

JAVA 2D Graphics

Índice



Graphics

Graphics 2D

Clase “Graphics”

- *El primer enfoque (JDK 1.0) que abordaba la gestión de gráficos se basaba en el uso de la clase Graphics y sus métodos*

Pintar

```
drawLine(int x1,int y, int x2, int y2);  
drawRect(int x1,int y, int w, int h);  
drawOval(int x1,int y, int w, int h);  
fillRect(int x1,int y, int w, int h);  
fillOval(int x1,int y, int w, int h);  
...  
drawString(String s , int x, int y);  
drawImage(Image i , int x, int y,...);
```

Propiedades

```
setColor(Color c);  
setFont(Font f);
```

Clase "Graphics"

- > *Problema: no hay clases asociados a las formas geométricas*
 - *No se pueden editar las figuras ya pintadas*
- > *La única excepción es el caso de las imágenes y las cadenas de caracteres*
- > *Las propiedades modificables son sólo el color y el formato del texto (para los que sí existen clases)*

Índice

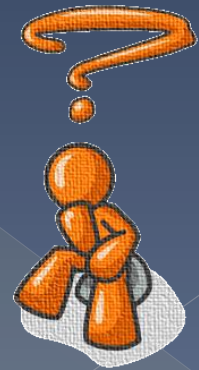


Graphics

Graphics 2D

JAVA 2D Graphics

- *Mejora (sustancialmente) el enfoque basado en la clase Graphics*
 - *Introduce clases asociadas a las formas geométricas*
 - *Incorpora nuevas formas y propiedades*



JAVA 2D Graphics

- > *Mejora (sustancialmente) el enfoque basado en la clase Graphics*
 - *Introduce clases asociadas a las formas geométricas*
 - *Incorpora nuevas formas y propiedades*
- > *Distingue entre:*
 - *Forma: Representa la geometría de la figura*
 - *Contexto: Atributos con los que se pintará*

JAVA 2D Graphics

- > *Mejora (sustancialmente) el enfoque basado en la clase Graphics*

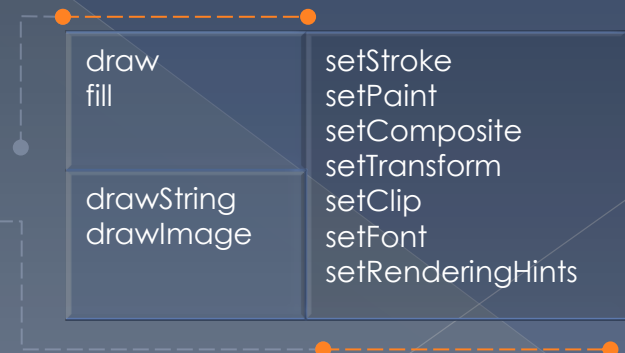
- *Introduce clases asociadas a las formas geométricas*
- *Incorpora nuevas formas y propiedades*

- > *Distingue entre:*

- *Forma: Representa la geometría de la figura*
- *Contexto: Atributos con los que se pintará*

- > *Incorpora una nueva clase **Graphics2D***

- *Hereda de Graphics*
- *Métodos para dibujar formas, imágenes o texto*
- *Métodos para cambiar atributos*
- *Ambos aceptan objetos como parámetros*



JAVA 2D Graphics

Cuando usábamos Graphics...

```
public void paint(Graphics g){  
    super.paint(g);  
  
    g.-----(--);  
  
}
```

> Incorpora una nueva clase *Graphics2D*

- *Hereda de Graphics*
- *Métodos para dibujar formas, imágenes o texto*
- *Métodos para cambiar atributos*
- *Ambos aceptan objetos como parámetros*

draw
fill

setStroke
setPaint
setComposite
setTransform

drawString
drawImage

setClip
setFont
setRenderingHints

JAVA 2D Graphics

Cuando usábamos Graphics...

```
public void paint(Graphics g){  
    super.paint(g);  
  
    g.-----(--);  
  
}
```

... ahora con Graphics2D

```
public void paint(Graphics g){  
    super.paint(g);  
    Graphics2D g2d = (Graphics2D)g;  
  
    g2d.-----(--);  
  
}
```

> Incorpora una nueva clase *Graphics2D*

- *Hereda de Graphics*
- *Métodos para dibujar formas, imágenes o texto*
- *Métodos para cambiar atributos*
- *Ambos aceptan objetos como parámetros*

draw fill	drawString drawImage
setStroke setPaint setComposite setTransform	setClip setFont setRenderingHints

Formas: la clase Shape

<i>Line2D</i>		<i>QuadCurve2D</i>	
<i>Rectangle2D</i>		<i>CubicCurve2D</i>	
<i>RoundRectangle2D</i>		<i>GeneralPath</i>	
<i>Ellipse2D</i>		<i>Polygon</i>	
<i>Arc2D</i>		<i>Area</i>	<div><p>Unión Intersección Diferencia XOR</p></div>

Contexto



The pen attribute is applied to the outline of a shape. This stroke attribute enables you to draw lines with any point size and dashing pattern and apply end-cap and join decorations to a line.

