Universidad Pontificia de Salamanca en Madrid

Programación de Aplicaciones

Introducción a la programación gráfica con Java2D

Gustavo Millán García

Facultad de Informática
Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software
Universidad Pontificia de Salamanca en Madrid

Introducción Java2D

- **Permite la creación gráficos avanzados o de efectos relacionados con imágenes**
- **Puede ser usada para el desarrollo de animaciones u otras presentaciones multimedia al combinarla con otras APIs de Java, como puedan ser JMF (*Java Media framework - Entorno de Trabajo* de Java para Médios Audiovisuales) o Java 3D.**
- **La API Java2D amplía muchas de las capacidades gráficas de la biblioteca AWT (*Abstract Window Toolkit)***
  - renderizado, figuras geométricas, uso de fuentes de letras, manipulación de imágenes y el enriquecimiento en la definición del color.
- **Demo \jdk1.2\demo\jfc\Java2D**

Clase Graphics2D

❑ Clase java.awt.Graphics2D (abstract)

- Es una clase que extiende a **java.awt.Graphics** proporcionando un control más potente sobre la presentación de texto, imágenes o figuras geométricas.

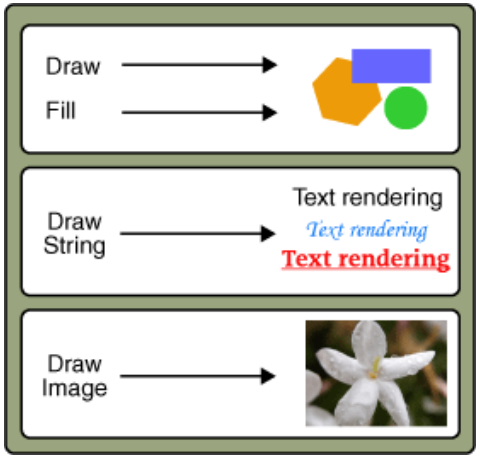
❑ Un objeto Graphics (es una clase abstracta) , representa un área de dibujo y el contexto en el que puede dibujarse algo. Ejemplo

```
public void paint (Graphics g) {
Rectangle2D r2 = new Rectangle2D.Float(75, 50, 100, 25);
Graphics2D g2 = (Graphics2D)g;
g2.draw(r2);
}
```

❑ Esta área de dibujo puede vincularse con distintos medios como un monitor, o una imagen en memoria

Objeto Graphics2D

- **Los métodos de Graphics2D se pueden dividir en dos grupos**
  - Métodos para **dibujar y pintar** formas geométricas, texto e imágenes



  - Métodos para **establecer artributo**s que afectan al dibujo y representación gráfica de objetos.

Objeto Graphics2D

❑ Métodos para dibujar

- **drawText** →Pintar texto
  - ✓`g.drawString("Prueba", 10, 10);`
- **drawImage**→Pintar una imagen
  - ✓
```
g.drawImage(img, 0, 0, width, height, 0,
  0, imageWidth, imageHeight, null);
```
- **drawLine, drawArc, drawRect, drawOval, drawPolygon** →Para dibujar figuras geométricas
  - ✓`g2.draw(new Line2D.Double(0, 0, 30, 40));`

Renderizado con Graphics2D

- El proceso de renderizado de Java2D está controlado por el objeto Graphics2D
- Renderizado=proceso de generar una representación (imagen) a partir de un modelo
- Los atributos de un objeto Graphics2D o características contextuales son a su vez objetos (un atributo contextual puede ser por ejemplo, el tamaño del pincel con que se dibuja una línea recta)
- El conjunto de estos atributos es lo que conforma el contexto del objeto Graphics2D.

