Para el operador de diseño propio pixel a pixel (RF-3.13), hacemos una clase llamada “PropioPP”, el cual hará la transformación de la imagen, recorre la imagen pixel a pixel aplicándole a cada pixel la función i = x * y.



Vemos como queda visualmente la función definida, intensifica los colores dándole un aspecto de dibujo infantil.



Incorporaremos la posibilidad de “tintar” la imagen (RF-3.14) de color rojo, para ello usaremos la clase “TintOp” del paquete “sm.image” que implementa el operador de tintado desarrollado en teoría. Cuando la imagen tiene el canal Alpha activo no tinta la imagen, intentamos darle solución a este error creando un método en la clase utilidades, que recorre la imagen y pixel a pixel convierte los colores al rojo, con la deficiencia que si tintas mas de dos veces la imagen ya no se distingue bien.

Para ecualizar (RF-3.15) la imagen seleccionada usaremos la clase “EcualizationOp” del paquete “sm.image”.

El operador de umbralización (RF-3.16) se define con nuestra propia clase” UmbralizacionOp” del paquete “sm.MCG.imagen” de nuestra librería, donde se define un Umbral al crear la clase, se recorre la imagen pixel a pixel, si la intensidad de la imagen supera el umbral se lleva el pixel de la imagen a 255 y si se queda por debajo del umbral se pone a 0.

Módulo de sonido.

Los requisitos de sonido son reproducir archivos de sonido, grabar archivos de sonido y de forma opcional vamos a mostrar la evolución de la reproducción.
Para la reproducción añadimos dos botones consecutivos a los que tenemos, uno para que empiece la reproducción y otro como stop. Un JLabel al que llamaremos “contador” para seguir la evolución tanto de la reproducción como del tiempo que estamos grabando. Un botón de dos posiciones para grabar el sonido.

Cuando abrimos un audio, o cuando le damos a grabar, se incorpora a la lista de reproducción de nuestra lista desplegable (JComboBox) el archivo, al pulsar el botón de reproducción situado en la barra de herramientas, se reproduciría el sonido que esté seleccionado en ese momento en la lista de reproducción. Para conseguir reproducir vamos a hacer uso del paquete sm.sound adjunto por él profesor y de su clase SMPlayer. Igualmente, para grabar audio haremos uso de SMRecorder, lo grabado quedará guardado en un archivo seleccionado antes de que empiece la misma.

Hacemos una clase interna ManejadoraAudio que se hará cargo de la gestión de los botones cuando estemos reproduciendo.

Para mostrar la evolución de la reproducción y del grabado de audio, en los métodos “grabarSonidoBarrHerramActionPerformed” y “playSonidoActionPerformed” creamos una

hebra que se encargará del proceso, información de como crear una hebra sacada de www.youtube.com/watch?v=LoKQvAQpL3w .

Módulo de video.

El objetivo de este módulo será reproducir videos previamente seleccionados, acceder a la cámara (webcam) y capturar imágenes instantáneas tanto de los videos en reproducción como de lo que visualiza la webcam.
Para la reproducción y captura de instantáneas podemos usar JMF o VLCj, tenemos las dos implementadas, aunque solo usamos JMF, VLCj en nuestro pc no funcionaba y no hemos podido corregirlo, así que nos quedamos con JMF. El uso de webcam es basándose en la *Webcam Capture API.*
Hacemos nuestras ventanas antes mencionadas "VentanaInternaJMFPlayer" y "VentanaInternaCamara", ambas clases las instanciamos. Tenemos el método común getShot para la captura de instantáneas que dependiendo si la ventana activa es de reproducción o la webcam, se accederá a una u otra clase.

Al abrir un nuevo archivo, si es formato video, se abre automáticamente una nueva ventana con los botones para reproducir el mismo. Para activar la webcam y la captura de instantáneas usaremos un botón.
Las instantáneas se podrán tratar como cualquier otra imagen.

En ventanaInternaCamara, no logro poner la dimensión, porque si la pongo no puedo capturar la imagen que hay en el momento emitiendo por la webcam, no he podido arreglarlo, así que se ve muy pequeñita.

