Tema 13

Gráficas en Java

APl de Java para Gráficas

En la actualidad, se usan intensivamente las gráficas e imágenes en computación tanto en la interfaz del usuario con la computadora, como un medio para que el usuario le suministre información a un programa o para que el programa despliegue los resultados. Lo anterior ha motivado la cración de bibliotecas especializadas en el manejo de gráficas e imágenes para casi cualquier lenguaje de programación, incluyendo Java que posee un conjunto de clases llamadas la Interfaz de Programación de Aplicaciones 2D de Java (la API 2D de Java). Esas clases se encuentran en los siguientes paquetes:

- `java.awt`
- `java.awt.image`
- `java.awt.color`
- `java.awt.font`
- `java.awt.geom`
- `java.awt.print`
- `java.awtiimage.renderable`
- `com.sun.image.codec.jpeg`

Todos los paquetes a excepción del último forman parte del nucleo de la plataforma de Java, por lo que estan disponibles en todas las implementaciones de java 2. La API de Java 2D nos permite hacer, entre otras cosas, lo siguiente:

- Representar figuras geométricas arbitrarias usando combinaciones de líneas y curvas. También hay un conjunto de herramientas para construir figuras estándar como rectángulos, arcos, elipses, etc.
- Dibujar contornos a figuras usando lineas sólidas o punteadas.
- Rellenar figuras usando colores sólidos, patrones, gradientes de color, etc.
- Estirar, comprimir, rotar figuras, imágenes o texto.
- Agregarle nuevos elementos a un dibulo existente.
- Limitar o recortar la extención de los dibujos.
- Suavizar las orillas de las imágenes quitando los bordes zizagueados.
- Usar las fuentes TrueType y type 1 instaladas en la computadora y manipular cadenas de caracteres en la misma forma en que se manipulan a las figuras.
- Representar colores independientemente del hardware.

- Manipular imágenes en la misma forma en que se manipulan a las figuras.
- Procesar imágenes para resaltar ciertas características o para limpiar imágenes.
- Imprimir (enviar la salida a una impresora).

La Clase `Graphics2D`

La clase `Graphics2D` es la clase medular de la API 2D de Java. Es la máquina de presentación, encargada de desplegar las primitivas gráficas (figuras, texto o imágenes) en los dispositivos de salida (pantalla o impresora). La máquina de presentación se encarga de tomar una colección de primitivas y de decidir de que color serán los pixeles en un dispositivo de salida.

Adicionalmente, la clase `Graphics2D` representa una superficie de dibujo, que es simplemente una colección de pixeles, cada uno de un color. Esa superficie puede estar dentro de una ventana o ser una página en una impresora o una imagen aún no desplegada.

Para desplegar figuras, texto o imágenes se requiere de un objeto del tipo `Graphics2D`. Ese objeto no se puede crear usando el operador `new`. En lugar de ello debemos obtenerlo de los objetos sobre los cuales se pueden desplegar figuras, texto o imágenes: Objetos de las subclases de `Component`.

La clase `Component`, Figura 13.1, tiene el método `paint(graphics g)` que recibe del sistema un objeto del tipo `Graphics2D` a través del parámetro `g`. En la subclase de en la que deseamos dibujar debemos sobreescribir su método `paint()`. Para tener acceso a los métodos de `Graphics2` hay que hacer una conversión explícita a `Graphics2`:

```
public void paint(Graphics g) {
   Graphics2D g2 = (Graphics2D)g;

   // Usamos g2 para desplegar figuras, texto o imágenes.
}
```

Si el componente sobre el que deseamos dibujar pertenece al paquete swing, debemos utilizar el método `paintComponent(graphics g)` que trabaja en forma similar al método `paint()`.

Una forma alternativa para obtener el objeto `Graphics2` es usando el método `Graphic getGraphics()` de la clase `Component`.

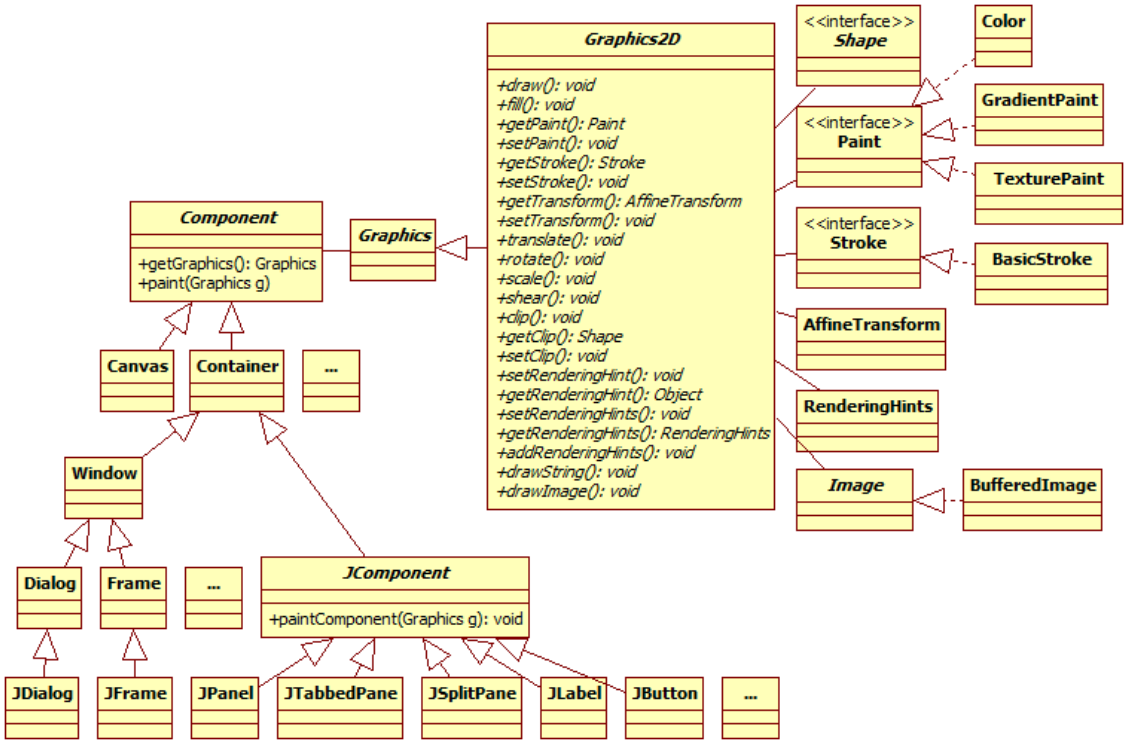

Figura 13.1 Clases sobre las que se puede dibujar

Procedimiento de Presentación

La figura 13.2 muestra el procedimiento empleado por la clase `Graphics2D` para decidir como las primitivas gráficas se convertirán en colores de pixeles. Este procedimiento consta de cinco pasos:

1. Determine la figura a ser presentada. Es diferente para cada operación de representación:

   - Si la figura va a rellenarse, la figura se transforma (mueve, estira, comprime, rota) usando la transformación actual. Todas las primitivas son transformadas antes de presentarse. La transformación convierte las primitivas del espacio del usuario al espacio del dispositivo. Por ausencia, la transformación mapea 72 coordenadas del usuario a una pulgada del dispositivo de salida.
   - Para las figuras cuyo contorno es dibujado por el método draw(), se usa el trazo (stroke) actual para convertir el contorno en una figura. Luego el contorno se transforma de igual forma que una figura rellena.

- El texto se despliega traduciendo los caracteres a figuras usando la fuente actual. Las figuras resultantes se transforman de igual forma que una figura rellena.
- En las imágenes, el contorno de la imagen se transforma usando la transformación actual.

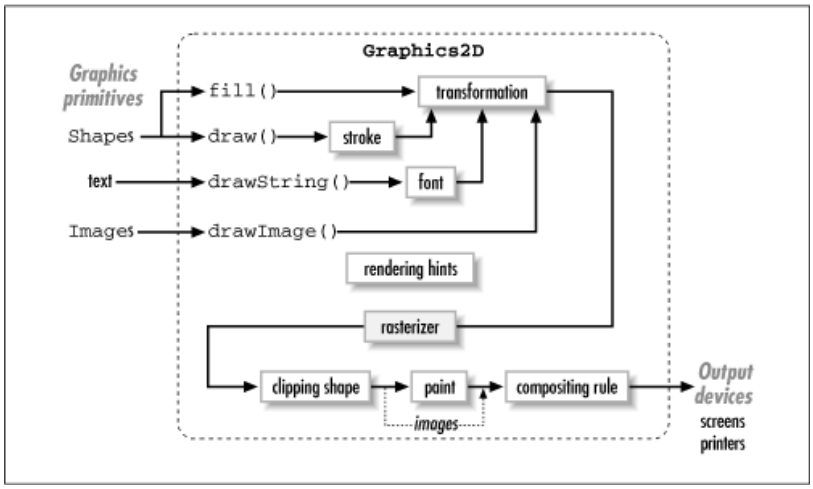

Figura 13.2 Procedimiento de Presentación

Como puede verse, la máquina de presentación sólo sabe rellenar figuras y dibujar imágenes. En realidad dibujar contornos de figuras y dibujar texto son casos especiales de rellenar figuras.

2. Entramar (rasterize) la figura. Es el proceso de convertir una figura ideal a un conjunto de valores de pixeles. En el caso de luna imagen, el contorno de la imagen es entramado. Para controlar el comportamiento del entramado, se emplean las sugerencias de Presentación (Rendering hints) que son un conjunto técnicas para desplegar primitivas.

3. Recorta los resultados usando la figura de recorte (clipping) actual. Sólo los pixeles dentro de la figura de recorte serán desplegados.

4. Determine los colores a usar. Para una figura rellena utilice el objeto `paint` actual para determinar los colores a usar para rellenar la figura. Para una imagen los colores provienen de la misma imagen.

5. Combine los colores con el dibujo existente, usando la regla de composición actual.

Valores Alfa

El proceso de despliegue es una aproximación. Al pedirle a la máquina de despliegue que rellene una figura, ésta determina cómo el dispositivo de salida debe colorearse para aproximar en lo mejor posible a la figura.

Efecto dentado y Suavizado de Bordes

Por ejemplo supongamos que le pedimos que llene a la figura con un color. Hay dos formas de hacerlo: la forma fácil y la forma correcta. En la forma fácil, sólo los pixeles que caen dentro de la figura son coloreados. Usando este algoritmo los pixeles son coloreados o dejados sin cambio. En la figura 13.3 (izquierda) se muestra un ejemplo de esta técnica. El contorno ideal también se muestra. La figura rellena muestra un efecto dentado (aliasing) no muy agradable.

Figura 13.3. Efecto Dentado (izquierda) y Suavizado de bordes (derecha)

La forma correcta de rellenar la figura involucra un poco más de trabajo. La idea básica es calcular la intersección de la figura con cada pixel del dispositivo de salida y colorearlo en proporción a la cantidad en que son cubiertos por la figura. Esto reduce el efecto dentado, figura 13.3 (derecha). En este ejemplo los pixeles en los bordes no solo son negros o blancos sino tienen varios tonos de grises. esta téctica se conoce como suavizamiento de bordes (antialiasing). La máquina de presentación se encarga del suavisado si se especifica en las sugerencias de presentación.

Entramado

El entramador toma una figura ideal y produce un valor de cobertura para cada pixel. Un valor de cobertura representa que tanto del pixel está cubierto por la figura. Ese valor se conoce como valor Alfa. Un pixel está definido por un color y su valor alfa. El valor alfo indica la transparencia del pixel. Se puede pensar que el valor alfa es parte del color. Los colores a veces se definen con un valor alfa asociado. En estos casos el valor alfa indica la transparencia del color.

Los valores típicos de los valores alfa van de 0.0 para no cobertura hasta 1.0 para cobertura completa. En la figura 13.3, el entramador no uso suavizado de bordes y produjo valores alfa de 0.0 o 1.0.

En la figura 13.3 /derecha) se usó suavizado de bordes y el entramador produjo valores alfa desde 0.0 en el exterior de la figura hasta 1.0 en el interior de la figura. En la figura 13.4 muestra los valores alfa de cada pixel en la esquina superior izquierda de la misma figura. Se sobreimpuso una cuadrícula por claridad.

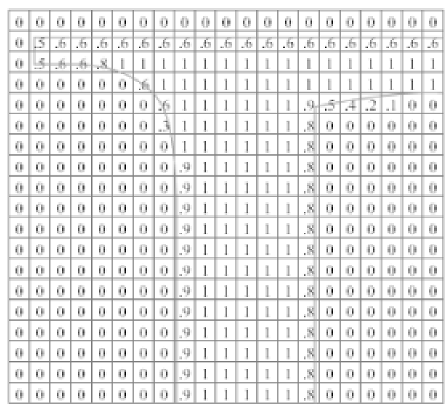

Figura 13.4. Valores Alfa Producidos por el Entramador

Composición

Despues de determinar los valores alfa para cada pixel de la figura, se usa una técnica llamada composición para determinar cómo esos valores alfa se traducen en información de color. para determinar cómo los colores de una nueva primitiva gráfica se combinan con los colores existentes en la superficie en la que se está dibujando, como se muestra en la figura 13.5.

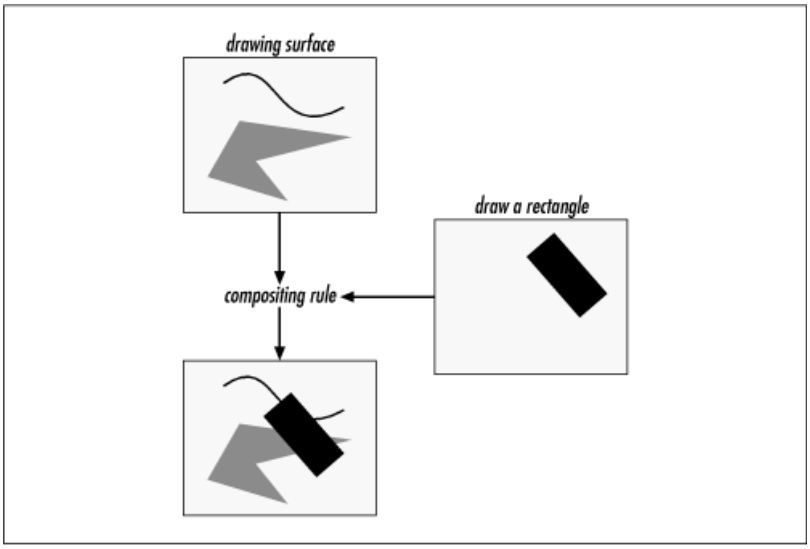

Figura 13.5 Composición

Espacio Coordenado

Los objetos 2D de Java viven en un plano definido por coordenadas Cartesianas. Este plano se llama el Espacio Coordenado del Usuario o simplemente Espacio del Usuario. Cuando los objetos se dibujan en la pantalla o en una impresora, las coordenadas del Espacio del Usuario se transforman en coordenadas del Espacio del Dispositivo. El Espacio del Dispositivo corresponde a un monitor o impresora en particular. Por lo general, una unidad en el Espacio del Dispositivo corresponde a un pixel de un dispositivo. Normalmente el Espacio del Usuario y el Espacio del Dispositivo están alineados con los ejes x y y como se muestran en la figura 13.6.

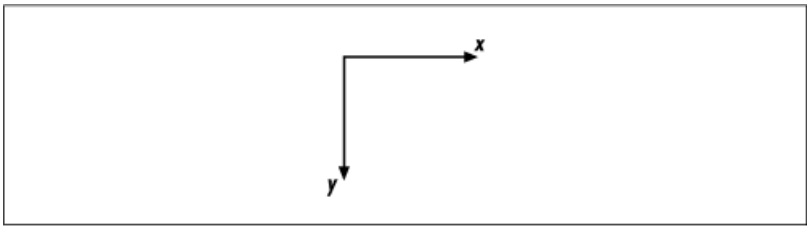

Figura 13.6. Sistema Coordenado del Espacio del Dispositivo

El eje x se incrementa de izquierda a derecha y el eje y de arriba hacia abajo. El origen se coloca en la izquierda superior derecha de la superficie de dibujo. Esto aplica para cualquier dispositivo.

Aunque el Espacio del Usuario y el Espacio del Dispositivo están alineados por ausencia, debe haber un cierto escalamiento para asegurarse que los objetos se dibujan del mismo tamaño sin importar el dispositivo de salida. el Espacio del Dispositivo está determinado por la resulución del dispositivo en particular. Un monitor, por ejemplo, por lo general tiene 72 pixeles por pulgada, mientras que una impresora laser por lo general tiene 300 0 600 pixeles por pulgada.

El Espacio del Usuario se convierte al Espacio del Dispositivo cuando se dibujan los objetos. Se utiliza una transformación para convertir de un sistema a otro. La conversión por ausencia convierte 72 coordenadas del Espacio del Usuario a una pulgada física.

Dibujo de Figuras

En esta sección se verá como se puede desplegar el primer tipo de las primitivas gráficas: las figuras. La clase Graphics2D tiene el método `draw()` que nos permite dibujar figuras. Su sintaxis se muestra en la Tabla 13.1.

Tabla 13.1 Método para Dibujar Figuras de la Clase Graphics2D

`public abstract void draw(Shape s)` Despliega el contorno de la figura del parámetro en el contexto actual de `Graphics2D`

La API 2D de Java permite la presentación de cualquier figura mediante una combinación de segmentos de linea rectos y curvos. Estos elementos de gráficas están contenidos en las interfaces, clases abstractas y clases de la figura 13.7.

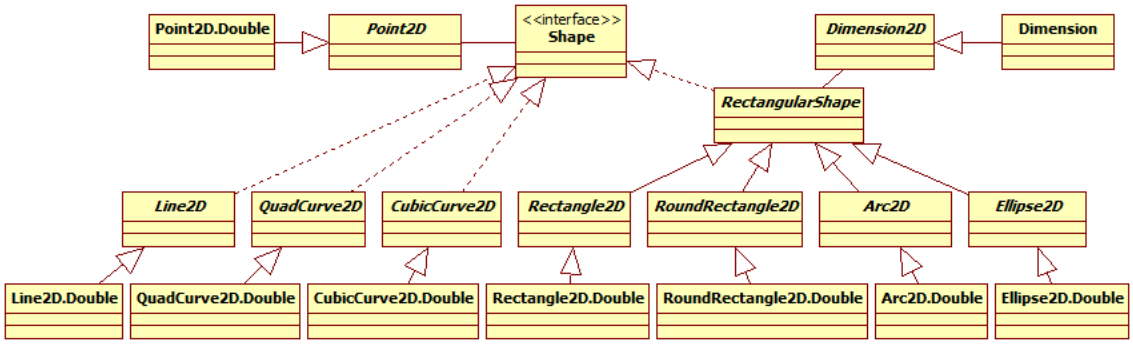

Figura 13.7 Clases que Representan Figuras

Puntos

Las clases `Punto2D` y `Punto2D.Double` encapsulan un solo punto (x, y) en el Espacio del Usuario. La clase `Punto2D` es una clase abstracta mientras que `Punto2D.Double` es una de sus subclases que implementa los métodos abstractos de `Punto2D` y está definida como una clase interna de `Punto2D`. El diagrama de clases de ésas clases se muestra en la figura 13.8.

`Punto2D` es la clase más básica de la API 2D de Java y se usa extensamente a lo largo de la API. Un punto no es un pixel. Un punto no tiene una superficie y por lo tanto no puede presentarse. Los puntos se usan para construir figuras que que si tienen superficie y por lo tanto sí pueden presentarse. La tabla 13.2 muestra los métodos de la clase `Punto2D` y la tabla 13.3 muestra los métodos de la clase clase `Punto2D.Double`.

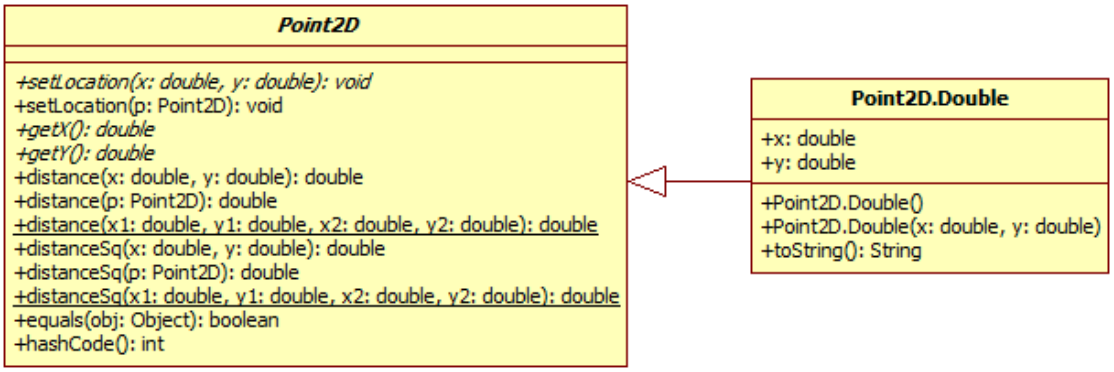

Figura 13.8 Clases que Representan un Punto

Tabla 13.2 Métodos de la Clase `Punto2D`

`public abstract void` **`setLocation`**`(double x, double y)`
`public void` **`setLocation`**`(Point2D pt)`
 	
Establecen la posición de este punto a las coordenadas de sus parámetros.

`public abstract double` **`getX`**`()`
`public abstract double` **`getY`**`()`
 	
Regresan las coordenadas X o Y de este punto, respectivamente.

`public double` **`distance`**`(double px, double py)`
`public double` **`distance`**`(Point2D pt)`
 	
Regresan la distancia de este punto al punto con las coordenadas de sus parámetros.

`public static double` **`distance`**`(double x1, double y1, double x2, double y2)` Regresa la distancia entre los dos puntos cuyas coordenadas están dadas por sus parámetros.

`public double` **`distanceSq`**`(double px, double py)`
`public double` **`distanceSq`**`(Point2D pt)`
 	
Regresan el cuadrado de la distancia de este punto al punto con las coordenadas de sus parámetros.

`public static double` **`distanceSq`**`(double x1, double y1,`
`double x2, double y2)`
 	
Regresa el cuadrado de la distancia entre los dos puntos cuyas coordenadas están dadas por sus parámetros.

Tabla 13.3 Métodos de la Clase `Punto2D.Double`

`public boolean` **`equals`**`(Object obj)` Regresa verdadero si este punto es igual al punto de su parámetro. Dos puntos son iguales si sus correspondientes coordenadas son iguales.
`public int` **`hashCode`**`()` Regresa el código hash para este punto.

Tabla 13.3 Métodos de la Clase `Punto2D.Double`. Cont.

`public` **`Point2D.Double`**`()` Construye e inicializa un objeto del tipo **`Point2D`** con coordenadas (0.0, 0.0).
`public` **`Point2D.Double`**`(double x, double y)` Construye e inicializa un objeto del tipo **`Point2D`** con las coordenadas dadas por sus parámetros.
`public String` **`toString`**`()` Regresa un objeto de tipo `String` que representa el valor de este punto.

Dimensiones

La clase `Dimension2D` es una clase abstracta que encapsula el ancho y el alto de una dimensión como dobles. La clase `Dimension` encapsula el ancho y el alto de una componente como enteros. El diagrama de clases de ésas clases se muestra en la figura 13.9.

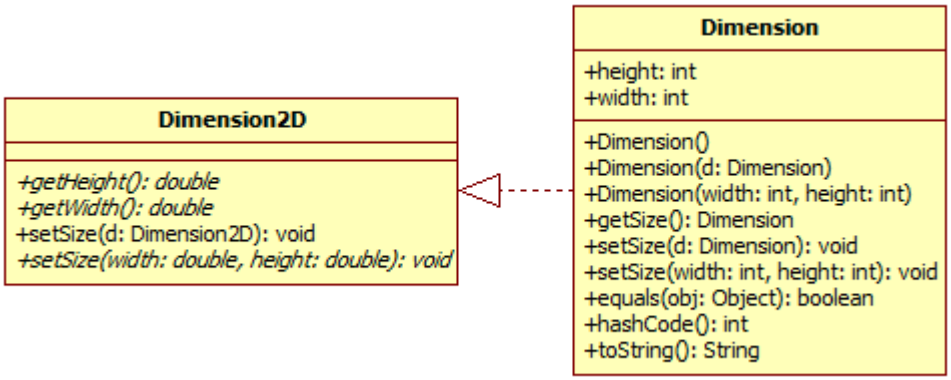

Figura 13.9 Clases que Representan una Dimensión

La tabla 13.4 muestra los métodos de la clase `Dimension2D`.

Tabla 13.4 Métodos de la Clase `Dimension2D`

`public abstract double` **`getHeight`**`()` Regresa la altura de esta dimensión.
`public abstract double` **`getWidth`**`()` Regresa el ancho de esta dimensión.
`public void` **`setSize`**`(Dimension2D d)` Establece el tamaño de esta dimensión al valor de su parámetro.
`public abstract void` **`setSize`**`(double width, double height)` Establece el tamaño de esta dimensión al valor de sus parámetros.

La tabla 13.5 muestra los métodos de la clase `Dimension`.

Tabla 13.5 Métodos de la Clase `Dimension`

`public` **`Dimension`**`()` Crea una instancia de `Dimension` con altura = 0 y ancho = 0.

`public` **`Dimension`**`(Dimension d)`
`public` **`Dimension`**`(int width, int height)`
 	
Crean una instancia de `Dimension` con altura y ancho dados por el parámetro.

`public` **`Dimension getSize`**`()` Obtiene el tamaño de este objeto.
`public void` **`setSize`**`(Dimension d)` Establece el tamaño de esta dimensión al tamaño de su parámetro.
`public void` **`setSize`**`(int width, int height)` Establece el tamaño de esta dimensión al ancho y alto de sus parámetros.
`public boolean` **`equals`**`(Object obj)` Determina si esta dimensión es la misma que la dimensión de su parámetro.
`public int` **`hashCode`**`()` Regresa el código Hash de esta dimensión.
`public String` **`toString`**`()` Regresa una cadena con la representación de esta dimensión.

La Interfaz Shape

Esta interfaz `Shape`, figura 13.10 establece los métodos que deben implementar todas las clases que representen una figura geométrica. La descripción de esos métodos se encuentra en la tabla 13.6.