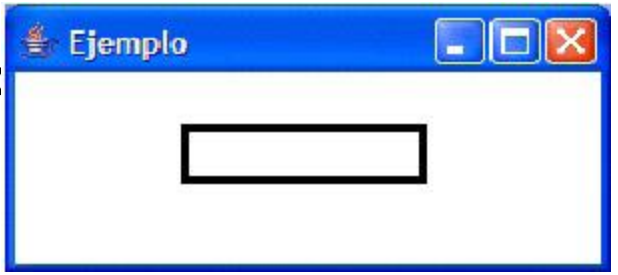
Contexto de dibujo

- **Para dibujar un objeto gráfico, se establece el contexto en que se realizará el renderizado**
- **Los atributos de estado que conforman el contexto y que pueden modificarse son los siguientes:**
  - ✓ Variar la anchura del pincel.
  - ✓ Definir colores o patrones de relleno para las figuras.
  - ✓ Delimitar un área concreta a renderizar (*clipping path).*
  - ✓ Trasladar, rotar, reducir o ampliar objetos cuando son renderizados.
  - ✓ Especificar la forma en que se componen las figuras superpuestas.

Contexto de dibujo

- **Cuando Graphics2D va a realizar un operación de renderizado simple, tal como dibujar un rectángulo, un texto o una imagen, lo hace en base a un contexto determinado.**
- **El contexto por defecto: trazo negro de 1 punto de ancho en el caso de las figuras geométricas.**

```
public void paint (Graphics g) {
Graphics2D g2 = (Graphics2D)g;
Rectangle2D r2 = new Rectangle2D.Float(75, 50, 100, 25);
Stroke pincel = new BasicStroke(4.0f, BasicStroke.CAP_R(
BasicStroke.JOIN_MITER);
g2.setStroke(pincel);
g2.draw(r2);
}
```



Clases Java2D

- **Entre las clases e interfaces más importantes, incluyendo las utilizadas por el contexto, podemos citar las siguientes (contenidas en los paquetes java.awt y java.awt.geom)**
- **Interfaces:**
  - **Composite**: define métodos para realizar composiciones de dibujos. Entre otras cosas permite definir transparencias.
  - **Paint:** extiende a Transparency y define la forma en que se construyen las tramas de color durante las operaciones draw() y fill().
  - **Stroke:** permite a Graphics2D generar un objeto Shape que representa el contorno de la figura que se quiere dibujar

Clases Java2D (I)

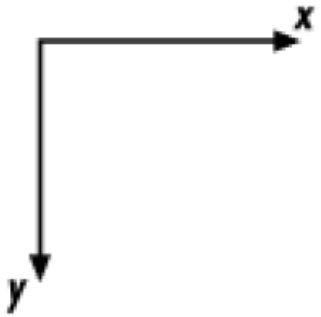
- **Clases:**
  - **AffineTransform:** representa una transformación en 2D: traslación, inversión, rotación, etc.
  - **AlphaComposite:** implementa a Composite. Gestiona la composición alfa (transparencias) básica para las figuras, textos e imágenes.
  - **BasicStroke:** implementa a Stroke. Define el estilo del pincel que se aplica al dibujar el contorno de una figura.
  - **Color:** implementa a Paint. Define por ejemplo el color del relleno o del contorno al dibujar una figura.
  - **GradientPaint:** implementa a Paint. Define un patrón de relleno

Clases Java2D (II)

- **Clases:**
  - **Graphics2D:** extiende a la clase Graphics. Es la **clase fundamental** para el renderizado 2D.
  - **TexturePaint:** define un patrón de relleno complejo al rellenar una figura. Este patrón -también llamado textura- está almacenado en un objeto de tipo BufferedImage.

El sistema de coordenadas de Java2D

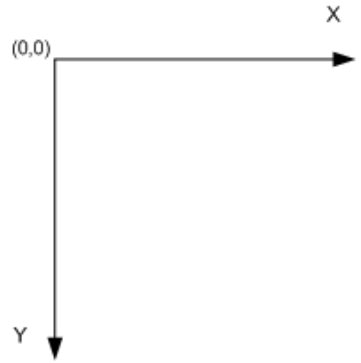
- **Los objetos Java 2D se ubican en un plano definido por coordenadas Cartesianas XY**



- **Java2D mantiene dos sistemas de coordenadas: coordenadas de usuario (*user space*) y *coordenadas de dispositivo (device space):***
  - El sistema de **coordenadas de usuario es totalmente independiente** del dispositivo final en el que se vaya a hacer el renderizado de los gráficos.
  - Java2D ejecuta automáticamente las conversiones entre los dos sistemas de coordenadas en el momento en que se realiza el renderizado real sobre un dispositivo.