Diagrama de paquetes:

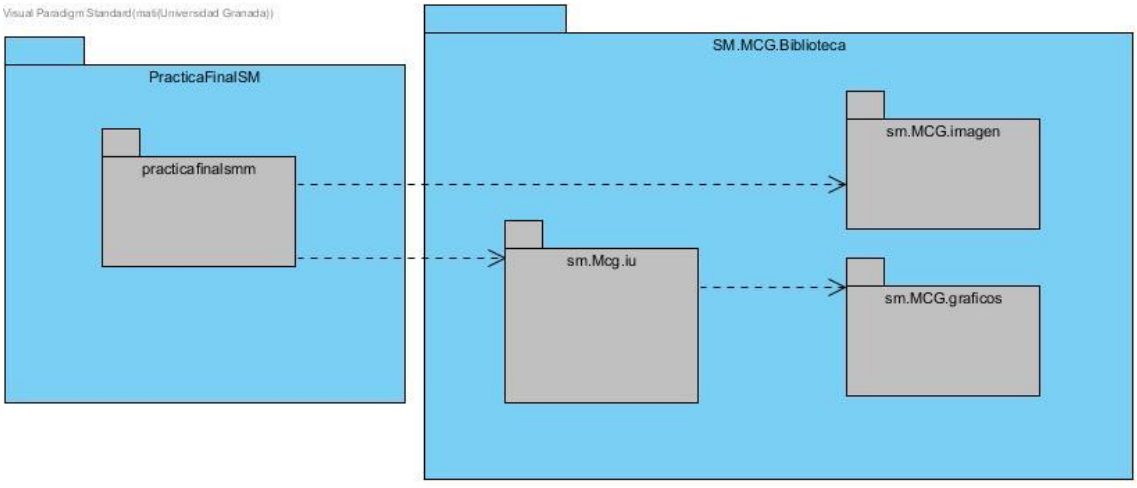


Diagrama de clases:

Mostramos el diagrama de clases de nuestra practicaFinal y la comunicación con nuestra biblioteca.