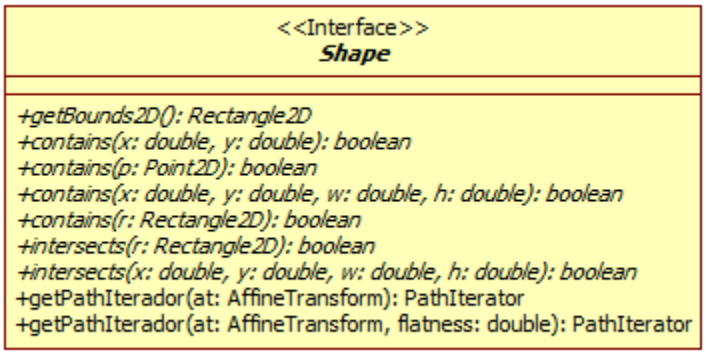

Figura 13.10. Interfaz Shape

Tabla 13.6 Métodos de la Interfaz Shape.

`Rectangle2D` **`getBounds2D`**`()` Regresa un rectángulo que encierra completamente a esta figura.
`boolean` **`contains`**`(double x, double y)` Regresa verdadero si las coordenadas de sus parámetros estan dentro de esta figura.
`boolean` **`contains`**`(Point2D p)` Regresa verdadero si el punto del parámetro cae dentro de esta figura.
`boolean` **`contains`**`(double x, double y, double w, double h)` Regresa verdadero si el área rectangular dada por las coordenadas de sus parámetros está dentro de esta figura.
`boolean` **`contains`**`(Rectangle2D r)` Regresa verdadero si el rectángulo dado por el parámetro está dentro de esta figura.
`boolean` **`intersects`**`(double x, double y, double w, double h)` Regresa verdadero si el interior de esta figura intersecta el interior del área rectangular dada por las coordenadas de sus parámetros. El área rectangular intersecta a esta figura si hay puntos que esten tanto en el interior de la figura y el área especificada.
`boolean` **`intersects`**`(Rectangle2D r)` Regresa verdadero si el interior de esta figura intersecta el interior del rectángulo dado por el parámetro.
`PathIterator` **`getPathIterator`**`(AffineTransform at)` Regresa un iterador que itera a lo largo de la frontera de la figura y provee acceso a la geometría de la orilla de la figura. Si el parámetro `at` no es `null`, las coordenadas regresadas en la iteración se transforman usando la transformación dada por el parámetro.
`PathIterator` **`getPathIterator`**`(AffineTransform at, double flatness)` Regresa un iterador que itera a lo largo de la frontera de la figura y provee acceso a una vista plana de la geometría de la orilla de la figura. Si el parámetro `at` no es `null`, las coordenadas regresadas en la iteración se transforman usando la transformación dada por el parámetro.

Líneas Rectas

La clase abstracta `Line2D` es la superclase de todas las clases que almacenan un segmento de línea 2D en el espacio coordenado (x, y). La clase `Line2D.Double` es una de sus subclases que implementa los métodos abstractos y está definida como una clase interna. El diagrama de clases de ésas clases se muestra en la figura 13.11. La tabla 13.7 muestra los métodos de la clase abstracta `Line2D`.

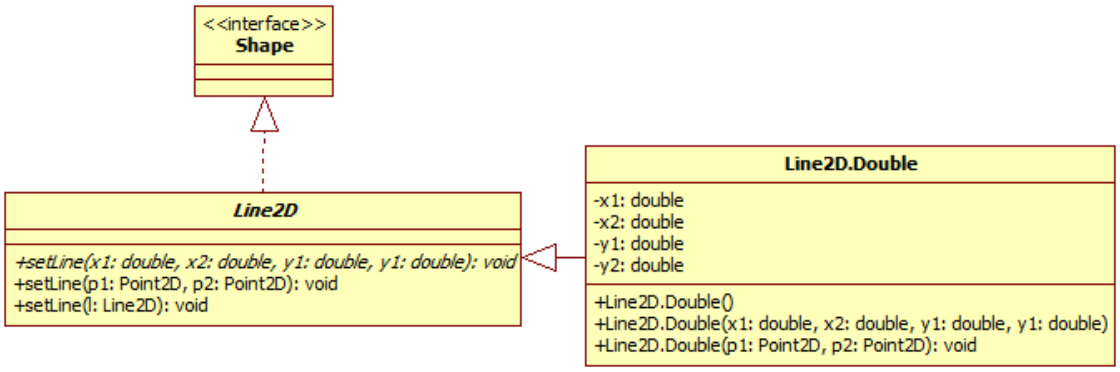

Figura 13.11 Clases que Representan una Línea

Tabla 13.7 Métodos de la Clase `Line2D`

`public abstract void` **`setLine`**`(double x1, double y1, double x2, double y2)` Establese la posición de los extremos de esta recta a las coordenadas de sus parámetros.
`public void` **`setLine`**`(Point2D p1, Point2D p2)` Establese la posición de los extremos de esta recta a las coordenadas de los puntos de sus parámetros.
`public void` **`setLine`**`(Line2D l)` Establese la posición de los extremos de esta recta a las coordenadas de los extremos de la recta de su parámetro.

La tabla 13.8 muestra los métodos de la clase `Line2D.Double`.

Tabla 13.8 Métodos de la Clase `Line2D.Double`.

`public` **`Line2D.Double`**`()` Construye e inicializa una línea con extremos en las coordenadas (0, 0) y (0, 0).
`public` **`Line2D.Double`**`(double x1, double y1, double x2, double y2)` Construye e inicializa una línea con extremos en las coordenadas de sus parámetros.
`public` **`Line2D.Double`**`(Point2D p1, Point2D p2)` Construye e inicializa una línea con extremos en los puntos de sus parámetros.

Ejemplo sobre Líneas

Para ilustrar cómo se despliegan las diferentes figuras en Java 2D, se implementa una aplicación con una interfaz de usuario gráfica en la que ventana principal se tienen dos paneles. El panel superior se utilizará como lienzo para desplegar las figuras, en el inferior hay dos botones: uno para dibujar la figura y otro para borrarla, Figura 13.12. Las opciones de la barra de menú permiten seleccionar el tipo de figura, texto u operación con imágenes a mostrar, Figura 13.13.

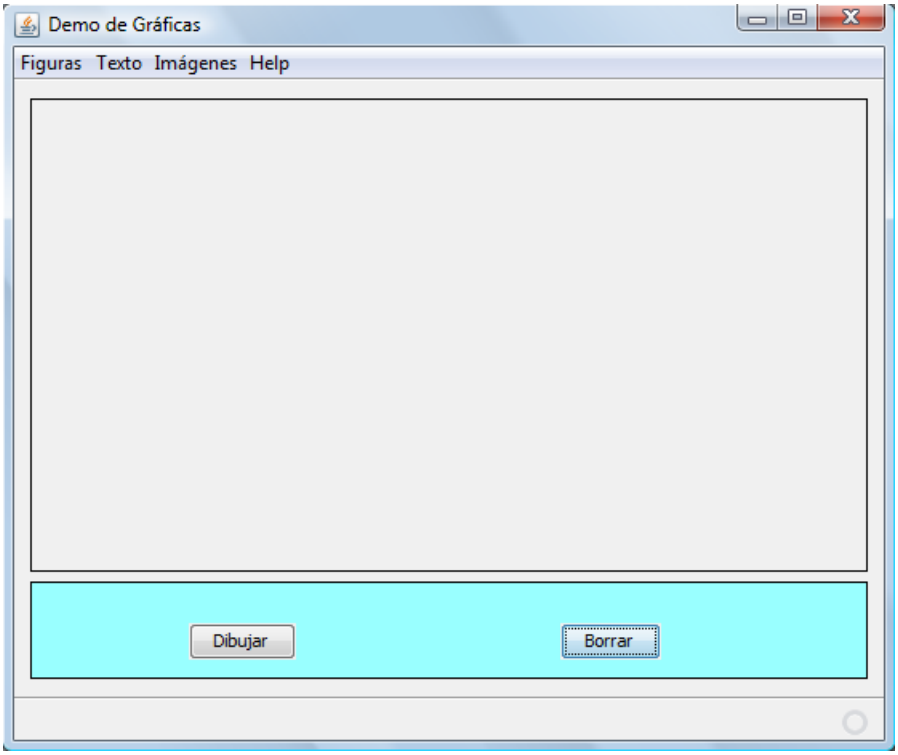

Figura 13.12.

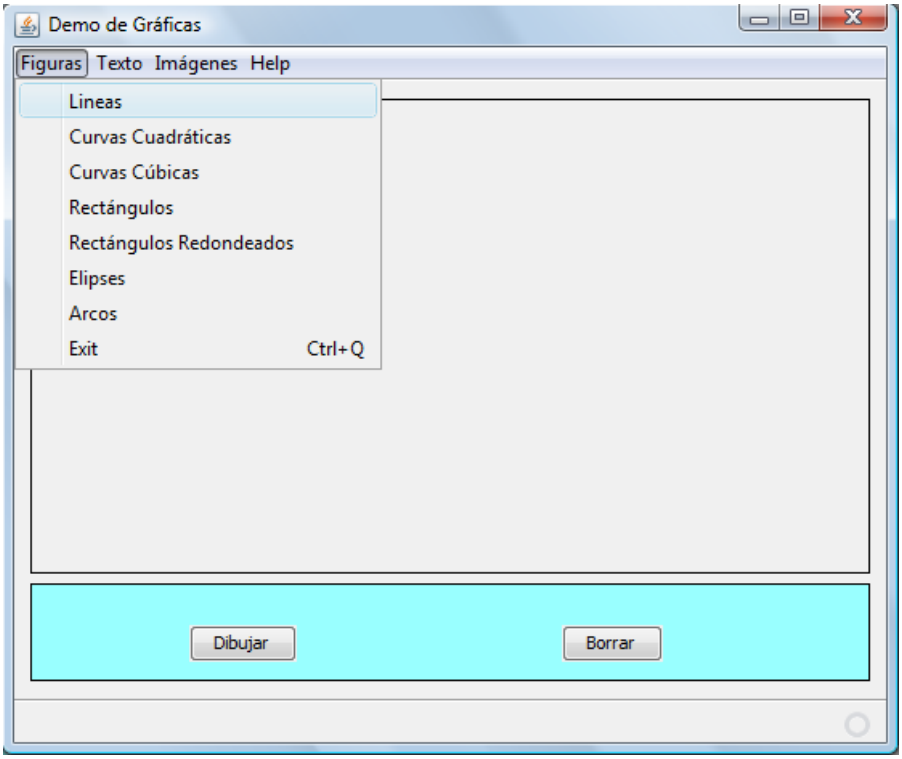

Figura 13.13

El siguiente fragmento de código muestra la porción de la clase de la ventana principal en la que se muestran las declaraciones de los paneles y los métodos oyentes de los eventos de hacer clic en el botón **Borrar** y seleccionar la opción **Lineas**.

Grafica2.View.java

```
/*
 * Grafica2View.java
 */
package grafica2;

import figuras.Operaciones;
import figuras.Trayectorias;
import figuras.Lineas;
import figuras.Rectangulos;
import org.jdesktop.application.Action;
import org.jdesktop.application.ResourceMap;
import org.jdesktop.application.SingleFrameApplication;
import org.jdesktop.application.FrameView;
import org.jdesktop.application.TaskMonitor;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Timer;
import javax.swing.Icon;
import javax.swing.JDialog;
import javax.swing.JFrame;

/**
 * Ventana de la aplicación Grafica2.
 */
public class Grafica2View extends FrameView {

    public Grafica2View(SingleFrameApplication app) {
        super(app);
        initComponents();
        ...
    }

    private void initComponents() {
        mainPanel = new javax.swing.JPanel();
        panelLienzo = new javax.swing.JPanel();
        ...
    }

    /**
     * Metodo oyente del boton Dibujar. Invoca al metodo que realiza la
     * tarea de dibujar
     * @param evt Evento de hacer clic sobre el boton
```

```
     */
    private void botonDibujarActionPerformed(
        java.awt.event.ActionEvent evt) {
        Operaciones oper;

        switch(operacion) {
            case BORRA: Lineas.borra(panelLienzo);
                        break;
            case LINEA: Lineas.dibujaLineas(panelLienzo);
                        break;
            case CUADRATICA: Lineas.dibujaCurvasCuadraticas(panelLienzo);
                        break;
            case CUBICA: Lineas.dibujaCurvasCubicas(panelLienzo);
                        break;
            case RECTANGULO: Rectangulos.dibujaRectangulos(panelLienzo);
                        break;
            case RECTANGULO_REDONDEADO:
                        Rectangulos.dibujaRectangulosRedondeados(
                           panelLienzo);
                        break;
            case ELIPSE: Rectangulos.dibujaElipses(panelLienzo);
                        break;
            case ARCO: Rectangulos.dibujaArcos(panelLienzo);
                        break;
            case TRAYECTORIA:
                        Trayectorias.dibujaTrayectoria(panelLienzo);
                        break;
            case COMBINA: Composicion.combinaFiguras(panelLienzo);
                        break;
            case RELLENA: oper = new Operaciones();
                        oper.rellenaFiguras(panelLienzo);
                        break;
            case CONTORNO: oper = new Operaciones();
                        oper.contornoFiguras(panelLienzo);
                        break;
            case TRANSLADA: Operaciones.TransladaFiguras(panelLienzo);
                        break;
            case ROTA: Operaciones.RotaFiguras(panelLienzo);
                        break;
            case ESCALA: Operaciones.escalaFiguras(panelLienzo);
                        break;
            case DEFORMA: Operaciones.deformaFiguras(panelLienzo);
                        break;
            case RECORTA: oper = new Operaciones();
                        oper.recortaFiguras(panelLienzo);
                        break;
            case DIBUJA_TEXTO: Texto.dibujaTexto(panelLienzo);
                        break;
            case ROTA_TEXTO: Texto.rotaTexto(panelLienzo);
```

```
                        break;
            }
        }

        /**
         * Metodo oyente del boton Borrar. Borra el contenido del panel
         * panelLienzo
         * @param evt Evento de hacer clic sobre el boton
         */
        private void botonBorrarActionPerformed(
            java.awt.event.ActionEvent evt) {
            Lineas.borra(panelLienzo);
        }

        /**
         * Metodo oyente de la opcion de menu Figuras. Dibuja varias lineas
         * en el panel panelLienzo
         * @param evt Evento de hacer clic sobre la opcion de menu
         */
        private void opcionLineasActionPerformed(
            java.awt.event.ActionEvent evt) {
            operacion = operaciones.LINEA;
        }

        /**
         * Metodo oyente de la opcion de menu Figuras. Dibuja varias curvas
         * cuadraticas en el panel panelLienzo
         * @param evt Evento de hacer clic sobre la opcion de menu
         */
        private void opcionCurvaCuadraticaActionPerformed(
            java.awt.event.ActionEvent evt) {
            operacion = operaciones.CUADRATICA;
        }

        /**
         * Metodo oyente de la opcion de menu Figuras. Dibuja varias curvas
         * cubicas en el panel panelLienzo
         * @param evt Evento de hacer clic sobre la opcion de menu
         */
        private void opcionCurvaCubicaActionPerformed(
            java.awt.event.ActionEvent evt) {
            operacion = operaciones.CUBICA;
        }

        /**
         * Metodo oyente de la opcion de menu Figuras. Dibuja varios
         * rectangulos en el panel panelLienzo
         * @param evt Evento de hacer clic sobre la opcion de menu
```

```
     */
    private void opcionRectanguloActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.RECTANGULO;
    }

    /**
     * Metodo oyente de la opcion de menu Figuras. Dibuja varios
     * rectangulos redondeados en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionRectanguloRedondeadoActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.RECTANGULO_REDONDEADO;
    }

    /**
     * Metodo oyente de la opcion de menu Figuras. Dibuja varias elipses
     * redondeados en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionElipseActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.ELIPSE;
    }

    /**
     * Metodo oyente de la opcion de menu Figuras. Dibuja varios arcos
     * en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionArcoActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.ARCO;
    }

    /**
     * Metodo oyente de la opcion de menu Figuras. Dibuja una trayectoria
     * arbitraria en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionTrayectoriasActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.TRAYECTORIA;
    }

    /**
     * Metodo oyente de la opcion de menu Operacioness. Dibuja diferentes
```

```
     * combinaciones de areas en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionCombinarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.COMBINA;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * figuras con relleno en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionRellenarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.RELLENA;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * tipos de contorno en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionContornoActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.CONTORNO;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * translaciones en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionTransladarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.TRANSLADA;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * rotaciones en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionRotarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.ROTA;
    }
```

```
    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * escalamientos en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionEscalarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.ESCALA;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * deformaciones en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionDeformarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.DEFORMA;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja diferentes
     * figuras recortadas en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionRecortarActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.RECORTA;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Dibuja texto
     * en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionDibujaTextoActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.DIBUJA_TEXTO;
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Rota texto
     * en el panel panelLienzo
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionRotaTextoActionPerformed(
        java.awt.event.ActionEvent evt) {
        operacion = operaciones.ROTA_TEXTO;
```

```
    }

    /**
     * Metodo oyente de la opcion de menu Operaciones. Permite
     * seleccionar una fuente de las fuentes disponibles en el sistema
     * @param evt Evento de hacer clic sobre la opcion de menu
     */
    private void opcionFuentesActionPerformed(
        java.awt.event.ActionEvent evt) {
        Texto texto = new Texto();

        texto.fuentes(this.getFrame());
    }

    private javax.swing.JPanel panelControl;
    private javax.swing.JPanel panelLienzo;
    private enum operaciones {BORRA, LINEA, CUADRATICA, CUBICA,
                              RECTANGULO, RECTANGULOREDONDEADO,
                              ELIPSE, ARCO , TRAYECTORIA, COMBINA,
                              RELLENA, CONTORNO, TRANSLADA, ROTA,
                              ESCALA, DEFORMA, RECORTA , DIBUJA_TEXTO,
                              ROTA_TEXTO };

    private operaciones operacion = operaciones.BORRA;
}
```

Note que para dibujar las lineas se invoca al método `dibujaLineas()` y se le pasa como parámetro una referencia al panel `panelLienzo` para que el método dibuje sobre él. El código del método `dibujaLineas()` es el siguiente:

Lineas.java

```
package figuras;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.geom.Line2D;
import javax.swing.JPanel;

/**
 * Clase con métodos estáticos para dibujar lineas
 * @author mdomitsu
 */
public class Lineas {
    /**
```

```
     * Este metodo borra el contenido del panel de su parametro.
     * @param lienzo Panel a borrar
     */
    public static void borra(JPanel lienzo) {
        // Obtiene un objeto de tipo Graphics del panelLienzo
        Graphics g = lienzo.getGraphics();

       // Al invocar al metodo paint se borra su contenido
        lienzo.paint(g);
    }

    /**
     * Este metodo dibuja sobre el panel de su parametro
     * un conjunto de lineas
     * @param lienzo Panel sobre el que se dibujan las lineas
     */
    public static void dibujaLineas(JPanel lienzo) {
        // Obtiene un objeto de tipo Graphics2D del panelLienzo
        Graphics2D g2 = (Graphics2D) lienzo.getGraphics();
        int numeroLineas = 25;
        Color[] colores = {Color.red, Color.green, Color.blue};

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        for (int i = 0; i < numeroLineas; i++) {
            // Establece el color de la linea
            g2.setPaint(colores[i % colores.length]);

            double radio = (double) i / numeroLineas;
            double iradio = 1 - radio;

            // Dibuja una linea en la esquina inferior izquierda
            Line2D linea1 = new Line2D.Double(0, radio * d.height,
                                              radio * d.width, d.height);
            g2.draw(linea1);

            // Dibuja una linea en la esquina superior derecha
            Line2D linea2 = new Line2D.Double(d.width, iradio * d.height,
                                              iradio * d.width, 0);
            g2.draw(linea2);
        }
    }
    ...
}
```

La figura 13.14 muestra las líneas desplegadas por el método `dibujaLineas()`.

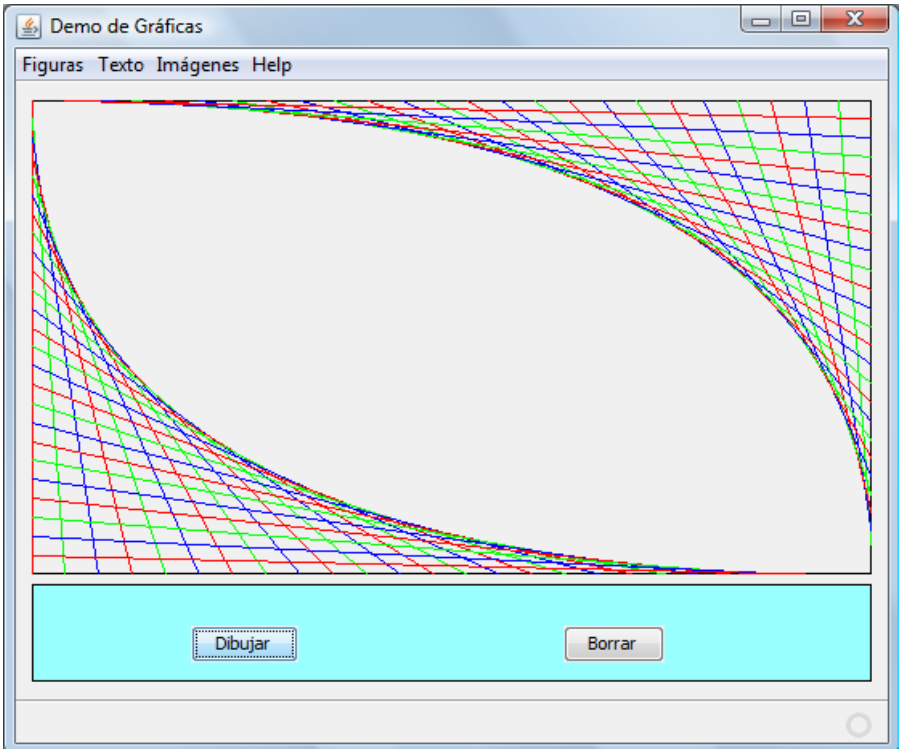

Figura 13.14. Líneas

Curvas Cuadráticas

Una curva cuadrática es una línea curva representada por una ecuación cuadrática (segundo órden). Esta curva se describe completamente por dos puntos en los extremos de la curva y un punto de control, fuera de la curva, que determina las tangentes de la curva en sus puntos extremos, figura 2.15.

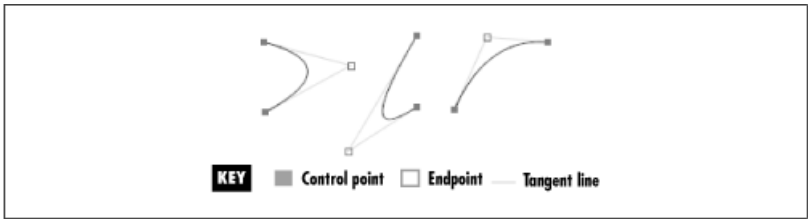

Figura 13.15. Puntos extremos, Punto de Control y Líneas tangentes de Curvas Cuadráticas.

La clase abstracta `QuadCurve2D` es la superclase de todas las clases que almacenan una curva cuadrática 2D en el espacio coordenado (x, y). La clase `QuadCurve2D.Double` es una de sus subclases que implementa los métodos abstractos y está definida como una clase interna. El diagrama de clases de ésas clases se muestra en la figura 13.16.

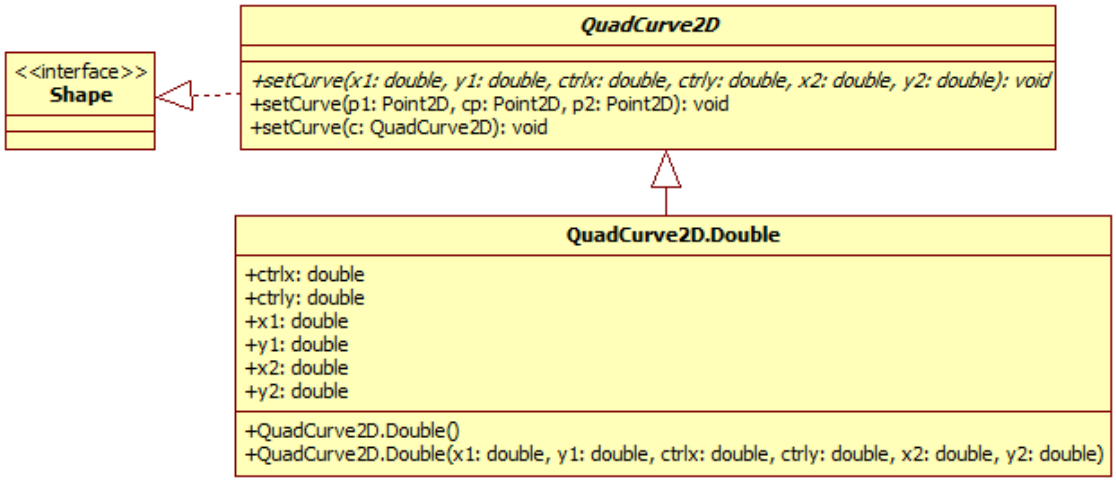

Figura 13.16 Clases que Representan una Curva Cuadrática

La tabla 13.9 muestra los métodos de la clase abstracta `QuadCurve2D`.

Tabla 13.9 Métodos de la Clase `QuadCurve2D`

`public abstract void` **`setCurve`**`(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)`
 	
`Establece la posición de los puntos de control de esta curva cuadrática a las coordenadas de sus parámetros.`

`public void` **`setCurve`**`(Point2D p1, Point2D cp, Point2D p2)` Establece la posición de los puntos de control de esta curva cuadrática a los puntos de sus parámetros.
`public void` **`setCurve`**`(QuadCurve2D c)` Establece la posición de los puntos de control de esta curva cuadrática a los mismos puntos de la curva cuadrática de su parámetro.

La tabla 13.10 muestra los métodos de la clase clase `QuadCurve2D.Double`.

Tabla 13.10 Métodos de la Clase `QuadCurve2D.Double`.

`public` **`QuadCurve2D.Double`**`()` Construye e inicializa una curva cuadrática a las coordenadas (0, 0, 0, 0, 0, 0).
`public` **`QuadCurve2D.Double`**`(double x1, double y1, double ctrlx, double ctrly, double x2, double y2)` Construye e inicializa una curva cuadrática a las coordenadas de sus parámetros.

