

---

# PRÁCTICA 6

## Gráficos 2D

Parte 2

---

El objetivo de esta práctica será seguir estudiando las principales características de la programación de gráficos usando la tecnología Java2D. Concretamente, nos centraremos en probar los diferentes atributos de dibujo. Una vez practicados estos fundamentos, ampliaremos la práctica 5 para añadirle nuevas funcionalidades.

Al igual que en la práctica anterior, se aconseja seguir el tutorial “[2D Graphics](#)” de Java.

## ■ Modificación de atributos

Para esta práctica crearemos una aplicación sencilla que incorpore una ventana principal (*JFrame*) donde sobrecargaremos el método *paint* e incorporaremos el código para ir modificando los diferentes atributos de dibujo. Se recomienda crear un método *pruebaAtributos* (*Graphics2D g2d*) en el que incluir todo el código de prueba; este método será llamado desde el método *paint*:

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

private void pruebaAtributos(Graphics2D g2d){
    //Trazo
    Stroke trazo;
    // TODO: Código para crear trazo
    g2d.setStroke(trazo);

    //Relleno
    Paint relleno;
    // TODO: Código para crear relleno
    g2d.setPaint(relleno);

    //Composición
    Composite composicion;
    // TODO: Código para crear composición
    g2d.setComposite(composicion);

    //Transformación
    AffineTransform transformacion;
    // TODO: Código para crear transformación
    g2d.setTransform(transformacion);

    //Fuente
    Font fuente;
    // TODO: Código para crear fuente
    g2d.setFont(fuente);

    //Renderización
    RenderingHints render;
    // TODO: Código para crear renderizado
    g2d.setRenderingHints(render);

    //Recorte
    Shape clip;
    // TODO: Código para crear clip
    g2d.setClip(clip);
}
```

## ■ Trazo

En este primer bloque probaremos el estilo de trazo (*Stroke*) asociado a las formas que dibujemos. Para ello, seguiremos la sección “[Stroking and Filling Graphics Primitives](#)” dentro del capítulo “*Working with Geometry*” del tutorial.

Así, por ejemplo, para usar un trazo discontinuo de grosor diez incluiríamos el siguiente código<sup>1</sup> en el método *setAtributos*:

```
//Trazo
Stroke trazo;
float patronDiscontinuidad[] = {15.0f, 15.0f};
trazo = new BasicStroke(10.0f,
                        BasicStroke.CAP_ROUND,
                        BasicStroke.JOIN_MITER, 1.0f,
                        patronDiscontinuidad, 0.0f);

g2d.setStroke(trazo);

//Pintamos una forma de prueba
g2d.draw(new Line2D.Float(40,40,160,160));
```

①

Usando el esquema anterior, probar las cuatro propiedades del trazo: grosor, estilos final de línea, estilos de unión de línea y discontinuidad (en este último caso, probar otros patrones).

## ■ Relleno

En este segundo bloque probaremos el color y relleno (*Paint*) de las formas que dibujemos (véase “[Stroking and Filling Graphics Primitives](#)” del tutorial). Como vimos, existen diferentes tipos de relleno (liso, degradado, basado en textura, etc.), cada uno de ellos con una clase asociada. Por ejemplo, para un relleno liso:

```
//Relleno
Paint relleno;
relleno = new Color(255, 100, 0);
g2d.setPaint(relleno);

g2d.draw(new Rectangle(170,40,120,120));
g2d.fill(new Rectangle(300,40,120,120));
```

②

En el caso de un relleno con degradado (para el ejemplo, diagonal de rojo a azul, desde la esquina superior izquierda a la esquina inferior derecha), el código sería:

```
Point pc1 = new Point(430,40), pc2 = new Point(550,160);
relleno = new GradientPaint(pc1, Color.RED, pc2, Color.BLUE);
g2d.setPaint(relleno);

g2d.fill(new Rectangle(430,40,120,120));
```

③

Usando el esquema anterior, probar otros estilos de relleno<sup>2</sup>. Para el caso del degradado, probar diferentes orientaciones.

---

<sup>1</sup> Además de cambiar los atributos, en los ejemplos de este guion incluiremos el código para dibujar una forma de prueba.

<sup>2</sup> Para el caso del *TexturePaint* es necesario indicar como parámetro una imagen textura. Cómo crear o leer imágenes será algo que veremos en siguientes temas.

## ■ Composición

Este atributo permite definir la regla de composición entre una nueva figura y el dibujo ya existente; además, permite definir el nivel de transparencia. Por ejemplo, el siguiente código establece como regla de composición la `SRC_OVER` (fuente sobre destino) y grado de transparencia 0,5:

```
//Composición
Composite comp;
comp = AlphaComposite.getInstance(AlphaComposite.SRC_OVER, 0.5f);
g2d.setComposite(comp);

g2d.fill(new Rectangle(190, 100, 200, 120));
```

4

Para conocer las distintas reglas de composición<sup>3</sup>, se aconseja probar la demo del tutorial que se encuentra en el capítulo [“Compositing Graphics”](#).

## ■ Transformación

En este bloque probaremos diferentes transformaciones (`AffineTransform`) que se aplicarán antes de dibujar las formas<sup>4</sup> (véase [“Transforming Shapes, Text, and Images”](#) del tutorial). Como sabemos, existen varias transformaciones predefinidas (traslación, escalado, rotación y deformación), cada una de ellas con métodos específicos para crear las correspondientes instancias. Por ejemplo, para una rotación de 45°:

```
//Transformación
Rectangle r = new Rectangle(430, 190, 120, 120);
g2d.draw(r); //Dibujamos rectángulo sin transformación

AffineTransform at;
at = AffineTransform.getRotateInstance(Math.toRadians(45.0),
                                     r.getCenterX(),
                                     r.getCenterY());

g2d.setTransform(at);

g2d.fill(r); //Dibujamos rectángulo con transformación
```

5

Nótese que en el ejemplo anterior, además del ángulo de rotación, se indica el eje (punto) de rotación ya que, si no se especifica, la rotación se haría respecto al punto (0,0). Hay que tener en cuenta que todas las transformaciones se hacen considerando el sistema de coordenadas con origen (0,0); si se quiere hacer la transformación focalizada en una determinada forma, habrá que asegurarse que se hace respecto al centroide de dicha forma. Por ejemplo, si continuando el ejemplo anterior creamos el siguiente escalado:

```
at = AffineTransform.getScaleInstance(0.5,0.5);
g2d.setTransform(at);
g2d.draw(r);
```

6

¿el resultado que obtenemos es el esperado? Como vemos, no escala respecto al centro del objeto. Para ello, habría que hacer lo siguiente<sup>5</sup>:

---

<sup>3</sup> El efecto de la composición no se apreciará si se implementa en el método `paint`, habrá que hacerlo usando imágenes como en el ejemplo del tutorial.

<sup>4</sup> Recordemos que no modifica las coordenadas originales de la forma, solo aplica la transformación para su visualización.

<sup>5</sup> Salvo para la rotación, la clase `AffineTransform` no ofrece métodos que acepten como parámetro un nuevo centro distinto al origen de coordenadas; por este motivo, es necesario implementarlo mediante concatenación de transformaciones.

```

at.setToIdentity();
at.translate(r.getCenterX(), r.getCenterY());
at.scale(0.5, 0.5);
at.translate(-r.getCenterX(), -r.getCenterY());
g2d.setTransform(at);
g2d.draw(r);

```

7

El código anterior traslada la forma, colocando su centro en el origen de coordenadas, la escala y después la traslada de nuevo a su posición original (obsérvese que las concatenaciones se han de hacer en el orden inverso).

Partiendo del ejemplo, y teniendo en cuenta las consideraciones anteriores, realizar los siguientes ejercicios:

- Escalar (a 0.25) y rotar (45°) el rectángulo (véase Figura 1 para ver el resultado esperado).
- Tras las transformaciones anteriores, dejar la identidad como transformación activa.

8

## ■ Fuente

En este bloque estableceremos la fuente (*Font*) que se usará al dibujar texto. En el capítulo “[Working with Text APIs](#)” del tutorial se explica en detalle la gestión de fuentes y las posibilidades que ofrece este atributo. En este guion proponemos un par de ejemplos muy sencillos; en primer lugar, estableceremos una fuente Arial de tamaño 45 y estilo negrita+itálica:

```

//Fuente
Font fuente;
fuente = new Font("Arial", Font.BOLD | Font.ITALIC, 45);
g2d.setFont(fuente);
g2d.drawString("Hola", 30, 220);

```

9

Como vemos, para el estilo existen variables estáticas que ofrecen diferentes alternativas. No obstante, no hay variables para todos los estilos; por ejemplo, no existen para el subrayado, tachado, subíndice o superíndice. Para estos otros estilos, hay que usar la clase *TextAttribute* (véase sección “[Using Text Attributes to Style Text](#)” del tutorial); por ejemplo, para incorporar el subrayado, el código sería:

```

Map atributosTexto = fuente.getAttributes();
atributosTexto.put(TextAttribute.UNDERLINE, TextAttribute.UNDERLINE_ON);
g2d.setFont( new Font(atributosTexto) );
g2d.drawString("mundo", 30, 260);

```

10

## ■ Renderización

Es posible mejorar la calidad del renderizado de formas, imágenes y texto. Para ello, y usando la clase *RenderingHints*, podemos activar/desactivar dichas mejoras (véase sección “[Controlling Rendering Quality](#)” del tutorial). Por ejemplo, el siguiente código activa la mejora en el alisado de formas y texto:

```

//Renderización
RenderingHints render;
g2d.draw(new Ellipse2D.Float(40,350,510,50)); //Elipse sin antialiasing

render = new RenderingHints(RenderingHints.KEY_ANTIALIASING,
                             RenderingHints.VALUE_ANTIALIAS_ON);
g2d.setRenderingHints(render);

g2d.draw(new Ellipse2D.Float(40,450,510,50)); //Elipse con antialiasing
g2d.drawString("Hola", 30, 170); //Texto con antialiasing

```

11

Dado un objeto *RenderingHints*, podemos activar nuevas mejoras incluyendo nuevos pares llave-valor en su mapa (también es posible crear el objeto pasándole directamente un mapa con varias llaves activadas). Por ejemplo, para mejorar el renderizado del color:

```
render.put(RenderingHints.KEY_COLOR_RENDERING,
           RenderingHints.VALUE_COLOR_RENDER_QUALITY);
```

## ■ Recorte

Es posible definir el área visible de nuestro *Graphics2D* (véase sección [“Clipping the Drawing Region”](#) del tutorial); dicha zona visible se definirá mediante objeto *Shape*:

```
//Recorte
Shape clipArea;
clipArea = new Ellipse2D.Float(100,100,500,500);
g2d.setClip(clipArea);
```

12

A partir de que activemos el *clip* anterior, sólo veremos lo que dibujemos dentro del área de recorte (en este ejemplo, un círculo)<sup>6</sup>. Si queremos marcar los límites de dicha área, podemos pintar la forma (aunque no es algo que se suele hacer):

```
g2d.draw(clipArea);
```

## ■ Introduciendo algunas mejoras...

En los ejemplos anteriores hemos ido probando los distintos atributos con parámetros fijos, siendo el objetivo conocer las posibilidades que ofrecen y no tanto la interacción con el usuario (la Figura 1 muestra el aspecto de la ventana principal tras realizar los ejercicios de las secciones anteriores). Para completar esta práctica, se proponen las siguientes mejoras que introducen la gestión de eventos:

- A. Incorporar a nuestra aplicación la lista de fuentes disponible, de forma que el texto aparezca con la fuente seleccionada por el usuario
  - En primer lugar incorporaremos una lista desplegable (*ComboBox*) a nuestra ventana principal (véase Figura 2). Como sabemos, el NetBeans incorporará automáticamente en nuestro código tanto la declaración de la variable (*private javax.swing.JComboBox listaFuentes*), como la construcción del objeto (*listaFuentes = new javax.swing.JComboBox()*) usando por defecto el constructor vacío.
  - Posteriormente incorporaremos a la lista desplegable las fuentes instaladas en el sistema. El siguiente código nos permite obtener las fuentes instaladas:

```
GraphicsEnvironment ge;
ge = GraphicsEnvironment.getLocalGraphicsEnvironment();
String []fuentesSistema = ge.getAvailableFontFamilyNames();
```

El vector de fuentes lo podemos pasar como parámetro en el constructor<sup>7</sup>:

---

<sup>6</sup> Si ponemos este código al final del método *setAtributos* que estamos desarrollando en esta práctica, el recorte se hará sobre lo que dibujemos nuevo (en particular, los rectángulos que dibujamos de forma interactiva), pero no sobre lo que ya teníamos dibujado. Si ponemos este código al principio del método *setAtributos*, el recorte sí se aplicará a todo lo que teníamos dibujado.

<sup>7</sup> Como sabemos, el NetBeans usa por defecto los constructores vacíos; si queremos modificar la llamada al constructor lo podemos hacer seleccionando la opción “Customize code” del menú contextual asociado a la lista desplegable (en la zona de diseño, clic con botón derecho sobre el *combobox*). En el diálogo que lanza,

```
| listaFuentes = new javax.swing.JComboBox(fuentesSistema);
```

o, si queremos mantener el constructor vacío, podemos actualizar el contenido de la lista desplegable de la siguiente forma<sup>8</sup>:

```
| listaFuentes.setModel(new DefaultComboBoxModel(fuentesSistema));
```

- Por último, se gestionará el evento *acción* sobre la lista desplegable; así, cuando el usuario seleccione una fuente, se actualizará una variable *String tipoFuente* que será la que se use al construir el objeto *Font*.

#### B. Efecto “ventana”: hacer que el área de recorte se mueva siguiendo al ratón

En el ejemplo que vimos sobre área de recorte, usábamos un círculo con una localización fija; en este nuevo ejemplo, haremos que la posición del círculo varíe en función de dónde esté el ratón (véase Figura 2). Para ello:

- En primer lugar, declaramos e inicializamos la variable<sup>9</sup>:

```
| Ellipse2D clipArea = new Ellipse2D.Float(100,100,500,500);
```

e introducimos el siguiente código en el método *paint* para activar el recorte (lo ponemos antes de la llamada a *pruebaAtributos*, para que el efecto ventana se vea sobre las pruebas que hemos hecho en las secciones anteriores):

```
| if (ventanaClipActiva) {  
|     g2d.setClip(clipArea);  
| }
```

donde *ventanaClipActiva* es una variable booleana vinculada a una caja de selección (*CheckBox*) de la ventana principal (véase Figura 2), que indicará si tenemos o no activa la opción.

Para evitar que se solape el recorte de este ejercicio con el recorte que hicimos en el ejercicio anterior, se aconseja comentar el código asociado al recorte dentro del método *pruebaAtributos* (o, preferiblemente, incluirlo dentro de un *if(!ventanaClipActiva)* para que sean excluyentes ambos ejercicios).

- Posteriormente, gestionamos el evento *mouseMove* para que actualice la localización del área de recorte (círculo) a la posición en la que está el ratón:

```
| private void formMouseMoved(java.awt.event.MouseEvent evt) {  
|     if(ventanaClipActiva){  
|         Point corner = evt.getPoint();  
|         corner.translate((int)clipArea.getWidth()/2,  
|                         (int)clipArea.getHeight()/2);  
|         clipArea setFrameFromCenter(evt.getPoint(), corner);  
|         this.repaint();  
|     }  
| }
```

---

aparece todo el código que se ha generado asociado a dicho elemento; si, en una línea dada, cambiamos de “*default code*” a “*custom creation*”, nos permitirá editar la línea (en particular, ponerle parámetros al constructor). Además, se pueden poner líneas de código antes de las ya existentes; en particular, y para el ejemplo que nos ocupa, antes de la llamada al constructor incluiríamos el código para obtener las fuentes instaladas y asignárselas al vector (que pasaríamos como argumento al constructor).

<sup>8</sup> En este caso, tanto el código para obtener las fuentes instaladas, como el de actualización de la lista desplegable, irían en el constructor de la ventana principal.

<sup>9</sup> Nótese que en el ejemplo anterior sobre recorte habíamos declarado la variable local al método *setAtributos*; puesto que en este nuevo ejemplo va a ser usada y modificada por otros métodos, debemos declararla como variable de la clase.



## ■ Ampliando la práctica 5

En los ejemplos anteriores hemos probado los diferentes atributos de dibujo que ofrece Java2D, para lo cual hemos optado por ejemplos sencillos que no implicasen interacción con el usuario. El objetivo ahora es incorporar algunos de estos nuevos atributos al Paint básico que se realizó en la práctica 4 y se amplió en la 5; concretamente, a la propiedad de color que ya tiene actualmente, se propone ahora incorporar tres nuevos atributos: (1) grosor del trazo, (2) transparencia<sup>13</sup> y (3) alisado de las formas.

El aspecto visual de la aplicación será el mostrado en la Figura 3, que amplía al de la práctica 5. A lo ya existente de práctica anteriores (selección de formas, color, relleno y la opción de mover figuras) se le incorporan tres nuevas opciones: el grosor del trazo (seleccionable mediante un *spinner*), si se aplica o no transparencia (seleccionable con una casilla de verificación tipo *JCheckBox*) y si se alisan o no las formas (seleccionable con una casilla de verificación tipo *JCheckBox*). Estas nuevas funcionalidades se mostrarán en la zona inferior derecha de la aplicación (véase Fig. 3).

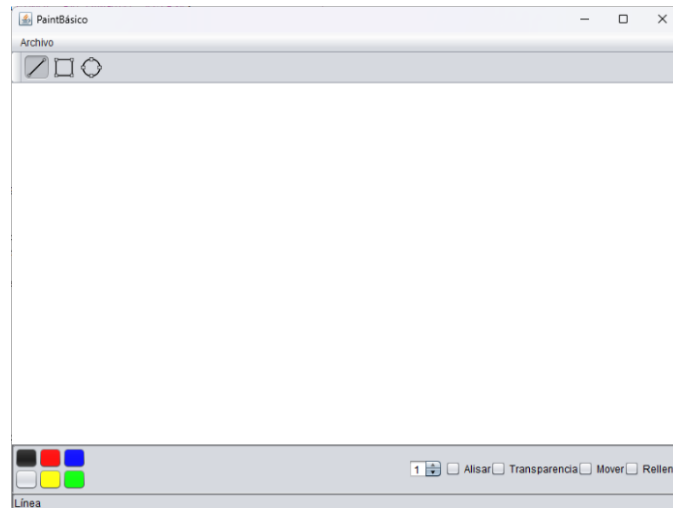


Figura 3. Aspecto de la aplicación

Para abordar las funcionalidades anteriores habrá que ampliar la clase *Lienzo*. Al igual que hicimos con las propiedades de color y relleno, será necesario definir variables miembro en la clase *Lienzo* que representen a las nuevas propiedades de grosor, transparencia y alisado; por ejemplo, para el caso del grosor del trazo:

```
| Stroke trazo = new BasicStroke();
```

De igual forma, habrá que implementar los métodos *set/get* que permitan modificar estas propiedades por terceros (en nuestro caso, desde la ventana principal); p.e., para el grosor<sup>14</sup>:

```
| public void setGrosor(int grosor) {  
|     this.stroke = new BasicStroke(grosor);  
| }
```

Por último, en el método *paint* de la clase *Lienzo*, habrá que incluir el código que active los atributos correspondientes<sup>15</sup>.

<sup>13</sup> Entendida como semitransparencia correspondiente a un alfa 0.5

<sup>14</sup> Nótese que el tipo de la variable que representa la propiedad no tiene que coincidir con el tipo del parámetro de la función *set* (o la salida del *get*). Por ejemplo, para representar la propiedad del grosor del trazo usaremos una variable de tipo *Stroke* (o *BasicStroke*), pero el método *set* ha de pedir como parámetro un entero, que es lo que esperaría un tercero como información relativa a un grosor. Lo mismo ocurre para las propiedades de alisado (que internamente sería un *RenderingHints*, pero en el *set/get* sería un *boolean*) y transparencia (internamente un *Composite*, pero en el *set/get* un *boolean*). En estos dos casos, puede optarse por tener también variables privadas booleanas que indiquen si está o no activa la propiedad (alisado y transparencia) y que permitan decidir si se aplica o no en el método *paint*.

<sup>15</sup> En este caso, la llamada a los métodos *setStroke* para el grosor (que se le pasará un objeto *Stroke*), *setRenderingHints* para el alisado (que se le pasará un objeto *RenderingHints*) y el *setComposite* para la transparencia (que se le pasará un objeto *AlphaComposite*).