Coordenadas de usuario

- El origen de las coordenadas (0,0) de usuario está situado en la esquina superior izquierda del objeto Graphics2D.
- A medida que se incrementa la x (primer valor de la coordenada) habrá un desplazamiento hacia la derecha y a medida que se incrementa la y (segundo valor de la coordenada) habrá un desplazamiento hacia abajo.



Coordenadas de dispositivo

- **En la tabla siguiente, a modo de resumen, se muestran las clases que intervienen en el sistema de coordenadas:**

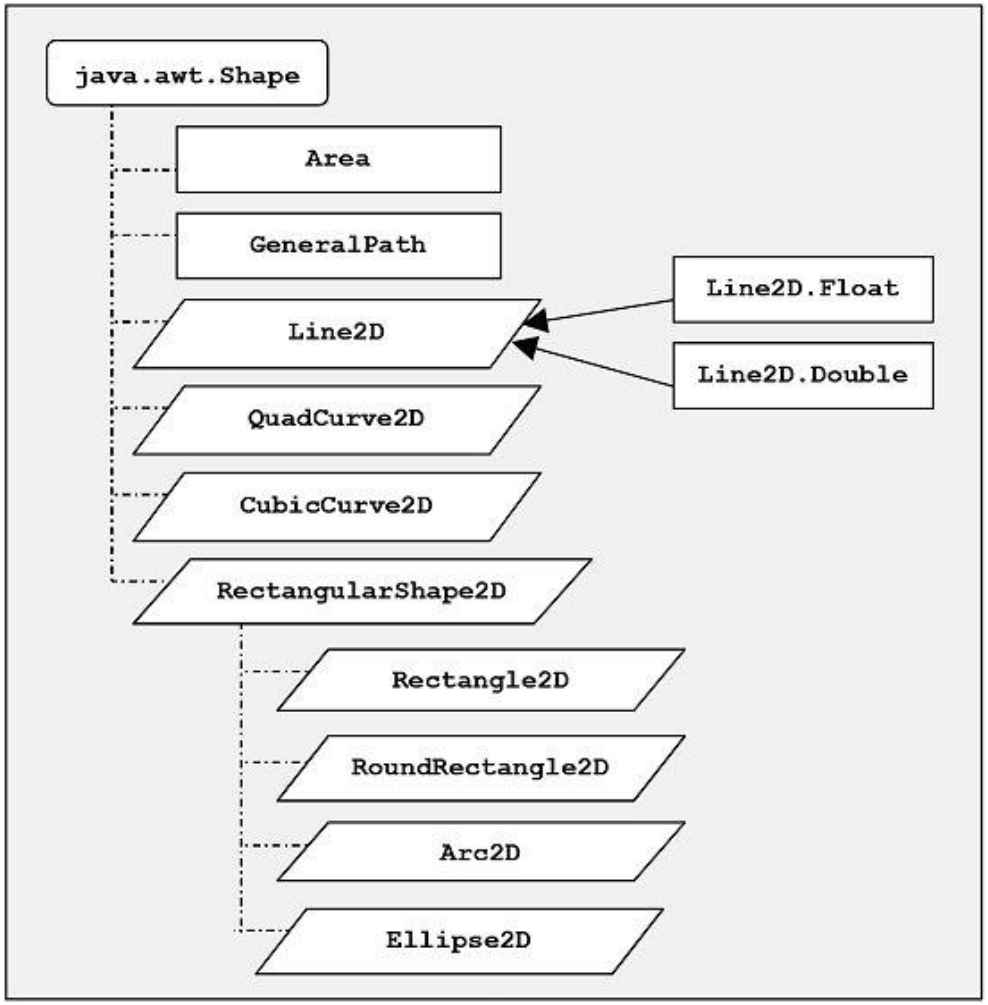
Clase	Descripción
GraphicsEnvironment	Define la colección de dispositivos visibles desde una plataforma Java.
GraphicsDevice	Define un dispositivo concreto de gráficos.
GraphicsConfiguration	Define un modo de ese dispositivo concreto.