`setStroke(..)`



The fill attribute is applied to a shape's interior. This paint attribute enables you to fill shapes with solid colors, gradients, and patterns.

`setPaint(...)`



The compositing attribute is used when rendered objects overlap existing objects.

`setComposite(...)`



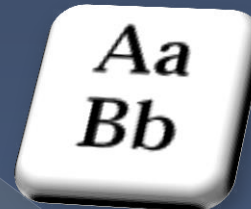
The transform attribute is applied during rendering to convert the rendered object from user space to device-space coordinates. Optional translation, rotation, scaling, or shearing transforms can also be applied through this attribute.

`setTransorm(...)`



The clip, type restricts rendering to the area within the outline of the Shape object used to define the clipping path. Any Shape object that is used to define the clip.

`setClip(...)`



The font attribute is used to convert text strings to glyphs

`setFont(...)`



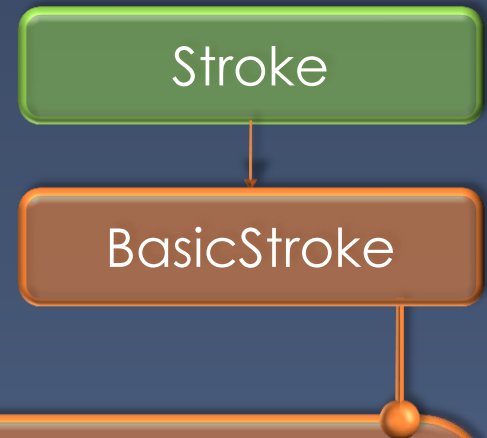
Rendering hints specify preferences in the trade-offs between speed and quality. For example, you can specify whether antialiasing should be used, if this feature available.

`setRenderingHints(...)`

Stroke

◉ *Establece el estilo del Trazo. Permite modificar cuatro atributos:*

- > *Grosor*
- > *Estilo final de líneas*
- > *Estilo de unión de líneas*
- > *Discontinuidad*



BasicStroke()

BasicStroke(float width)

BasicStroke(float width, int cap, int join)

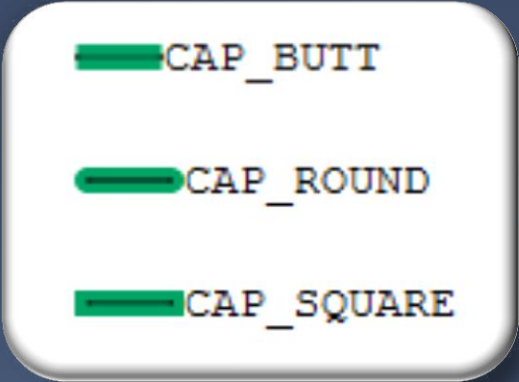
BasicStroke(float width, int cap, int join, float miterlimit)

BasicStroke(float width, int cap, int join, float miterlimit,
float[] dash, float dash_phase)

Stroke

◉ Establece el estilo del Trazo. Permite modificar cuatro atributos:

- > Grosor
- > Estilo final de líneas
- > Estilo de unión de líneas
- > Discontinuidad



CAP_BUTT

CAP_ROUND

CAP_SQUARE

BasicStroke()

BasicStroke(float width)

BasicStroke(float width, int cap, int join)

BasicStroke(float width, int cap, int join, float miterlimit)

BasicStroke(float width, int cap, int join, float miterlimit,
float[] dash, float dash_phase)

Stroke

◉ Establece el estilo del Trazo. Permite modificar cuatro atributos:

- > Grosor
- > Estilo final de líneas
- > Estilo de unión de líneas
- > Discontinuidad



BasicStroke()

BasicStroke(float width)

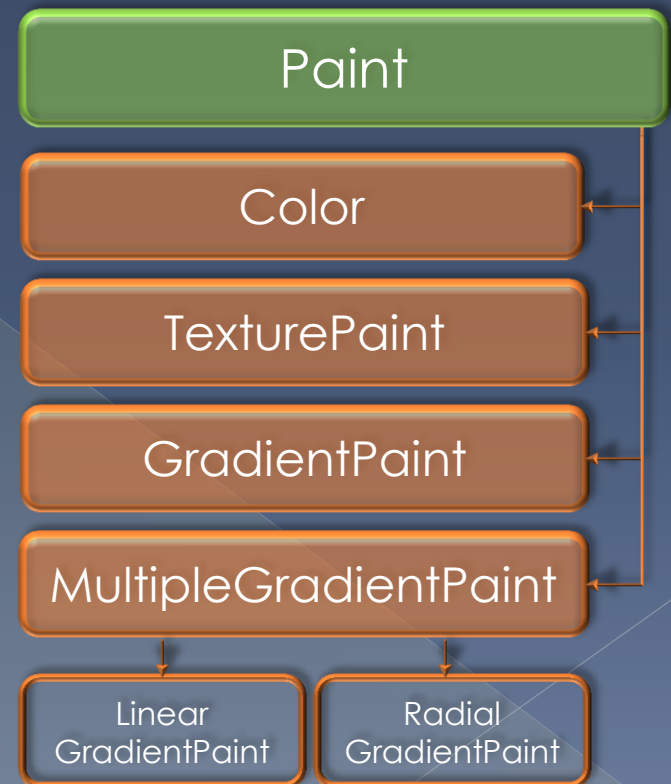
BasicStroke(float width, int cap, int join)

