ÍNDICE

Descripción.

Se trata de una aplicación multimedia que permite hacer dibujos simples, mostrar imágenes y reproducir sonido y vídeo. A cada sección del programa se puede acceder de forma sencilla e intuitiva desde la ventana principal (mediante paneles tabulados, ventanas internas o cualquier otra disposición).

1.1 Sección Dibujo.

Esta sección incluye en su interfaz:

- Panel de herramientas de dibujo
- Panel de colores
- Panel de atributos

En el en el primer panel aparece un botón por cada forma de dibujo disponible (el botón deberá incluir un icono). Al menos, el usuario podrá elegir una de las **siguientes formas**:

- Punto
- Línea recta
- Rectángulo
- Círculo
- Texto formateado
- Trazo libre
- Goma
- Curva con un punto de control
- Arco
- Rectángulo redondeado
- Curva con dos puntos de control
- Formas basadas en áreas

El usuario podrá dibujar sobre el lienzo utilizando la herramienta que tenga seleccionada en el panel de herramientas (el botón asociado a la forma que se está utilizando deberá estar pulsado -sólo puede estar pulsado uno de los botones-). Dicho dibujo se irá almacenando sobre una imagen y no será necesario permitir la edición de las formas ya dibujadas (de forma similar al programa *Paint* del sistema Windows).

La única excepción se aplicará a la última forma dibujada, la cual se podrá mover por el lienzo en cualquier momento **usando el botón derecho** (ésta se volcará sobre la imagen cuando se cambie de forma o se dibuje una nueva). El dibujo deberá realizarse utilizando las clases y métodos de **Graphics2D.**

Mediante el panel de colores el usuario podrá elegir el color del trazo y el de relleno. En dicho panel deberán aparecer una serie de colores predeterminados (como en el programa *Paint* de Windows) y, opcionalmente, la posibilidad de lanzar un diálogo de selección de colores. Para seleccionar el color del trazo se pulsará con el botón izquierdo, mientras que el derecho permitirá cambiar el color de relleno. En el panel de colores deberá haber una referencia al color de trazo y de relleno actuales.

En el panel de atributos el usuario podrá modificar los siguientes elementos:

- Trazo
- Relleno
- Fuente
- Composición
- Transformación

Respecto al trazo, se podrán modificar las cuatro propiedades básicas: el grosor del trazo, el estilo final de línea, el estilo de unión de líneas y el estilo del trazo. En este último caso, se podrán dibujar, al menos, líneas continuas o líneas punteadas (opcionalmente se pueden fijar más estilos de discontinuidad).

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, éste se aplicará utilizando los dos colores seleccionados en ese momento (para la dirección del degradado, se ofrecerá al menos dos posibilidades: horizontal y vertical). Una vez que la figura está dibujada, no es necesario permitir el posterior relleno de la misma. Opcionalmente, se podrá incorporar el relleno mediante imágenes predeterminadas.

La aplicación permitirá establecer la fuente, tamaño y estilo del texto. La escritura del texto se podrá hacer directamente sobre el área de dibujo o bien utilizando un diálogo previo en el que introducir la cadena. Independientemente de la forma de introducir el texto, éste deberá de aparecer en el punto de la imagen donde se haga el clic y con el formato indicado.

En relación a la composición, el usuario podrá fijar un grado de transparencia para la figura (opcionalmente, se puede incluir estilos de composición). Además, se deberá permitir la rotación de la figura, considerando para ello un número de ángulos fijos y predeterminados.

1.2 Sección Imagen.

En esta sección se mostrará una imagen abierta por el usuario (para el caso en el que la imagen sea mayor que la zona visible, deberá aparecer una barra de desplazamiento). Para ello, a través de la opción abrir del menú archivo, el usuario seleccionará una imagen (formatos GIF y JPEG) y ésta se mostrará en el panel. El usuario podrá repetir esta operación cuantas veces quiera, mostrándose únicamente la última imagen abierta.

Esta sección deberá incluir un panel de operaciones sobre la imagen. Desde dicho panel, el usuario podrá realizar las siguientes operaciones:

- Modificar el brillo mediante un deslizador
- Realzar la imagen
- Detectar fronteras

1.3 Sección Sonido y Video.

En esta sección se podrá reproducir sonido y vídeo. Para ello el usuario seleccionará un fichero (sonido o vídeo) a través de la opción abrir (menú archivo) y éste se reproducirá mostrando un panel en la parte inferior de esta sección que permitirá controlar la reproducción (iniciarla, pararla, avance rápido, etc.). En caso de que el fichero corresponda a un vídeo, este se mostrará en la zona central de esta sección.

1.4 Menú

En el menú principal deberán aparecer las opciones: ***Archivo, Ver*** **y** ***Ayuda***.

Dentro de la opción Archivo habrá cuatro opciones:

- **Nuevo:** Permite crear un dibujo nuevo, borrando el anterior.

- **Abrir:** Abre el dialogo "Abrir fichero". Su función dependerá de la sección que esté activa. Para la sección “Dibujo” se podrán abrir ficheros en formato GIF y JPEG, de forma que la imagen leída aparecerá en el fondo del lienzo (borrándose la que hubiese en ese momento). Para las secciones “Imagen” y “Sonido-Vídeo” la función de esta opción se explica en los apartados anteriores.

- **Guardar:** Lanza el dialogo "Guardar fichero" y permite guardar la **imagen** que hay en el lienzo en formato JPEG. El nombre y carpeta del fichero serán los indicados por el usuario a través del diálogo.

Esta opción no se usará en las secciones “Imagen” y “Sonido-Vídeo”

- **Salir:** Para abandonar el programa.

Dentro de la opción "*Ver"* se incluirá una opción "*Ver barra de estado*" que activará/desactivará la barra de estado. Dicha barra de estado deberá mostrar un mensaje indicando la herramienta elegida.

Dentro de la opción "*Ayuda"* se incluirá una opción "*Acerca de...*". Dicho diálogo incluirá una imagen logotipo del programa y el nombre del autor.

Las especificaciones indicadas en este documento son los requisitos mínimos, si bien el alumno podrá incorporar todas las alternativas que considere oportunas para mejorar la calidad de la aplicación (nuevas formas de dibujo, posibilidad de modificar más atributos del contexto, etc.)

1. Análisis.

La aplicación debe de constar de las tres secciones mencionadas en el apartado anterior, para ello pueden utilizar los componentes swing de java. Con ello conseguiremos estructurar la aplicación y darle el aspecto visual que deseemos de tal forma que sea una aplicación atractiva, fácil e intuitiva al mismo tiempo.

Para el ***dibujo*** tenemos que tener en cuenta que si utilizamos swing en vez de awt no será necesario la implementación del doble buffer ya que todos los componentes de swing lo traen implementado ya por defecto, por lo tanto la elección de swing sería la más acertada para la realización de la práctica, además de que swing posee muchas más opciones y características que awt. Deberemos de dibujar en un componente swing teniendo en cuenta el repintado de éste para ver los cambios que estamos produciendo al dibujar, estos cambios se producirán mediante el ratón fundamentalmente, ya que tendremos que ofrecer una serie opciones de dibujado que el usuario elegirá mediante éste. Por este motivo debemos facilitar en la medida de lo posible tanto la elección del tipo de figura a dibujar como sus características (color, grosor, tipo de línea, relleno etc.).