Figuras geométricas en Java2D

- **Java2D proporciona varias clases que representan objetos geométricos simples tales como puntos, líneas, curvas y rectángulos**
- **Paquete java.awt.geom .**
  - Rectangle2D, Line2D, Point2D ,Ellipse2D.

Figuras geométricas interface Shape

- **Por compatibilidad, las clases en versiones previas de JDK, como Rectangle, Point y Polygon, permanecen en el paquete java.awt y se pueden seguir usando.**



Interfaz Shape

- La interfaz Shape abstrae el comportamiento de una figura que puede dibujarse
- Cualquier figura que implementa Shape posee un contorno (denominado *path en inglés)*
- La interfaz Shape proporciona un mecanismo estándar para describir e interpretar el contorno de un objeto geométrico

Interfaz Shape

Método	Devuelve un	Descripción
contains(double x, double y)	boolean	Determina si las coordenadas caen dentro de la Shape
contains(Point2D p)	boolean	Igual al anterior pero con un Point2D
contains(double x, double y, double ancho, double largo)	boolean	Determina si el área rectangular entra dentro de la Shape
contains(Rectangle2D r)	boolean	Igual que el anterior pero con un Rectangle2D
getBounds()	Rectangle	Devuelve el rectángulo mínimo que recubre la Shape
getBounds2D()	Rectangle2D	Devuelve un rectángulo más optimizado que en el anterior caso
getPathIterator(AffineTransform at)	PathIterator	Devuelve un Iterator que itera sobre los sub-trazos de la figura. Si queremos transformar las iteraciones, usamos la transformada, sino sencillamente pasamos null en el método
intersects(double x, double y, double ancho, double largo)	boolean	Testea si el interior de la Shape interseca con el interior del área rectangular pasada como parámetro

Figura Geométrica: Punto

❑ Punto2D

- Esta **clase no dibuja nada**, sino que es la representación de un punto en Java2D.
- Se pueden dibujar figuras a partir de puntos (ej líneas)
- Ejemplo de creación de una línea

```
public void paint (Graphics g) {
Graphics2D g2 = (Graphics2D)g;
g2.setStroke(new BasicStroke(3.0f));
Point2D p1 = new Point2D.Float(23.5f, 48.9f);
Point2D p2 = new Point2D.Float(158.0f, 173.0f);
Line2D l = new Line2D.Float(p1, p2);
g2.draw(l);
}
```

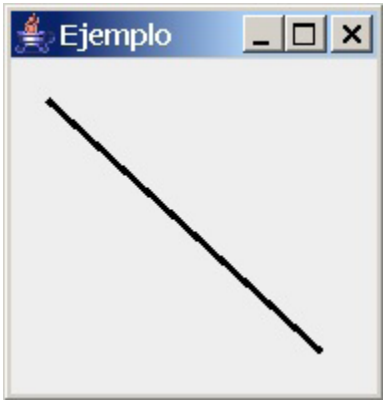


Figura Geométrica: Línea

❑ Line2D

- **Constructores**
  - ✓ Line2D.Float(float X1, float Y1, float X2, float Y2)
  - ✓ Line2D.Float(Point2D p1, Point2D p2)
- **Ejemplo**

```
public void paint (Graphics g) {
super.paint(g);
Graphics2D g2 = (Graphics2D)g;
// Dibujo de la línea
g2.setColor(Color.pink);
g2.setStroke(new BasicStroke(3.0f));
Line2D linea = new Line2D.Float(50.0f, 50.0f, 200.0f, 200.0f);
g2.draw(linea);
}
```