Ejemplo sobre Curvas Cuadráticas

El código del método `dibujaCurvasCuadraticas()` dibuja varias curvas cuadráticas:

Lineas.java

```
...
import java.awt.geom.Point2D;
import java.awt.geom.QuadCurve2D;
...
public class Lineas {
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * un conjunto de curvas cuadraticas
     * @param lienzo Panel sobre el que se dibujan las curvas
     */
    public static void dibujaCurvasCuadraticas(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        QuadCurve2D q = new QuadCurve2D.Double();
        Color[] colores = {Color.red, Color.green, Color.blue};
        int numeroCurvas = 7;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        Point2D pi = new Point2D.Double(50, d.height / 2);
        Point2D pf = new Point2D.Double(d.width - 50, d.height / 2);
        Point2D pc;

        for (int i = 0; i < numeroCurvas; i++) {
            double radio = (double) i / numeroCurvas;
            double iradio = 1 - radio;

           // Dibuja el punto de control externo de la curva
            pc = new Point2D.Double(50 + (d.width - 100) * radio,
                    (d.height - 100) * iradio);
            dibujaPunto(lienzo, pc, colores[i % colores.length]);

            // Establece el color de la curva
            g2.setPaint(colores[i % colores.length]);

            // Dibuja la curva cuadrática
            q.setCurve(pi, pc, pf);
            g2.draw(q);
        }
    }

    /**
     * Dibuja un punto como un circulo
     * @param lienzo Panel sobre el que se dibuja el punto
     * @param p coordenadas del punto a dibujar
```

```
     * @param paint Color del punto a dibujar
     */
    public static void dibujaPunto(JPanel lienzo, Point2D p,
        Paint paint) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

       // Establece el color del punto
       g2.setPaint(paint);

       // Dibuja el punto como un circulo de radio 5
        Ellipse2D e = new Ellipse2D.Double(p.getX(), p.getY(), 5, 5);
        g2.fill(e);
    }
}
```

La figura 13.17 muestra las curvas cuadráticas desplegadas por el método `dibujaCurvasCuadraticas()`.

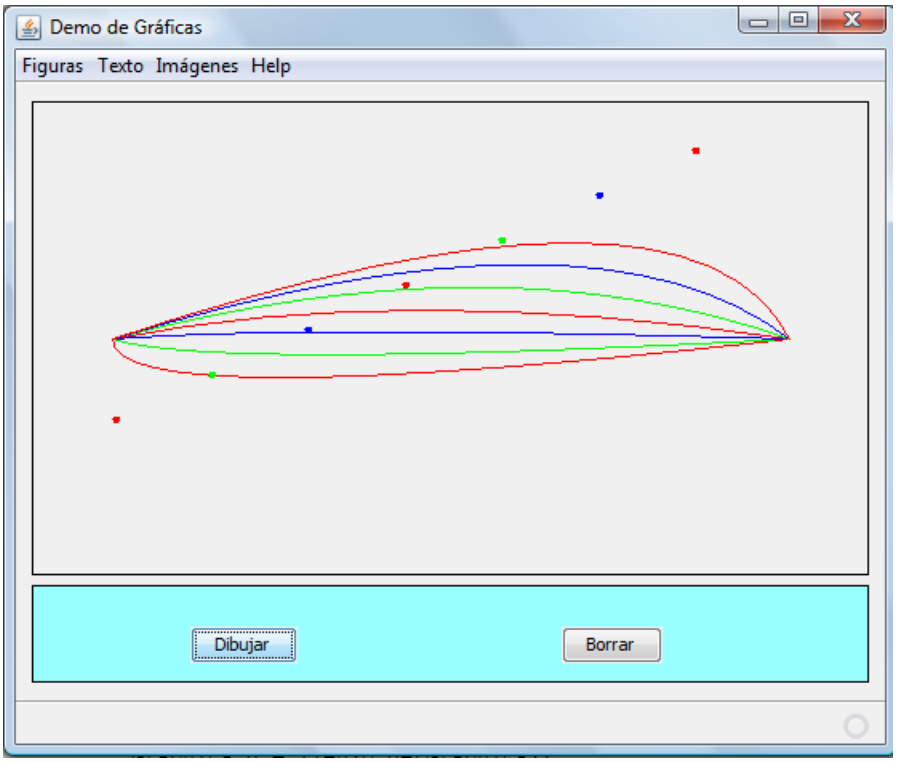

Figura 13.17. Curvas Cuadráticas

Curvas Cúbicas

Una curva cúbica es una línea curva cúbica de Bézier representada por una ecuación cúbica (tercer órden). Esta curva se describe completamente por dos

puntos en los extremos de la curva y dos puntos de control, fuera de la curva, que determinan las tangentes de la curva en sus puntos extremos, figura 2.18.

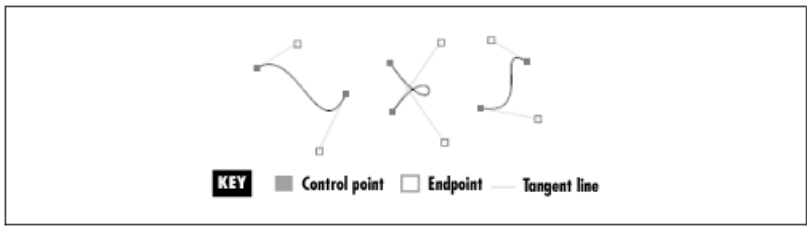

Figura 13.18. Puntos extremos, Punto de Control y Líneas tangentes de Curvas Cúbicas.

La clase abstracta `CubicCurve2D` es la superclase de todas las clases que almacenan una curva cuadrática 2D en el espacio coordenado (x, y). La clase `CubicCurve2D.Double` es una de sus subclases que implementa los métodos abstractos y está definida como una clase interna. El diagrama de clases de ésas clases se muestra en la figura 13.19.

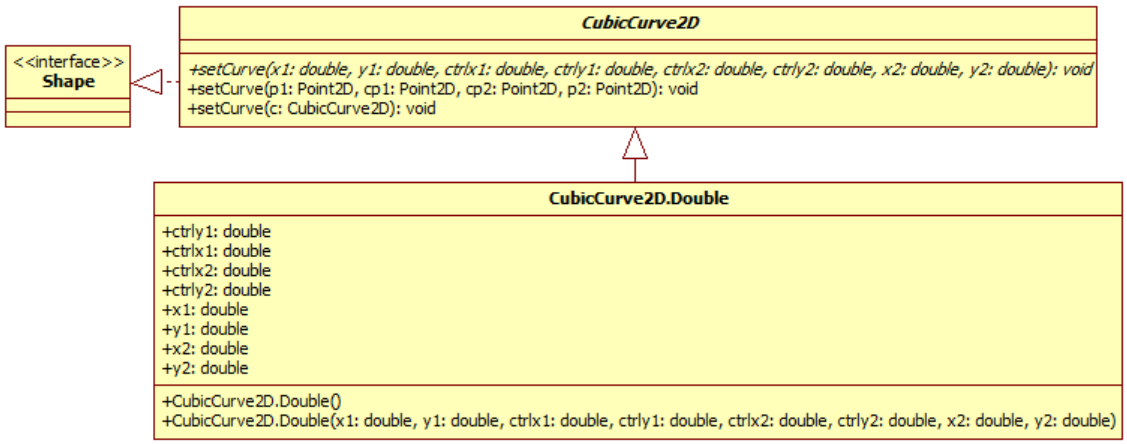

Figura 13.19 Clases que Representan una Curva Cúbica

La tabla 13.11 muestra los métodos de la clase abstracta `CubicCurve2D` y la tabla 13.12 muestra los métodos de la clase clase `CubicCurve2D.Double`.

Tabla 13.11 Métodos de la Clase `CubicCurve2D`

`public abstract void setCurve(double x1, double y1, double ctrlx1, double ctrly1, double ctrlx2, double ctrly2, double x2, double y2)` Establece la posición de los puntos de control de esta curva cúbica a las coordenadas de sus parámetros.
`public void setCurve(Point2D p1, Point2D cp1, Point2D cp2, Point2D p2)` Establece la posición de los puntos de control de esta curva cúbica a los puntos de sus parámetros.

Tabla 13.11 Métodos de la Clase `CubicCurve2D`. Cont.

`public void` **`setCurve`**`(CubicCurve2D c)` Establece la posición de los puntos de control de esta curva cúbica a los mismos puntos de la curva cúbica de su parámetro.

Tabla 13.12 Métodos de la Clase `CubicCurve2D.Double`.

`public` **`QuadCubic2D.Double`**`()` Construye e inicializa una curva cúbica a las coordenadas (0, 0, 0, 0, 0, 0, 0, 0).

`public` **`QuadCubic2D.Double`**`(double x1, double y1, double ctrlx1,`
`double ctrly1, double ctrlx2,`
`double ctrly2, double x2, double y2)`
 	
`Construye e inicializa una curva cúbica a las coordenadas dadas.`

Ejemplo sobre Curvas Cúbicas

El código del método `dibujaCurvasCubicas()` dibuja varias curvas cúbicas:

Lineas.java

```
...
import java.awt.geom.CubicCurve2D;
...
public class Lineas {
    ...
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * un conjunto de curvas cubicas
     * @param lienzo Panel sobre el que se dibujan las curvas
     */
    public static void dibujaCurvasCubicas(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        CubicCurve2D q = new CubicCurve2D.Double();
        Color[] colores1 = {Color.red, Color.green, Color.blue};
        int numeroCurvas = 5;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        Point2D pi = new Point2D.Double(50, d.height / 2);
        Point2D pf = new Point2D.Double(d.width - 50, d.height / 2);
        Point2D pc1, pc2;

        for (int i = 0; i < numeroCurvas; i++) {
            double radio = (double) i / numeroCurvas;
            double iradio = 1 - radio;

           // Dibuja el punto de control 1 externo de la curva
            pc1 = new Point2D.Double(50 + (d.width - 100) * radio / 2,
                    (d.height - 100) * iradio);
```

```
            dibujaPunto(lienzo, pc1, colores1[i % colores1.length]);

           // Dibuja el punto de control 2 externo de la curva
            pc2 = new Point2D.Double(d.width * (1 + radio) / 2,
                    50 + (d.height - 100) * radio);
            dibujaPunto(lienzo, pc2, colores1[i % colores1.length]);

            // Establece el color de la curva
            g2.setPaint(colores1[i % colores1.length]);

            // Dibuja la curva cubica
            q.setCurve(pi, pc1, pc2, pf);
            g2.draw(q);
        }
    }
    ...
}
```

La figura 13.20 muestra las curvas cúbicas desplegadas por el método `dibujaCurvasCubicas()`.

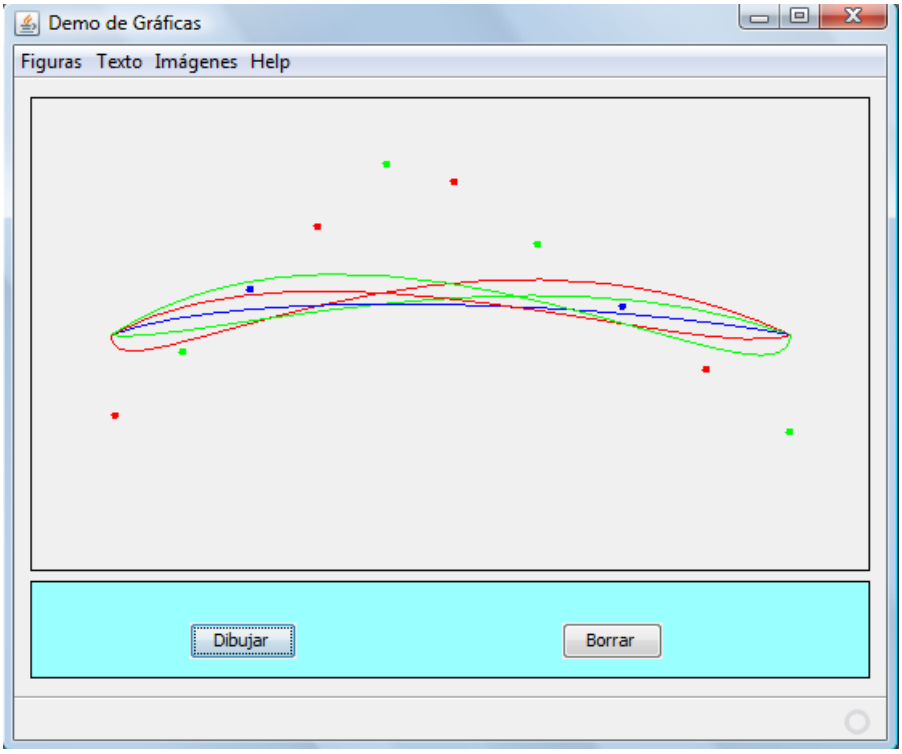

Figura 13.20. Curvas Cúbicas

La Interfaz `RectangularShape`

Esta interfaz `RectangularShape`, figura 13.21 es la clase base para un número de objetos del tipo `Shape` cuya geometría está definida por un marco rectangular.

Esta clase no establece una geometría específica sino que establece los métodos que permiten manipularlos.

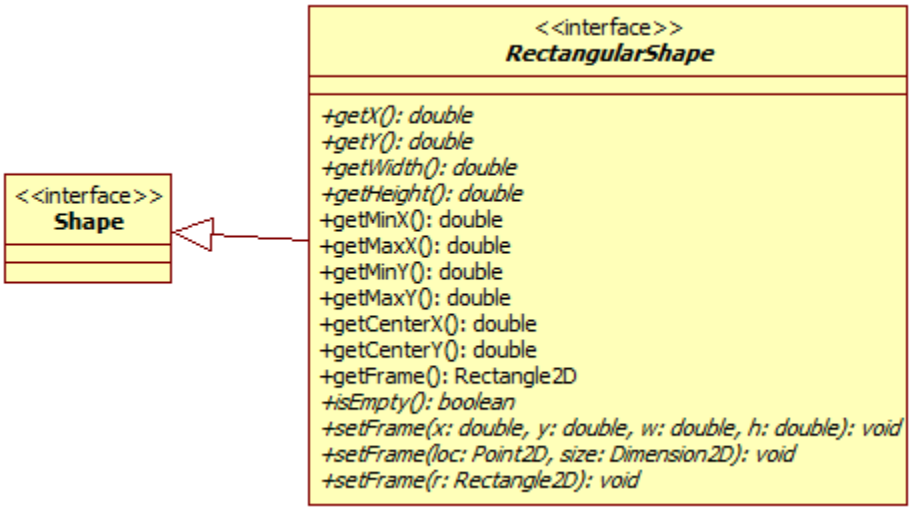

Figura 13.21. Interfaz `RectangularShape`

Los métodos de la interfaz `RectangularShape` pueden usarse para obtener o modificar el marco rectangular. La descripción de esos métodos se encuentra en la tabla 13.13.

Tabla 13.13 Métodos de la Interfaz `RectangularShape`

`public abstract double` **`getX`**`()`
`public abstract double` **`getY`**`()`
 	
Regresa las coordenadas de la esquina superior izquierda del rectángulo que enmarca está figura.

`public abstract double` **`getWidth`**`()`
`public abstract double` **`getHeight`**`()`
 	
Regresa el ancho y el alto del rectángulo que enmarca está figura.

`public double` **`getMinX`**`()`
`public double` **`getMaxX`**`()`
 	
Regresa el mínimo y el máximo valor de la coordenada X del rectángulo que enmarca está figura.

`public double` **`getMinY`**`()`
`public double` **`getMaxY`**`()`
 	
Regresa el mínimo y el máximo valor de la coordenada Y del rectángulo que enmarca está figura.

`public double` **`getCenterX`**`()`
`public double` **`getCenterY`**`()`
 	
Regresa las coordenadas del centro del rectángulo que enmarca está figura

`public Rectangle2D` **`getFrame`**`()`
 	
`Regresa el rectángulo que enmarca está figura.`

Tabla 13.13 Métodos de la Interfaz `RectangularShape`. Cont.

`public abstract boolean` **`isEmpty`**`()` Determina si la figura rectangular está vacía. Esto es, que no encierra una área.
`public void` **`setFrame`**`(double x, double y, double w, double h)` Establece la posición del rectángulo que enmarca está figura a los valores de sus parámetros.
`public void` **`setFrame`**`(Point2D loc, Dimension2D size)` Establece la posición del rectángulo que enmarca está figura a los valores de sus parámetros.
`public void` **`setFrame`**`(Rectangle2D r)` Establece la posición del rectángulo que enmarca está figura al valor de su parámetro.

Rectángulos

La clase abstracta `Rectangle2D` describe un rectángulo definido por una posición (x, y) y una dimensión (w x h). Es la superclase de todas las clases que almacenan un rectángulo 2D. La clase `Rectangle2D.Double` es una de sus implementaciones. El diagrama de clases se muestra en la figura 13.22.

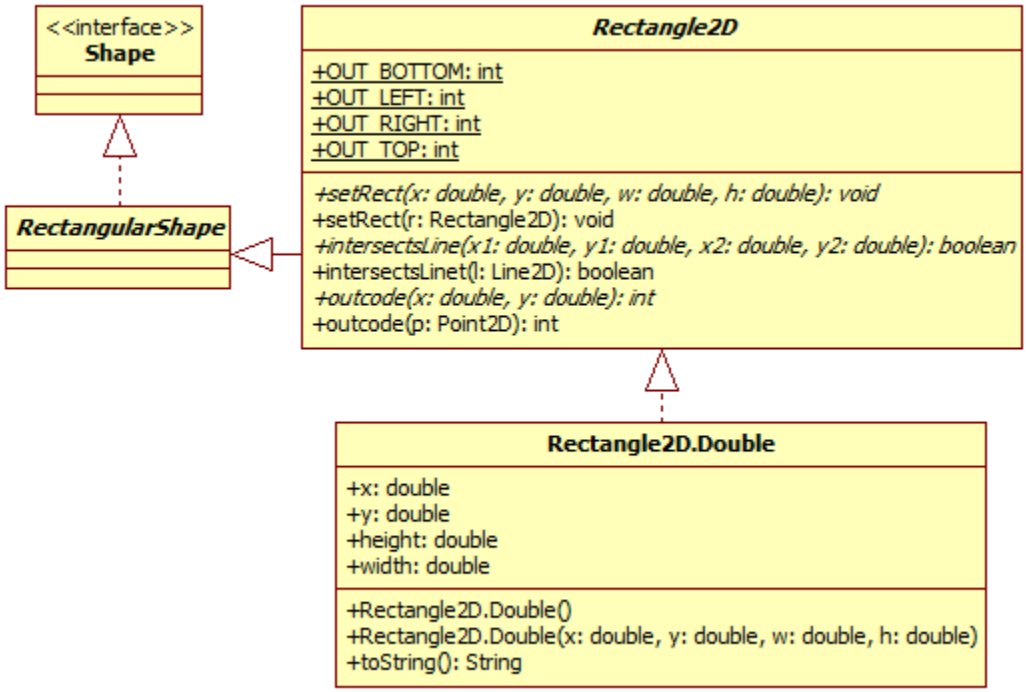

Figura 13.22 Clases que Representan un Rectángulo

La tabla 13.14 muestra los métodos de la clase abstracta `Rectangle2D`.

Tabla 13.14 Métodos de la Clase `Rectangle2D`.

`public abstract void` **`setRect`**`(double x, double y, double w, double h)`
 	
`Establece la posición de este rectángulo a los valores de sus parámetros.`

`public void` **`setRect`**`(Rectangle2D r)` Establece la posición de este rectángulo a los valores del rectángulo de su parámetro.
`public boolean` **`intersectsLine`**`(double x1, double y1, double x2, double y2)` Determina si el segmento de línea especificado por sus parámetros intersecta a este rectángulo.

`public boolean` **`intersectsLine`**`(Line2D l)`
`public int` **`outcode`**`(Point2D p)`
 	
Determina si el segmento de línea del parámetro intersecta a este rectángulo.

`public abstract int` **`outcode`**`(double x, double y)` Determina en que lugar de este rectángulo se encuentran las coordenadas del parámetro. El método puede regresar alguno de los valores mostrados en la figura 13.23, donde `OUT_LEFT, OUT_TOP, OUT_RIGHT, OUT_BOTTOM`, son constantes simbólicas definidas como atributos de esta clase.

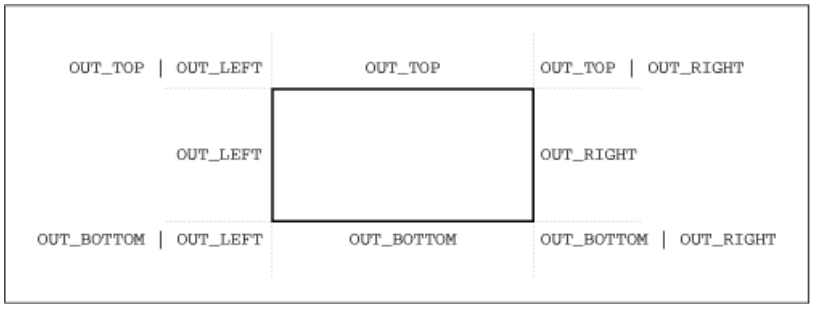

Figura 13.23 Valores Regresados por el Método `outcode()` de la Clase `Rectangle2D`

La tabla 13.15 muestra los métodos de la clase `Rectangle2D.Double`.

Tabla 13.15 Métodos de la Clase `Rectangle2D.Double`

`public` **`Rectangle2D.Double`**`()` Construye un nuevo rectángulo y lo inicializa a la posición (0, 0) y tamaño (0, 0).
`public` **`Rectangle2D.Double`**`(double x, double y, double w, double h)` Construye un nuevo rectángulo y lo inicializa a los valores de sus parámetros.
`public String` **`toString`**`()` Regresa una cadena con la representación de este rectángulo.

Ejemplo sobre Rectángulos

El código del método `dibujaRectangulos()` dibuja varios rectángulos y cuadrados:

Rectangulos.java

```
package figuras;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Rectangle2D;
import javax.swing.JPanel;

/**
 * Clase con métodos estáticos para dibujar rectangulos, rectangulos
 * redondeados, elipses y arcos
 * @author mdomitsu
 */
public class Rectangulos {
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * un conjunto de rectangulos
     * @param lienzo Panel sobre el que se dibujan los rectangulos
     */
    public static void dibujaRectangulos(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        int numeroRectangulos = 5;
        Color[] colores = {Color.red, Color.green, Color.blue};

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double x = 0;

        for (int i = 0; i < numeroRectangulos; i++) {
            // Establece el color de la rectangulo
            g2.setPaint(colores[i % colores.length]);

            double paso = d.width/numeroRectangulos;
            double alto = paso/(i+1);

            // Dibuja un rectangulo en superior
            Rectangle2D rectangulo1 = new Rectangle2D.Double(i*paso, 0,
                    paso, alto);
```

```
            g2.draw(rectangulo1);

            // Dibuja un cuadrado en la parte inferior
            Rectangle2D rectangulo2 = new Rectangle2D.Double(x,
                        d.height- alto, alto, alto);
            x += alto;
            g2.draw(rectangulo2);
        }
    }
    ...
}
```

La figura 13.24 muestra los rectángulos desplegados por el método `dibujaRectangulos()`.

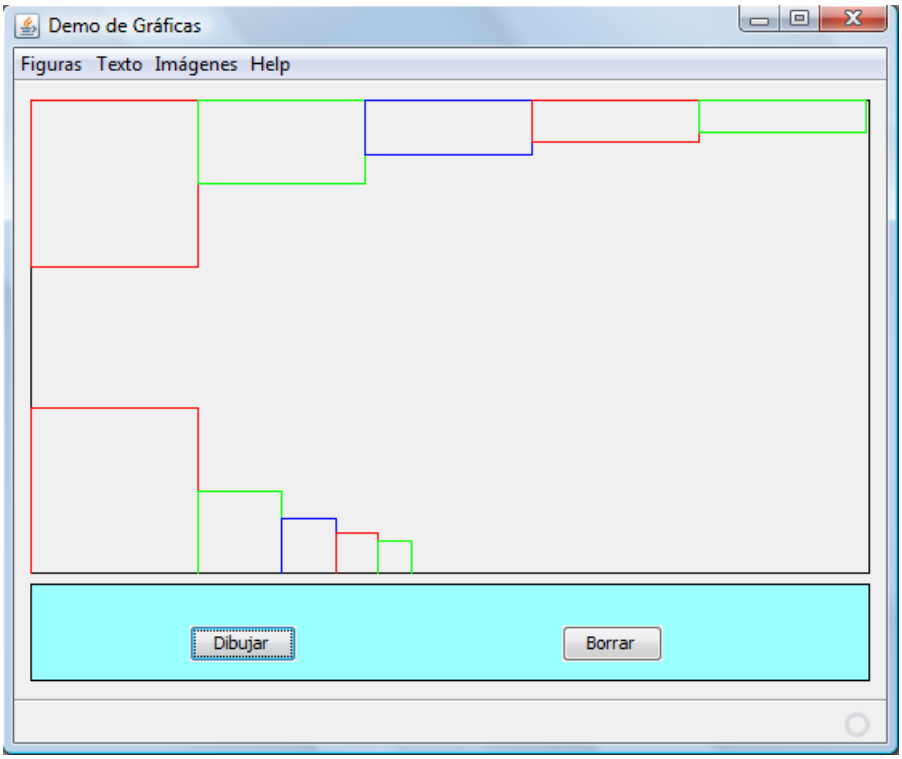

Figura 13.24. Rectángulos

Rectángulos Redondeados

La clase abstracta `RoundRectangle2D` describe un rectángulo redondeado (de esquinas redondeadas) definido por una posición (x, y) y una dimensión (w x h) y el ancho y alto de los arcos que redondean las esquinas, figura 13.25.

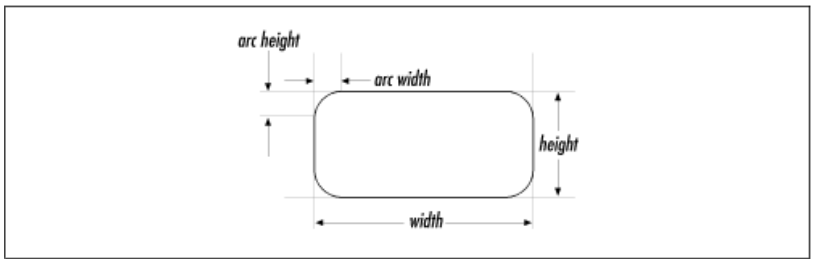

Figura 13.25. Rectángulo Redondeado

La clase `RoundRectangle2D` es la superclase de todas las clases que almacenan un rectángulo 2D de esquinas redondeadas. La clase `RoundRectangle2D.Double` es una de sus subclases que implementa los métodos abstractos y está definida como una clase interna. El diagrama de clases de ésas clases se muestra en la figura 13.26.

