Unidad 1

Elementos de Interfaces Gráficas

Computación Gráfica

Esquema general de elementos gráficos

Para ubicar los objetos gráficos se utiliza un sistema de coordenadas. En computación gráfica, el sistema de coordenadas es disinto al tradicional. Vease la imagen 1.2 para una comparativa entre ambos sistemas.

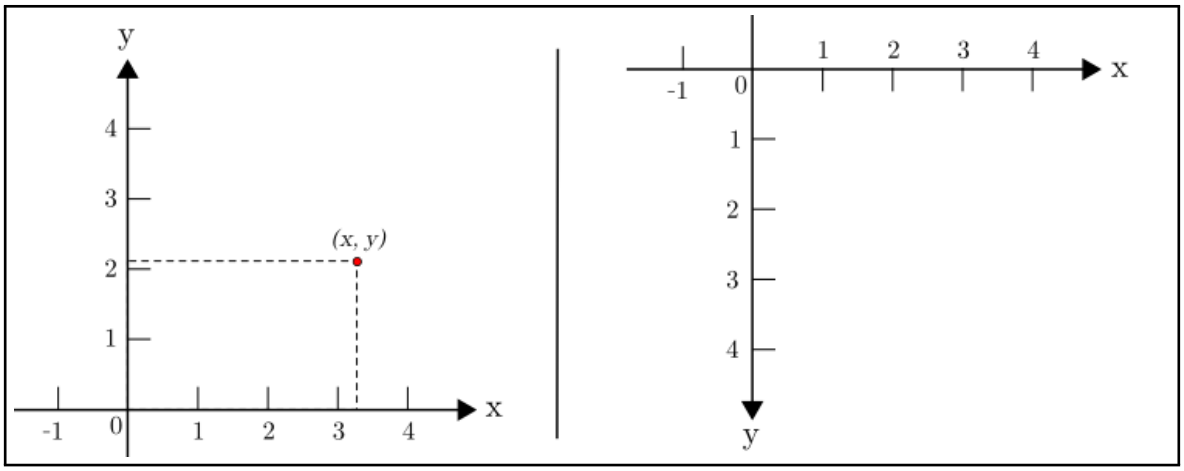

Figura 1.1: Izquierda: Sistema coordenado tradicional; Derecha: Sistema usado en computación gráfica

Prism

Prism es el componente de la arquitectura de JavaFX encargado de los trabajos de renderizado de gráficas. Posee un hilo de ejecución independiente para el renderizado, lo cual permite que un frame N sea renderizado mientras se está procesando el frame N+1 favoreciendo el desempeño general de las aplicaciónes.

Pulsos

«A pulse is an event that indicates to the JavaFX scene graph that it is time to synchronize the state of the elements on the scene graph with Prism.»

Java Documentation of Oracle[1]

En pocas palabras, cuando un pulso sucede, JavaFX se encarga de renderizar la sección o el elemento gráfico que disparó el pulso. Hay que resaltar que los programadores no tienen que lidiar directamente con los pulsos, estos eventos son generados tras bambalinas por el propio sistema de gráficos de JavaFX.

Un ejemplo de un evento desencadenador de un pulso es el cambio de ubicación de un control o la adicción o remoción de un nodo (con una parte gráfica) a un gráfo de escena vivo[2].

Canvas

La clase `javafx.scene.canvas.Canvas` es una subclase de `Node` que funciona como un lienzo para trazar primitivas gráficas y dibujos en 2D, inclusive se pueden aplicar transformaciones a los trazos. La parte interna del canvas en que se pueden realizar los trazos se conoce como **contexto gráfico**.

- En JavaFX se utiliza un objeto como motor de interpretación de los trazos en el contexto gráfico (también se le conoce como brocha). La brocha es un objeto de la clase `javafx.scene.canvas.GraphicsContext`.
- El método **getGraphiscContext2D()** sirve para generar una brocha del canvas.

Código 1.1: Estructura general para el manejo del contexto gráfico.