Por otra parte java nos ofrece clases pertenecientes al paquete awt.geom.* que nos podrían servir para el almacenamiento de figuras geométricas de una forma fácil y eficaz. Casi todas estas clases que contiene el paquete implementan la interfaz Shape que nos podría servir para englobarlas todas para cambiar ciertas características de las figuras. Otra facilidad que nos daría utilizar la interfaz shape es la de dibujar sobre el objeto Graphics o Graphics2D de cualquier elemento swing que elijamos simplemente pasándoselo como parámetro al objeto Graphics. Otras clases que nos harán falta son: awt.Point2D ó awt.Point para el manejo de puntos en el dibujo, las clases para las posibles transformaciones de los shapes contenidas en la clase awt.geom.AffineTransform que permite rotar, escalar, etc.

Para las características utilizaremos la clase awt.Color para el manejo de colores y modificación de éstos, awt.BasicStroke para definir estilos de línea, awt.RenderingHints para definir la calidad del dibujado en el objeto Graphics para así mejorar la eficiencia de la aplicación, awt.GradientPaint para la opción degradado de una figura. Otra clase que nos puede ser de utilidad es la clase Font para almacenar el tipo de letra que se utilizará para la opción insertar texto de la aplicación.

Para el la entrada y salida de la imagen se necesitarán las clases javax.imageio.ImageIO y awt.image.BufferedImage, y las clases para la codificación de las imágenes almacenadas en los BufferedImage al estándar JPEG: com.sun.image.codec.jpeg.JPEGImageEncoder junto con el códec correspondiente com.sun.image.codec.jpeg.JPEGCodec.

Estos son las clases fundamentales que se utilizarán para la sección de dibujo junto con los contenedores swing y otras clases fundamentales del paquete awt como pueden ser vectores, dimensiones etc.

Para la sección de ***edición de imagen*** se utilizarán prácticamente las mismas clases que para la sección anterior menos las relacionadas con figuras geométricas como eran awt.geom.*. Se utilizarán también para el manejo de imágenes la clase awt.Image, awt.Image.BufferedImage, javax.imageio.ImageIO y los codec mencionados anteriormente para almacenar y cargar las imágenes en JPEG.

Para llevar un control de las imágenes cargadas y asegurarnos que la imagen se ha cargado en su totalidad antes de visualizarla o trabajar con ella se utilizará la clase awt.MediaTracker.

Para realizar las transformaciones en las imágenes se utilizarán las clases awt.image.Kernel, para definir matrices que servirán de filtro para las imágenes, awt.image.ConvolveOp para aplicar los filtros Kernel definidos a una imagen determinada y awt.image.RescaleOp para cambiar el brillo de las imágenes.

Para la sección de ***Sonido y Video*** utilizaremos las herramientas que nos proporciona Sun para reproducir sonidos y videos con el nuevo conjunto de clases contenidas en el Java Media Framework (utilizaremos la versión 2.1), estas herramientas estas herramientas deben de ser instaladas aparte de la JDK ya que no están incluidas en ésta. Una vez instaladas se encuentran en el paquete javax.media.*, algunas de las clases más importantes utilizadas en esta sección serían javax.media.Player para la reproducción de los elementos multimedia, javax.media.Control para la obtención de los controles de reproducción para los medios etc. Otras clases de utilidad serían la clase java.io.File para cargar archivos y javax.swing.Timer para mandar realizar tareas periódicas durante la reproducción.

Además en todas las secciones se añadirán otras posibles clases creadas especialmente para el desarrollo de la aplicación que serán también incluidas en el proyecto.

En el proyecto también se añadirán iconos, imágenes y cursores para hacer que la aplicación sea más vistosa y hacerla más intuitiva.

2. Diseño.

2.1 Aplicación

La aplicación estará formada por tres módulos independientes:

1. Un módulo llamado Paint, encargado de la sección de dibujo.
2. Otro llamado EditorImagen para la sección de imagen.
3. Y el último llamado ReproductorVideoSonido que se encarga de la sección de reproducción de archivos multimedia.

Estos tres módulos están implementados sobre un JPanel de forma independiente por lo que se podrían considerar incluso programas sin dependencia alguna, de tal forma que el objeto Paint funciona por si mismo sin la necesidad de cualquier otro módulo de la aplicación, de igual forma ocurre con los otros dos.

La clase principal es **PaintMultimedia.java** que contiene un objeto de cada módulo y que sólo se encarga de que estas tres secciones sean alternativas de tal forma que sólo se muestre al usuario una de ellas a la vez, dándole a elegir en todo momento la sección. Para conseguir esto se han incrustado las tres secciones (que como hemos dicho antes son paneles en sí) en un JFrame que es el que engloba toda la aplicación. En este JFrame se incluyen los menús y la barra de archivo que son comunes para todas las secciones aunque con algunas diferencias dependiendo de la sección en la que el usuario se encuentre actualmente y las funcionalidades de la sección elegida. Un ejemplo de esto es que en la sección VideoSonido no se da la opción de Guardar como en las demás, sin embargo si que hay una opción en el menú que permite ocultar una barra de información que solo se encuentra en esta sección.

El método main del programa está en la clase mainMultimedia que simplemente contiene un objeto EditorMultimedia y lanza la aplicación.

Las tres secciones de que consta el programa se añadirán al EditorMultimedia en la parte central de éste y serán gestionadas mediante un CardLayout para poder visualizar de una en una cada sección ocultando al mismo tiempo las demás y mostrándolas las tres en la misma posición de la aplicación (centro). En la parte superior se incluirá una barra de archivo con iconos que indiquen las posibles operaciones fundamentales para las tres secciones; guardar, abrir, cerrar, nuevo, ayuda. En la parte central de esta barra se sitúan tres JTogglebutton indicando que sección está activa en cada momento, para ello se utiliza un ButtonGroup que gestiona que solo uno de ellos puede estar seleccionado al mismo tiempo, es decir, estos tres botones conmutan el tipo de sección dentro de la aplicación.

Ésta clase también se encarga de pedirle confirmación al usuario siempre que éste desee realizar alguna acción que implique pérdida de datos en ese momento, como por ejemplo al salir si no se ha guardado el archivo actual con modificaciones, cargar un nuevo archivo si no se ha guardado el actual o cambiar de sección sin

guardar el archivo. También pide confirmación siempre que se desee salir de la aplicación y se encuentre un archivo abierto y modificado en ese momento.

Otro elemento de la clase EditorMultimedia es el About_Box que es un JDialog con la información del autor, fecha, descripción del programa, logotipo etc. Éste diálogo es una variable miembro de la clase y de tipo EditorMultimedia_AboutBox.

2.2 Dibujo

En la parte del dibujo se pueden apreciar dos partes diferentes, una sería la relacionada con el dibujado en sí como serían las clases que se encargan de almacenar información de las figuras, sus características, sus datos y puntos, su dibujado y repintado en la pantalla, etc. Y por otro lado está la parte que se encarga del aspecto visual y entorno gráfico con botones, listas, etc., que sirven para fijar las características y las distintas formas de dibujo.

3.2.1 Primero empezaremos por la parte más interna y menos visible de ésta sección que se corresponde con las **clases encargadas de almacenar la información** a dibujar.

En primer lugar tenemos una clase **interfaz** llamada **Dibujable** que como su propio nombre indica marca unas pautas bien definidas para cualquier objeto que se quiera dibujar. En esta interfaz se definen funciones tales como:

- dibujar(Graphics g) que nos permite dibujar el objeto en el gráfico pasado por parámetro.
- rotar un cierto número de grados el objeto al ser dibujado.
- trasladar un objeto en una dirección una determinada distancia al ser dibujado.

Esta interfaz se ha creado con el objetivo de marcar unas pautas para las clases que la implementen, sin la necesidad de heredar de ella y así para permitirles heredar de otra clase diferente. Otro motivo por el que he creado la interfaz es porque nos permite crear objetos dibujables con diferentes implementaciones y mediante el polimorfismo tratarlos todos aunque sean de diferente tipo como objetos dibujables así cambiar poder cambiar sus propiedades de dibujo sin conocer su tipo exacto (es muy útil cuando se realizan bucles para pintar un vector de objetos de diferentes tipos pero todos ellos dibujables).