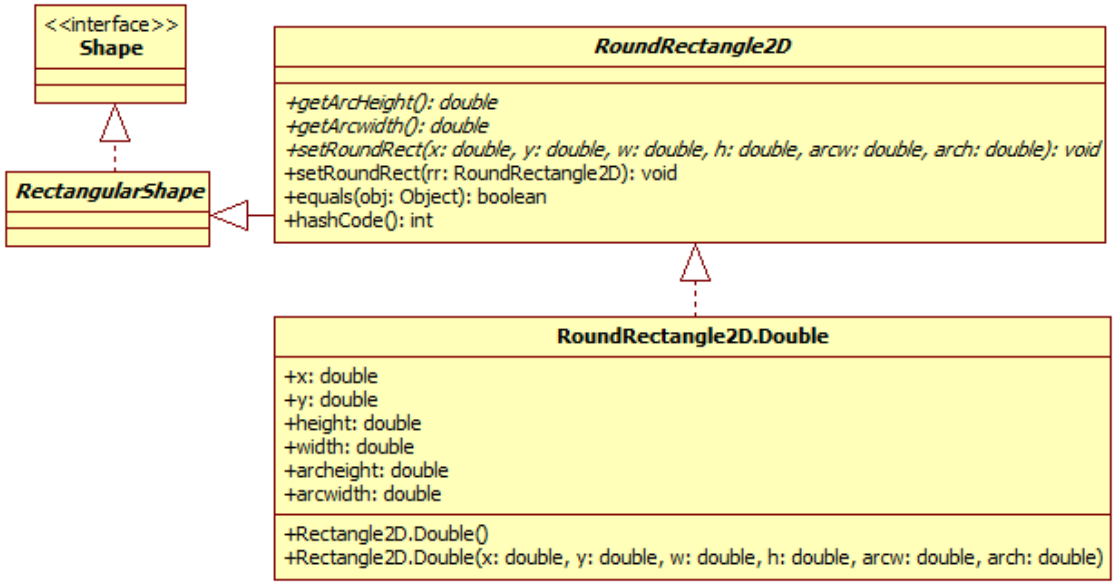

Figura 13.26 Clases que Representan un Rectángulo Redondeado

La tabla 13.16 muestra los métodos de la clase abstracta `Roundectangle2D`.

Tabla 13.16 Métodos de la Clase `RoundRectangle2D`

`public abstract double` **`getArcHeight`**`()`
`public abstract double` **`getArcWidth`**`()`
 	
Obtienen la altura y el ancho del arco que redondea las esquinas del rectángulo.

`public abstract void` **`setRoundRect`**`(double x, double y, double w, double h, double arcw, double arch)` Establece la posición y las dimenciones de los arcos de este rectángulo redondeado a los valores de sus parámetros.

Tabla 13.16 Métodos de la Clase `RoundRectangle2D`. Cont.

`public void` **`setRoundRect`**`(RoundRectangle2D rr)` Establece la posición de este rectángulo redondeado a los valores del rectángulo redondeado de su parámetro.
`public boolean` **`equals`**`(Object obj)` Determina si este rectángulo edonrdeado es el mismo que el rectángulo redondeado de su parámetro.
`public int` **`hashCode`**`()` Regresa el código Hash de este rectángulo de redondeada.

La tabla 13.17 muestra los métodos de la clase `Rectangle2D.Double`.

Tabla 13.17 Métodos de la Clase `RoundRectangle2D.Double`

`public` **`RoundRectangle2D.Double`**`()` Construye un nuevo rectángulo redondeado y lo inicializa a la posición (0, 0), tamaño (0, 0) y tamaño de los arcos de las esquinas redondeadas de (0, 0).
`public Round`**`Rectangle2D.Double`**`(double x, double y, double w, double h, double arcw, double arch)` Construye un nuevo rectángulo y lo inicializa a los valores de sus parámetros.

Ejemplo sobre Rectángulos Redondeados

El código del método `dibujaRectangulosRedondeados()` dibuja varios rectángulos y cuadrados redondeados:

Rectangulos.java

```
package figuras;
...
import java.awt.geom.RoundRectangle2D;

/**
* Clase con métodos estáticos para dibujar rectangulos, rectangulos
 * redondeados, elipses y arcos
 * @author mdomitsu
*/
public class Rectangulos {
    ...

    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * un conjunto de rectangulos redondeados
     * @param lienzo Panel sobre el que se dibujan los rectangulos
```

```
     * redondeados
     */
    public static void dibujaRectangulosRedondeados(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        int numeroRectangulos = 5;
        Color[] colores = {Color.red, Color.green, Color.blue};

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double x = 0;

        for (int i = 0; i < numeroRectangulos; i++) {
            // Establece el color de la rectangulo
            g2.setPaint(colores[i % colores.length]);

            double paso = d.width/numeroRectangulos;
            double alto = paso/(i+1);
            double aw = paso/5;
            double ah = alto/5;

            // Dibuja una rectangulo en parte superior
            RoundRectangle2D rectangulo1 = new
               RoundRectangle2D.Double(i*paso, 0, paso, alto, aw, ah);
            g2.draw(rectangulo1);

            // Dibuja una cuadradoo en la parte inferior
            RoundRectangle2D rectangulo2 = new
               RoundRectangle2D.Double(x, d.height - alto, alto, alto,
                                       ah, ah);
            x += alto;
            g2.draw(rectangulo2);
        }
    }
}
```

La figura 13.27 muestra los rectángulos redondeados desplegadas por el método `dibujaRectangulosRedondeados()`.

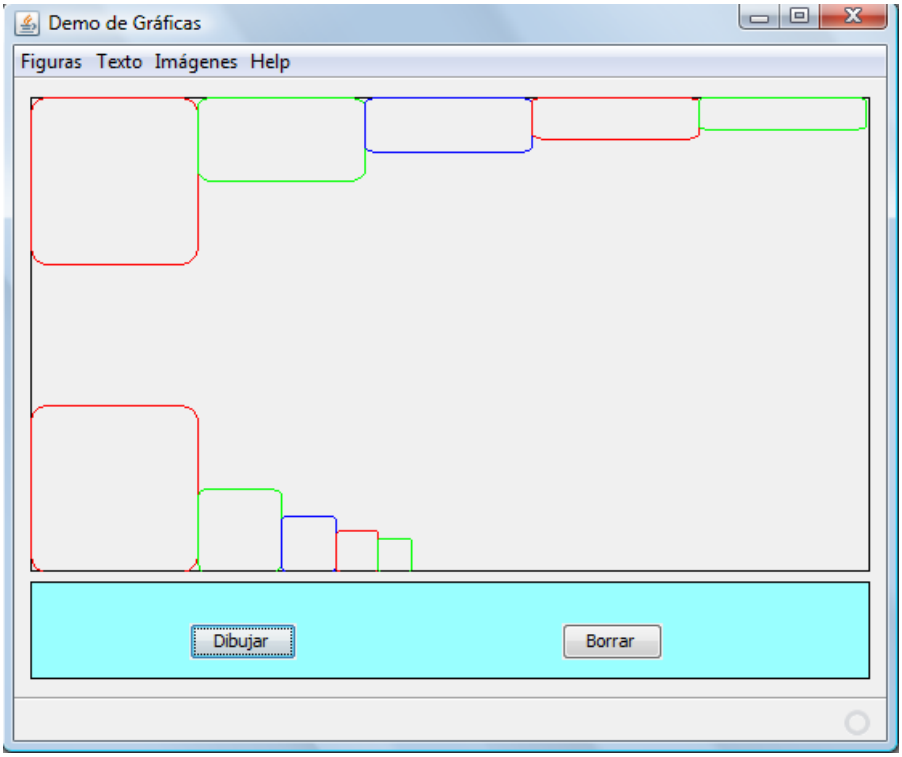

Figura 13.27. Rectángulos Redondeados

Elipses

La clase abstracta `Ellipse2D` describe una elipse definida por el rectángulo que la enmarca. Es la superclase de todas las clases que almacenan una elipse 2D. La clase `Ellipse2D.Double` es una de sus subclases que implementa los métodos abstractos y está definida como una clase interna. El diagrama de clases de ésas clases se muestra en la figura 13.28.

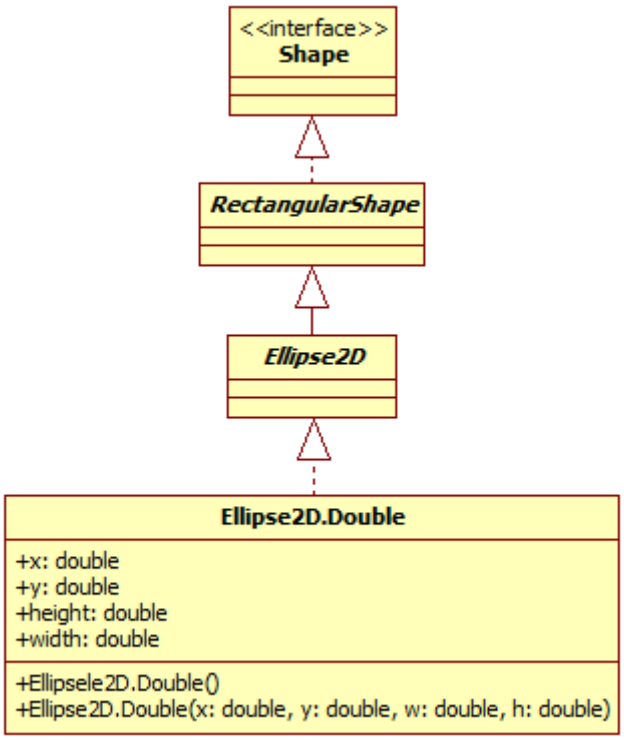

Figura 13.28 Clases que Representan una Elipse

La tabla 13.18 muestra los métodos de la clase `Ellipse2D.Double`.

Tabla 13.18 Métodos de la Clase `Ellipse2D.Double`

`public` **`Ellipse2D.Double`**`()` Construye una nueva relipse y la inicializa a la posición (0, 0) y tamaño (0, 0).
`public Ellips`**`e2D.Double`**`(double x, double y, double w, double h)` Construye una nueva relipse y la inicializa a los valores de sus parámetros.

Ejemplo sobre Elipses

El código del método `dibujaElipses()` dibuja varias elipses y círculos:

Rectangulos.java

```
package figuras;
...
import java.awt.geom.RoundRectangle2D;

/**
 * Clase con métodos estáticos para dibujar rectangulos, rectangulos
 * redondeados, elipses y arcos
```

```
 * @author mdomitsu
 */
public class Rectangulos {
    ...

    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * un conjunto de elipses
     * @param lienzo Panel sobre el que se dibujan las elipses
     */
    public static void dibujaElipses(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        int numeroElipses = 5;
        Color[] colores = {Color.red, Color.green, Color.blue};

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        for (int i = 0; i < numeroElipses; i++) {
            // Establece el color de la elipse
            g2.setPaint(colores[i % colores.length]);

            double ancho = d.width/(i + 1);
            double alto = d.height/(i+1);
            double x1 = (d.width - ancho)/2;
            double y = (d.height - alto)/2;
            double x2 = (d.width - alto)/2;

            // Dibuja una elipse
            Ellipse2D elipse1 = new Ellipse2D.Double(x1, y, ancho, alto);
            g2.draw(elipse1);

            // Dibuja un circulo
            Ellipse2D elipse2 = new Ellipse2D.Double(x2, y, alto, alto);
            g2.draw(elipse2);
        }
    }
    ...
}
```

La figura 13.29 muestra las elipses desplegadas por el método `dibujaElipses()`.

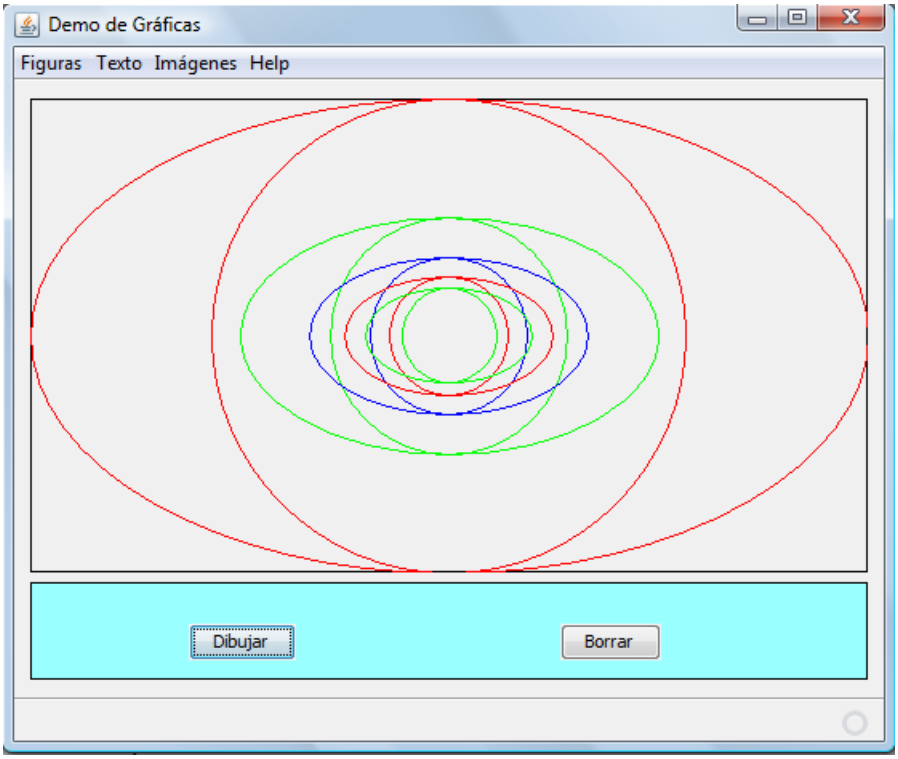

Figura 13.29. Elipses

Arcos

La clase abstracta `Arc2D` describe un segmento de una elipse definido por la posición (x, y) y dimensión (w x h) del rectánngulo que enmarca la elipse, los ángulos iniciales y finales del arco, figura 13.30 y el tipo de arco, figura 13.31.

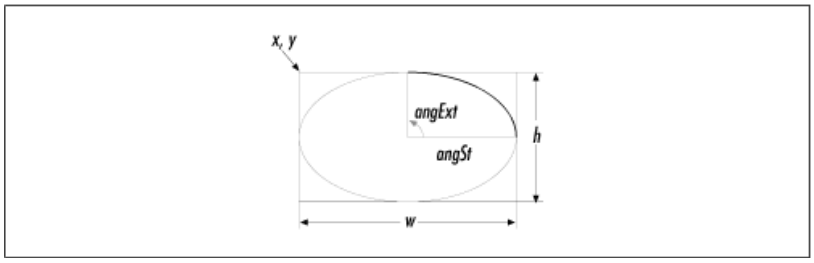

Figura 13.30. Parámetros de un Arco

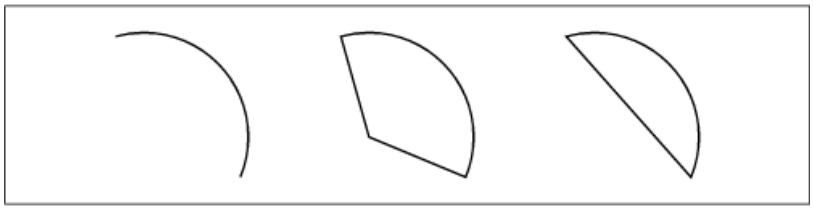

Figura 13.31. Tipos de Arcos: `OPEN`, `PIE`, `CHORD`.

Los tipos de arco están definidos por constantes estáticas:

- **`OPEN`**: Un arco abierto, es simplemente un segmento de una elipse.
- **`PIE`**: Un arco en la forma de una rebanada de pie. está formado por un segmento de elipse y dos líneas rectas entre los extremos del arco y el centro de la elipse.
- **`CHORD`**: Un arco con una línea recta que une los extremos del arco.

La clase `Arc2D.Double` es una de sus subclases que implementa los métodos abstractos y está definida como una clase interna. El diagrama de clases de ésas clases se muestra en la figura 13.32.

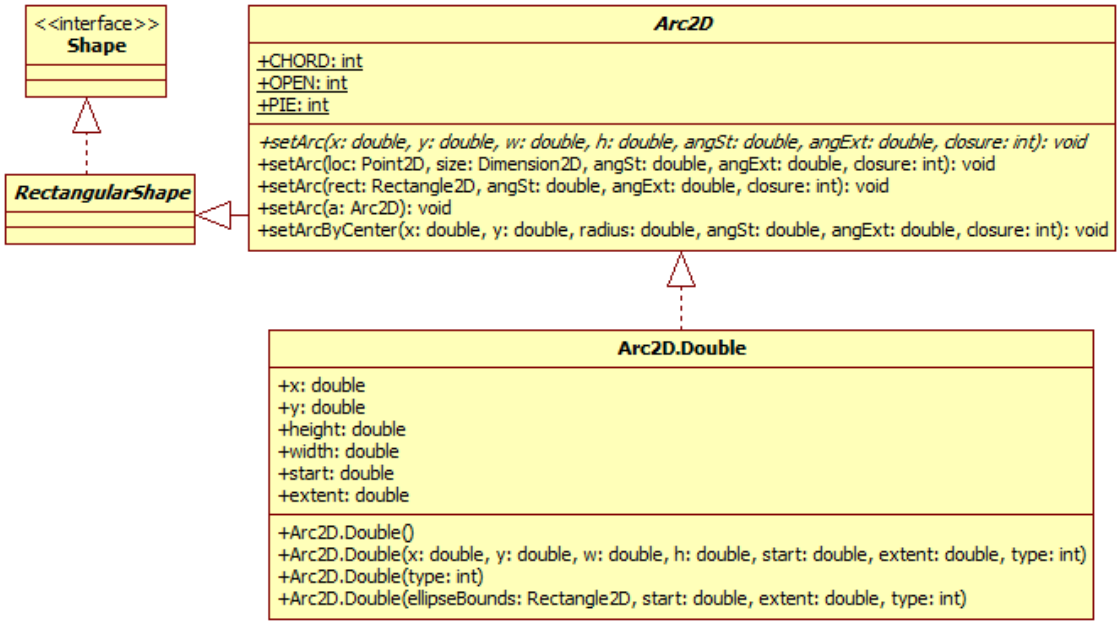

Figura 13.32 Clases que Representan Arcos

La tabla 13.19 muestra los métodos de la clase abstracta `Arc2D`.

Tabla 13.19 Métodos de la Clase `Arc2D`

`public abstract void setArc(double x, double y, double w, double h, double angSt, double angExt, int closure)` Establece la posición y las dimenciones del arco a los valores de sus parámetros.
`public abstract void setArc(Point2D loc, Dimension size, double angSt, double angExt, int closure)` Establece la posición y las dimenciones del arco a los valores de sus parámetros.

Tabla 13.19 Métodos de la Clase `Arc2D`. Cont.

`public void` **`setArc`**`(Rectangle2D rec, double angSt, double angExt, int closure)` Establece la posición y las dimenciones del arco a los valores de sus parámetros.
`public void` **`setArc`**`(Arc2D a)` Establece las dimensiones del arco a los valores del arco del parámetro.
`public void` **`setArcByCenter`**`(double x, double y, double radius, double angSt, double angExt, int closure)` Establece las dimensiones de un arco circular dados su centro, radio, ángulo inicial y extención del ángulo y tipo de arco a los valores de sus parámetros.

La tabla 13.20 muestra los métodos de la clase `Arc2D.Double`.

Tabla 13.20 Métodos de la Clase `Arc2D.Double`

`public` **`Arc2D.Double`**`()` Construye un nuevo arco abierto y lo inicializa a la posición (0, 0), tamaño (0, 0) y ángulo inicial y extención del ángulo (0, 0).
`public Arc`**`2D.Double`**`(double x, double y, double w, double h, double start, double ext, int type)` Construye un nuevo arco y lo inicializa a los valores de sus parámetros.
`public Arc`**`2D.Double`**`(int type)` Construye un nuevo arco del tipo especificado por el parámetro y lo inicializa a la posición (0, 0), tamaño (0, 0), ángulo inicial y extención del ángulo (0, 0).
`public Arc`**`2D.Double`**`(Rectangle2D ellipseBounds, double start, double ext, int type)` Construye un nuevo arco y lo inicializa a los valores de sus parámetros.

Ejemplo sobre Arcos

El código del método `dibujaArcos()` dibuja varios arcos:

Rectangulos.java

```
package figuras;
...
import java.awt.geom.RoundRectangle2D;

/**
 * Clase con métodos estáticos para dibujar rectangulos, rectangulos
 * redondeados, elipses y arcos
 * @author mdomitsu
 */
public class Rectangulos {
```

```
    ...
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * un conjunto de arcos
     * @param lienzo Panel sobre el que se dibujan las rectangulos
     */
    public static void dibujaArcos(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        int numeroArcos = 4;
        Color[] colores = {Color.red, Color.green, Color.blue};

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        for (int i = 0; i < numeroArcos; i++) {
            // Establece el color del arco
            g2.setPaint(colores[i % colores.length]);
            double ancho = d.width/(i + 1);
            double alto = d.height/(i+1);
            double x1 = (d.width - ancho)/2;
            double y = (d.height - alto)/2;
            double x2 = (d.width - alto)/2;

            // Dibuja un arco elíptico
            Arc2D arco1 = new Arc2D.Double(x1, y, ancho, alto, 90*i, 45,
                                           i % 3);
            g2.draw(arco1);

            // Dibuja un arco circular
            Arc2D arco2 = new Arc2D.Double(x2, y, alto, alto, 90*i, 45,
                                           i % 3);
            g2.draw(arco2);
        }
    }
    ...
}
```

La figura 13.33 muestra las elipses desplegadas por el método `dibujaArcos()`.

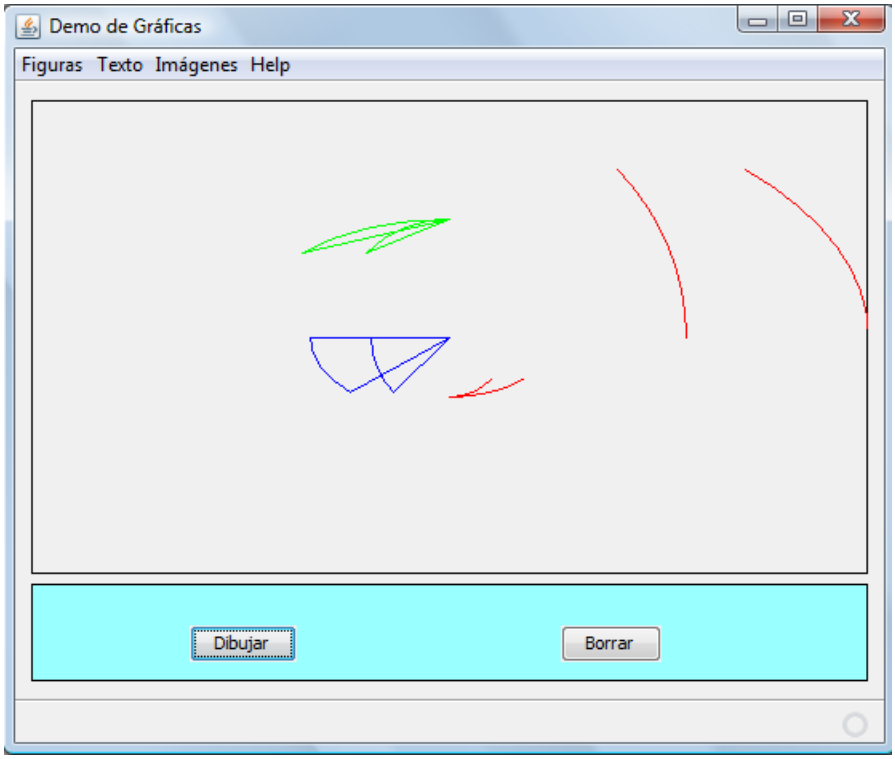

Figura 13.33. Arcos

Trayectorias

Si deseamos dibujar una figura diferente a las figuras disponibles en Java 2D y que implementan la interfaz `Shape`, entonces se requiere describir la trayectoria que define el contorno de la figura, a partir de líneas rectas y curvas. Una trayectoria es es una serie de instrucciones para ir de un lugar a otro. Cada instrucción describe un segmento o pieza de la trayectoria. Por ejemplo, podemos describir el contorno de un cuadrado con las siguientes instrucciones:

1. Colócate en la posición 0, 0.
2. Dibuja una línea a la posición 72, 0.
3. Dibuja una línea a la posición 72, 72.
4. Dibuja una línea a la posición 0, 72.
5. Dibuja una línea de regreso a 0, 0.

Java 2D, utiliza las clases `Path2D` y `Path2D.Double` para describir una trayectoria. El diagrama de clases para `Path2D` y `Path2D.Double` se muestra en la figura 13.34.

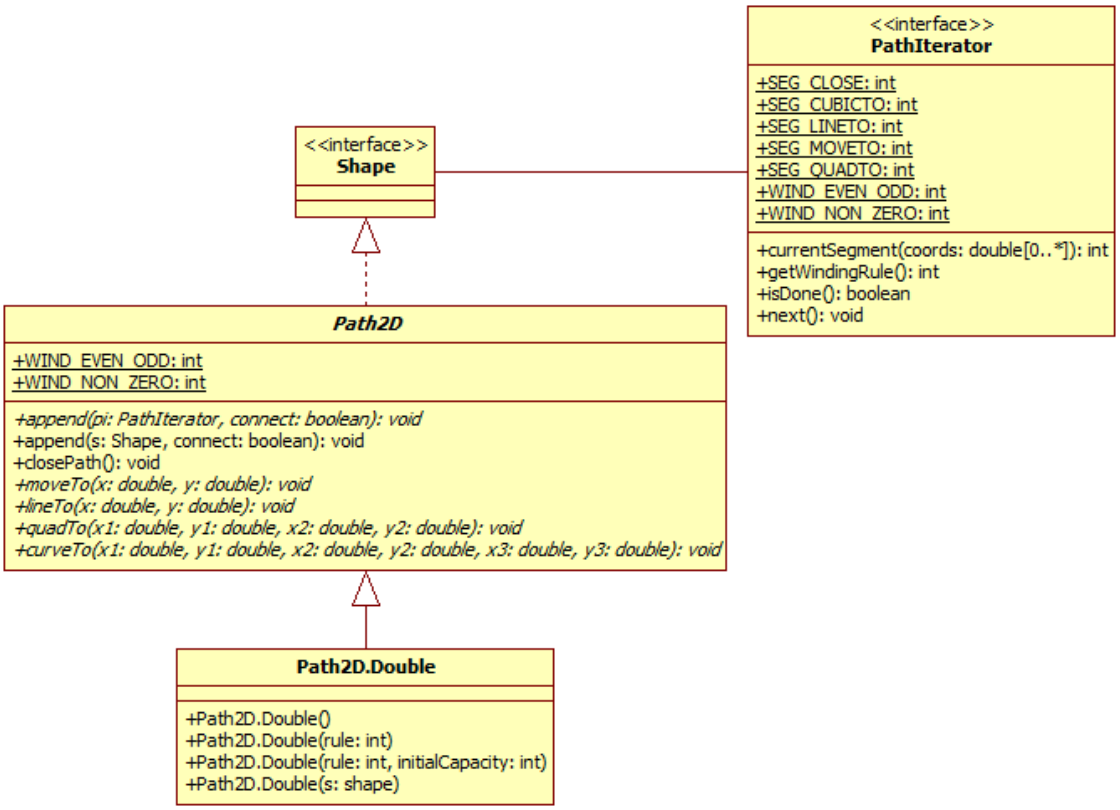

Figura 13.34 Clases que Representan Trayectorias

La tabla 13.21 muestra los métodos de la clase abstracta `Path2D`.