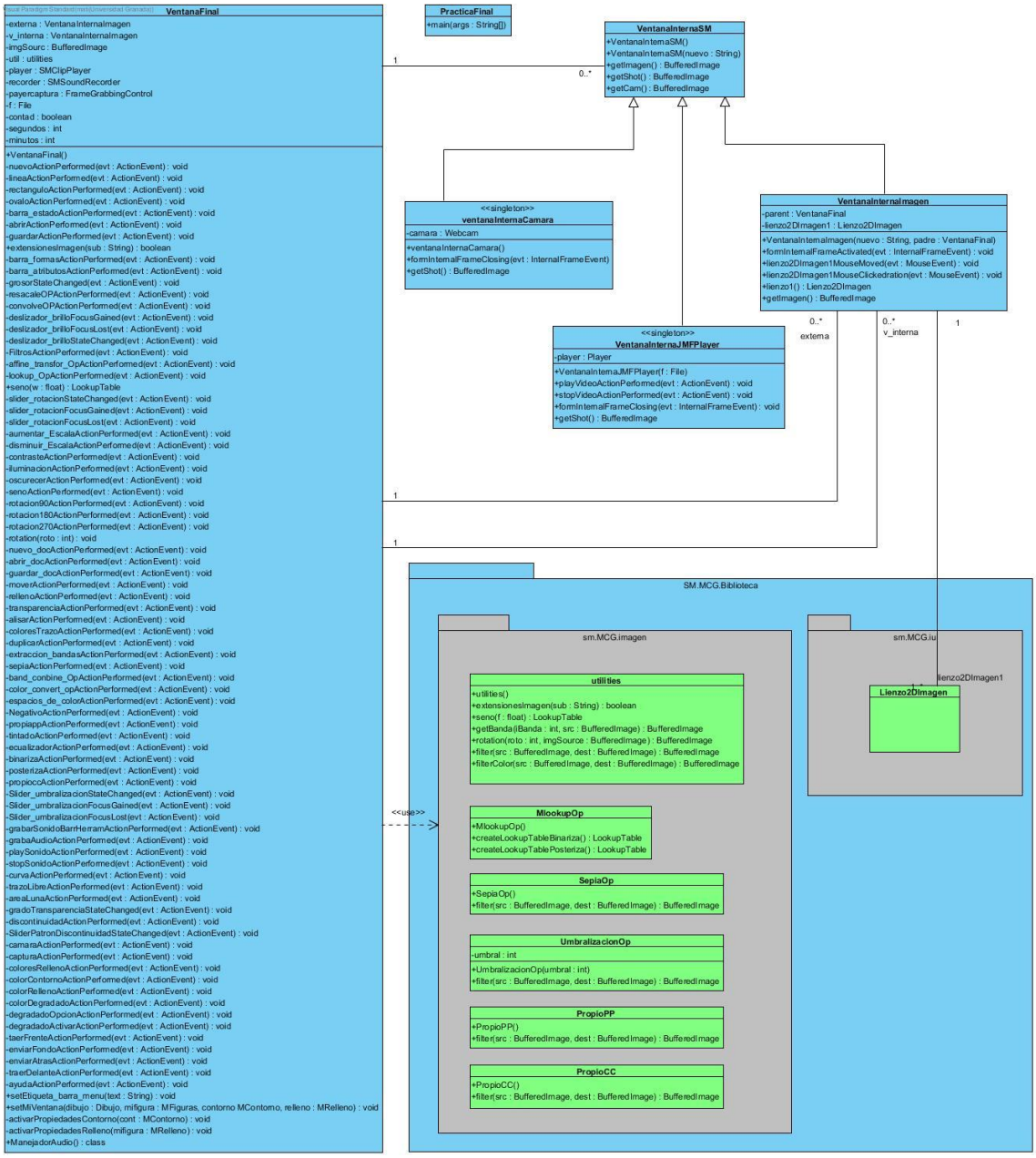
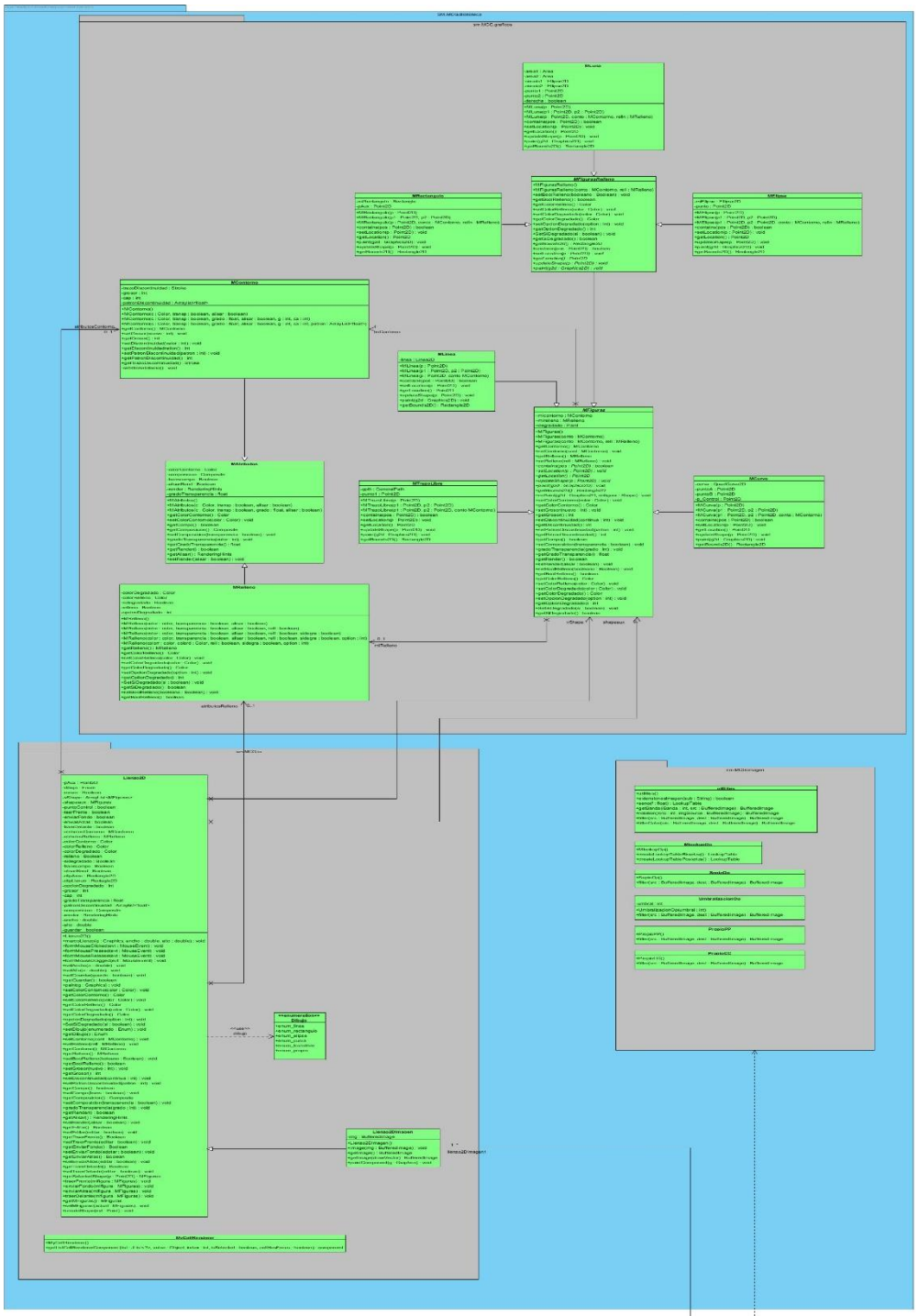