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;

public class MyCanvas extends Canvas{

   public MyCanvas(double width, double height) {
      super(width, height);
      draw();
   }

   public final void draw() {
      GraphicsContext gc = this.getGraphicsContext2D();
      ...
   }
}
```

[1] http://docs.oracle.com/javase/8/javafx/get-started-tutorial/jfx-architecture.htm#JFXST788
[2] Un grafo de escena vivo es un grafo que está añadido a un objeto `Stage`.

Contexto Gráfico

Como se mencionó previamente, el uso de un hilo de ejecución independiente da la ventaja de que, por un lado se puedan procesar los trazos mientras que por otro se realiza el renderizado de los gráficos.

La clase `GraphicsContext` realiza los trazos del canvas mediante un buffer[3]. Cada ocasión que se usa alguno de sus métodos de trazado, se colocan en el buffer los parámetros necesarios para renderizarse en la imagen del canvas al final de cada **pulso** en el hilo de renderizado.

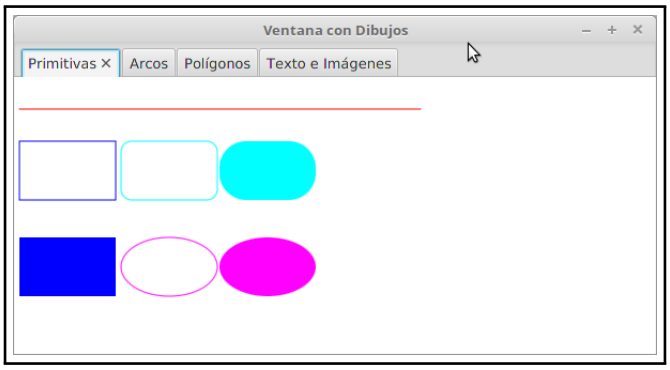

Figura 1.2: Ejemplo de trazado de primitivas gráficas.

Código 1.2: Aplicación de demostración para el manejo del contexto gráfico.

```
public class Main extends Application {

   public static final double WIDTH = 600;
   public static final double HEIGTH = 600;

   @Override
   public void start(Stage primaryStage) {
      Parent root = new GraphicContextPane(WIDTH, HEIGTH);
      primaryStage.setTitle("Ventana con Dibujos");
      primaryStage.setScene(new Scene(root, WIDTH, HEIGTH));
      primaryStage.show();
   }

   public static void main(String[] args) { launch(args); }
}
```

[3]Un buffer de datos es un espacio de la memoria en un disco o en un instrumento digital reservado para el almacenamiento temporal de información digital, mientras que está esperando ser procesada. - Wikipedia

Código 1.3: Clase auxiliar para la construcción de la interface.