La clase **ContextoGrafico** almacena información sobre características de dibujado tales como el color, color de relleno, tipo de línea, grosor, degradado, fuente, estilo unión de líneas, estilo final de línea etc.

Esta clase posee cinco variables miembro que sirven para el almacenamiento de las propiedades de dibujo. Estas variables son:

- Color colorLinea. Indica el color de línea.
- Color colorRelleno. Indica si posee relleno o no, y en caso de estar rellenado su color.
- Font fuente. Indica el estilo y tipo de fuente a la hora de introducir texto.
- BasicStroke estiloLinea. Indica el estilo de línea con el que será dibujado el objeto.
- GradientPaint degradado. Indica si el objeto posee o no degradado, y en caso de tenerlo su valor.

Para poder almacenar y dibujar las distintas formas que permite la aplicación, dividimos en *dos grupos diferentes* los posibles elementos a dibujar. Por un lado se podrían considerar los elementos geométricos como serían rectángulos, óvalos etc. Y por otros elementos que no son geométricos y que constan de un determinado número de puntos sueltos como podrían ser el dibujo con lápiz, pincel o goma. Los primeros serán codificados por la clase **GeometríaDibujable** y sus descendientes y los segundos serán codificados por la clase **DibujoLapizDibujable** y sus descendientes. Todos los elementos de ambos grupos implementan la interfaz dibujable, si no la implementan directamente la implementa el padre.

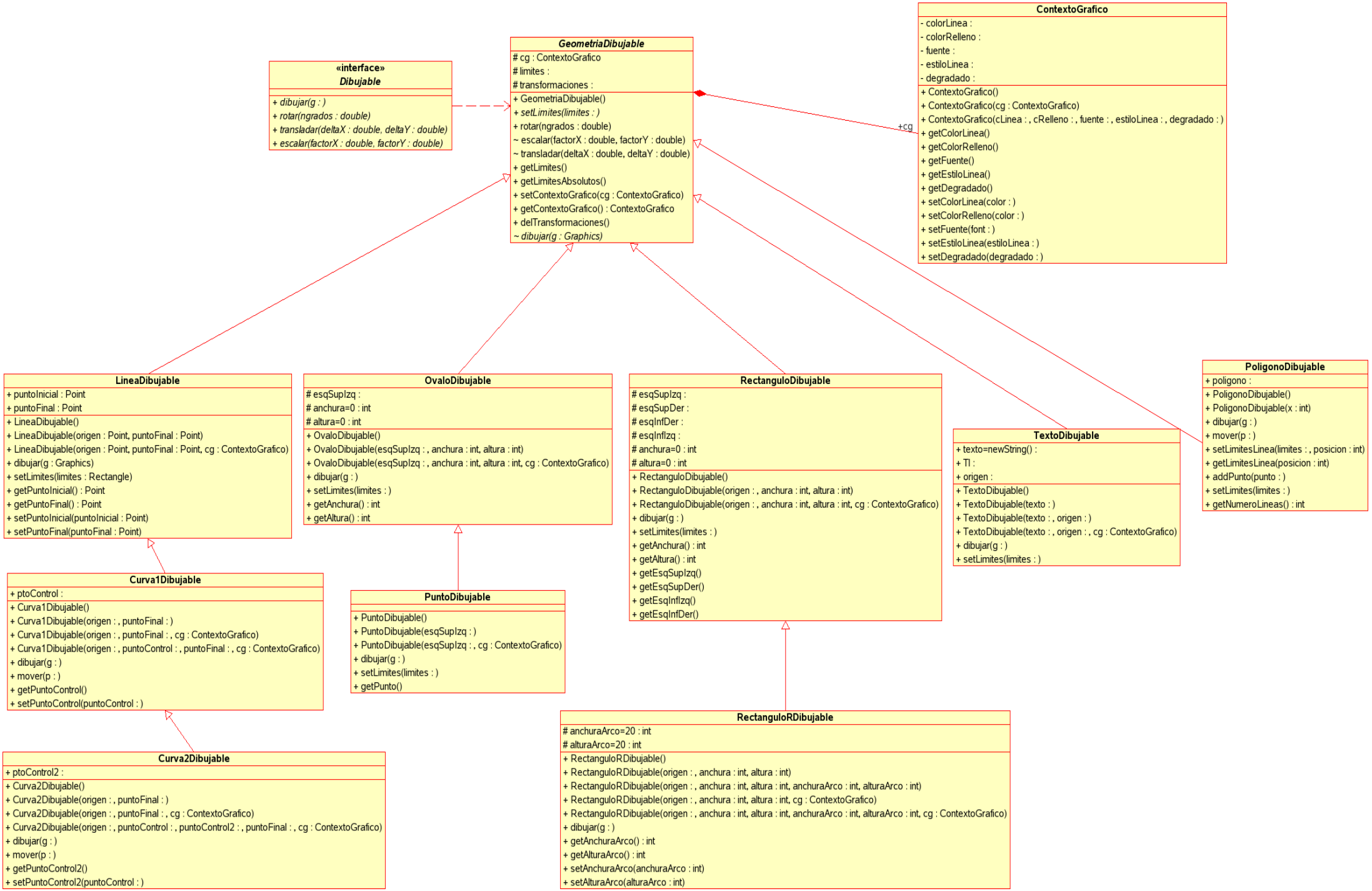
- Veamos la organización del ***primer grupo***:

Tenemos la **clase abstracta GeometríaDibujable** que representa a cualquier figura geométrica que pueda ser dibujada. Esta clase implementa la interfaz Dibujable para que los elementos de su tipo puedan ser dibujados. Esta clase he decidido hacerla abstracta ya que cualquier figura geométrica posee unas características comunes a las demás pero tiene algunos métodos cuya implementación depende del tipo de figura geométrica que sea. Por lo tanto métodos como dibujar(Graphics g) dependen del tipo de figura de que se trate y por lo tanto debe ser implementado por las clases hijas y no por el padre. Sin embargo hay métodos que están definidos dentro de la clase abstracta y que no son abstractos porque no dependen del tipo de figura geométrica y solo dependen de sus características como GeometriaDibujable o de métodos abstractas de ésta que deben ser implementadas por sus clases hijas.

La clase GeometriaDibujable contiene tres variables miembro:

- ContextoGrafico cg. Almacena las características de dibujo de la figura geométrica. Pertenece a la clase ContextoGrafico que se explicó anteriormente.
- Rectangle limites. Almacena los límites de la figura geométrica de tal forma que ésta siempre está circunscrita en éstos límites.
- AffineTransform transformaciones. Almacena las posibles transformaciones que se le apliquen a la figura pero sin modificar sus siendo los mismos solo que se rota a la hora de dibujarlo.