BasicStroke(float width, int cap, int join, float miterlimit)

BasicStroke(float width, int cap, int join, float miterlimit,
float[] dash, float dash_phase)

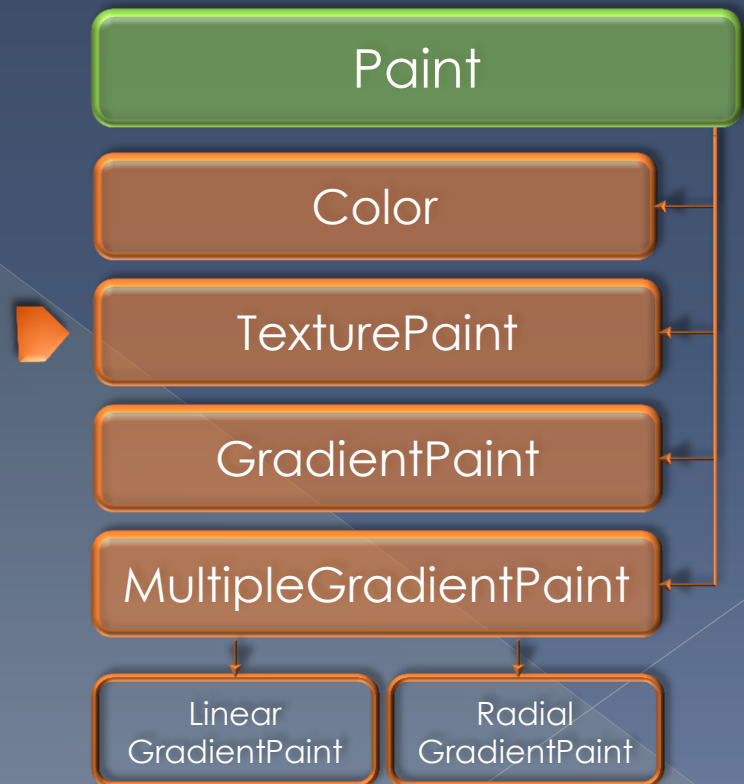
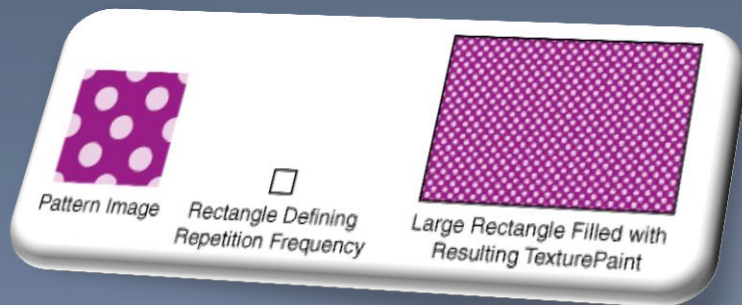
Paint

- Establece el tipo y estilo de color del trazo y el relleno
 - Color liso
 - Basado en imagen
 - Color degradado
 - Lineal, 2 puntos
 - Lineal, N puntos
 - Radial



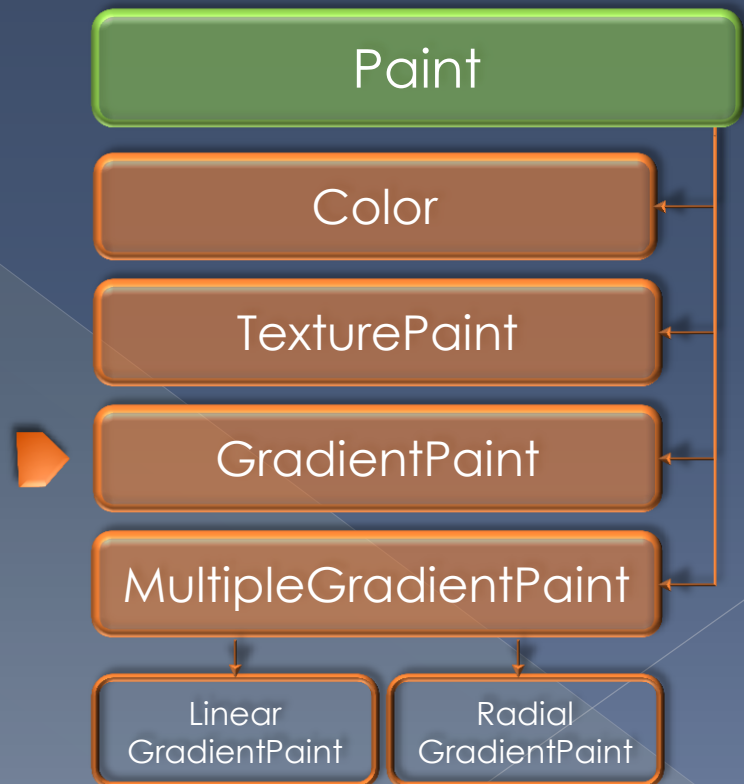
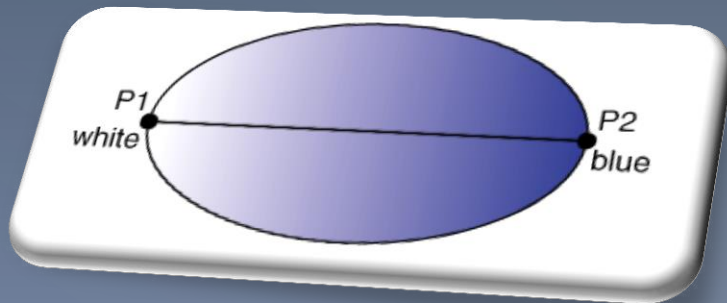
Paint