```
public class GraphicContextPane extends TabPane {

   public GraphicContextPane(double width, double height) {
      setBackground(new Background(new BackgroundFill(Color.WHITE, null, null)));
      getTabs().add(new Tab("Primitivas", new PrimitivesPane(width, height)));
      getTabs().add(new Tab("Arcos", new ArcsPane(width, height)));
      getTabs().add(new Tab("Polígonos", new PolygonsPane(width, height)));
      getTabs().add(new Tab("Texto e Imágenes", new TextImagesPane(width, height)));
   }
}
```

Primitivas Gráficas

La clase GraphicsContext proporciona los siguientes métodos para presentar las primitivas gráficas:

Trazos

- void strokeLine(double x1, double y1, double x2, double y2)
- void strokeRect(double x, double y, double w, double h)
- void strokeOval(double x, double y, double w, double h)
- void strokeRoundRect(double x, double y, double w, double h, double arcWidth, double arcHeight)

Figuras con relleno

- void fillRect(double x, double y, double w, double h)
- void fillOval(double x, double y, double w, double h)
- void fillRoundRect(double x, double y, double w, double h, double arcWidth, double arcHeight)

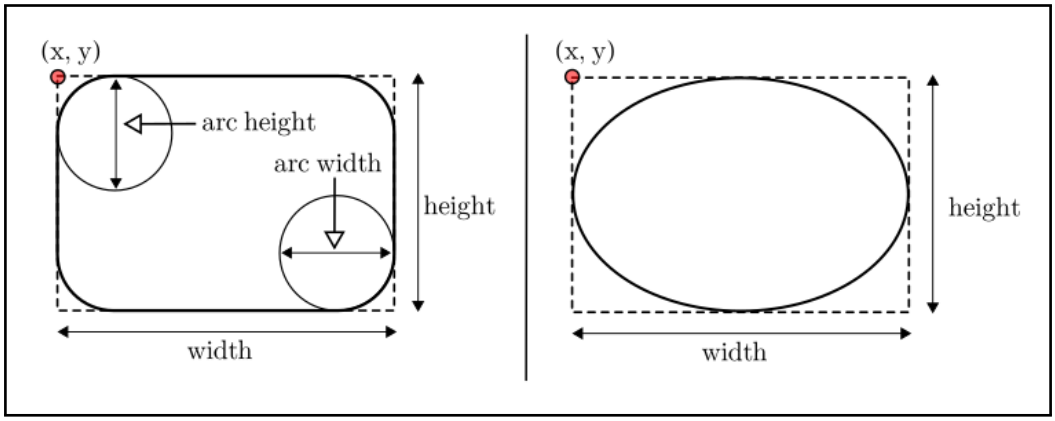

Figura 1.3: Izquierda: Cuadrado redondeado y sus parámetros; Derecha: Óvalo y sus parámetros.

Color

Para definir un nuevo color de contorno o relleno de un trazo, se utilizan los métodos

`setStroke(Paint p)` y **`setFill(Paint p)`**

respectivamente.

- Se pueden utilizar los colores definidos en la clase `javafx.scene.paint.Color` (`RED, MAGENTA, CYAN, BLACK, ...`, entre muchos otros).
  - `gc.setStroke(Color.CYAN);`
  - `gc.setFill(Color.ORANGE);`
- Se puede crear un color a partir de sus componentes RGB (rojo, verde y azul) y su opacidad.
  - `gc.setStroke(new Color(0.4, 0.4, 0, 1));`
  - `gc.setFill(new Color(0.4, 0.4, 0, 1));`
- Los componentes de color y la opacidad tienen valores entre 0 y 1.

Código 1.4: Clase PrimitivesPane.

```
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.paint.Color;

public class PrimitivesPane extends Canvas {

   public PrimitivesPane(double width, double height) {
      super(width, height);
      draw();
   }

   public final void draw() {
      GraphicsContext gc = getGraphicsContext2D();
      gc.setStroke(Color.RED);
      gc.strokeLine(5, 30, 380, 30);

      gc.setStroke(Color.BLUE);
      gc.strokeRect(5, 60, 90, 55);
      gc.setFill(Color.BLUE);
      gc.fillRect(5, 150, 90, 55);

      gc.setStroke(Color.CYAN);
      gc.strokeRoundRect(100, 60, 90, 55, 20, 20);
      gc.setFill(Color.CYAN);
      gc.fillRoundRect(192, 60, 90, 55, 50, 50);

      gc.setStroke(Color.MAGENTA);
      gc.strokeOval(100, 150, 90, 55);
      gc.setFill(Color.MAGENTA);
      gc.fillOval(192, 150, 90, 55);
   }
}
```

Arcos

- void strokeArc(double x, double y, double w, double h, double sa, double aa, ArcType closure)
- void fillArc(double x, double y, double w, double h, double sa, double aa, ArcType closure)

ArcType Indica como será el cierre del arco.

ROUND: Cierra el arco con líneas que parten desde los puntos de inicio y fin y se conectan en el centro del arco.

CORD: Cierra el arco con una línea recta que conecta los puntos de inicio y fin.

OPEN: No cierra el arco.

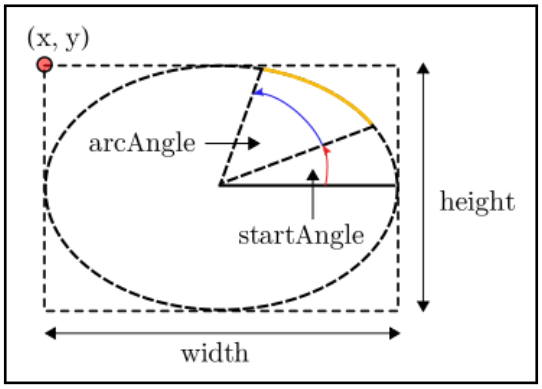

Figura 1.4: Arco y sus parámetros.

Código 1.5: Clase ArcsPane.