Tabla 13.21 Métodos de la Clase `Path2D`

`public abstract void` **`append`**`(PathIterator pi, boolean connect)`
`public final void` **`append`**`(Shape s, boolean connect)`
 	
La agrega a esta trayectoria la geometría especificada por el iterador o la figura del parámetro. Si el parámetro connect es verdadero y esta trayectoria no está vacía cualquier `moveTo()` inicial en la geometría de la figura inicial se convierte en un segmento `lineTo()`.

`public final void` **`closePath`**`()` Cierra esta trayectoria dibujando una línea recta a las coordenadas de la última `moveTo()`. Si la trayectoria ya está cerrada, el método no tiene efecto.
`public abstract void` **`moveTo`**`(double x, double y)` Agrega un punto a esta trayectoria moviendose a las coordenadas del parámetro.
`public abstract void` **`lineTo`**`(double x, double y)` Agrega un punto a esta trayectoria dibujando una línea recta desde la posición actual hasta las coordenadas del parámetro.

Tabla 13.21 Métodos de la Clase `Path2D`. Cont.

`public abstract void` **`quadTo`**`(double x1, double y1, double x2, double y2)` Agrega un segmento curvo definido por dos nuevos puntos a esta trayectoria dibujando una curva cuadrática desde la posición actual hasta las coordenadas (`x2, y2`) usando el punto (`x1, y1`) como punto de control.
`public abstract void` **`curveTo`**`(double x1, double y1, double x2, double y2 , double y3)` Agrega un segmento curvo definido por tress nuevos puntos a esta trayectoria dibujando una curva cúbica desde la posición actual hasta las coordenadas (`x3, y3`) usando los puntos (`x1, y1`) y (`x2, y2`) como puntos de control.

La tabla 13.22 muestra los métodos de la clase `Path2D.Double`.

Tabla 13.22 Métodos de la Clase `Path2D.Double`

`public` **`Path2D.Double`**`()` Construye una nueva trayectoria vacía con la regla de curvado por omisión: `Path2D.WIND_NON_ZERO`.
`public` **`Path2D.Double`**`(int rule)` Construye una nueva trayectoria vacía con la regla de curvado de su parámetro.
`public` **`Path2D.Double`**`(int rule, int initialCapacity)` Construye una nueva trayectoria vacía con la regla de curvado y capacidad inicial de sus parámetros. La capacidad inicial es una estimación de segmentos que contendrá la trayectoria, pero esa capacidad crece conforme se requiere.
`public` **`Path2D.Double`**`(Shape s)` Construye una nueva trayectoria a partir de una figura arbitraria dada por su parámetro.

Reglas de Curvado

Una regla de curvado determina que parte de la figura se define como el interior y por lo tanto que parte de la figura será rellenada con el método `fill()`. Para figuras simples como un rectángulo o una elipse, determinar el interior es sencillo, pero no lo es para figuras más complicadas como la mostrada en la figura 13.35.

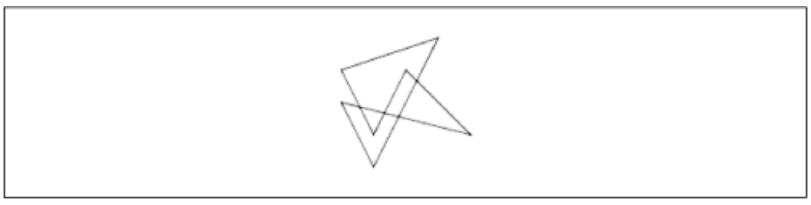

Figura 13.35. Una Figura Compleja.

Hay dos reglas que nos permiten determinar el interior de una figura: La Regla de Curvado Par-Non y la Regla de Curvado No-Cero.

Regla de Curvado Par-Non

Esta regla representada por la constante entera **`WIND_EVEN_ODD`** trabaja de la siguiente manera: Dibuje una línea que cruce la figura completamente. Cada vez que la línea cruce la orilla de la figura, incremente en uno un contador. Cuando el contador sea par, la línea se encuentra fuera de la figura. Cuando el contador sea non, la línea está en el interior de la figura, como se muestra en la figura 13.36. Esta regla sólo nos dice el interior de la figura para los puntos sobre la línea de prueba. Para determinar el interior total de la figura deben de dibujarse muchas lineas que atraviecen la figura.

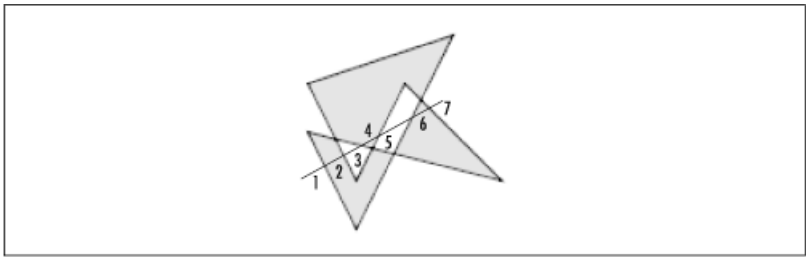

Figura 13.36. Regla de Curvado Par-Non.

Regla de Curvado No-Cero

Esta regla representada por la constante entera **`WIND_NON_ZERO`** trabaja de la siguiente manera: Dibuje una línea que cruce la figura completamente. Cada vez que la línea cruce la orilla de la figura, incremente en uno un contador si la orilla se dibuja de izquierda a derecha y decremente en uno el contador si la orilla se dibuja de derecha a izquierda. Las porciones de la línea donde el contador no sea cero se consideran dentro de la figura, como se muestra en la figura 13.37.

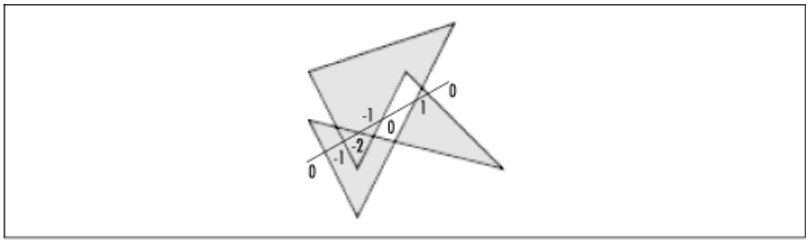

Figura 13.37. Regla de Curvado No Cero.

Ejemplo sobre Trayectorias

El código del método `dibujaTrayectoria()` dibuja una trayectoria arbitraria:

Trayectorias.java

```
package figuras;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Path2D;
import javax.swing.JPanel;

/**
 * Esta clase contiene métodos estáticos para trabajar con trayectorias
 * @author mdomitsu
 */
public class Trayectorias {
    /**
     * Este metodo estatico dibuja dibuja sobre el panel de su parametro
     * una trayectoria arbitraria
     * @param lienzo Panel sobre el que se dibuja la trayectoria
     */
    public static void dibujaTrayectoria(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();
        // Establece el color de la trayectoria
        g2.setPaint(Color.blue);

        // crea la trayectoria
        Path2D trayectoria = new Path2D.Double(Path2D.WIND_EVEN_ODD);
        trayectoria.moveTo(50, 50);
        trayectoria.lineTo(70, 44);
        trayectoria.curveTo(100, 10, 140, 80, 160, 80);
        trayectoria.lineTo(190, 40);
        trayectoria.lineTo(200, 56);
        trayectoria.quadTo(100, 150, 70, 60);
        trayectoria.closePath();

        // Dibuja la trayectoria
        g2.draw(trayectoria);
    }
}
```

La figura 13.38 muestra la trayectoria desplegada por el método `dibujaTrayectoria()`.

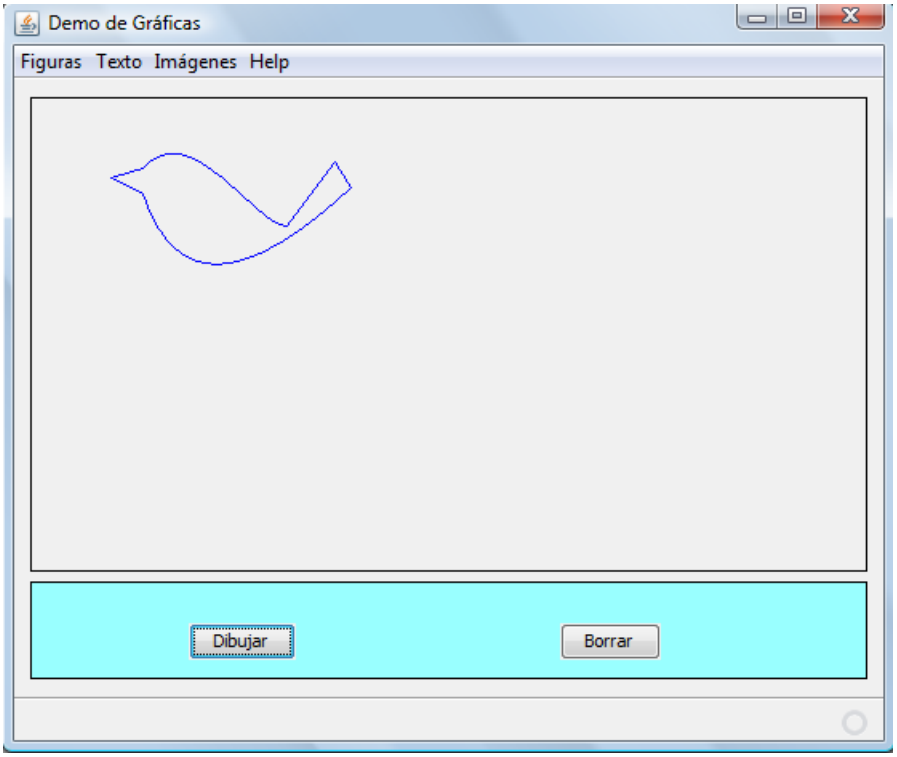

Figura 13.38. Trayectorias

La interfaz `PathIterator`

Un objeto que implemente la interfaz `PathIterador`, cuyo diagrama de clases se muestra en la figura 13.34, nos permite recorrer los segmentos de una trayectoria. Podemos obtener un objeto del tipo `PathIterador` que describa el contorno de una figura invocando a su método `getPathIterador()`. Inicialmente el iterador se encuentra posicionado al inicio de la trayectoria, podemos movernos a lo largo de los diferentes segmentos de la trayectoria hasta alcanzar el final. El iterador es de sólo lectura. No nos permite que modifiquemos la trayectoria.

Como ya se vió en la clase Path2D, para formar una trayectoria se pueden emplear cinco diferentes tipos de segmentos, usando los métodos `moveTo()`, `lineTo()`, `quadTo`, `curveTo()` y `closePath()`. La interfaz `PathIterador` define cinco constantes que representan cada uno de esos tipos de segmentos:

- **`SEG_MOVETO`**: Este tipo segmento se usa para actualizar la posición de la trayectoria sin dibujar una línea.

- **SEG_LINETO**: Este tipo segmento es una línea recta, dibujada desde el último punto de la trayectoria.

- **SEG_QUADTO**: Este tipo segmento es una línea curva representada por una ecuación cuadrática (segundo órden). Este segmento se describe completamente por dos puntos extremos y un punto de control que determina las tangentes de la curva en sus puntos extremos. El último punto de la trayectoria es el primer punto extremo. se debe especificar el otro punto extremo y el punto de control.

- **SEG_CUBICTO**: Este tipo segmento es una línea curva cúbica de Bézier. Es representada por una ecuación cúbica (tercer órden). Este segmento se describe completamente por dos puntos extremos y dos puntos de control que determinan las tangentes de la curva en sus puntos extremos. El último punto de la trayectoria es el primer punto extremo. se debe especificar el otro punto extremo y los dos puntos de control.

- **SEG_CLOSE**: Este tipo de segmento dibuja una línea de regreso al final del último segmento del tipo **SEG_MOVETO**. Una trayectoria puede componerse de varias subtrayectorias. El principio de cada subtrayectoria se marca con un segmento del tipo **SEG_MOVETO**, de tal manera que el efecto de SEG_CLOSE es de cerrar la última subtrayectoria con una línea recta.

La descripción de los métodos de la interfaz `PathIterador` se encuentra en la tabla 13.23.

Tabla 13.23 Métodos de la Interfaz `PathIterador`

`int` **`currentSegment`**`(double[] coords)`

Regresa las coordenadas y tipo del segmento de trayectoria actual en la iteración. Se le debe pasar un arreglo de dobles de tamaño 6. El número de puntos regresado depende del tipo de segmento.

Valor regresado	Puntos regresados en el arreglo
SEG_MOVETO	1
SEG_LINETO	1
SEG_QUADTO	2
SEG_CUBICTO	3
SEG_CLOSE	0

`int` **`getWindingRule`**`()`

Regresa la regla de curvado para la determinación del interior de la trayectoria.

`boolean` **`isDone`**`()`

Prueba si la iteración se ha completado. Regresa verdadero si se han leído todos los segmentos. Falso en caso contrario.

Tabla 13.23 Métodos de la Interfaz `PathIterador`. Cont.

`void next()` Mueve el iterador al siguiente segmento de la trayectoria.

Combinación de Figuras

La API 2D de Java permite combinar las áreas de 2 figuras de cuatro formas diferentes:

- **Adición o Unión**: La adición o unión de dos figuras es el área cubierta por una o ambas figuras.
- **Intersección**: La intersección de dos figuras es el área cubierta por ambas figuras simultáneamente.
- **Substracción**: El resultado de substraer una figura de la otra es es el área cubierta por una y que no es cubierta por la otra.
- **Unión Exclusiva**: La unión exclusiva es la inversa de la operación de intersección. La unión exclusiva de dos figuras es el área cubierta por una o la otra figura pero no por ambas.

La figura 13.39 muestra el resultado de combinar dos figuras sobrepuestas, usando las cuatro operaciones.

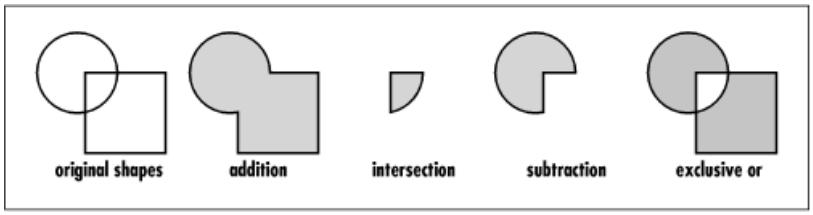

Figura 13.39 Diferentes Formas de Combinar Dos Figuras

En la API 2D de Java, las cuatro operaciones anteriores se implementan en la clase `Area`. Un objeto de tipo `Area` almacena y manipula la descripción de un área cerrada. La clase `Area` implementa la interfaz `Shape` pero es específico que una trayectoria en varias formas:

- Solo se almacenan trayectorias y subtrayectorias cerradas. Objetos de tipo `Area` construidos de trayectorias abiertas son implicitamente cerrados durante la construcción si esas trayectorias se han llenado usando el método `fill()`.

- El interior de todas las subareas individuales almacenadas son no vacías y no se traslapan.
- La geometría de la trayectoria que describe el contorno del `Area` se parece a la trayectoria de la que fue construida sólo en que describe a la misma area bidimensional, pero puede usar diferentes tipos de elementos y ordenamiento de los segmentos.

El diagrama de clases para `Area` se muestra en la figura 13.40.

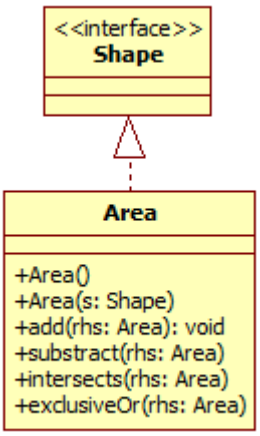

Figura 13.40 Clase que Representa una Área

La descripción de los métodos de la clase `Area` se encuentra en la tabla 13.24.

Tabla 13.24 Métodos de la Clase `Area`

`public` **`Area`**`()` Construye un área vacía.
`public` **`Area`**`(Shape s)` Construye un área a partir de una figura. Si la figura no es cerrada, se cierra explicitamente.
`public void` **`add`**`(Area rhs)` Le agrega a esta área la figura del área del parámetro. La figura resultante incluirá la unión de ambas figuras, o todas las áreas que estaban contenidas en esta o en el área del parámetro.
`public void` **`subtract`**`(Area rhs)` Le resta a esta área la figura del área del parámetro. La figura resultante incluirá las áreas de esta área y no en el área del parámetro.
`public void` **`intersect`**`(Area rhs)` Establece la figura de esta área a la intersección de la figura actual y la figura especificada por el parámetro. La figura resultante de esta área incluirá sólo las áreas que estaban en esta área y el área del parámetro.

Tabla 13.24 Métodos de la Clase `Area`. Cont.

`public void` **`exclusiveOr`**`(Area rhs)` Establece la figura de esta área a la unión de la figura actual y la figura especificada por el parámetro menos su intersección. La figura resultante de esta área incluirá sólo las áreas que estan en esta área o en el área del parámetro pero no en ambas.

Ejemplo sobre Combinación de Áreas

El código del método `combinaFiguras()` dibuja las combinaciones de varias figuras:

Operaciones.java

```
package figuras;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Area;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Rectangle2D;
import javax.swing.JPanel;

/**
 * Esta clase ilustra las operaciones de combinacion de figuras,
 * relleno de figuras y seleccion del tipo de contorno
 * @author mdomitsu
 */
public class Operaciones {

    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * combinaciones de figuras
     * @param lienzo Panel sobre el que se combinan las figuras
     */
    public static void combinaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        Area area1, area2;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double ancho = d.width / 6;
        double alto = d.height / 4;
        double x = 50;
        double y = 25;

        // Obten el area de una figura rectangular
        area1 = new Area(new Rectangle2D.Double(x, y, ancho, alto));
        x += ancho/2;
        y += alto/2;
```

```
        // Obten el area de una figura eliptica
        area2 = new Area(new Ellipse2D.Double(x, y, ancho, alto));
        // Dibuja las dos areas
        g2.setPaint(Color.black);
        g2.draw(area1);
        g2.draw(area2);
        // Obten la union de las areas
        area2.add(area1);
        // Rellana la union de las areas
        g2.setPaint(Color.red);
        g2.fill(area2);

        x += 100+ancho;
        y = 25;

        // Obten el area de una figura rectangular
        area1 = new Area(new Rectangle2D.Double(x, y, ancho, alto));
        x += ancho/2;
        y += alto/2;
        // Obten el area de una figura eliptica
        area2 = new Area(new Ellipse2D.Double(x, y, ancho, alto));
        // Dibuja las dos areas
        g2.setPaint(Color.black);
        g2.draw(area1);
        g2.draw(area2);
        // Obten la resta de las areas
        area2.subtract(area1);
        // Rellana la resta de las areas
        g2.setPaint(Color.yellow);
        g2.fill(area2);

        x = 50;
        y = 100+alto;

        // Obten el area de una figura rectangular
        area1 = new Area(new Rectangle2D.Double(x, y, ancho, alto));
        x += ancho/2;
        y += alto/2;
        // Obten el area de una figura eliptica
        area2 = new Area(new Ellipse2D.Double(x, y, ancho, alto));
        // Dibuja las dos areas
        g2.setPaint(Color.black);
        g2.draw(area1);
        g2.draw(area2);
        // Obten la interseccion de las areas
        area2.intersect(area1);
        // Rellena la interseccion de las areas
        g2.setPaint(Color.green);
        g2.fill(area2);

        x += 100+ancho;
        y = 100+alto;

        // Obten el area de una figura rectangular
        area1 = new Area(new Rectangle2D.Double(x, y, ancho, alto));
        x += ancho/2;
```

```
        y += alto/2;
        // Obten el area de una figura eliptica
        area2 = new Area(new Ellipse2D.Double(x, y, ancho, alto));
        // Dibuja las dos areas
        g2.setPaint(Color.black);
        g2.draw(area1);
        g2.draw(area2);
        // Obten la union exclusiva de las areas
        area2.exclusiveOr(area1);
        // Rellena la union exclusiva de las areas
        g2.setPaint(Color.cyan);
        g2.fill(area2);
    }
}
```

La figura 13.41 muestra las combinaciones de figuras desplegada por el método `combinaFiguras()`.

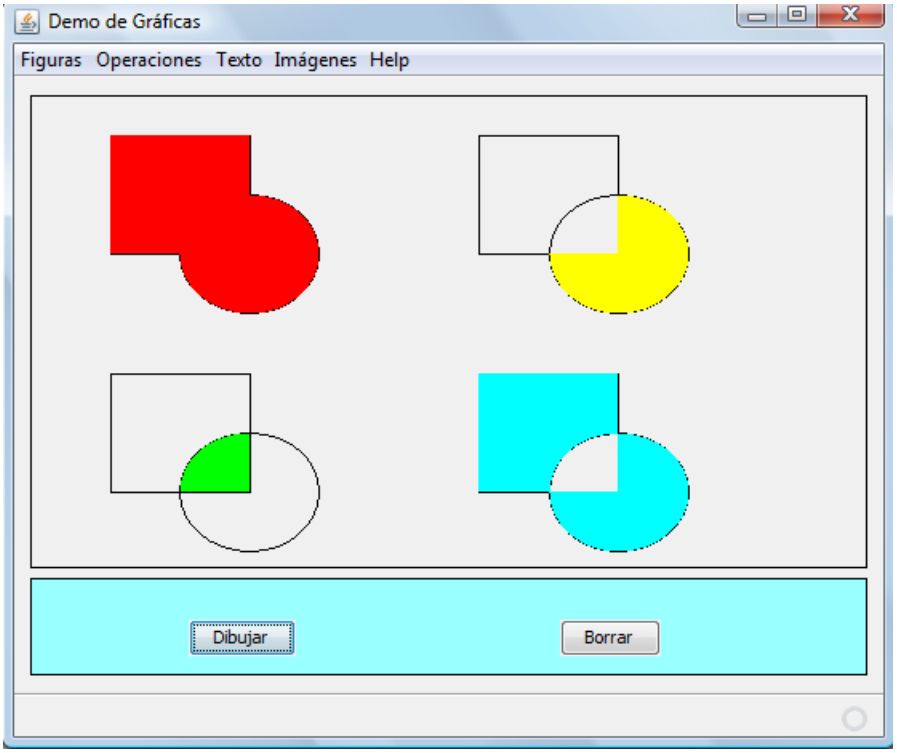

Figura 13.41 Combinación de Figuras: a) Unión, b) Substracción, c) Intersección, d) Unión exclusiva

Relleno de Figuras

Una figura cerrada puede rellenarse usando un color sólido, un gradiente de colores o una textura. Los métodos de la clase Graphics2D que nos permiten rellenar figuras se muestran en la Tabla 13.25.

Tabla 13.25 Métodos para Rellenar Figuras de la Clase Graphics2D

`public abstract void fill(Shape s)` Rellena la figura del parámetro utilizando las propiedades del contexto actual de `Graphics2D`
`public abstract Paint getPaint()` Regresa el atributo `Paint` del contexto de `Graphics2D`. Este atributo determina el tipo de relleno de una figura.
`public abstract void setPaint(Paint paint)` Establece el atributo `Paint` del contexto de `Graphics2D`. Este atributo determina el tipo de relleno de una figura.

Llenar una figura es un proceso de dos pasos:

1. Establecer el color, gradiente de color o textura a usar como relleno usando el método `setPaint()`.
2. Rellenar la figura usando el método `fill()`.

El parámetro del método `setPaint()` es un objeto que implemente la interface `Paint`. Graphics2D almacena ese objeto como parte de su estado y lo utiliza para rellenar las figuras. El diagrama de clases de la figura 13.42 muestra las clases que implementan la interfaz `Paint`.

La clase `Color` nos sirve para rellenar una figura con un color sólido. Esta clase define una serie de constantes simbólicas para algunos colores, tabla 13.26. Todas esas constantes son públicas y estáticas y del tipo `Color`.

Tabla 13.26 Colores Predefinidos de la clase `Color`

Constante	Color	Constante	Color	Constante	Color
`black`	Negro	**`green`**	Verde	**`red`**	Rojo
`blue`	Azul	**`lightGray`**	Gris Claro	**`white`**	Blanco
`cyan`	Cian	**`magenta`**	Magenta	**`yellow`**	Amarillo
`darkGray`	Gris oscuro	**`orange`**	Naranja		
`gray`	Gris	**`pink`**	Rosa		

Si se requieren otros colores diferentes a los predefinidos se pueden construir otros colores a partir de los colores primarios y valores alfa. Para ello, la clase `Color` tiene los constructores de la tabla 13.27.