Después tenemos las **clases** encargadas de implementar las **distintas figuras geométricas** que se utilizarán en la aplicación. Aquí podemos ver una aproximación al diagrama de clases que forman:


«interface»
Dibujable
+ dibujar(g :)
+ rotar(ngrados : double)
+ transladar(deltaX : double, deltaY : double)
+ escalar(factorX : double, factorY : double)
GeometriaDibujable
cg : ContextoGrafico
limites :
transformaciones :
+ GeometriaDibujable()
+ setLimites(limites :)
+ rotar(ngrados : double)
~ escalar(factorX : double, factorY : double)
~ transladar(deltaX : double, deltaY : double)
+ getLimites()
+ getLimitesAbsolutos()
+ setContextoGrafico(cg : ContextoGrafico)
+ getContextoGrafico() : ContextoGrafico
+ delTransformaciones()
~ dibujar(g : Graphics)
+cg
ContextoGrafico
- colorLinea :
- colorRelleno :
- fuente :
- estiloLinea :
- degradado :
+ ContextoGrafico()
+ ContextoGrafico(cg : ContextoGrafico)
+ ContextoGrafico(cLinea : , cRelleno : , fuente : , estiloLinea : , degradado :)
+ getColorLinea()
+ getColorRelleno()
+ getFuente()
+ getEstiloLinea()
+ getDegradado()
+ setColorLinea(color :)
+ setColorRelleno(color :)
+ setFuente(font :)
+ setEstiloLinea(estiloLinea :)
+ setDegradado(degradado :)
LineaDibujable
+ puntoInicial : Point
+ puntoFinal : Point
+ LineaDibujable()
+ LineaDibujable(origen : Point, puntoFinal : Point)
+ LineaDibujable(origen : Point, puntoFinal : Point, cg : ContextoGrafico)
+ dibujar(g : Graphics)
+ setLimites(limites : Rectangle)
+ getPuntoInicial() : Point
+ getPuntoFinal() : Point
+ setPuntoInicial(puntoInicial : Point)
+ setPuntoFinal(puntoFinal : Point)
OvaloDibujable
esqSupIzq :
anchura=0 : int
altura=0 : int
+ OvaloDibujable()
+ OvaloDibujable(esqSupIzq : , anchura : int, altura : int)
+ OvaloDibujable(esqSupIzq : , anchura : int, altura : int, cg : ContextoGrafico)
+ dibujar(g :)
+ setLimites(limites :)
+ getAnchura() : int
+ getAltura() : int
RectanguloDibujable
esqSupIzq :
esqSupDer :
esqInfDer :
esqInfIzq :
anchura=0 : int
altura=0 : int
+ RectanguloDibujable()
+ RectanguloDibujable(origen : , anchura : int, altura : int)
+ RectanguloDibujable(origen : , anchura : int, altura : int, cg : ContextoGrafico)
+ dibujar(g :)
+ setLimites(limites :)
+ getAnchura() : int
+ getAltura() : int
+ getEsqSupIzq()
+ getEsqSupDer()
+ getEsqInfIzq()
+ getEsqInfDer()
TextoDibujable
+ texto=newString() :
+ TI :
+ origen :
+ TextoDibujable()
+ TextoDibujable(texto :)
+ TextoDibujable(texto : , origen :)
+ TextoDibujable(texto : , origen : , cg : ContextoGrafico)
+ dibujar(g :)
+ setLimites(limites :)
PoligonoDibujable
+ poligono :
+ PoligonoDibujable()
+ PoligonoDibujable(x : int)
+ dibujar(g :)
+ mover(p :)
+ setLimitesLinea(limites : , posicion : int)
+ getLimitesLinea(posicion : int)
+ addPunto(punto :)
+ setLimites(limites :)
+ getNumeroLineas() : int
Curva1Dibujable
+ ptoControl :
+ Curva1Dibujable()
+ Curva1Dibujable(origen : , puntoFinal :)
+ Curva1Dibujable(origen : , puntoFinal : , cg : ContextoGrafico)
+ Curva1Dibujable(origen : , puntoControl : , puntoFinal : , cg : ContextoGrafico)
+ dibujar(g :)
+ mover(p :)
+ getPuntoControl()
+ setPuntoControl(puntoControl :)
PuntoDibujable
+ PuntoDibujable()
+ PuntoDibujable(esqSupIzq :)
+ PuntoDibujable(esqSupIzq : , cg : ContextoGrafico)
+ dibujar(g :)
+ setLimites(limites :)
+ getPunto()
Curva2Dibujable
+ ptoControl2 :
+ Curva2Dibujable()
+ Curva2Dibujable(origen : , puntoFinal :)
+ Curva2Dibujable(origen : , puntoFinal : , cg : ContextoGrafico)
+ Curva2Dibujable(origen : , puntoControl : , puntoControl2 : , puntoFinal : , cg : ContextoGrafico)
+ dibujar(g :)
+ mover(p :)
+ getPuntoControl2()
+ setPuntoControl2(puntoControl :)
RectanguloRDibujable
anchuraArco=20 : int
alturaArco=20 : int
+ RectanguloRDibujable()
+ RectanguloRDibujable(origen : , anchura : int, altura : int)
+ RectanguloRDibujable(origen : , anchura : int, altura : int, anchuraArco : int, alturaArco : int)
+ RectanguloRDibujable(origen : , anchura : int, altura : int, cg : ContextoGrafico)
+ RectanguloRDibujable(origen : , anchura : int, altura : int, anchuraArco : int, alturaArco : int, cg : ContextoGrafico)
+ dibujar(g :)
+ getAnchuraArco() : int
+ getAlturaArco() : int
+ setAnchuraArco(anchuraArco : int)
+ setAlturaArco(alturaArco : int)

Estas clases **heredan de GeometriaDibujable** por lo que todas contienen además de su información sobre puntos etc, un contexto gráfico para que puedan ser dibujadas. Todas ellas implementan de forma diferente el método abstracto Dibujar de la clase padre ya que no se dibuja igual un óvalo que un cuadrado por ejemplo. Además estas clases deben modificar la variable del padre límites cada vez que realicen alguna modificación en la que el significado de ésta se vea afectado para mantener en todo momento su consistencia. Las características y variables internas de estas clases están especificadas en la documentación de éstas mediante comentarios que pueden ser visualizadas mediante javadoc.

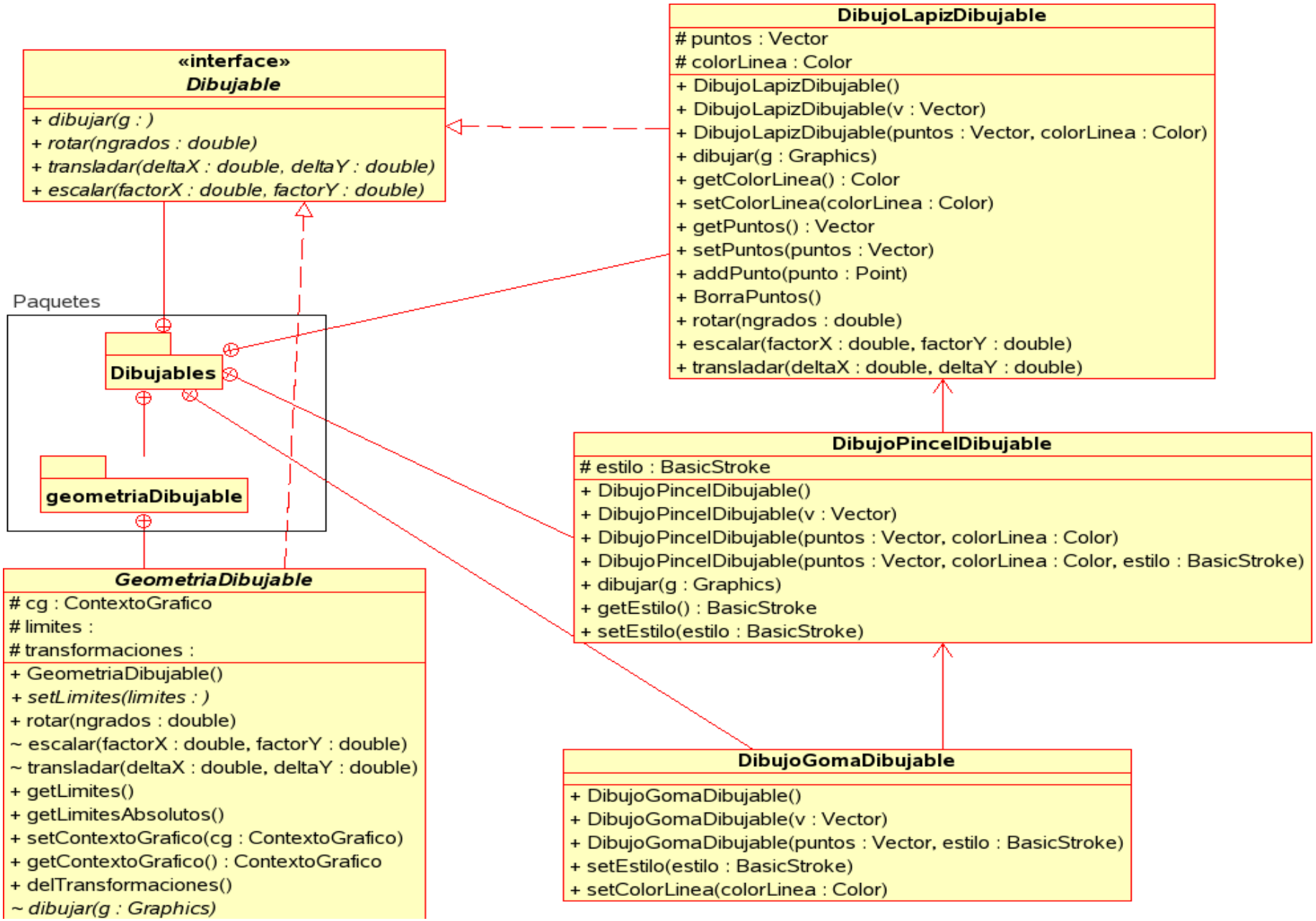
Como se puede observar algunas de estas clases heredan unas de otras como es el ejemplo de RectanguloDibujable y RectanguloRDibujable (éste último hereda del primero) ya que son un caso especial que poseen todas las propiedades del padre y algunas adicionales que lo hacen diferente.