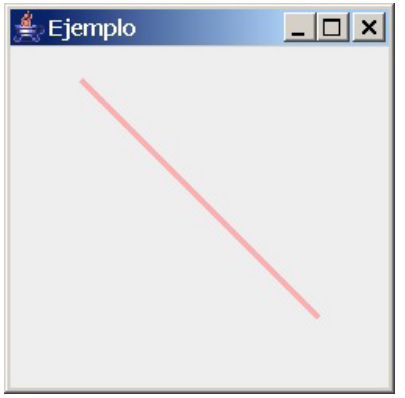
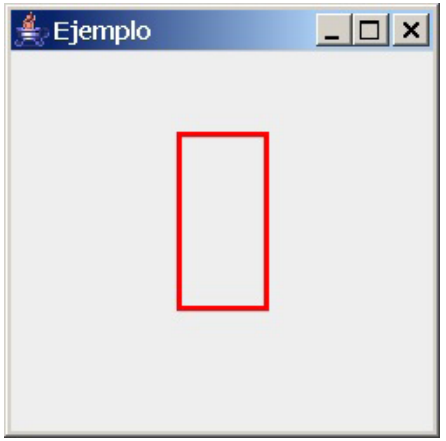


Figura Geométrica: Rectángulo

❑ Rectangle2D

- El constructor especifica en los dos primeros parámetros la posición de la esquina superior izquierda con respecto al sistema de coordenadas de la ventana, y en los dos siguientes el ancho y largo respectivamente.




```
public void paint (Graphics g) {
super.paint(g);
Graphics2D g2 = (Graphics2D)g;
// Creación del Rectangle2D
g2.setColor(Color.red);
g2.setStroke(new BasicStroke(3.0f));
Rectangle2D r = new Rectangle2D.Float(100.0f, 75.0f, 50.0f, 100.0f);
g2.draw(r);
}
```

Tratamiento de texto con Java2D

- En Java2D el texto es sencillamente un tipo especial de figura que puede ser representado y con el que se puede trabajar de diferentes maneras, más o menos parecidas a las disponibles con la interfaz Shape.
- Una fuente de texto es un conjunto completo de caracteres con ciertas características gráficas distintivas
- La fuente define la vista, tamaño y estilo característico (bold (negrita), italic (cursiva) o plain (normal)) con el que se puede dibujar una cadena de texto.

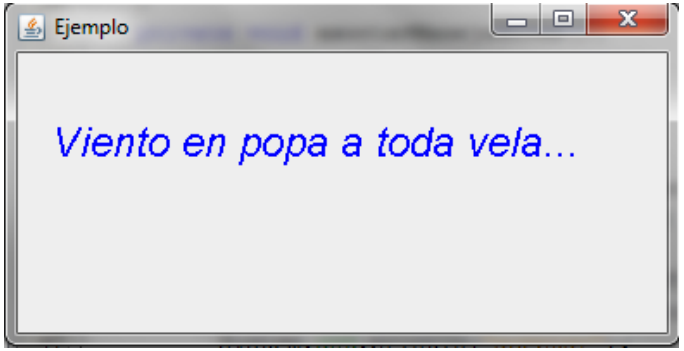
Trabajar con fuentes tipográficas

- **En Java2D para trabajar con texto hay que utilizar un objeto de la clase Font**
- **Es necesario saber qué fuentes están disponibles en el sistema operativo**

```
for (Font f: raphicsEnvironment.getLocalGraphicsEnvironment().getAllFonts())
    System.out.println(f.getName()+" - "+f.getFontName()+" - "+f.getFamily());
```

Dibujar Texto

- **Ejemplo de dibujo de texto**




```
public void paint (Graphics g) {
super.paint(g);
Graphics2D g2 = (Graphics2D)g;
g2.setColor(Color.blue);
g2.setFont(new Font("Arial", Font.ITALIC, 24));
g2.drawString("Viento en popa a toda vela...", 20, 60);
}
```

Cargar una imagen

- El método general usado para cargar una imagen desde una URL o desde un fichero es el método getImage() del paquete java.awt.Toolkit (si no se usa buffer).
- Para poder usar esta función hay que obtener antes el objeto Toolkit por defecto invocando a la función estática Toolkit.getDefaultToolkit().