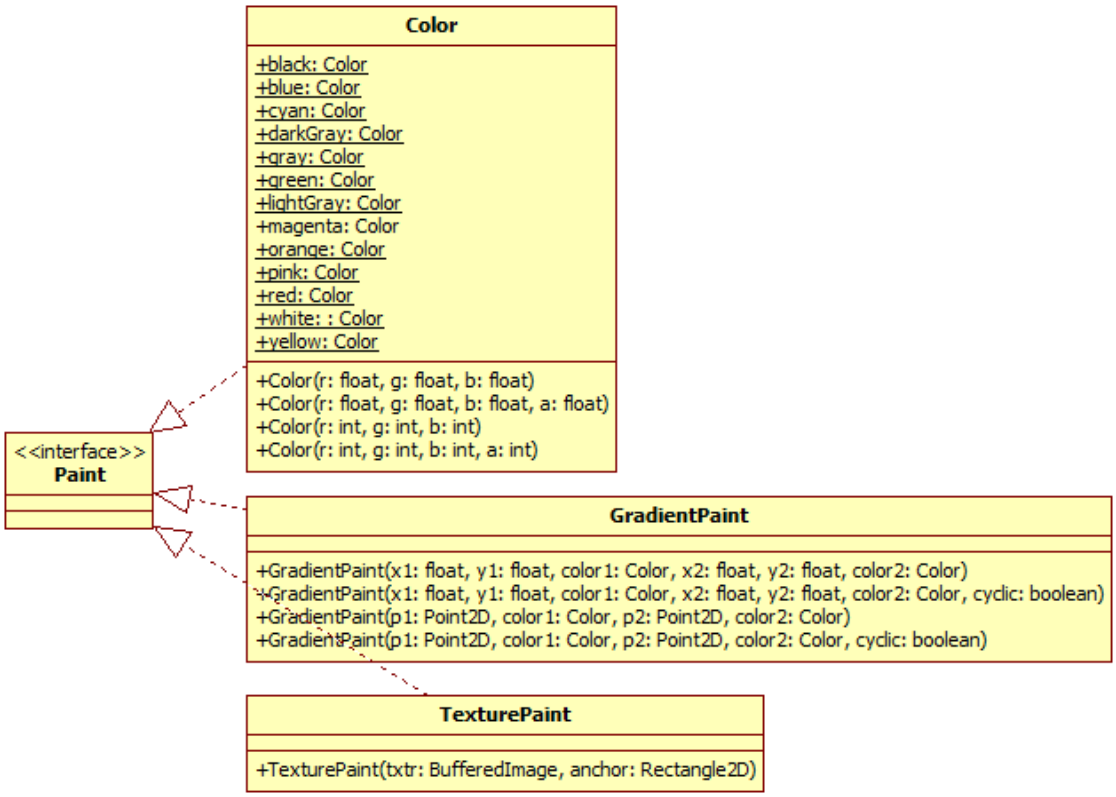

Figura 13.42 Clases que Representan un Relleno de una Figura

Tabla 13.27 Constructores de la Clase `Color`

`public Color(int r, int g, int b)`
`public Color(int r, int g, int b, int a)`
 	
Crean una color sRGB con los componentes rojo, verde, azul y alfa de sus parámetros. Los valores están en el rango 0 a 255, inclusive.
 	
En el primer constructor el valor de alfa es de 255.

`public Color(float r, float g, float b)`
`public Color(float r, float g, float b, float a)`
 	
Crean una color sRGB con los componentes rojo, verde, azul y alfa de sus parámetros. Los valores están en el rango 0.0 a 1.0, inclusive.
 	
En el primer constructor el valor de alfa es de 1.0.

La clase `GradientPaint` provee de una forma para llenar una figura con un patrón de gradiente de color. Si los puntos `p1` y `p2` del tipo `Point2D` con colores `c1` y `c2` del tipo Color se encuentran en el espacio del usuario, los colores a lo largo de la línea que conecta a los puntos `p1` y `p2` cambia proporcionalmente de `c1` a `c2`.

Cualquier punto q que no se encuentra en la línea que conecta a los puntos `p1` y `p2` tiene el color del punto p que es la proyección perpendicular de q en la línea entre `p1` y `p2`.

Cualquier punto p que se encuentra en la línea que conecta a los puntos `p1` y `p2` y por fuera del segmento entre los puntos `p1` y `p2`, se colorean de dos formas diferentes:

- Si el gradiente es cíclico entonces los puntos en la línea que conectan a los puntos `p1` y `p2` y por fuera del segmento entre los puntos `p1` y `p2`, se colorean usando el mismo gradiente de `c1` a `c2`.
- Si el gradiente es acíclico entonces los puntos del ñado de `p1` tendrán el color `c1` mientras que los puntos del lado `p2` tendrán el color `c2`.

Para especificar el gradiente de color, la clase `GradientPaint` tiene los constructores de la tabla 13.28.

Tabla 13.28 Constructores de la Clase `GradientPaint`

`public` **`GradientPaint`**`(float x1, float y1, Color color1,`
`float x2, float y2, Color color2)`
`public` **`GradientPaint`**`(float x1, float y1, Color color1,`
`float x2, float y2, Color color2, boolean cyclic)`
 	
Construyen gradientes al valor de sus parámetros. El primer constructor construye un gradiente acíclico.

`public` **`GradientPaint`**`(Point2D p1, Color color1, Point2D p2, Color color2)`
`public` **`GradientPaint`**`(Point2D p1, Color color1, Point2D p2, Color color2,`
`boolean cyclic)`
 	
Construyen gradientes al valor de sus parámetros. El primer constructor construye un gradiente acíclico.

La clase `TexturePaint` provee de una forma para llenar una figura con una textura especificada por un objeto del tipo `BufferedImage`. El tamaño del objeto `BufferedImage` debe ser pequeño ya que los datos del objeto `BufferedImage` serán copiados por el objeto `TexturePaint`. En el momento de la construcción, la textura es anclada en la esquina superior izquierda del rectángulo del tipo Rectangle2D. La textura se crea repitiendo el rectángulo en forma infinita en todas las dimensiones y mapeando la imagen a cada rectágulo copiado. El constructor de la clase que nos permite hacer esto se muestra en la tabla 13.29.

Tabla 13.29 Constructor de la Clase `TexturePaint`

`public` **`TexturePaint`**`(BufferedImage txtr, Rectangle2D anchor)` Construye un objeto del tipo `TexturePaint` al valor de sus parámetros.

Ejemplo sobre Rellenado de Figuras

El código del método `rellenanaFiguras()` rellena varias figuras:

Operaciones.java

```
    /**
     * Este metodo rellena figuras sobre el panel de su parametro
     * @param lienzo Panel sobre el que se rellenan las figuras
     */
    public void rellenaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double ancho = d.width / 4;
        double alto = d.height / 4;
        double x = 50;
        double y = 25;

        // Rellena una elipse con un color solido
        Ellipse2D elipse = new Ellipse2D.Double(x, y, ancho, alto);
        g2.setPaint(Color.yellow);
        g2.fill(elipse);

        x += 100 + ancho;
        // Crea un gradiente de color
        GradientPaint gp = new GradientPaint((int) x, (int) y, Color.red,
                (int) (x + ancho), (int) (y + alto), Color.blue);

        // Rellena un rectangulo con un gradiente de color
        Rectangle2D rect = new Rectangle2D.Double(x, y, ancho, alto);
        g2.setPaint(gp);
        g2.fill(rect);

        x = 50 + ancho;
        y = 100 + alto;

        // Carga la imagen a ser usada como textura
        BufferedImage bi;
        try {
            bi = getImage("fondo.jpg");
        } catch (IOException ioe) {
            JOptionPane.showMessageDialog(lienzo,
                                          "Error al cargar imagen");
            return;
```

```
        }
        // Crea el rectangulo ancla del tamaño de la imagen
        Rectangle2D tr = new Rectangle2D.Double(0, 0, bi.getWidth(),
                                                bi.getHeight());
        // Crea la textura
        TexturePaint tp = new TexturePaint(bi, tr);

        // Rellena el rectangulo con la textura
        RoundRectangle2D rrect = new RoundRectangle2D.Double(x, y, ancho,
                                               alto, ancho / 5, alto / 5);
        g2.setPaint(tp);
        g2.fill(rrect);
    }

    /**
     * Lee la imagen a ser usada como textura de un archivo
     * @param filename Nombre del archivo con la imagen
     * @return Imagen
     * @throws IOException si hay un problema al cargar la imagen
     * @throws ImageFormatException Si hay un problema al formatear la
     * imagen
     */
    public BufferedImage getImage(String filename)
            throws IOException, ImageFormatException {
        // Lee la imagen JPEG del archivo.
        InputStream in = getClass().getResourceAsStream(filename);
        JPEGImageDecoder decoder = JPEGCodec.createJPEGDecoder(in);

        // Convierte la imagen a una del tipo BufferedImage
        BufferedImage bi = decoder.decodeAsBufferedImage();
        in.close();
        return bi;
    }
}
```

La figura 13.43 muestra las combinaciones de figuras desplegada por el método `rellenaFiguras()`.

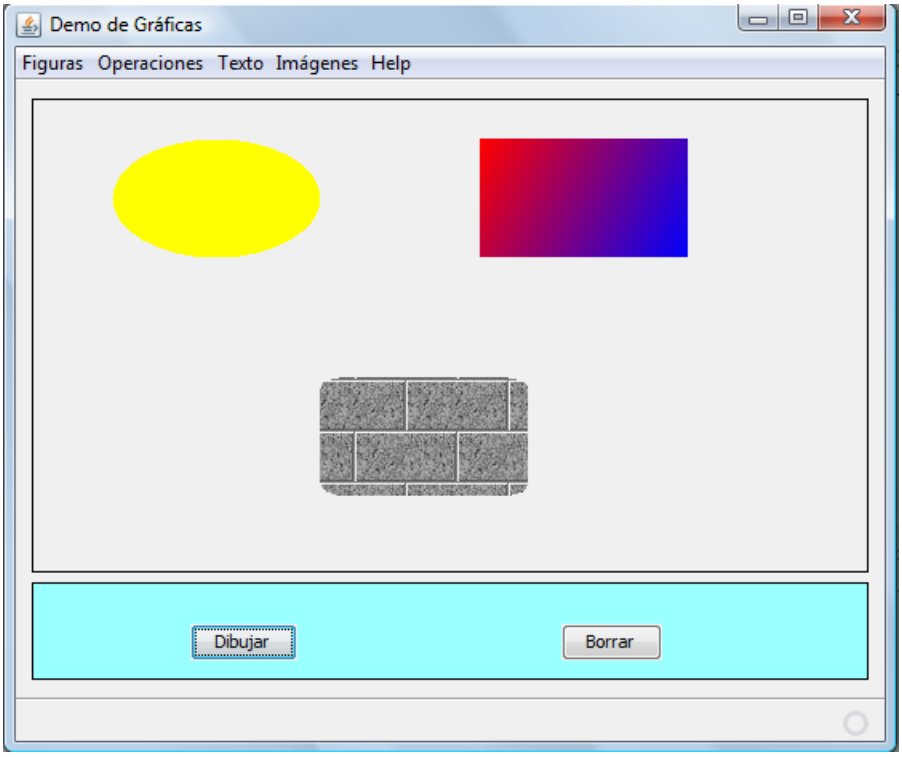

Figura 13.43 Rellenado de Figuras

Tipo de Contorno

Para dibujar el contorno de una figura se siguen los siguientes tres pasos:

1. Especificar el tipo de contorno a usar, llamando al método `setStroke()` de la clase `Graphics2D`. Este método acepta cualquier objeto que implementa la interfaz `Stroke`. La sintaxis de este método se muestra en la tabla 13.30.

Tabla 13.30 Método para Acceder al Tipo de Contorno de las Figuras de la Clase Graphics2D

`public abstract Stroke` **`getStroke`**`()` Obtiene el tipo de contorno del contexto actual de Graphics2D.
`public abstract void` **`setStroke`**`(Stroke s)` Establece el tipo de contorno a usar para dibujar una figura al valor del parámetro. El tipo de contorno se almacena como una propiedad del contexto actual de `Graphics2D.`

2. Establecer el color, gradiente de color o textura a usar como relleno del contorno, usando el método `setPaint()`de la clase `Graphics2D`. El contorno, de la misma forma que el interior de las figuras puede dibujarse

empleando un color, gradiente de color, textura o cualquier cosa que implemente la interfaz `Paint`.

3. Dibuja el borde de la figura usando el método `draw()` de la clase `Graphics2D`. La clase `Graphics2D` usa el objeto de tipo `Stroke` para determinar el tipo de contorno a emplear y utiliza el objeto de tipo `Paint` para producir el contorno.

Los objetos que encapsulan la información de tipo de contorno de una figura implementan la interfaz `Stroke`. Una clase que implemeta esa interfaz es la clase `BasicStroke`. El diagrama de clases de ambas se muestra en la figura 13.44.

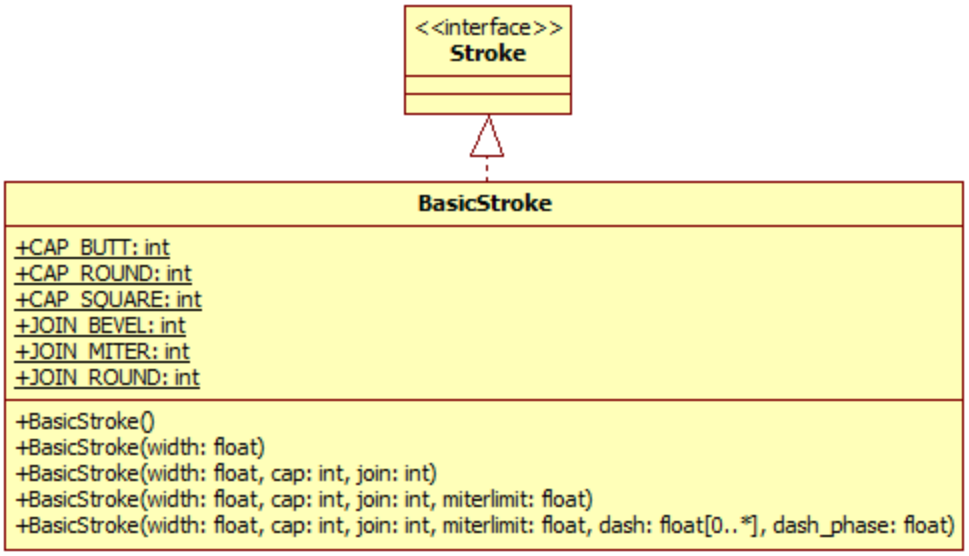

Figura 13.44 Clases que Representan el Tipo de Contorno de una Figura

La clase `BasicStroke` soporta líneas sólidas y punteadas de cualquier grosor. También maneja dos detalles para líneas gruesas: Estilos para los extremos y para las uniones de líneas.

Hay tres estilos para los extremos en la clase `BasicStroke` representados por las constantes simbólicas:

- `CAP_BUTT`: En este estilo de extremo, los extremos aparecen como rectángulos.
- `CAP_ROUND`: En este estilo de extremo, los extremos son semicirculos de radio de la mitad del ancho de la línea.
- `CAP_SQUARE`: En este estilo de extremo, los extremos son rectángulos cuya longitud es la mitad del ancho de la línea.

El efecto de cada estilo se muestra en la figura 13.45:

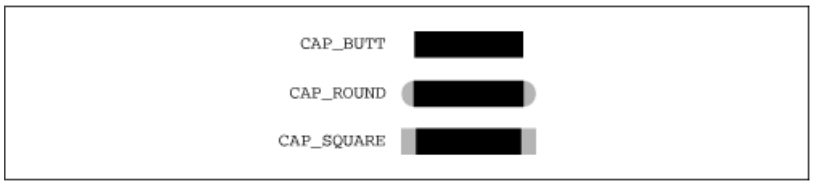

Figura 13.45 Estilos de Extremos para las líneas.

Hay tres estilos para las uniones de líneas en en la clase `BasicStroke` representados por las constantes simbólicas:

- `JOIN_BEVEL`: En este estilo, las líneas se unen conectando los bordes externos de sus extremos.
- `JOIN_MITER`: En este estilo, los bordes externos de las líneas se extienden hasta que se intersectan.
- `JOIN_ROUND`: En este estilo, cada segmento termina con un semicirculo. Esto crea un efecto redondeado en la intersección de los segmentos de línea.

El efecto de cada estilo se muestra en la figura 13.46:

Figura 13.46 Estilos de Uniones para las líneas.

`BasicStroke` controla la forma de las lineas punteadas mediante un arreglo de flotantes llamado `dash` que representan las longitudes de de las secciones sólidas y borradas de la línea. Por ejemplo un arreglo con los valores `{12, 12}` produce una línea que es visible por 12 unidades e invisible por otras 12 unidades. Los elementos pares del arreglo empezando por el de índice 0 determinan donde la línea será visible. Los elementos nones determinan donde la línea será invisible. En la figura 13.47 se muestra una trayectoria arbitraria con dos diferentes lineas punteadas. A la izquierda con una línea sólida, en medio usando una arreglo `dash` de `{12, 12}` y a la derecha con areglo `dash` de `{4, 4, 12, 4}`.

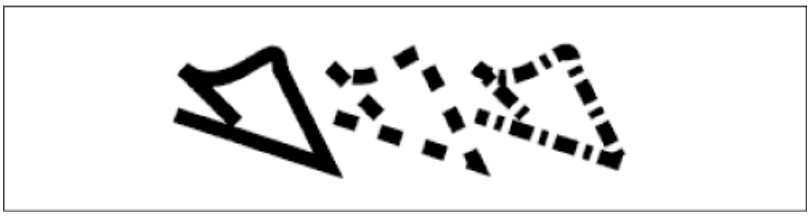

Figura 13.47 Diferentes Tipos de Líneas Punteadas.

Adicionalmente al arreglo `dash`, `BasicStroke` utiliza un flotante llamado `dash_phase` que actua como un desfase en el el patrón de puntos. Por ejemplo, con un arreglo `dash` con los valores `{12, 12}`, Si el valor de `dash_phase` es 0, el patrón será de una línea que es visible por 12 unidades e invisible por otras 12 unidades. Si el valor de `dash_phase` es 8, el patrón será de una línea que es visible por 4 unidades, invisible por otras 12 unidades, visible por otras 12 unidades, etc., como se muestra en la figura 13.48.

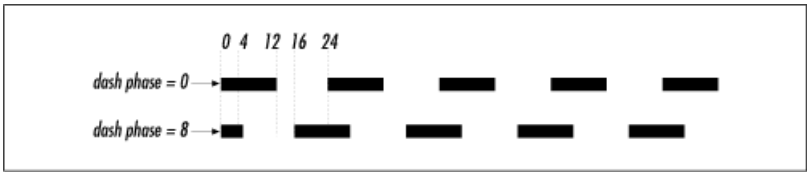

Figura 13.48. Diferentes valores de dash_phases.

La tabla 13.31 muestra los métodos de la clase `BasicStroke`.

Tabla 13.31 Métodos de la Clase `BasicStroke`

`public` **`BasicStroke`**`()` Construye un objeto del tipo basicStroke con los valores para ausencia para todos los atributos: Una línea sólida de un grosor de 1.0, `CAP_SQUARE`, `JOIN_MITER` y `miter_limit` de 10.0. Si la extención de la unión es mayor que `miterlimit`, entonces se usa una unión del tipo `JOIN_BEVEL` en lugar de `JOIN_MITER`.
`public` **`BasicStroke`**`(float width)` Construye un objeto del tipo basicStroke con el grosor de línea especificado por el parámetro y valores para ausencia para los demás atributos.
`public` **`BasicStroke`**`(float width, int cap, int join)` Construye un objeto del tipo basicStroke con los valores especificados para los atributos.
`public` **`BasicStroke`**`(float width, int cap, int join, float miterlimit)` Construye un objeto del tipo basicStroke con los valores especificados para los atributos.

`public` **`BasicStroke`**`(float width, int cap, int join, float miterlimit,`
`float[] dash, float dash_phase)`
 	
Construye un objeto del tipo basicStroke con los valores especificados para los atributos.

Ejemplo sobre Tipos de Contornos de Figuras

El código del método `contornoFiguras()` dibuja diferentes contornos de figuras:

Operaciones.java

```
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * tipos de contornos
```

```
     * @param lienzo Panel sobre el que se dibujan tipos de contornos
     */
    public void contornoFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        Area area1, area2;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();
        // Crea dos contornos
        BasicStroke bs1 = new BasicStroke(3);
        BasicStroke bs2 = new BasicStroke(15);

        double ancho = d.width / 6;
        double alto = d.height / 4;
        double x = 50;
        double y = 25;

        // Obten el area de una figura rectangular
        area1 = new Area(new Rectangle2D.Double(x, y, ancho, alto));
        x += ancho / 2;
        y += alto / 2;
        // Obten el area de una figura eliptica
        area2 = new Area(new Ellipse2D.Double(x, y, ancho, alto));
        // Establece el tipo de contorno
        float[] dash1 = {10, 5};
        BasicStroke bs3 = new BasicStroke(1, BasicStroke.CAP_BUTT,
                                    BasicStroke.JOIN_BEVEL, 0, dash1, 0);
        g2.setStroke(bs3);
        // Dibuja las dos areas
        g2.setPaint(Color.black);
        g2.draw(area1);
        g2.draw(area2);
        // Obten la union de las areas
        area2.add(area1);
        // Dibuja la union de las areas
        g2.setStroke(bs1);
        g2.setPaint(Color.red);
        g2.draw(area2);

        x += 100 + ancho;
        y = 25;

        // Obten el area de una figura rectangular
        area1 = new Area(new Rectangle2D.Double(x, y, ancho, alto));
        x += ancho / 2;
        y += alto / 2;
        // Obten el area de una figura eliptica
        area2 = new Area(new Ellipse2D.Double(x, y, ancho, alto));
        // Establece el tipo de contorno
        float[] dash2 = {5, 10, 10, 5};
        BasicStroke bs4 = new BasicStroke(1, BasicStroke.CAP_BUTT,
                                    BasicStroke.JOIN_BEVEL, 0, dash2, 0);
        g2.setStroke(bs4);
        // Dibuja las dos areas
        g2.setPaint(Color.black);
```

```
        g2.draw(area1);
        g2.draw(area2);
        // Obten la resta de las areas
        area2.subtract(area1);
        // Dibuja la resta de las areas
        g2.setStroke(bs1);
        g2.setPaint(Color.yellow);
        g2.draw(area2);

        x = 50;
        y = 100 + alto;

        // Crea una figura rectangular
        Rectangle2D rect = new Rectangle2D.Double(x, y, 2*ancho, alto);
        // Crea un gradiente de color
        GradientPaint gp = new GradientPaint((int) x, (int) y, Color.red,
                     (int) (x + 2*ancho), (int) (y + alto), Color.blue);
        // Establece el tipo de contorno
        g2.setStroke(bs2);
        // Dibuja el rectangulo
        g2.setPaint(gp);
        g2.draw(rect);

        x += 50 + 2*ancho;
        y = 100 + alto;

        // Crea de una figura eliptica
        Ellipse2D elipse = new Ellipse2D.Double(x, y, 2*ancho, alto);
        // Carga la imagen a ser usada como textura
        BufferedImage bi;
        try {
            bi = getImage("fondo.jpg");
        } catch (IOException ioe) {
            JOptionPane.showMessageDialog(lienzo,
                                          "Error al cargar imagen");
            return;
        }
        // Crea el rectangulo ancla del tamaño de la imagen
        Rectangle2D tr = new Rectangle2D.Double(0, 0, bi.getWidth(),
                                                bi.getHeight());
        TexturePaint tp = new TexturePaint(bi, tr);
        // Establece el tipo de contorno
        g2.setStroke(bs2);

        // Dibuja la elipse
        g2.setPaint(tp);
        g2.draw(elipse);
    }
```

La figura 13.49 muestra las combinaciones de figuras desplegada por el método `contornoFiguras()`.

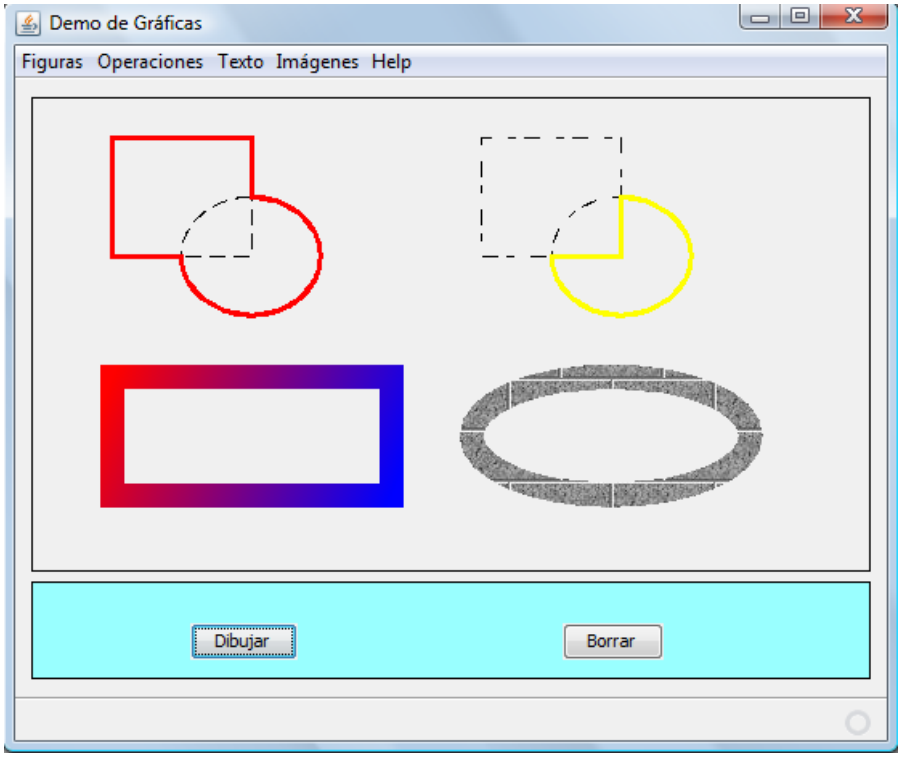

Figura 13.49 Tipos de Contorno

Transformaciones de Coordenadas

La API de java 2D nos permite transladar, rotar, escalar o deformar el sistema coordenado. Por lo general es más facil transformar el sistema coordenado que calcular las coordenadas de todos los puntos. Los métodos de la clase `Graphics2D` que nos permiten hacer esas transformaciones se muestran en la tabla 13.32.

Tabla 13.32 Métodos de la Clase `Graphics2D` para Transformar Coordenadas.

`public abstract AffineTransform` **`getTransform`**`()` Regresa una copia de la transformación actual en el contexto de `Graphics2D`.
`public abstract void` **`setTransform`**`(AffineTransform Tx)` Establece la transformación en el contexto de `Graphics2D`.
`public abstract void` **`translate`**`(double tx, double ty)` Le concatena a la transformación actual en el contexto de `Graphics2D`, la transformación de translación. Todas las coordenadas empleadas en las operaciones de despliegue en este contexto gráfico serán relativas a este nuevo origen (x, y).

Tabla 13.32 Métodos de la Clase `Graphics2D` para Transformar Coordenadas. Cont.