- Establece el tipo y estilo de color del trazo y el relleno
 - Color liso
 - Basado en imagen
 - Color degradado
 - Lineal, 2 puntos
 - Lineal, N puntos
 - Radial



Paint

- Establece el tipo y estilo de color del trazo y el relleno
 - Color liso
 - Basado en imagen
 - Color degradado
 - Lineal, 2 puntos
 - Lineal, N puntos
 - Radial



Paint

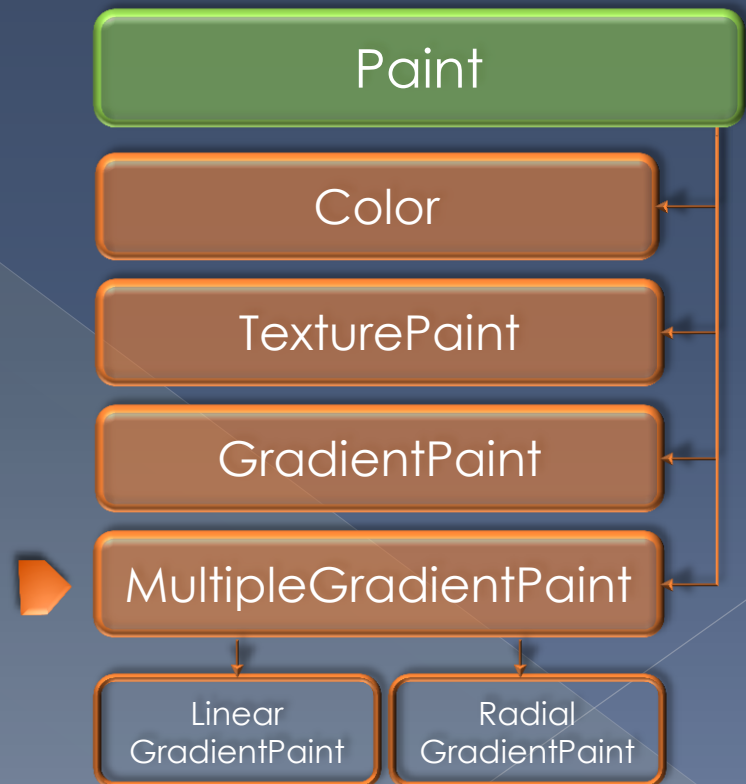
- Establece el tipo y estilo de color del trazo y el relleno

- > Color liso
- > Basado en imagen
- > Color degradado
 - Lineal, 2 puntos
 - Lineal, N puntos
 - Radial



```
Point2D start = new Point2D.Float(0, 0);  
Point2D end = new Point2D.Float(50, 50);  
float[] dist = {0.0f, 0.2f, 1.0f};  
Color[] colors = {Color.RED, Color.WHITE, Color.BLUE};
```

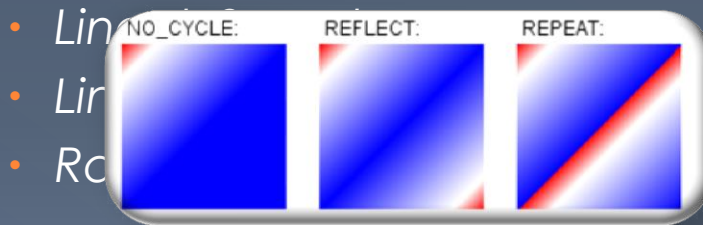
```
LinearGradientPaint p =  
    new LinearGradientPaint(start, end, dist, colors);
```