Gracias a ésta estructuración la aplicación ***no sólo nos permite dibujar*** en todo momento lo que se desee sino que también ***nos permite la edición de los objetos*** pudiéndoles modificar en todo momento sus características como tamaño, posición, color, relleno, estilo de línea etc.

-Veamos la organización del ***segundo grupo***:

En el segundo grupo tenemos como hemos dicho antes a la clase que se encarga de almacenar el dibujo realizado con un lápiz. Esta clase además de implementar la interfaz Dibujable, sirve de base para el dibujo con un pincel (posee estilo de línea y grosor) y ésta a su vez para otra clase que permite el borrado con una goma. Estas clases como es evidente no se han incluido dentro del paquete geometríaDibujable porque como su propio nombre indica no son figuras geométricas aunque sí dibujables por lo que implementan la interfaz por ellas mismas.

El esquema quedaría de la siguiente forma:


«interface»
Dibujable
+ dibujar(g :)
+ rotar(ngrados : double)
+ transladar(deltaX : double, deltaY : double)
+ escalar(factorX : double, factorY : double)
DibujoLapizDibujable
puntos : Vector
colorLinea : Color
+ DibujoLapizDibujable()
+ DibujoLapizDibujable(v : Vector)
+ DibujoLapizDibujable(puntos : Vector, colorLinea : Color)
+ dibujar(g : Graphics)
+ getColorLinea() : Color
+ setColorLinea(colorLinea : Color)
+ getPuntos() : Vector
+ setPuntos(puntos : Vector)
+ addPunto(punto : Point)
+ BorraPuntos()
+ rotar(ngrados : double)
+ escalar(factorX : double, factorY : double)
+ transladar(deltaX : double, deltaY : double)
Paquetes
Dibujables
geometriaDibujable
DibujoPincelDibujable
estilo : BasicStroke
+ DibujoPincelDibujable()
+ DibujoPincelDibujable(v : Vector)
+ DibujoPincelDibujable(puntos : Vector, colorLinea : Color)
+ DibujoPincelDibujable(puntos : Vector, colorLinea : Color, estilo : BasicStroke)
+ dibujar(g : Graphics)
+ getEstilo() : BasicStroke
+ setEstilo(estilo : BasicStroke)
GeometriaDibujable
cg : ContextoGrafico
limites :
transformaciones :
+ GeometriaDibujable()
+ setLimites(limites :)
+ rotar(ngrados : double)
~ escalar(factorX : double, factorY : double)
~ transladar(deltaX : double, deltaY : double)
+ getLimites()
+ getLimitesAbsolutos()
+ setContextoGrafico(cg : ContextoGrafico)
+ getContextoGrafico() : ContextoGrafico
+ delTransformaciones()
~ dibujar(g : Graphics)
DibujoGomaDibujable
+ DibujoGomaDibujable()
+ DibujoGomaDibujable(v : Vector)
+ DibujoGomaDibujable(puntos : Vector, estilo : BasicStroke)
+ setEstilo(estilo : BasicStroke)
+ setColorLinea(colorLinea : Color)

3.2.2 Además de las clases encargadas de almacenar la información del dibujo tenemos otras clases que se utilizan en el objeto Paint y que están **orientadas al entorno gráfico de la aplicación** para que el usuario pueda cambiar las características de los objetos que está dibujando y así modificar las propiedades de los objetos del apartado 3.2.1.

Una de las clases necesarias para poder ofrecerle al usuario la elección de un color es la clase **PaletaColores**. Esta clase nos permite crear una paleta de colores con un determinado número de colores distribuidos como celdas de una matriz de una cierta dimension. La paleta de colores hereda de JPanel por lo que se trata un panel con etiquetas de colores distribuidos uniformemente sobre éste. Posee métodos que permiten cambiar el color de las etiquetas, devolver el color de las etiquetas dada su posición en la matriz (fila, columna) y su posición absoluta según el orden que posea. Este objeto no gestiona los eventos recibidos de ratón sino que solo se encarga de mostrarlos en pantalla, sin embargo se pueden obtener fácilmente ya que se puede llamar al método de la paleta de colores getComponentAt(punto del click), éste nos devuelve el componente en cual se a picado (hay que tener en cuenta que si no se pica sobre ningún color nos devolverá la propia paleta). Una vez que tenemos el componente sobre el cual se ha hecho el click solo se tiene que obtener su color de Background.

Otra clase es **DialogoDegradado** que simplemente muestra un diálogo que permite elegir los colores de degradado y la dirección de éste para el relleno de las figuras dibujadas. Además este diálogo presenta un pequeño cuadro donde muestra al usuario el resultado del degradado que está seleccionando en cada momento. Para poder mostrarle al usuario en todo momento el resultado y que sea repintado cada vez que altere cualquier parámetro se utiliza un objeto Lienzo que describimos a continuación.