`public abstract void` **`rotate`**`(double theta)` Le concatena a la transformación actual en el contexto de `Graphics2D`, la transformación de rotación. Todas las coordenadas empleadas en las operaciones de despliegue en este contexto gráfico estarán giradas por el ángulo `theta` en radianes relativas al origen anterior.
`public abstract void` **`scale`**`(double sx, double sy)` Le concatena a la transformación actual en el contexto de `Graphics2D`, la transformación de rescalamiento. Todas las coordenadas empleadas en las operaciones de despliegue en este contexto gráfico estarán escaladas de acuerdo a los factores de escala relativos a la escala anterior.
`public abstract void` **`shear`**`(double shx, double shy)` Le concatena a la transformación actual en el contexto de `Graphics2D`, la transformación de deformación. Todas las coordenadas empleadas en las operaciones de despliegue en este contexto gráfico estarán deformadas de acuerdo a los multiplicadores especificados relativos a la posición anterior. `shx` es el multiplicador por el que las coordenadas son recorridas en la dirección positiva del eje X como una función de de la coordenada Y. `shy` es el multiplicador por el que las coordenadas son recorridas en la dirección positiva del eje Y como una función de de la coordenada X.

Ejemplos sobre Transformaciones

El código del método `transladaFiguras()` dibuja diferentes translaciones de figuras:

Operaciones.java

```
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * translaciones de figuras
     * @param lienzo Panel sobre el que se dibujan translaciones de
     * figura
     */
    public static void transladaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double ancho = d.width / 4;
        double alto = d.height / 4;
        double x = 50;
        double y = 25;

        // Establece el tipo de contorno
        BasicStroke bs = new BasicStroke(2);
        g2.setStroke(bs);
```

```
        // Establece el color del contorno
        g2.setPaint(Color.blue);

        // Crea una figura rectangular
        Rectangle2D rect = new Rectangle2D.Double(x, y, ancho, alto);
        // Crea de una figura eliptica
        Ellipse2D.Double elipse = new Ellipse2D.Double(x, y, ancho,
                                                       alto);

        // Establece el origen inicial
        g2.translate(10, 10);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Translada el origen
        g2.translate(2*ancho, 0);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Translada el origen
        g2.translate(0, 2*alto);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);
    }

    /**
     * Este método estático dibuja unos ejes coordenados en el contexto
     * g2
     * @param g2 Contexto de Graphics2D en el que se dibujan los ejes
     * @param ancho Longitud del eje X
     * @param alto Longitud del eje Y
     */
    public static void dibujaEjes(Graphics2D g2, double ancho,
                                  double alto) {
        BasicStroke bs1 = new BasicStroke(2, BasicStroke.CAP_BUTT,
                                          BasicStroke.JOIN_BEVEL, 0,
                                          new float[] { 5, 5}, 0);
        // Crea los ejes coordenados
        Line2D ejeX = new Line2D.Double(-10, 0, ancho, 0);
        Line2D ejeY = new Line2D.Double(0, -10, 0, alto);
```

```
        // Guarda los valores de Paint y Stroke actuales
        Paint paint = g2.getPaint();
        Stroke stroke = g2.getStroke();

        // Establece los nuevos valores de Paint y Stroke
        g2.setStroke(bs1);
        g2.setPaint(Color.black);

        // Dibuja los ejes coordenados
        g2.draw(ejeX);
        g2.draw(ejeY);

        // Reestablece los valores de Paint y Stroke originales
        g2.setStroke(stroke);
        g2.setPaint(paint);
    }
```

La figura 13.50 muestra las translaciones de figuras desplegada por el método `transladaFiguras()`.

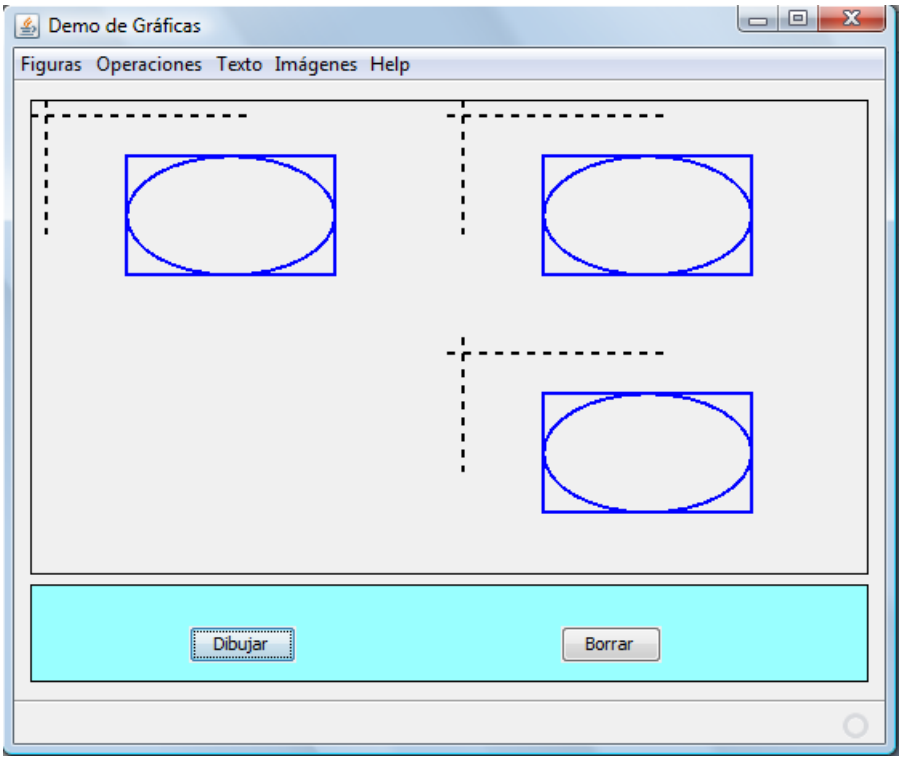

Figura 13.50 Translaciones

El código del método `rotaFiguras()` dibuja diferentes translaciones de figuras:

Operaciones.java

```
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * rotaciones de figuras
     * @param lienzo Panel sobre el que se dibujan rotaciones de figura
     */
    public static void rotaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double ancho = d.width / 4;
        double alto = d.height / 4;

        // Establece la posición inicial del origen
        g2.translate(2*ancho, 2*alto);

        // Establece el tipo de contorno
        BasicStroke bs = new BasicStroke(2);
        g2.setStroke(bs);

        // Establece el color del contorno
        g2.setPaint(Color.blue);

        // Crea una figura rectangular
        Rectangle2D.Double rect = new Rectangle2D.Double(-ancho/2,
                                                  -alto/2, ancho, alto);
        // Crea de una figura eliptica
        Ellipse2D.Double elipse = new Ellipse2D.Double(-ancho/2, -alto/2,
                                                       ancho, alto);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo
        g2.draw(rect);
        g2.draw(elipse);

        // Rota el origen
        g2.rotate(Math.PI/4);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo
        g2.draw(rect);
        g2.draw(elipse);

        // Rota el origen
        g2.rotate(Math.PI/4);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);
```

```
        // Dibuja el rectangulo
        g2.draw(rect);
        g2.draw(elipse);
    }
```

La figura 13.51 muestra rotaciones de figuras desplegada por el método `rotaFiguras()`.

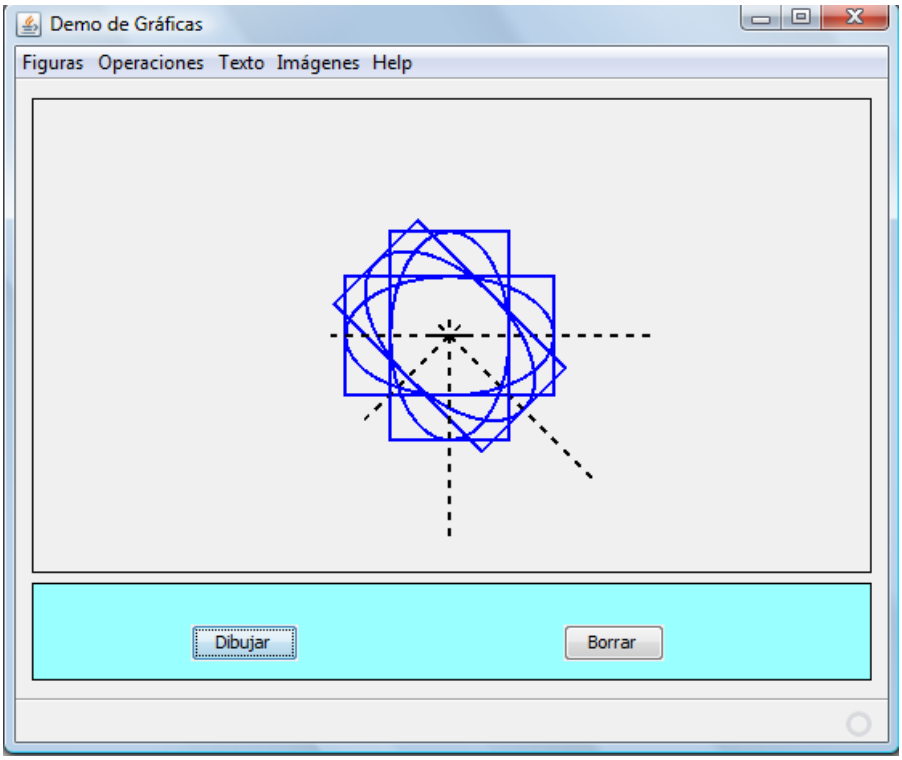

Figura 13.51 Rotaciones

El código del método `escalaFiguras()` dibuja diferentes escalamientos de figuras:

Operaciones.java

```
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * figuras escaladas
     * @param lienzo Panel sobre el que se dibujan figuras escaladas
     */
    public static void escalaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();
```

```
        double ancho = d.width / 4;
        double alto = d.height / 4;
        double x = 50;
        double y = 25;

        // Establece el tipo de contorno
        BasicStroke bs = new BasicStroke(2);
        g2.setStroke(bs);

        // Establece el color del contorno
        g2.setPaint(Color.blue);

        // Crea una figura rectangular
        Rectangle2D.Double rect = new Rectangle2D.Double(x, y, ancho,
                                                         alto);
        // Crea de una figura eliptica
        Ellipse2D.Double elipse = new Ellipse2D.Double(x, y, ancho,
                                                       alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Escala la imagen
        g2.scale(1.5, 1.5);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Escala la imagen
        g2.scale(1.5, 1.5);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);
    }
```

La figura 13.52 muestra escalamientos de figuras desplegada por el método `escalaFiguras()`.

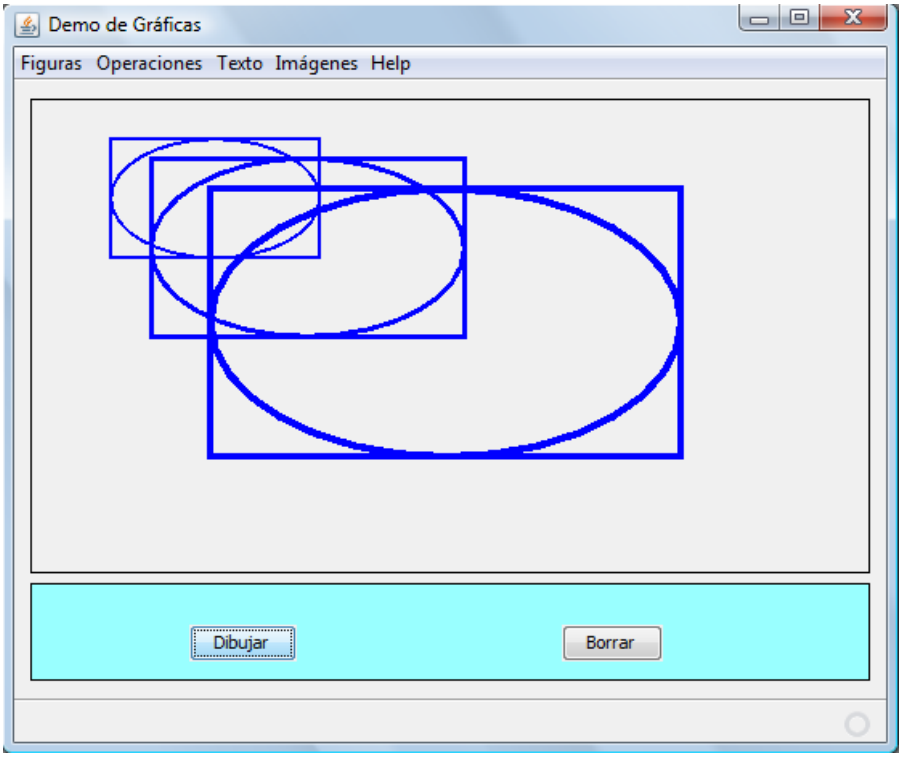

Figura 13.52 Escalamientos

El código del método `escalaFiguras()` dibuja diferentes escalamientos de figuras:

Operaciones.java

```
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * deformaciones de figuras
     * @param lienzo Panel sobre el que se dibujan translaciones de
     * figura
     */
    public static void deformaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double ancho = d.width / 4;
        double alto = d.height / 4;
        double x = 50;
        double y = 10;

        // Establece el tipo de contorno
        BasicStroke bs = new BasicStroke(2);
        g2.setStroke(bs);

        // Establece el color del contorno
```

```
        g2.setPaint(Color.blue);

        // Crea una figura rectangular
        Rectangle2D rect = new Rectangle2D.Double(x, y, ancho, alto);
        // Crea de una figura eliptica
        Ellipse2D.Double elipse = new Ellipse2D.Double(x, y, ancho,
                                                       alto);

        // Establece el origen inicial
        g2.translate(10, 10);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Translada el origen
        g2.translate(2*ancho, 0);

        // Deforma horizontalmente
        g2.shear(0.5, 0);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Quita la deformación horizontal
        g2.shear(-0.5, 0);

        // Translada el origen
        g2.translate(-2*ancho, 2*alto);

        // Deforma verticalmente
        g2.shear(0, 0.2);

        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);

        // Quita la deformación vertical
        g2.shear(0, -0.2);

        // Translada el origen
        g2.translate(2*ancho, 0);

        // Deforma horizontal y verticalmente
        g2.shear(0.5, 0.2);
```

```
        // Dibuja los ejes coordenados
        dibujaEjes(g2, ancho, alto);

        // Dibuja el rectangulo y la elipse
        g2.draw(rect);
        g2.draw(elipse);
    }
```

La figura 13.52 muestra las deformaciones de figuras desplegada por el método `deformaFiguras()`.

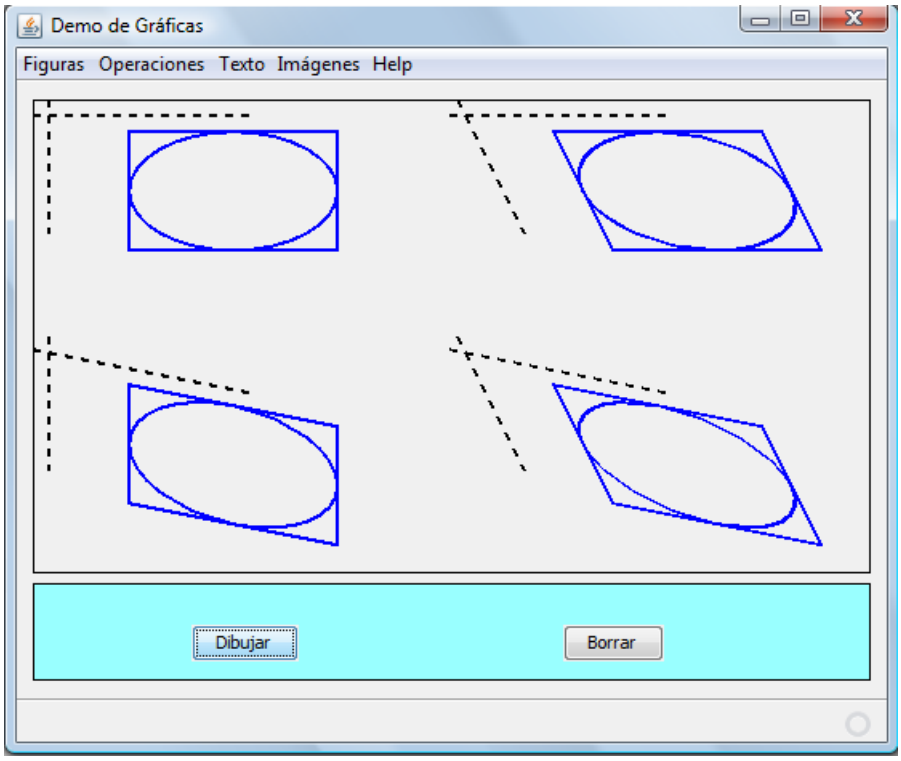

Figura 13.53 Deformaciones

Recorte de Figuras

El recorte de figuras es una técnica que limita la extensión de un dibujo. `Graphics2D` puede recortar los dibujos usando cualquier figura. De hecho `Graphics2D` que mantiene la figura de recorte como parte de su estado.La figura 13.54 muestra un dibujo simple en tres estados: Sin recorte, recortado a un triángulo y recortado a un círculo. Los métodos de la clase `Graphics2D` que nos permiten hacer esas transformaciones se muestran en la tabla 13.33.

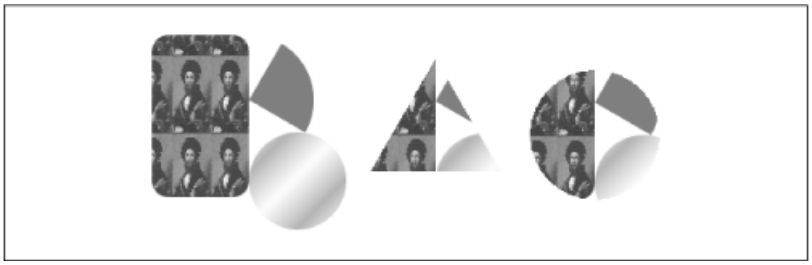

Figura 13.54 Recortado

Tabla 13.33 Métodos de la Clase `Graphics2D` para Recortar Figuras.

`public abstract void` **`clip`**`(Shape s)` Intersecta la fiura de recorte actual con el interior de la figura del parámetro y establece el resultado como la nueva figura de recorte
`public abstract Shape` **`getClip`**`()` Obtiene la figura de recorte actual.
`public abstract void` **`setClip`**`(Shape clip)` Establece la figura de recorte a la figura del parámetro. No todas las figuras que implementan la interfaz `Shape` pueden usarse como figuras de recorte. Las unicas figuras que se garantizan como figuras de recorte son las obtenidas por el método **`getClip`**`()` o mediante objetos del tipo `Rectangle`.

El código del método `recortaFiguras()` dibuja diferentes escalamientos de figuras:

Operaciones.java

```
    /**
     * Este metodo dibuja sobre el panel de su parametro
     * figuras recortadas
     * @param lienzo Panel sobre el que se dibujan figuras recortadas
     */
    public void recortaFiguras(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        Area area1, area2;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        double ancho = d.width / 3;
        double alto = d.height / 3;
        double x = 50;
        double y = 25;

        // Carga la imagen a ser usada como textura
        BufferedImage bi;
        try {
            bi = getImage("fondo.jpg");
        } catch (IOException ioe) {
```

```
            JOptionPane.showMessageDialog(lienzo,
                                          "Error al cargar imagen");
            return;
        }
        // Crea el rectangulo ancla del tamaño de la imagen
        Rectangle2D tr = new Rectangle2D.Double(0, 0, bi.getWidth(),
                                                bi.getHeight());
        TexturePaint tp = new TexturePaint(bi, tr);

        // establece el tipo de relleno
        g2.setPaint(tp);

        // Crea una figura rectangular
        Rectangle2D rect1 = new Rectangle2D.Double(x, y, ancho, alto);

        // Dibuja el rectangulo
        g2.fill(rect1);

        x += 100 + ancho;

        // Crea una figura rectangular
        Rectangle2D rect2 = new Rectangle2D.Double(x, y, ancho, alto);

        // Crea la figura de recorte
        Ellipse2D elipse1 = new Ellipse2D.Double(x, y, ancho, alto);
        g2.clip(elipse1);

        // Dibuja el rectangulo
        g2.fill(rect2);

        // Elimina el recorte
        g2.setClip(null);

        x = 50;
        y = 50 + alto;

        // Crea una figura rectangular
        Rectangle2D rect3 = new Rectangle2D.Double(x, y, ancho, alto);

        // Crea la figura de recorte
        Ellipse2D elipse2 = new Ellipse2D.Double(x, y, ancho, alto);
        g2.clip(elipse2);

        // Crea otra figura de recorte
        Rectangle2D rect4 = new Rectangle2D.Double(x, y, alto, alto);
        g2.clip(rect4);

        // Dibuja el rectangulo
        g2.fill(rect3);
    }
```

La figura 13.55 muestra las figuras recortadas desplegadas por el método `recortaFiguras()`.

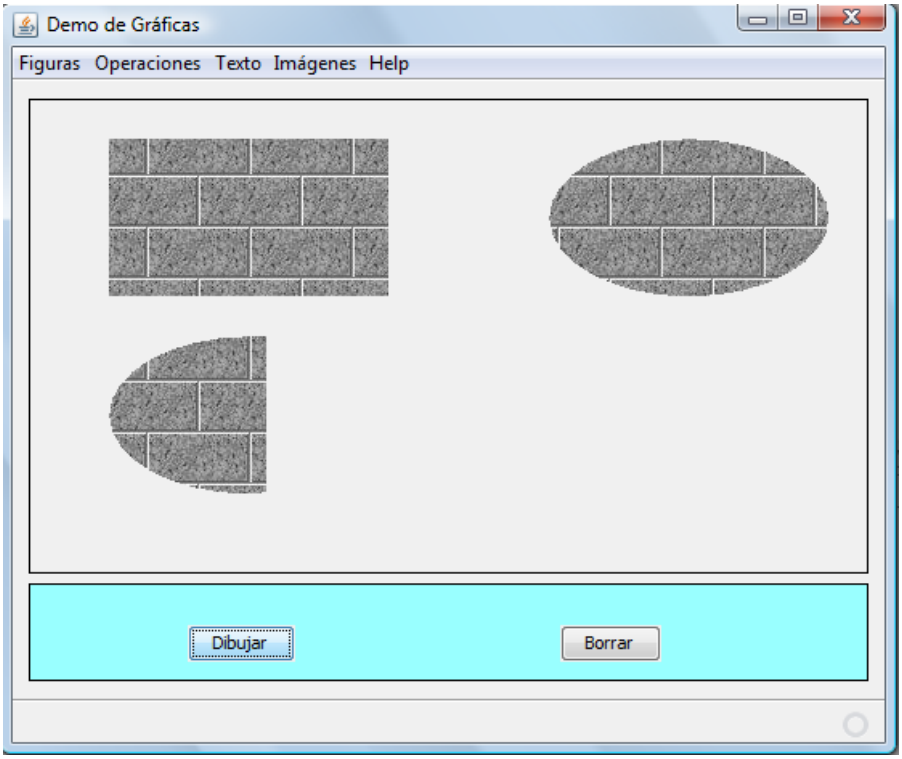

Figura 13.55 Deformaciones

Sugerencias de Despliegue

La API 2D de java permite el control de la calidad de su despliegue mediante el uso de **Sugerencias de Despliegue**. Son sugerencias ya que no controlan la máquina de despliegue sólo son sugerencias de preferencias. La máquina de despliegue puede seguir las sugerencias o ignorarlas. Esas sugerencias están encapsuladas en la clase `RenderingHints`.

la máquina de despliegue, Graphics2D sabe más de una forma de hacer las cosas. Por ejemplo, al dibujar una figura puede utilizar suavizamiento, que se ve bien, o puede no usarlo lo que lo hace rápido. Este tipo de compromisos entre calidad y velocidad ocurren en otras partes del proceso de despliegue. Las sugerencias de despliegue nos permiten tener un poco de control en esas decisiones. Las sugerencias de despliegue también se utilizan en el procesamiento de imágenes.

Las sugerencias de despliegue se especifican usando un esquema *llave – valor*. Los diferentes valores que una llave puede tomar son constantes simbólicas. Las llaves y los posibles valores que una llave puede tomar se muestran en la tabla 13.34.

Tabla 13.34. Sugerencias de Despliegue.

Sugerencia	Llave	Posibles valores

Controla si se va usar suavizamiento para el desplegado de figuras y texto	`KEY_ANTIALIASING`	`VALUE_ANTIALIAS_ON`
		`VALUE_ANTIALIAS_OFF`
		`VALUE_ANTIALIAS_DEFAULT`

Le dice a `Graphics2D` si se prefiere un dibujado rápido o el mejor dibujado.	`KEY_RENDERING`	`VALUE_RENDER_QUALITY`
		`VALUE_RENDER_SPEED`
		`VALUE_RENDER_DEFAULT`

Le dice a `Graphics2D` si se va a usar colores aproximados en dispositivos que soportan un limitado número de colores.	`KEY_DITHERING`	`VALUE_DITHER_DISABLE`
		`VALUE_DITHER_ENABLE`
		`VALUE_DITHER_DEFAULT`

Controla si los colores serán corregidos para un determinado dispositivo usando un perfil de dispositivo.	`KEY_COLOR_RENDERING`	`VALUE_COLOR_RENDER_QUALITY`
		`VALUE_COLOR_RENDER_SPEED`
		`VALUE_COLOR_RENDER_DEFAULT`

Controla si en las fuentes se usaran mediciones con enteros o fotantes. Las medidas flotantes permiten mayor precisión en el posicionamiento del texto.	`KEY_FRACTIONALMETRICS`	`VALUE_FRACTIONALMETRICS_ON`
		`VALUE_FRACTIONALMETRICS_OFF`
		`VALUE_FRACTIONALMETRICS_DEFAULT`

Controla el suavizamiento del texto en forma separada del de las otras figuras.	`KEY_TEXT_ANTIALIASING`	`VALUE_TEXT_ANTIALIAS_ON`
		`VALUE_TEXT_ANTIALIAS_OFF`
		`VALUE_TEXT_ANTIALIAS_DEFAULT`

Utiliza el algoritmo empleado para transformar las imágenes.	`KEY_INTERPOLATION`	`VALUE_INTERPOLATION_BICUBIC`
		`VALUE_INTERPOLATION_BILINEAR`
		`VALUE_INTERPOLATION_NEAREST_NEIGHBOR`

Determina como se calcularán los valores alfa.	`KEY_ALPHA_INTERPOLATION`	`VALUE_ALPHA_INTERPOLATION_QUALITY`
		`VALUE_ALPHA_INTERPOLATION_SPEED`
		`VALUE_ALPHA_INTERPOLATION_DEFAULT`

Los métodos de `Graphics2D` que nos permiten establecer o cuestionar el valor o valores para esas sugerencias de despliegue se muestran en la tabla 13.35.