Ejemplo cargar una imagen

```
public class Imagenes extends JFrame {
public Imagenes() {
super("Abrir Imagen");
}
public void paint(Graphics g) {
Graphics2D g2 = (Graphics2D)g;
Image im = Toolkit.getDefaultToolkit().getImage("Imagen.jpg");
g2.drawImage(im, 0, 20, this);
}
public static void main (String args[]) {
Imagenes v = new Imagenes();
v.setDefaultCloseOperation(EXIT_ON_CLOSE);
v.setSize(405,450);
v.setVisible(true);
}
}
```

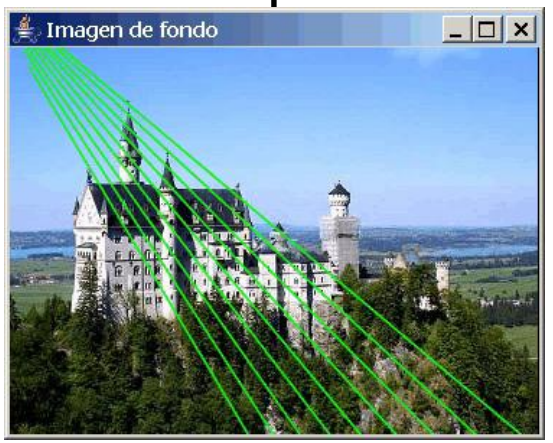
Cargar una imagen como BufferedImage

- **Java 2D soporta la carga de imágenes externas con Buffer mediante la clase ImageIO del paquete javax.imageio**
- ***Esta clase esta especializada en leer los formatos GIF,PNG,JPEG, BMP y WBMP***
- ***Ejemplo***

```
BufferedImage img = null;
 try { img = ImageIO.read(new File("map.jpg")); }
catch (IOException e) { }
```

Dibujar sobre imágenes

- **La clase Image provee métodos para este propósito**
  - `public Graphics2D createGraphics()`: este método sirve para representar una BufferedImage como una superficie de dibujo.



- **Los pasos a seguir son:**
  - Crear un objeto BufferedImage nuevo.
  - Llamar a su método **createGraphics().**
  - Establecer su configuración gráfica.
  - Dibujar sobre el objeto Graphics2D lo que se desee.
  - Pintar BufferedImage en el panel del JFrame.

Dibujar sobre imágenes

- En el siguiente código se utiliza BufferedImage , se crea una Imágen con Java2D Graphics2D con el método createGraphics()
- este Graphics2D se usa como superficie de dibujo sobre la que renderiza una imagen y luego se pintan unas líneas de color verde
- El quid de la cuestión estriba en obtener el objeto Graphics2D asociado a BufferedImage.

Ejemplo dibujar sobre imágenes

```
public void paint(Graphics g) {
BufferedImage mImagen;
Graphics2D g2 = (Graphics2D)g;
Dimension d = getSize();
int a = d.width, l = d.height;
mImagen = new BufferedImage(a, l, BufferedImage.TYPE_INT_RGB);
Graphics2D gOculta = mImagen.createGraphics();
gOculta.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON);
Image im = Toolkit.getDefaultToolkit().getImage("neuschwanstein.jpg");
gOculta.drawImage(im,0,20,this);
gOculta.setStroke(new BasicStroke(1.5f));
Color[] colors = { Color.red, Color.blue, Color.green };
gOculta.setPaint(Color.green);
for (int i = -32; i < 40; i += 8) {
gOculta.drawLine(i, i, a - i * 2, l - i * 2);
gOculta.rotate(0.05f);
} g2.drawImage(mImagen, 0, 0, this);
}
```