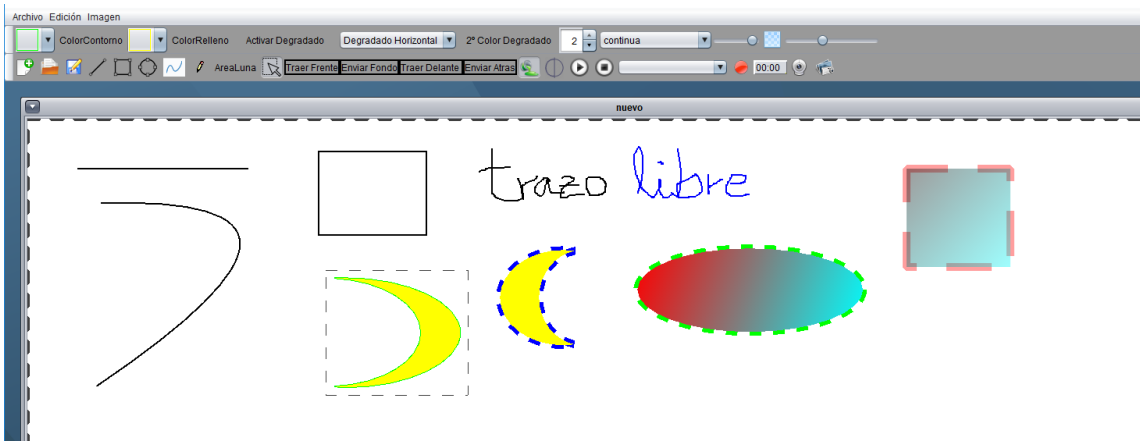