Tabla 13.35. Métodos de la Clase `Graphics2D` para Manejar las Sugerencias de Despliegue.

`public abstract void` **`setRenderingHint`**`(RenderingHints.Key hintKey, Object hintValue)` Establece el valor de una sola sugerencia de despliegue para los algoritmos de despliegue. Establece el valor de la sugerencia `hintKey` al valor de `hintValue`.
`public abstract Object` **`getRenderingHint`**`(RenderingHints.Key hintKey)` Regresa el valor de una sola sugerencia de despliegue para los algoritmos de despliegue. Regresa el valor de la sugerencia `para la llave hintKey`.
`public abstract void` **`setRenderingHints`**`(Map<?,?> hints)` Reemplaza los valores de todas las sugerencias para los algoritmos de despliegue con las sugerencias del parámetro `hints`. Los valores existentes de todas las sugerencias de despliegue se descartan y son reemplazados por las sugerencias especificadas en el mapa.

Tabla 13.35. Métodos de la Clase `Graphics2D` para Manejar las Sugerencias de Despliegue. Cont.

`public abstract RenderingHints` **`getRenderingHints`**`()` Regresa los valores de las sugerencias para los algoritmos de despliegue.
`public abstract void` **`addRenderingHints`**`(Map<?,?> hints)` Establece los valores de un número arbitrario de sugerencias para los algoritmos de despliegue. Solo los valores de las sugerencias que se encuentran presente en el mapa del parámetro son modificados. Las otras sugerencias que no están en el mapa del parámetro no se ven modificados.

Para establecer que se use suavizamiento al desplegar las figuras podemos hacer lo siguiente:

```
public void recortaFiguras(JPanel lienzo) {
    Graphics g = lienzo.getGraphics();
    Graphics2D g2 = (Graphics2D) g;

    g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);
    ...
   }
```

Si deseamos establecer varias sugerencias de despliegue al mismo tiempo, podemos podemos podemos guardarlos en una instancia de `RenderingHints` o cualquier otra clase que implemente la interfaz `Map`, como se muestra en la siguiente pieza de código.

```
public void recortaFiguras(JPanel lienzo) {
    Graphics g = lienzo.getGraphics();
    Graphics2D g2 = (Graphics2D) g;

    RenderingHints hints = new RenderingHints(
       RenderingHints.KEY_ANTIALIASING,
       RenderingHints.VALUE_ANTIALIAS_ON);

    hints.add(RenderingHints.KEY_RENDERING,
              RenderingHints.VALUE_RENDER_ON);

    g2.setRenderingHints(hints);
    ...
}
```

Texto

La forma más simple de dibujar texto y controlar su apariencia es mediante los métodos de la clase `Graphics2D` que se muestran en la tabla 13.36.

Tabla 13.36 Métodos para Controlar y Dibujar Texto de la Clase `Graphics2D`.

`public abstract void` **`drawString`**`(String str, int x, int y)`
`public abstract void` **`drawString`**`(String str, float x, float y)`
 	
Dibujan el texto especificado por el parámetro `str`, usando los atributos de texto del contexto gráfico. La línea base del primer carácter se dibuja en las coordenadas (x, y).

`public abstract void` **`drawString`**`(AttributedCharacterIterator iterator, int x, int y)`
`public abstract void` **`drawString`**`(AttributedCharacterIterator iterator, float x, float y)`
 	
Dibujan el texto especificado por el iterador `iterator`, usando sus atributos de acuerdo a la especificación de la clase `TextAttribute`. La línea base del primer carácter se dibuja en las coordenadas (x, y).

`public abstract Font` **`getFont`**`()` Regresa la fuente actual en el contexto gráfico.
`public abstract void` **`setFont`**`(Font font)` Establece la fuente actual en el contexto gráfico. Todas las operaciones subsecuentes operaciones con texto usan esa fuente.

`public FontMetrics` **`getFontMetrics`**`()`
`public abstract FontMetrics` **`getFontMetrics`**`(Font f)`
 	
La primera versión obtiene las métricas para la fuente actual.
La segunda versión obtiene las métricas para la fuente del parámetro.

`public abstract FontRenderContext` **`getFontRenderContext`**`()` Obtiene el contexto de despliegue de la fuente dentro del contexto de **`Graphics2D`**. El objeto de tipo `FontRenderContext` encapsula las sugerencias de despliegue así como la información especial del dispositivo destino como los puntos por pixel. Esta información es util cuando se requiere posicionar texto.

Línea Base

La línea base de una cadena es una línea sobre la que se dibujan los caracteres de la cadena, figura 13.56.

Figura 13.56. Línea Base de una Cadena.

Métricas de Fuente

Las métricas de fuente es un conjunto de mediciones que describen la fuente. la figura 13.57 muestra algunas de esas métricas. La altura total de una fuente es la suma de la separación entre líneas (leading), el ascenso (ascent) y el descenso

(descent) de la fuente. El ascenso mide la altura del carácter por encima de la línea base. El descenso mide la distancia de algunos caracteres por debajo de la línea base.

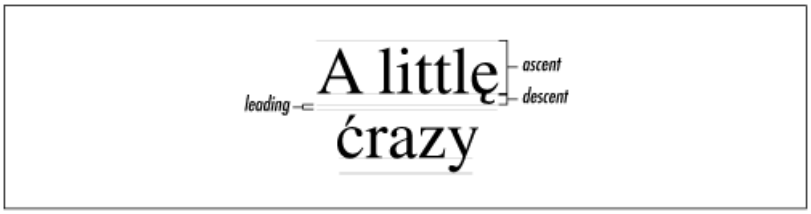

Figura 13.57. Métricas de una Fuente.

El siguiente código ilustra el uso del método `drawString()` para dibujar texto.

Texto.java

```
package figuras;

import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import javax.swing.JPanel;

/**
 * Esta clase muestra diferentes operaciones para dibujar texto
 * @author mdomitsu
 */
public class Texto {
    /**
     * Este metodo estatico dibuja texto sobre el panel de su parametro
     * @param lienzo Panel sobre el que se dibuja texto
     */
    public static void dibujaTexto(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        float x, y;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        // Crea un tipo de fuente
        Font font = new Font("Serif", Font.PLAIN, 96);
        // Establece el tipo de fuente
        g2.setFont(font);

        // Dibuja una cadena de texto
        g2.drawString("Hola mundo", 20, 100);

        // Establece la sugerencia de usar suavizamiento de orillas
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);
```

```
        // Dibuja una cadena de texto
        g2.drawString("Hola mundo", 20, 200);
    }
}
```

La figura 13.58 muestra el texto desplegado por el método `dibujaTexto()`.

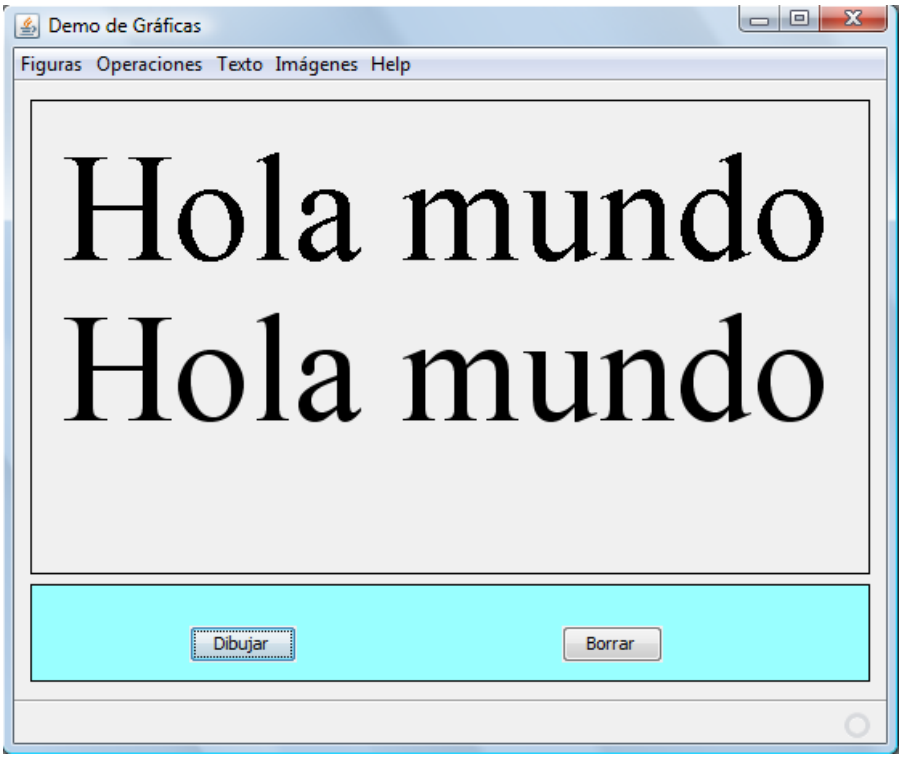

Figura 13.58 Dibujado de Texto

El siguiente código ilustra la forma de rotar texto.

Texto.java

```
    /**
     * Este metodo estatico dibuja sobre el panel de su parametro
     * texto rotado
     * @param lienzo Panel sobre el dibuja texto rotado
     */
    public static void rotaTexto(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        float x, y;

        // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        // Crea un tipo de fuente
        Font font = new Font("Serif", Font.PLAIN, 48);
        // Establece el tipo de fuente
```

```
        g2.setFont(font);

        // Establece la sugerencia de usar suavizamiento de orillas
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                            RenderingHints.VALUE_ANTIALIAS_ON);

        // Establece el origen de coordenadas
        g2.translate(80, 250);

        // Dibuja una cadena de texto
        g2.drawString("Hola mundo", 0, 0);

        // Gira el sistema coordenado 45 grados
        g2.rotate(-Math.PI/4);

        // Dibuja una cadena de texto
        g2.drawString("Hola mundo", 0, 0);

        // Gira el sistema coordenado otros 45 grados
        g2.rotate(-Math.PI/4);

        // Dibuja una cadena de texto
        g2.drawString("Hola mundo", 0, 0);
    }
```

La figura 13.59 muestra el texto desplegado por el método `rotaTexto()`.

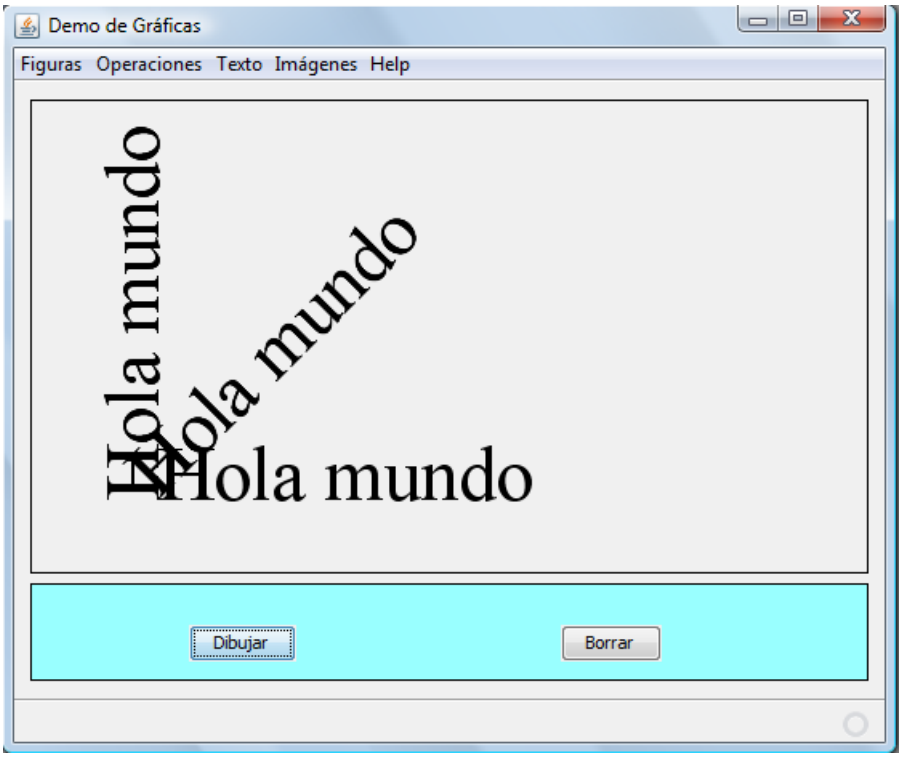

Figura 13.59 Rotado de Texto

Caracteres, Glifos y Fuentes

Un carácter es un símbolo empleado para representar una letra, un dígito o un símbolo de puntuación.
Un glifo es una figura que se usa para desplegar un carácter o secuencia de caracteres. A veces un glifo representa un carácter pero por lo general no hay una relación uno a uno entre los glifos y los caracteres. Por ejemplo la sílaba fi formadas por dos caracteres puede representarse por un solo glifo.

Una fuente encapsula la colección de glifos necesarios para desplegar un conjunto selecto de caracteres asi como las tablas necesarias para mapear secuencias de carateres a sus correspondientes secuencias de glifos.

Fuentes Físicas y Fuentes Lógicas

La plataforma de java distingue entre dos tipos de fuentes: fuentes físicas y fuentes lógicas.

las fuentes físicas son las bibliotecas actuales de las fuentes con los datos de los glifos y las tablas para mapear de las secuencias de caracteres a las secuencias de glifos, usando tecnologías como *True Type* o *PostScript Type 1*. Todas las implementaciones de Java deben soportar las fuentes *True Type*, aunque cada implementación puede soportar otras tecnologías de fonts. Las fuentes físicas usan nombres como Helvetica, Palatino, HonMincho, etc.

Las fuentes lógicas son las cinco familias definidas por la plataforma de Java y que deben ser soportadas por cualquier ambiente de ejecución de Java: **Serif**, **SansSerif**, **Monospaced**, **Dialog**, and **DialogInput**. Estas fuentes no son bibliotecas reales, en lugar de ello los nombre lógicos son mapeados a los nombres físicos por el ambiente de ejecución de Java. Este mapeado es dependiente de la implementación y localización. Por lo general cada nombre lógico se mapea se mapea a varias fuentes físicas para cubrir un rango amplio de caracteres.

Caras de fuentes y Nombres

Una fuente puede tener varias caras como heavy, medium, oblique, gothic and regular. Todas esas caras tienen un diseño tipográfico similar.

Hay tres nombres que se pueden obtener de un objeto de tipo `Font`. El nombre lógico de la fuente es el nombre usado para construir la fuente. El nombre de la

cara de la fuente o simplemente, el nombre de la fuente, es el nombre de una cara de una fuente, por ejemplo Hevetic bold. El nombre de de familia es el nombre de la familia de la fuente que determina el diseño tipográfico a lo largo de varias caras como Helvetica.
La clase `Font` representa una instancia de de una cara de fuente de una colección de caras de fuente que están presentes en en los recursos del sistema del sistema huesped, por ejemplo Arial Bold y Courier bold. Puede haber varios objetos del tipo `Font` asociados a una cara de fuente, cada uno diferiendo en tamaño, estilo, transformación y características de fuente.

La tabla 13.36 muestra algunos de los métodos de la clase `Font`.

Tabla 13.36 métodos de la clase `Font`.

`public` **`Font`**`(String name, int style, int size)` Crea una fuente para el nombre, estilo y tamaño dados por sus parámetros. El nombre de la fuente puede ser el nombre de la cara o nombre lógico. Los estilos de la fuente pueden ser cualquiera de kas siguientes constantes definidas en esta clase: `PLAIN`, `ITALIC`, `BOLD` o `ITALIC | BOLD`. El tamaño de la fuente debe darse en puntos tipográficos. Un punto es 1/72 pulgadas. Si la fuente especificada no existe, se regresará la fuente por omisión.

`public Font` **`deriveFont`**`(float size)`
`public Font` **`deriveFont`**`(int style)`
`public Font` **`deriveFont`**`(int style, float size)`
 	
Crea una nueva fuente, replicando esta fuente y aplicando un nuevo tamaño, estilo o ambos.

`public String` **`getFamily`**`()` Regresa el nombre de la familia de esta fuente.
`public String` **`getFontName`**`()` Regresa el nombre de la cara de esta fuente.
`public String` **`getName`**`()` Regresa el nombre lógico de esta fuente.

Familia de Fuentes Instaladas en el Sistema

La clase `GraphicsEnvirement` tiene el método `getAvailableFontFamilyNames()` que nos regresa un arreglo una lista de las familias de fuentes instaladas en el sistema. Su sintaxis me muestra en la tala 13.37.

Tabla 13.36 método `getAvailableFontFamilyNames()`de la Clase `GraphicsEnvirement`.

`public abstract String[]` **`getAvailableFontFamilyNames`**`()` Regresa un arreglo con los nombres de todas las familias de fuentes en el sistema.

Ejemplo sobre Familia de Fuentes Instaladas en el Sistema

El siguiente código ilustra el método que invoca al cuadro de diálogo usado para seleccionar una fuente de las instaladas en el sistema.

Texto.java

```
    /**
     * Este metodo despliega un cuadro de dialogo para seleccioanar
     * una fuente de las instaladas en el sistema.
     *
     * @param frame frame sobre el que se desplegará el cuadro de dialogo
     */
    public void fuentes(JFrame frame) {
        Font font = new Font("Serif", Font.PLAIN, 12);

        DlgSelectorFuentes dlg = new DlgSelectorFuentes(frame,
                "Selector de fuentes", true, font);
    }
```

DlgSelectorFuentes.java

```
/*
 * DlgSelectorFuentes.java
 *
 * Created on 25/10/2011, 12:55:32 PM
 */
package grafica2;

import figuras.Lineas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.GraphicsEnvironment;
import java.awt.Point;
import javax.swing.DefaultComboBoxModel;
import javax.swing.JPanel;

/**
 * Este cuadro de dialogo permite seleccionar una fuente
 * @author mdomitsu
 */
public class DlgSelectorFuentes extends javax.swing.JDialog {

    /**
     * Constructor del cuadro de dialogo
     * @param parent Frame sobre el que se dibuja el cuadro de dialogo
     * @param title Titulo del cuadro de dialogo
     * @param modal true si permite acceder fuera de los limites del
     * cuadro de dialogo, false en caso contrario
     * @param font Fuente seleccionada en el cuadro de dialogo
     */
    public DlgSelectorFuentes(java.awt.Frame parent, String title,
            boolean modal, Font font) {
```

```
        super(parent, title, modal);
        this.font = font;
        initComponents();
        // establece la fuente inicial
        font = new Font(fuenteSel, estiloSel, tamanosel);

        // Centra el cuadro de dialogo sobre el frame
        centraCuadroDialogo(parent);
        // Muestra el cuadro de diálogo
        setVisible(true);
    }

    /**
     * Este método regresa una instancia de la clase DefaultComboMoxModel
     * formada a partir de los nombres de las familias de fuentes
     * instaladas en el sistema.
     */
    private DefaultComboBoxModel
        getNombresFamiliaFuentesComboBoxModel() {
        GraphicsEnvironment gEnv =
                GraphicsEnvironment.getLocalGraphicsEnvironment();

        // Obtiene los nombres de las familias de fuentes instaladas
        // en el sistema
        String[] nombresFamiliaFuentes =
           gEnv.getAvailableFontFamilyNames();
        // Crea la instancia de la clase DefaultComboMoxModel
        // formada a partir de los nombres de las familias de fuentes
        // instaladas en el sistema
        DefaultComboBoxModel dcbmNombresFamiliaFuentes =
           new DefaultComboBoxModel(nombresFamiliaFuentes);

        return dcbmNombresFamiliaFuentes;
    }

    /**
     * Este método centra el cuadro de dialogo sobre la ventana de la
     * aplicación.
     * @param parent Ventana sobre la que aparecerá el cuadro de diálogo
     */
    private void centraCuadroDialogo(java.awt.Frame parent) {
        // Obtiene el tamaño y posición de la ventana de la aplicación
        Dimension frameSize = parent.getSize();
        Point loc = parent.getLocation();

        // Obtiene el tamaño del cuadro de diálogo
        Dimension dlgSize = getPreferredSize();

        // Centra el cuadro de diálogo sobre la ventana padre
        setLocation((frameSize.width - dlgSize.width) / 2 + loc.x,
                (frameSize.height - dlgSize.height) / 2 + loc.y);
    }

    ...

    /**
```

```
     * Método oyente del botón botonAceptar
     * @param evt Evento al que escucha
     */
    private void btnAceptarActionPerformed(
       java.awt.event.ActionEvent evt) {
       // Destruye el cuadro de díalogo
       dispose();
    }

    /**
     * Metodo oyente de la caja combinada para seleccionar la familia de
     * fuente a usar en la fuente
     * @param evt Evento al que escucha
     */
    private void cbFuentesActionPerformed(
        java.awt.event.ActionEvent evt) {
        // Obtiene la fuente seleccionada en la caja combinada
        fuenteSel = (String) cbFuentes.getSelectedItem();

        // Crea una fuente con la familia seleccionada
        font = new Font(fuenteSel, estiloSel, tamanosel);

        // Dibuja una cadena con la fuente seleccionada
        dibujaTextoDemoFuentes(panelLienzoFuentes);
    }

    /**
     * Metodo oyente de la caja combinada para seleccionar el estilo de
     * fuente a usar en la fuente
     * @param evt Evento al que escucha
     */
    private void cbEstilosActionPerformed(
        java.awt.event.ActionEvent evt) {
        // Obtiene el estilo de la fuente de la caja combinada
        estiloSel = cbEstilos.getSelectedIndex();

        // Crea una fuente con la familia seleccionada
        font = new Font(fuenteSel, estiloSel, tamanosel);

        // Dibuja una cadena con la fuente seleccionada
        dibujaTextoDemoFuentes(panelLienzoFuentes);
    }

    /**
     * Metodo oyente del spinner para seleccionar el tamano de
     * fuente a usar en la fuente
     * @param evt Evento al que escucha
     */
    private void spTamanosStateChanged(
        javax.swing.event.ChangeEvent evt) {
        // Obtiene el tamano de la fuente del spinner
        String tamano = spTamanos.getModel().getValue().toString();
        tamanosel = Integer.parseInt(tamano);

        // Crea una fuente con la familia seleccionada
        font = new Font(fuenteSel, estiloSel, tamanosel);
```

```
        // Dibuja una cadena con la fuente seleccionada
        dibujaTextoDemoFuentes(panelLienzoFuentes);
    }

    /**
     * Metodo oyente que se invoca al desplegarse el cuadro de dialogo
     * @param evt Evento al que escucha
     */
    private void formWindowOpened(java.awt.event.WindowEvent evt) {
        // Dibuja una cadena con la fuente seleccionada
        dibujaTextoDemoFuentes(panelLienzoFuentes);
    }

    /**
     * Dibuja una cadena con la fuente seleccionada
     * centrada horizontalmente sobre el panel del parametro
     * @param lienzo Panel sobre el que se dibuja la cadena
     */
    public void dibujaTextoDemoFuentes(JPanel lienzo) {
        Graphics g = lienzo.getGraphics();
        Graphics2D g2 = (Graphics2D) g;
        String s = "Hola mundo";

        // Establece la fuente a usar
        g2.setFont(font);
        g2.setPaint(Color.black);

        // Borra el panel
        Lineas.borra(lienzo);

        // Obtiene una instancia de FontRenderContext que contiene
        // información para obtener el tamaño de una cadena
        FontRenderContext frc = g2.getFontRenderContext();
        // Obtiene el rectangulo que limita la cadena
        Rectangle2D bounds = g2.getFont().getStringBounds(s, frc);
        // Obtiene el ancho del rectangulo que limita la cadena
        float width = (float) bounds.getWidth();

         // Obtiene el tamaño del panel
        Dimension d = lienzo.getSize();

        // Calcula la posición horizontal para centrar el texto
        int x = (int)((float)d.width - width)/2;
        int y = (int) ((float)d.height - 50);

        // Dibuja la cadena de texto
        g2.drawString(s, x, y);
    }

    // Variables declaration - do not modify
    private javax.swing.JButton btnAceptar;
    private javax.swing.JComboBox cbEstilos;
    private javax.swing.JComboBox cbFuentes;
    private javax.swing.JLabel jLabel1;
    private javax.swing.JLabel jLabel2;
```

```
    private javax.swing.JLabel jLabel3;
    private javax.swing.JPanel panelLienzoFuentes;
    private javax.swing.JSpinner spTamanos;
    // End of variables declaration
    private String fuenteSel = "Dialog";
    private int estiloSel = 0;
    private int tamanosel = 12;
    private Font font;
}
```

La figura 13.60 muestra el cuadro de diálogo empleado para seleccionar una fuente de las instaladas en el sistema.

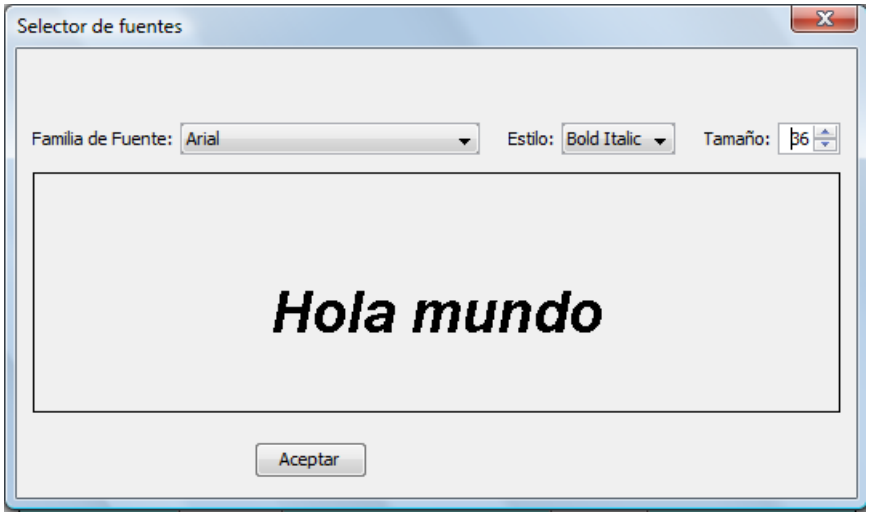

Figura 13.60 Cuadro de Diálogo para seleccionar una Fuente de las Instaladas en el Sistema