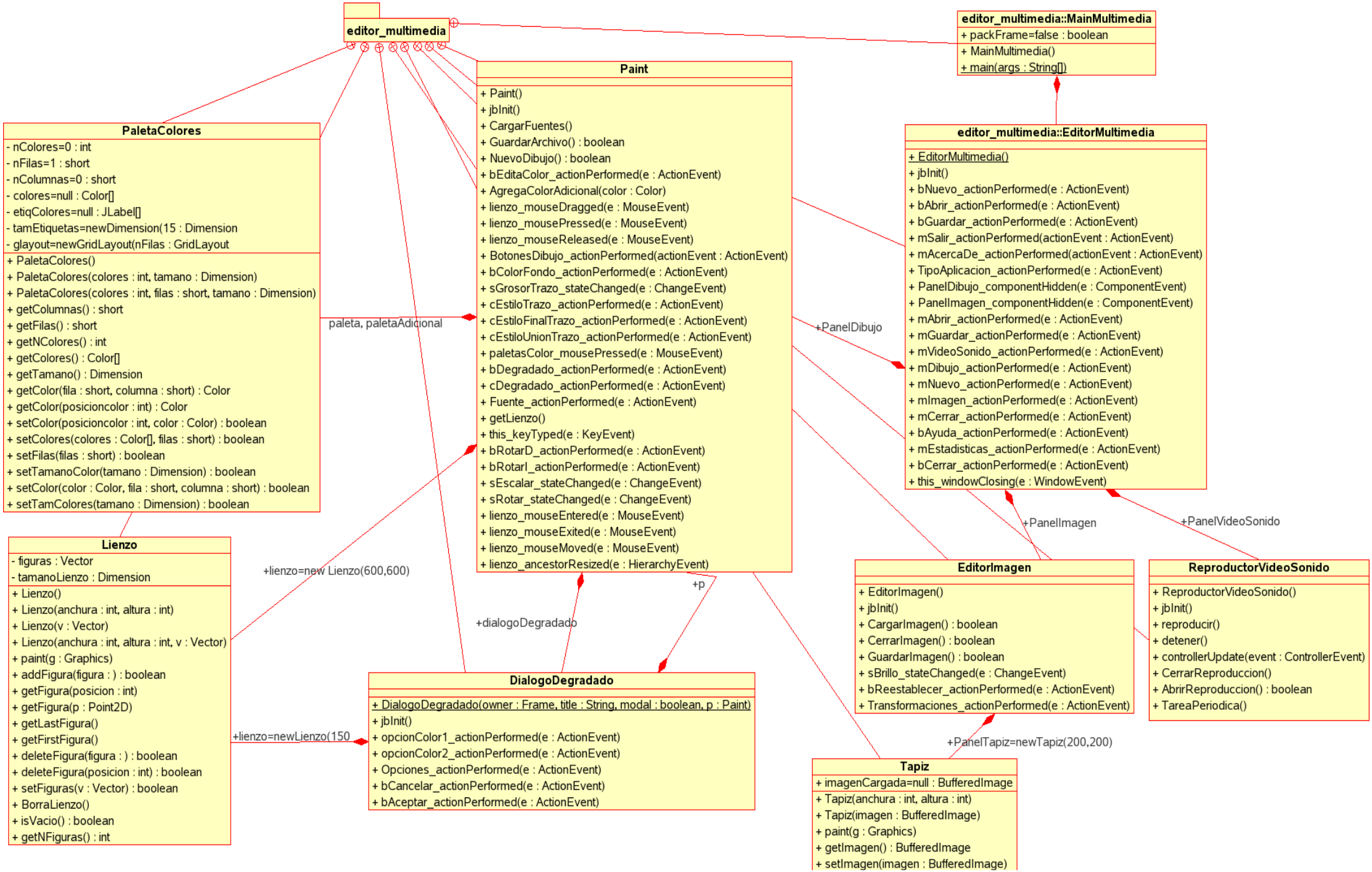
El objeto **Lienzo** es de los elementos más importantes en el dibujo. Mediante el lienzo el usuario puede ver lo que está dibujando en todo momento e incluso es capaz de ser repintado si se ocultara mediante cualquier otra ventana u objeto. Para poder realizar esto, el lienzo posee un vector de figuras que son las que debe de pintar siempre que sea necesario. De ésta forma el lienzo no es un simple panel sino que es un panel junto con todos los objetos que se dibujan sobre él. La única restricción que tiene es que cualquier objeto que se añada al lienzo debe ser un objeto que pueda ser dibujado, por lo que **debe implementar** la interfaz **Dibujable**. Al lienzo se le pueden añadir figuras, obtenerlas y borrarlas de éste. El programador que utilice el lienzo solo tiene que tener en cuenta que si obtiene una referencia de una figura mediante un método get y modifica los el aspecto de ésta, los cambios no se mostrarán en el lienzo hasta que necesite ser repintado o hasta que se le indique explícitamente que debe ser repintado mediante el método repaint().

Otro objeto útil para el aspecto gráfico del Paint es el **RendererEstiloLinea.** Este objeto nos permite mostrar imágenes en un comboBox de tal forma que

podemos mostrar los distintos tipos de líneas de forma visual al usuario. Este objeto hereda de un JLabel (que será el que contendrá la imagen a mostrar en cada celda del comboBox) e implementa la interfaz ListCellRenderer que es la que nos permite alterar el comboBox.

Por último tenemos la clase **Paint** que contiene uno o más objetos de todos los anteriores y los distribuye junto con botones y otros elementos sobre él mismo ya que hereda de JPanel. En la parte central contiene un objeto de tipo Lienzo que ocupa la mayor parte del panel, a la derecha se encuentra una barra de herramientas que puede ser arrastrada por el usuario y que contiene todas las posibles transformaciones sobre las figuras dibujadas (botones para rotar, trasladar, escalar, cambiar el grosor, el tipo de trazo y de letra etc.). En la parte inferior se encuentran dos barras de herramientas independientes, la barra de colores y la barra de figuras. En la barra de colores se sitúan dos objetos de tipo PaletaColores, una para los colores preestablecidos y otros para los colores personalizados, junto con las opciones de relleno y degradado. En la otra barra de figuras se encuentran botones de las distintas figuras que pueden ser dibujadas en la aplicación, éstos botones son de tipo JToggleButton por lo que se quedan pulsados indicando la figura seleccionada. Estos botones forman parte de un ButtonGroup que no permite que haya más de un botón pulsado al mismo tiempo.

A continuación se muestra un diagrama de clases en el que se indican las dependencias entre éstas. No se han incluido la herencia de las clases respecto a las clases JPanel, JFrame y JDialog para facilitar la lectura de éste. También se han ocultado las variables miembro de las clases que representan la interfaz gráfica para no mostrar los botones, paneles, etiquetas etc. Las conexiones entre las clases acabadas en rombo indican que la clase que posee la punta del rombo incluye una variable miembro del mismo tipo de la otra clase (el nombre de esta variable viene en una etiqueta sobre la conexion), por ejemplo el objeto EditorMultimedia contiene un objeto de tipo Paint llamado PanelDibujo.