Paint

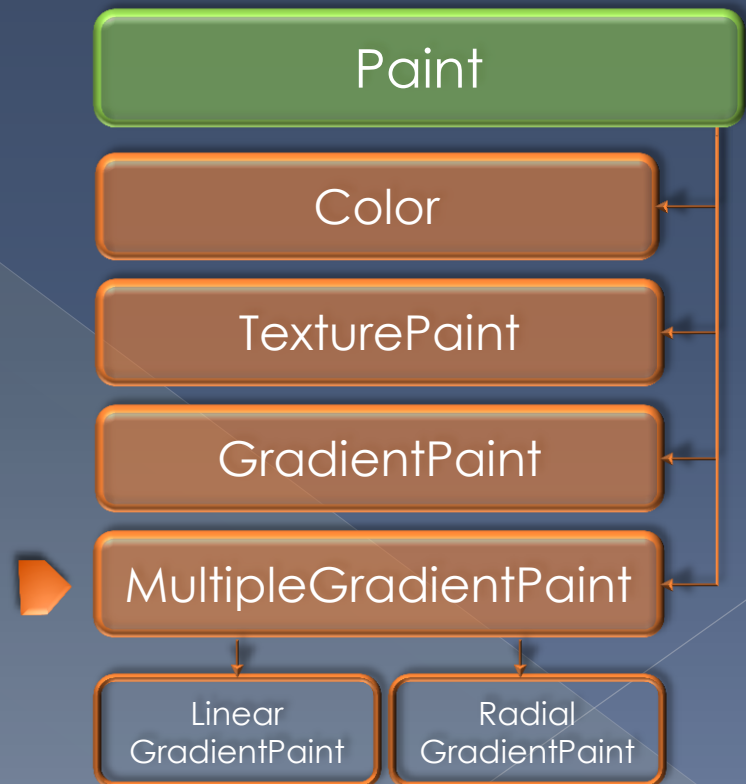
- Establece el tipo y estilo de color del trazo y el relleno

- > Color liso
- > Basado en imagen
- > Color degradado



```
Point2D start = new Point2D.Float(0, 0);  
Point2D end = new Point2D.Float(50, 50);  
float[] dist = {0.0f, 0.2f, 1.0f};  
Color[] colors = {Color.RED, Color.WHITE, Color.BLUE};
```

```
LinearGradientPaint p =  
    new LinearGradientPaint(start, end, dist, colors);
```



Paint

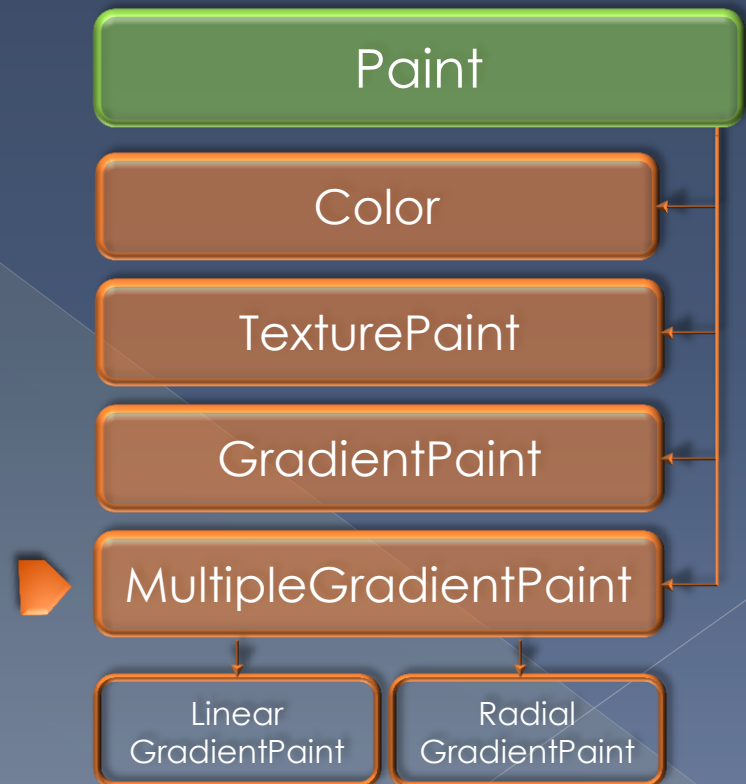
- Establece el tipo y estilo de color del trazo y el relleno

- > Color liso
- > Basado en imagen
- > Color degradado
 - Lineal, 2 puntos
 - Lineal, N puntos
 - Radial



```
Point2D center = new Point2D.Float(50, 50);  
float r = 25;  
float[] dist = {0.0f, 0.2f, 1.0f};  
olor[] colors = {Color.RED, Color.WHITE, Color.BLUE};
```

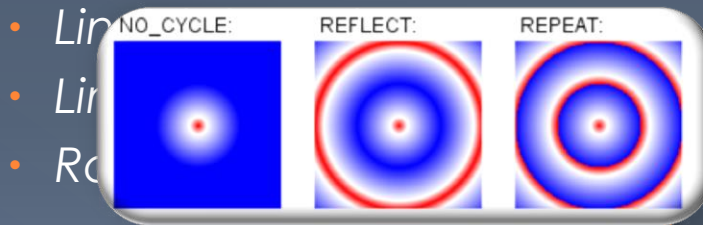
```
RadialGradientPaint p =  
    new RadialGradientPaint(center, r, dist, colors);
```