```
...
import javafx.scene.shape.ArcType;

public class ArcsPane extends Canvas {

   public ArcsPane(double width, double height) {
      super(width, height); draw();
   }

   public final void draw() {
      GraphicsContext gc = getGraphicsContext2D();

      gc.setStroke(Color.RED);
      gc.strokeRect(40, 35, 80, 80);
      gc.strokeRect(160, 35, 80, 80);
      gc.strokeRect(280, 35, 80, 80);
      gc.setStroke(Color.BLACK);
      gc.strokeArc(40, 35, 80, 80, 0, 360, ArcType.ROUND);
      gc.strokeArc(160, 35, 80, 80, 0, 110, ArcType.ROUND);
      gc.strokeArc(280, 35, 80, 80, 0, -270, ArcType.ROUND);
      gc.setFill(Color.ORANGE);
      gc.fillArc(40, 150, 80, 70, 0, 360, ArcType.ROUND);
      gc.fillArc(160, 150, 80, 70, 270, - 90, ArcType.ROUND);
      gc.fillArc(280, 150, 80, 70, 0, -270, ArcType.ROUND);
   }
}
```

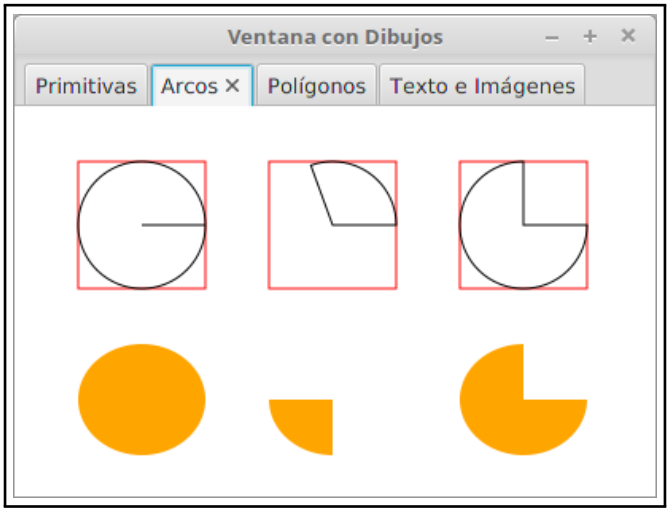

Figura 1.5: Canvas con arcos dibujados.

Polígonos y Poli-líneas

- void strokePolygon(double[] xPoints, double[] yPoints, int nPoints)
- void fillPolygon(double[] xPoints, double[] yPoints, int nPoints)
- void strokeLine(double x1, double y1, double x2, double y2)

Código 1.6: Clase PolygonsPane.

```
...

public class PolygonsPane extends Canvas {

   public PolygonsPane(double width, double height) {
      super(width, height);
      draw();
   }

   public final void draw() {
      GraphicsContext gc = getGraphicsContext2D();
      double[] x1 = {50, 140, 160, 120, 100, 45};
      double[] y1 = {50, 100, 120, 190, 190, 60};
      gc.strokePolygon(x1, y1, x1.length);

      int incX = 200;
      int incY = 0;
      double[] x3 = {incX + 50, incX + 140, incX + 160, incX + 120, incX + 100, incX + 45};
      double[] y3 = {incY + 50, incY + 100, incY + 120, incY + 190, incY + 190, incY + 60};
      gc.setFill(Color.BLUE);
      gc.fillPolygon(x3, y3, 6);