editor_multimedia
editor_multimedia::MainMultimedia
+ packFrame=false : boolean
+ MainMultimedia()
+ main(args : String[])
Paint
+ Paint()
+ jbInit()
+ CargarFuentes()
+ GuardarArchivo() : boolean
+ NuevoDibujo() : boolean
+ bEditaColor_actionPerformed(e : ActionEvent)
+ AgregaColorAdicional(color : Color)
+ lienzo_mouseDragged(e : MouseEvent)
+ lienzo_mousePressed(e : MouseEvent)
+ lienzo_mouseReleased(e : MouseEvent)
+ BotonesDibujo_actionPerformed(actionEvent : ActionEvent)
+ bColorFondo_actionPerformed(e : ActionEvent)
+ sGrosorTrazo_stateChanged(e : ChangeEvent)
+ cEstiloTrazo_actionPerformed(e : ActionEvent)
+ cEstiloFinalTrazo_actionPerformed(e : ActionEvent)
+ cEstiloUnionTrazo_actionPerformed(e : ActionEvent)
+ paletasColor_mousePressed(e : MouseEvent)
+ bDegradado_actionPerformed(e : ActionEvent)
+ cDegradado_actionPerformed(e : ActionEvent)
+ Fuente_actionPerformed(e : ActionEvent)
+ getLienzo()
+ this_keyTyped(e : KeyEvent)
+ bRotarD_actionPerformed(e : ActionEvent)
+ bRotarI_actionPerformed(e : ActionEvent)
+ sEscalar_stateChanged(e : ChangeEvent)
+ sRotar_stateChanged(e : ChangeEvent)
+ lienzo_mouseEntered(e : MouseEvent)
+ lienzo_mouseExited(e : MouseEvent)
+ lienzo_mouseMoved(e : MouseEvent)
+ lienzo_ancestorResized(e : HierarchyEvent)
PaletaColores
- nColores=0 : int
- nFilas=1 : short
- nColumnas=0 : short
- colores=null : Color[]
- etiqColores=null : JLabel[]
- tamEtiquetas=newDimension(15 : Dimension
- glayout=newGridLayout(nFilas : GridLayout
+ PaletaColores()
+ PaletaColores(colores : int, tamano : Dimension)
+ PaletaColores(colores : int, filas : short, tamano : Dimension)
+ getColumnas() : short
+ getFilas() : short
+ getNColores() : int
+ getColores() : Color[]
+ getTamano() : Dimension
+ getColor(fila : short, columna : short) : Color
+ getColor(posicioncolor : int) : Color
+ setColor(posicioncolor : int, color : Color) : boolean
+ setColores(colores : Color[], filas : short) : boolean
+ setFilas(filas : short) : boolean
+ setTamanoColor(tamano : Dimension) : boolean
+ setColor(color : Color, fila : short, columna : short) : boolean
+ setTamColores(tamano : Dimension) : boolean
editor_multimedia::EditorMultimedia
+ EditorMultimedia()
+ jbInit()
+ bNuevo_actionPerformed(e : ActionEvent)
+ bAbrir_actionPerformed(e : ActionEvent)
+ bGuardar_actionPerformed(e : ActionEvent)
+ mSalir_actionPerformed(actionEvent : ActionEvent)
+ mAcercaDe_actionPerformed(actionEvent : ActionEvent)
+ TipoAplicacion_actionPerformed(e : ActionEvent)
+ PanelDibujo_componentHidden(e : ComponentEvent)
+ PanelImagen_componentHidden(e : ComponentEvent)
+ mAbrir_actionPerformed(e : ActionEvent)
+ mGuardar_actionPerformed(e : ActionEvent)
+ mVideoSonido_actionPerformed(e : ActionEvent)
+ mDibujo_actionPerformed(e : ActionEvent)
+ mNuevo_actionPerformed(e : ActionEvent)
+ mImagen_actionPerformed(e : ActionEvent)
+ mCerrar_actionPerformed(e : ActionEvent)
+ bAyuda_actionPerformed(e : ActionEvent)
+ mEstadisticas_actionPerformed(e : ActionEvent)
+ bCerrar_actionPerformed(e : ActionEvent)
+ this_windowClosing(e : WindowEvent)
paleta, paletaAdicional
+PanelDibujo
+PanelImagen
+PanelVideoSonido
Lienzo
- figuras : Vector
- tamanoLienzo : Dimension
+ Lienzo()
+ Lienzo(anchura : int, altura : int)
+ Lienzo(v : Vector)
+ Lienzo(anchura : int, altura : int, v : Vector)
+ paint(g : Graphics)
+ addFigura(figura :) : boolean
+ getFigura(posicion : int)
+ getFigura(p : Point2D)
+ getLastFigura()
+ getFirstFigura()
+ deleteFigura(figura :) : boolean
+ deleteFigura(posicion : int) : boolean
+ setFiguras(v : Vector) : boolean
+ BorraLienzo()
+ isVacio() : boolean
+ getNFiguras() : int
+lienzo=new Lienzo(600,600)
+lienzo=newLienzo(150
+dialogoDegradado
+p
DialogoDegradado
+ DialogoDegradado(owner : Frame, title : String, modal : boolean, p : Paint)
+ jbInit()
+ opcionColor1_actionPerformed(e : ActionEvent)
+ opcionColor2_actionPerformed(e : ActionEvent)
+ Opciones_actionPerformed(e : ActionEvent)
+ bCancelar_actionPerformed(e : ActionEvent)
+ bAceptar_actionPerformed(e : ActionEvent)
EditorImagen
+ EditorImagen()
+ jbInit()
+ CargarImagen() : boolean
+ CerrarImagen() : boolean
+ GuardarImagen() : boolean
+ sBrillo_stateChanged(e : ChangeEvent)
+ bReestablecer_actionPerformed(e : ActionEvent)
+ Transformaciones_actionPerformed(e : ActionEvent)
ReproductorVideoSonido
+ ReproductorVideoSonido()
+ jbInit()
+ reproducir()
+ detener()
+ controllerUpdate(event : ControllerEvent)
+ CerrarReproduccion()
+ AbrirReproduccion() : boolean
+ TareaPeriodica()
+PanelTapiz=newTapiz(200,200)
Tapiz
+ imagenCargada=null : BufferedImage
+ Tapiz(anchura : int, altura : int)
+ Tapiz(imagen : BufferedImage)
+ paint(g : Graphics)
+ getImagen() : BufferedImage
+ setImagen(imagen : BufferedImage)

2.3 Imagen.

Para crear esta sección de la aplicación solo hemos hecho uso de **dos clases**:

1. La primera es **EditorImagen** que se trata de un panel sobre el cual se visualizan las imágenes sus transformaciones y las posibles opciones y filtros a aplicar, se puede decir que es la totalidad de esta sección. Posee en la parte central de éste un objeto de tipo Tapiz que es el que se encarga de visualizar la imagen en todo momento, en la parte de abajo se encuentra tanto barra que permite modificar el brillo de la imagen como unos paneles tabulados con los distintos filtros que se pueden aplicar a la imagen. Dentro de esta clase se encuentran definidas las constantes que indican los distintos tipos de filtros que se pueden aplicar a la imagen.

2. La segunda es la clase **Tapiz** que como hemos dicho antes se encarga de dibujar una imagen de tipo BufferedImage sobre ella. Esta clase también se encarga de repintar la imagen cuando sea necesario debido a la superposición de otras ventanas u objetos.

2.4 Video y Sonido.

Para crear esta sección de la aplicación solo se ha utilizado una clase que se trata de **ReproductorVideoSonido**, es un panel que se encarga de mostrarnos la reproducción en curso junto con todas las opciones y e información de la reproducción. En la parte central se sitúa la zona en la que serán visualizados los elementos multimedia reproducidos, en la parte derecha se sitúan un panel de información sobre la reproducción del elemento multimedia y en la parte inferior se encuentran la barra de reproducción y los botones para controlarla. Dentro de esta clase se encuentra un objeto **Player** que es el encargado de la reproducción en sí. Este objeto player pertenece a la Java Media Framework que es la que contiene las clases necesarias para la reproducción de archivos de video y audio. Las barras de control de la reproducción y el panel de reproducción nos los devuelve como components el propio objeto player por lo que sólo tenemos que situarlos dentro del JPanel. La clase ReproductorVideoSonido también contiene un objeto Timer que actuará de temporizador para poder actualizar cada cierto tiempo las estadísticas de reproducción que serán mostradas en el panel de la derecha.

3. Codificación.

La estructuración e información de la codificación se puede observar en la documentación generada por javadoc ya que nos indica la distribución de las clases, tipos de datos, restricciones de los objetos y explicaciones sobre el uso de éstos.

A continuación se listarán los detalles más importantes en la implementación de las distintas clases:

- La clase PaletaColores se ha implementado mediante un Jpanel al cual se le añaden un jLabel y se distribuyen mediante un GridLayout, además cada etiqueta es dibujada con un color diferente por medio de su color de Background, estos colores son obtenidos intentando distribuir uniformemente todos los colores de la paleta para que aparezcan repetidos (se controla en la clase Paint ya que la clase PaletaColores no se encarga de asignar los colores). La paleta de color es de tamaño variable y se encarga de situar correctamente el número de etiquetas incluso si el número de filas y columnas es incorrecto (crea una paleta de color cuadrada aunque las últimas posiciones de la última fila no se completen del todo).