Paint

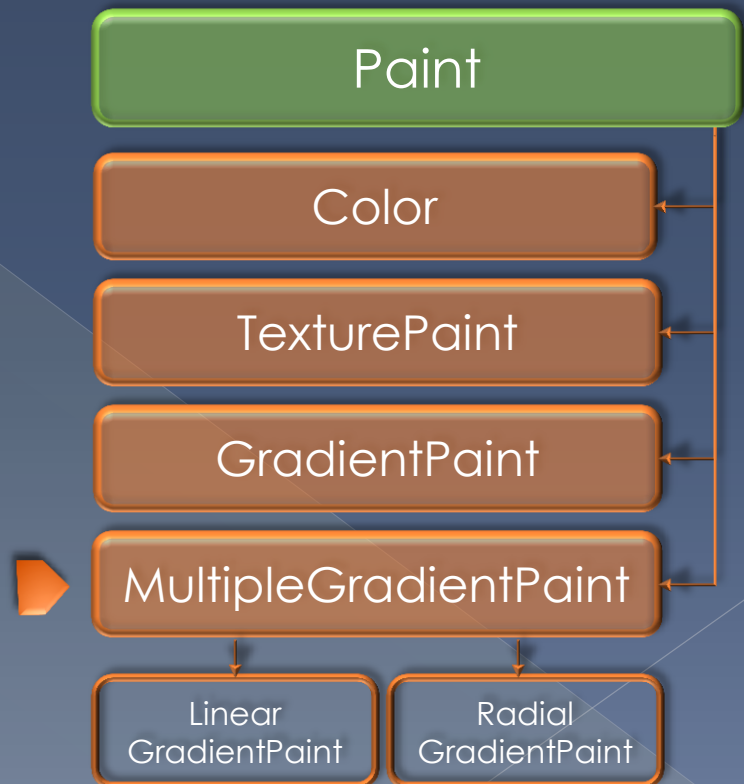
- Establece el tipo y estilo de color del trazo y el relleno

- > Color liso
- > Basado en imagen
- > Color degradado



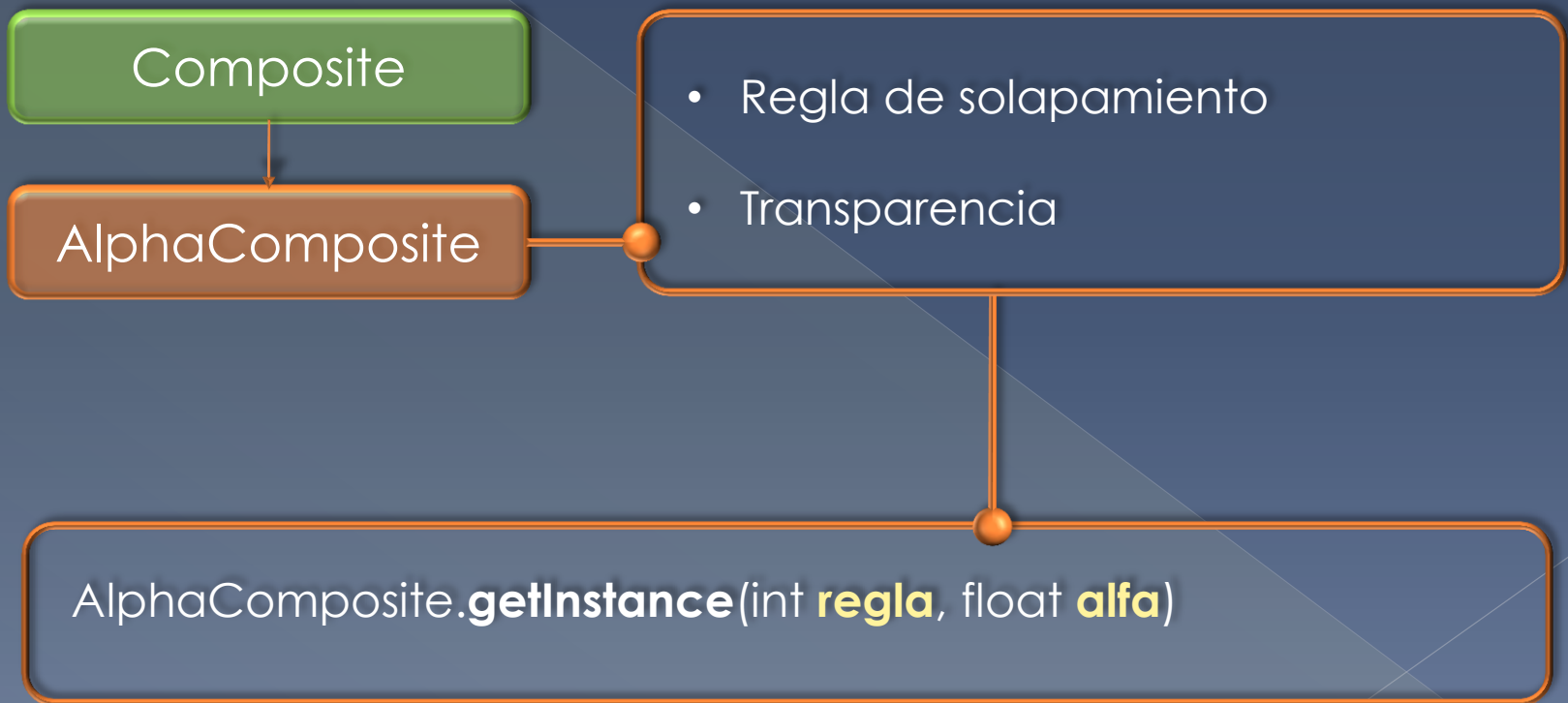
```
Point2D center = new Point2D.Float(50, 50);  
float r = 25;  
float[] dist = {0.0f, 0.2f, 1.0f};  
olor[] colors = {Color.RED, Color.WHITE, Color.BLUE};
```

```
RadialGradientPaint p =  
    new RadialGradientPaint(center, r, dist, colors);
```