      incY = 0;
      incX = 400;
      double[] x2 = {incX + 50, incX + 140, incX + 160, incX + 120, incX + 100, incX + 45};
      double[] y2 = {incY + 50, incY + 100, incY + 120, incY + 190, incY + 190, incY + 60};
      gc.setFill(Color.RED);
      gc.strokePolyline(x2, y2, 6);
   }
}
```

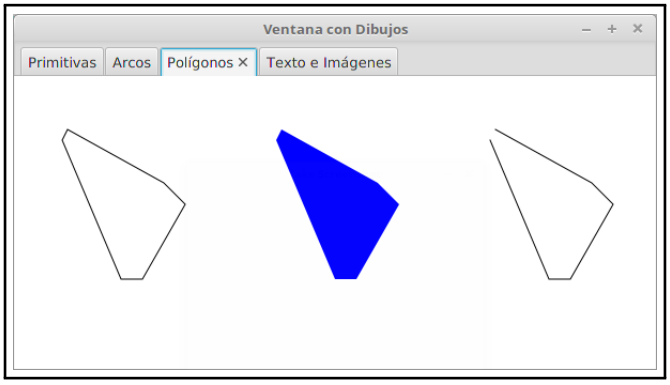

Figura 1.6: Canvas con polígonos y poli-línea.

Texto e Imágenes

`GraphicsContext` es capaz de dibujar texto e imágenes.

- void strokeText(String text, double x, double y)
- void strokeText(String text, double x, double y, double maxWidth)
- void fillText(String text, double x, double y)
- void fillText(String text, double x, double y, double maxWidth)
- void drawImage(Image img, double x, double y)
- void drawImage(Image img, double x, double y, double w, double h)
- void drawImage(Image img, double sx, double sy, double sw, double sh, double dx, double dy, double dw, double dh)

Se puede especificar la fuente que se usará para escribir en el canvas mediante el método `setFont(Font f)`.

- Se puede crear una fuente a través de alguno de los constructores de la clase Font:
  - `Font(double size)`
  - `Font(String name, double size)`
- Se puede crear una fuente a través de alguno de los métodos estáticos de la clase Font:
  - *font*`(String family)`
  - *font*`(double size)`
  - *font*`(String family, double size)`
  - *font*`(String family, FontPosture p, double size)`
  - *font*`(String family, FontWeight w, double size)`
  - *font*`(String family, FontWeight w, FontPosture p, double size)`
- Se pueden usar las fuentes cargadas en la máquina o alguno de los cinco tipos de fuente base (SansSerif, Serif, Monospaced, Dialog y DialogInput).

- FontWeight es un enumerado que puede ser: `THIN, EXTRA_LIGHT, LIGHT, NORMAL, MEDIUM, SEMI_BOLD, BOLD, EXTRA_BOLD, BLACK`.
- FontPosture es un enumerado que puede ser: `ITALIC, NORMAL`.

Código 1.7: Clase TextImagesPane.

```
import javafx.scene.image.Image;
import javafx.scene.paint.Color;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;

public class TextImagesPane extends Canvas {

   public TextImagesPane(double width, double height) {
      super(width, height);
      draw();
   }

   public final void draw() {
      GraphicsContext gc = getGraphicsContext2D();

      gc.setStroke(Color.BLUE);
      gc.strokeText("Mensaje en azul", 20, 20);

      gc.setStroke(new Color(0.4, 0.4, 0, 1));
      gc.strokeText("Mensaje en otro color", 400, 20);

      Font font;
      font = Font.font("Arial", FontWeight.BOLD, 24);
      gc.setFont(font);
      gc.setStroke(Color.RED);
      gc.strokeText("Mensaje en Arial", 20, 350);

      font = Font.font("Courier", FontWeight.BOLD, FontPosture.ITALIC, 18);
      gc.setStroke(Color.MAGENTA);
      gc.setFont(font);
      gc.strokeText("Mensaje en Courier", 400, 350);

      Image img = new Image(getClass().getResourceAsStream("/images/duke.jpg"));
      double imgWidth = img.getWidth();
      double imgHeight = img.getHeight();

      double canvasWidth = this.getWidth();
      double canvasHeight = this.getHeight();

      double x = canvasWidth / 2 - imgWidth / 2;
      double y = canvasHeight / 2 - imgHeight / 2;

      gc.drawImage(img, x, y);
   }

}
```

Figura 1.7: Canvas con texto e imagen.