- También se ha añadido al paint otra paleta adicional para los colores personalizados recientemente usados y así poder seleccionarlos con posterioridad.

- Otro detalle de la implementación es que no se incluido un JColorChooser dentro de la clase Paint (aunque parezca que de este modo no se necesite una paleta de colores personalizados porque ya la trae incluida) por que no funciona del todo bien ya que el JColorChooser solo añade a su paleta los colores que se eligen en la primera pestaña y no en las demás.

- Utilizo el uso de una variable modificado en las secciones de dibujo e imagen por la cual se detecta si el dibujo actualmente ha sido modificado y puede haber pérdida de datos o no. Por ejemplo si va a cargar una imagen nueva le pregunta, en caso de que el archivo actual esté modificado, si desea guardar o no, pero si el archivo no está modificado realiza la carga sin pedir confirmación. Igualmente ocurre cuando se sale de la aplicación o se desea cambiar de sección.

- El motivo de crear todas las clases geométricas y no utilizar las de java es que los objetos Shape no se pueden pintar si tienen alturas o anchuras negativas aunque sí que te dejan almacenarlas sin ningún error, esto es debido a que consideran que el Shape está vacío y no lo dibujan. Además así en clases como rectanguloDibujable siempre se conocen las esquinas de forma concreta, aunque el propio objeto que se está dibujando cambie de forma o posición.

- Para mostrar los dibujos de las líneas que posee el comboBox he tenido que crear un objeto Renderer que permite a cada celda asignarle una imagen y así mejorar el uso y la comprensión del programa.

- Para detectar en el apartado de dibujo qué botón se ha pulsado para el tipo de figura a dibujar no he buscado entre todos para ver si está pulsado, sino que lo obtengo automáticamente por su índice de posición en el panel.

- Otro detalle a tener en cuenta es la forma de creación de figuras antes de añadirlas al Lienzo, ya que se le pasa al constructor un nuevo objeto "copia" del contexto gráfico para que cada figura posea su propio contexto gráfico, de no ser así al cambiar una propiedad en una figura se modificarían las propiedades de todas las figuras que tuvieran ese mismo contexto gráfico.

- El objeto poligonoDibujable posee siempre una línea menos ya que la última línea no se incluye porque sirve de cierre de éste. En esta clase también ocurre que el método mover está redefinido, sobrescribiendo así la implementación del padre. Esto es debido a que no se ha implementado el método setLimites, por lo tanto no funcionaría el método mover del padre porque depende de éste método.

- Una pequeña excepción en el significado de límites de las figuras está en las clases curva1Dibujable y curva2Dibujable, ya que límites no indica los límites de la curva circunscrita sino el rectángulo que circunscribe a la recta que une los dos extremos de la curva.

- En la parte del dibujo cuando se crean transformaciones sobre los objetos hay que tener cuidado ya que el nº de grados que se rota la figura no es el nº de grados absolutos desde la posición de origen sino el incremento desde la posición actual (se concatenan). Lo mismo ocurre con las translaciones y los cambios de tamaño. Se podría solucionar si se realizaran dos transformaciones desde la clase Paint(), una para volverlo al estado inicial y otra para el nuevo valor de la propiedad que se ha modificado en la interfaz.

- Para la selección de una figura para ser editada se tiene que buscar la figura geométrica en el Lienzo, esta figura debe contener el punto en el que se ha realizado el evento del click (la búsqueda se realiza empezando por las figuras anteriormente dibujadas, es decir, por las primeras que fueron añadidas). Para detectar si el punto pertenece a la figura compruebo si el punto está contenido dentro de los límites de ésta. El problema es que el método contains() de Rectangle de los límites no funciona si posee la anchura o altura negativos porque lo considera vacío. Esto es solucionado

con el método getLimitesAbsolutos de la clase Geometria dibujable que devuelve los mismos límites pero con anchura y altura positivos.

- El método mover de la clase GeometríaDibujable está pensado para mover la figura completamente cambiando sus variables de posición mientras que trasladar solo cambia la forma de mostrar la figura en pantalla al igual que rotar y escalar.

4. Extras.

A la aplicación se le han añadido algunos extras para mejorar su aspecto, eficiencia, manejabilidad y funcionalidad. Estos extras son comentados y explicados a continuación.

- El dibujo con pincel posee estilo de línea (grosor, tipo de línea, estilo final etc.), lo mismo ocurre con la goma excepto en el tipo de línea que no se le puede definir un patrón de línea aunque sí el estilo final, grosor, etc.
- He simplificado el código que genera jBuilder para el manejo de eventos para aquellos elementos en los que se pueden unificar el método que los gestiona.
- Muestro en la medida de lo posible cualquier error producido por medio de mensajes al usuario indicándole que es lo que ha ocurrido en la aplicación.
- Guardo los archivos con la extensión correcta aunque el usuario no la incluya explícitamente en el diálogo de guardado (para ello escribo la extensión que esté seleccionada en el filtro de tipo de archivos). Compruebo también que no se sobrescribe un archivo al guardar en cuyo caso pregunto si sobrescribir.
- Los extremos del degradado los sitúo a un ¼ de distancia de los extremos del lienzo para que se pueda apreciar en todo el lienzo y no solo en la esquina superior izquierda.
- Uno de los extras más importantes es el hecho de que se pueda **editar las figuras geométricas; posición, color, tamaño, estilo de línea, relleno etc. Incluso se pueden eliminar una vez seleccionadas pulsando el botón SUPR.** Para mostrar que una figura ha sido seleccionada se refleja un efecto en la pantalla por el cual se identifica la figura seleccionada claramente.
- Muestro una barra de cargado en la sección de sonido y video. También añado estadísticas sobre el estado de la reproducción con un panel adicional que se puede ocultar si se desea.
- He mejorado el aspecto gráfico de la aplicación mediante el uso de iconos, imágenes y punteros.
- Hago más eficiente la aplicación repintando solamente (en la medida de lo posible) los elementos que son modificados (por medio de sus límites).
- Se controla la pérdida de datos si se carga algún archivo o se desea salir d e la aplicación
- Añado un panel en la sección de dibujo sobre la barra de estado que indica la posición del cursor dentro del Lienzo.
- Puedo reestablecer los cambios en la sección imagen volviendo otra vez a la imagen original.
- Mejoro la eficiencia en el repintado de figuras con degradado reduciendo la calidad mediante los Renderinthits.

- La sección de imagen es capaz de cargar más tipos de imágenes ya que utiliza la clase imageIO para después ser pasada a un BufferedImage.

5. Futura mejoras.

La aplicación todavía se puede mejorar en muchos aspectos. Muchos cambios no se han realizado por falta de tiempo pero podrían tenerse en cuenta en un futuro para la mejora de la aplicación. Las posibles mejoras podrían ser:

1. Mejorar la eficiencia en general revisando el código y creando métodos más eficientes.
2. Manejar los errores mediante excepciones y intentar corregirlos antes de mostrar el mensaje de error y cerrar la aplicación.
3. Mejorar el degradado ya que se podría realizar de forma local solo dentro de la figura (por medio de sus límites) y no por medio del Lienzo.
4. Crear una paleta de colores en la cual se distribuyan uniformemente todos los tipos de colores pudiendo el usuario variar el tamaño de ésta. También se podrían crear paletas predefinidas.
5. Incrementar el tipo de archivos a modificar y a guardar.
6. Incluir la opción guardar proyecto en la sección Dibujo para que el usuario pueda continuar con el trabajo después de cerrar la aplicación. Esto se podría conseguir serializando los objetos y variables.
7. Incluir autoformas a dibujar.
8. Añadir botones de deshacer y rehacer en la aplicación.

• • • • • • •