Composite

- Establece la composición entre una nueva figura y el dibujo ya existente



Composite

SRC_OVER



SRC_IN



SRC_OUT



CLEAR



DST_OVER



DST_IN



DST_OUT



Transform

- *Permite realizar transformaciones afines (rotaciones, traslaciones, escalados,...) en la visualización de una forma*
- *No modifica las coordenadas de la forma*

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & d_x & t_x \\ d_y & s_y & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$x' = s_x x + d_x y + t_x$$

$$y' = d_y x + s_y y + t_y$$

Transform

- Clase: *AffineTransform*
- Hay dos formas de construir un objeto:
 - > Mediante el constructor, pasándole como parámetros la matriz de transformación
 - > Mediante métodos estáticos, creando transformaciones predeterminadas:

`AffineTransform.getTranslateInstance(double tx, double ty)`

`AffineTransform.getScaleInstance(double sx, double sy)`

`AffineTransform.getRotateInstance(double theta,...)`

`AffineTransform.getShearInstance(double shx, double shy)`

Transform

● Concatenación de transformaciones

- > Mediante el método “concatenate”

`concatenate(AffineTransform Tx)`

- > Mediante métodos específicos

translate(double tx, double ty)

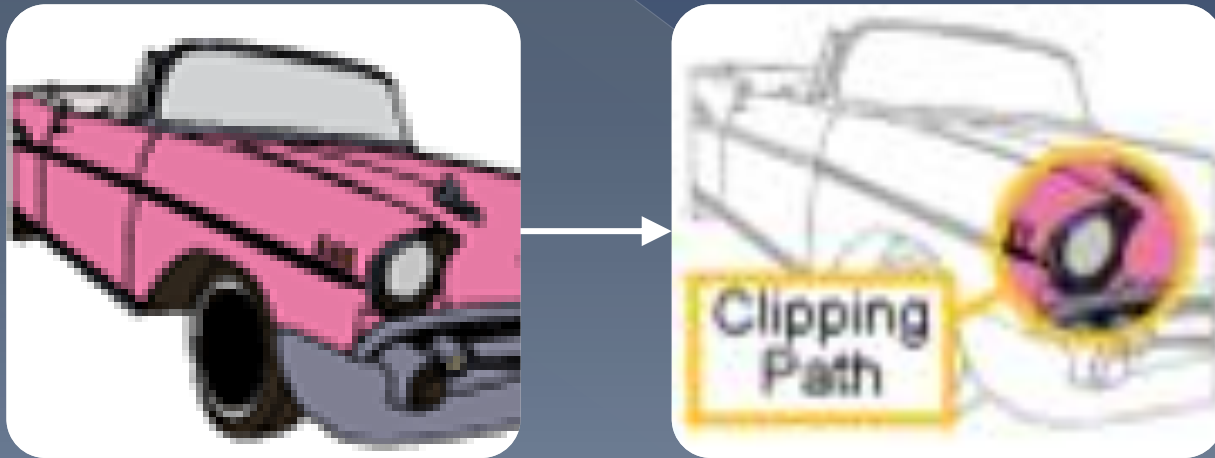
scale(double sx, double sy)

rotate(double theta,...)

shear(double shx, double shy)

Clip

- Define el área que será visualizada
- Para definir dicha área, se usará un Shape como parámetro



Fuente

- ◉ *Permite definir la fuente del texto*
 - > *Tipo de letra*
 - > *Tamaño*
 - > *Estilo (negrita, cursiva,...)*

Font

Font(String **tipoLetra**, int **estilo**, int **tamaño**)

Font(Map<AttributedCharacterIterator.Attribute, ?> **attributes**)

Fuente

- ◉ *Permite definir la fuente del texto*
 - > *Tipo de letra*
 - > *Tamaño*
 - > *Estilo (negrita, cursiva,...)*

Font

```
Font font = new Font("Arial", Font.BOLD, 45);  
g2d.setFont(font);  
g2d.drawString("Hola mundo", 100, 100);
```

Fuente

- ◉ *Permite definir la fuente del texto*
 - > *Tipo de letra*
 - > *Tamaño*
 - > *Estilo (negrita, cursiva,...)*

Font

```
Font font = new Font("Arial", Font.BOLD | Font.ITALIC, 45);  
g2d.setFont(font);  
g2d.drawString("Hola mundo", 100, 100);
```

Fuente

- La clase *Font* ofrece diferentes constantes de estilo (*negrita*, *itálica*), si bien éstas son muy limitadas. Para definir estilos más variados, se usa la clase *TextAttribute*

```
Map atributos = new HashMap();
atributos.put(TextAttribute.UNDERLINE,TextAttribute.UNDERLINE_ON);
atributos.put(TextAttribute.SIZE, 40);
...
Font font = new Font(atributos);
g2d.setFont(font);
g2d.drawString("Hola mundo", 100, 100);
```