Diagrama de clases global, la representación de las clases de nuestra biblioteca en verde, las clases del paquete practicaFinal en azul.

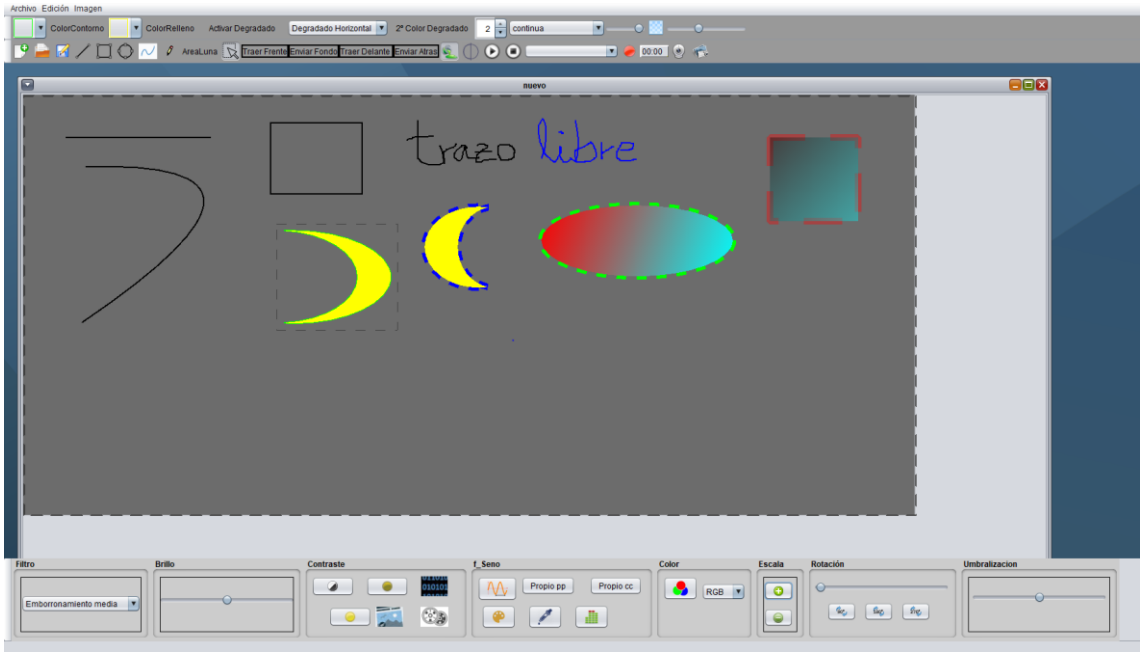


Ejemplo de uso de la aplicación:

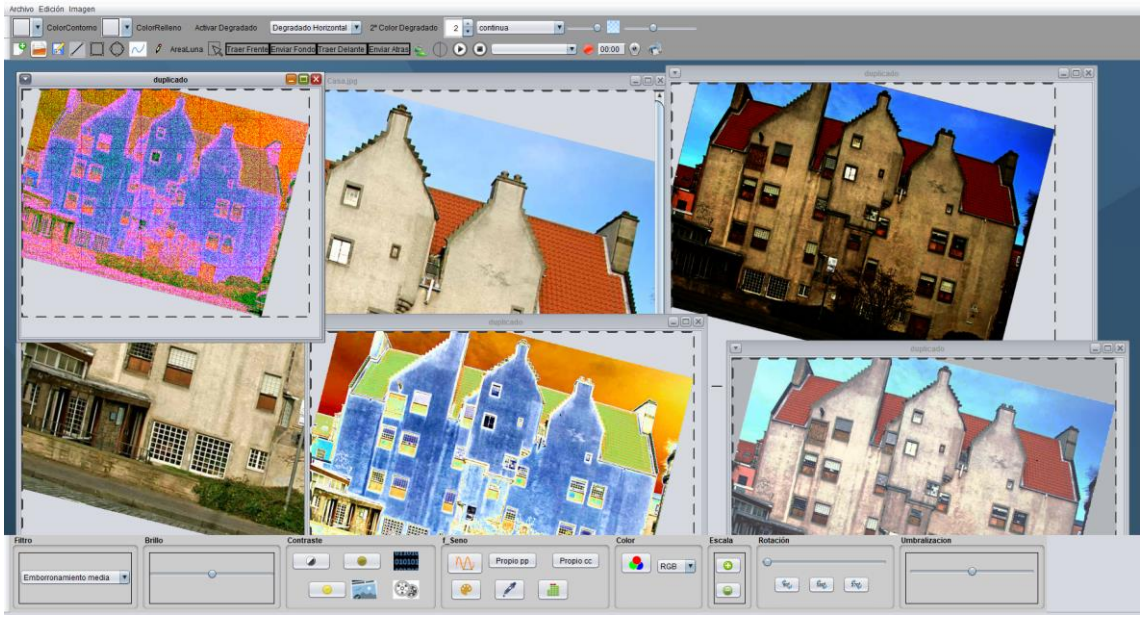
Haciendo figuras:



Cambio brillo



Aplicando operaciones a la imagen



Captura de imagen con la webcam.

Captura pantalla con video en funcionamiento.