Fuente

- La clase *Font* ofrece diferentes constantes de estilo (negrita, itálica), si bien éstas son muy limitadas. Para definir estilos más variados, se usa la clase *TextAttribute*

```
Map atributos = otra_fuente.getAttributes();
atributos.put(TextAttribute.UNDERLINE,TextAttribute.UNDERLINE_ON);
atributos.put(TextAttribute.SIZE, 40);
...
Font font = new Font(atributos);
g2d.setFont(font);
g2d.drawString("Hola mundo", 100, 100);
```

Fuente

- La clase *Font* ofrece diferentes constantes de estilo (negrita, itálica), si bien éstas son muy limitadas. Para definir estilos más variados, se usa la clase *TextAttribute*

```
Map atributos = new HashMap();
atributos.put(TextAttribute.UNDERLINE,TextAttribute.UNDERLINE_ON);
atributos.put(TextAttribute.SIZE, 40);
...
Font font = otra_fuente.deriveFont(atributos);
g2d.setFont(font);
g2d.drawString("Hola mundo", 100, 100);
```

RenderingHints

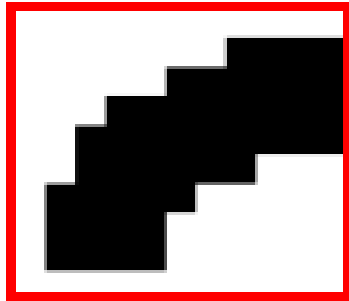
- ◉ *Permite mejorar la calidad del rendering*
- ◉ *Clase: RenderingHints*

RenderingHints(RenderingHints.Key **key**, Object **value**)

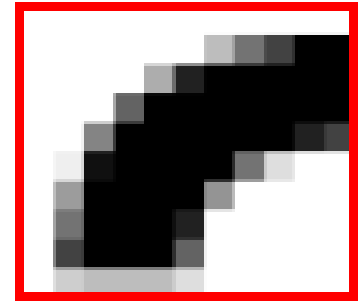
[Propiedad a mejorar] [Valor]



a



a



RenderingHints

- ◉ *Permite mejorar la calidad del rendering*
- ◉ *Clase: RenderingHints*

RenderingHints(RenderingHints.Key **key**, Object **value**)

[Propiedad a mejorar] [Valor]

```
public void paint (graphics g){  
    Graphics2D g2 = (Graphics2D)g;  
    RenderingHints rh = new RenderingHints(  
        RenderingHints.KEY_TEXT_ANTIALIASING,  
        RenderingHints.VALUE_TEXT_ANTIALIAS_ON);  
    g2.setRenderingHints(rh);  
    ....  
}
```

RenderingHints

Hint	Key	Values
Antialiasing	KEY_ANTIALIASING	VALUE_ANTIALIAS_ON VALUE_ANTIALIAS_OFF VALUE_ANTIALIAS_DEFAULT
Alpha Interpolation	KEY_ALPHA_INTERPOLATION	VALUE_ALPHA_INTERPOLATION_QUALITY VALUE_ALPHA_INTERPOLATION_SPEED VALUE_ALPHA_INTERPOLATION_DEFAULT
Color Rendering	KEY_COLOR_RENDERING	VALUE_COLOR_RENDER_QUALITY VALUE_COLOR_RENDER_SPEED VALUE_COLOR_RENDER_DEFAULT
Dithering	KEY_DITHERING	VALUE_DITHER_DISABLE VALUE_DITHER_ENABLE VALUE_DITHER_DEFAULT
Fractional Text Metrics	KEY_FRACTIONALMETRICS	VALUE_FRACTIONALMETRICS_ON VALUE_FRACTIONALMETRICS_OFF VALUE_FRACTIONALMETRICS_DEFAULT
Image Interpolation	KEY_INTERPOLATION	VALUE_INTERPOLATION_BICUBIC VALUE_INTERPOLATION_BILINEAR VALUE_INTERPOLATION_NEAREST_NEIGHBOR
Rendering	KEY_RENDERING	VALUE_RENDER_QUALITY VALUE_RENDER_SPEED VALUE_RENDER_DEFAULT
Stroke Normalization Control	KEY_STROKE_CONTROL	VALUE_STROKE_NORMALIZE VALUE_STROKE_DEFAULT VALUE_STROKE_PURE
Text Antialiasing	KEY_TEXT_ANTIALIASING	VALUE_TEXT_ANTIALIAS_ON VALUE_TEXT_ANTIALIAS_OFF VALUE_TEXT_ANTIALIAS_DEFAULT VALUE_TEXT_ANTIALIAS_GASP VALUE_TEXT_ANTIALIAS_LCD_HRGB VALUE_TEXT_ANTIALIAS_LCD_HBGR VALUE_TEXT_ANTIALIAS_LCD_VRGB VALUE_TEXT_ANTIALIAS_LCD_VBGR