Unidad 1

Elementos de Interfaces Gráficas

Conceptos sobre archivos

Desde una perspectiva de "bajo nivel"se puede definir un archivo como: Un conjunto de bits almacenados en un dispositivo y accesible a través de una ruta de acceso (path) que lo identifica. En general, existen dos criterios para clasificar a los archivos.

- **De acuerdo a su contenido:** Los archivos de caracteres (o de texto) y los de bytes (binarios).
  - **Archivo de texto:** Es aquél formado exclusivamente por caracteres y pueden crearse y visualizarse usando un editor (los archivos de código fuente de Java).
  - **Archivo binario:** No está formado por caracteres si no por los bytes que contiene y pueden representar imágenes, sonido, etc.
- **De acuerdo de acceso:** Acceso secuencial o acceso directo.
  - **Acceso secuencial:** La información del archivo es una secuencia de elementos (bytes o caracteres) de manera que para acceder al i-ésimo elemento se debe haber accedido a los i-1 elementos anteriores.
  - **Acceso directo:** La información del archivo puede ser accesada de forma directa a través de apuntadores o índices.

Lor archivos desde Java

Se utilizan las clase del paquete `java.io` para manipulación de archivos. El código que maneja archivos debe considerar que varias cosas pueden fallar al tratar de manipularlos, por ejemplo: el achivo está dañado, desconexión inesperada de la fuente de datos, etc.

Manejo de excepciones

Las excepciones son un mecanismo que permite a los métodos indicar si algún error ha sucedido (una situación excepcional), de manera que quien ha invocado al método puede detectar la situación errónea y actuar acorde al caso.

Cuando un error sucede, el método lanza (throw) una excepción y en lugar de seguir la ejecución normal de instrucciones, se busca hacia atrás en la secuencia de llamadas si existe alguna que pueda atraparla (catch). Si no se pueden atrapar, el programa acaba su ejecución y se informa del error que ha producido la excepción.

Las excepciones pueden ser:

- **Excepciones de tiempo de ejecución:** Estas no obligan al programador a tratarlas explícitamente.
- **Excepciones verificadas:** Obligan al programador a atrapar la excepción (bloque **try-catch**) o indicar que dicho método puede lanzar la excepción (declaración **throws**).

Lectura de archivos secuenciales de texto

- La lectura de un archivo se realiza utilizando **flujos** o streams.
- **FileReader** permite leer caracteres de un archivo.
- El método tt**read()** lee el siguiente carácter no leído del archivo.
- El método read() devuelve -1 cuando ya no hay más caracteres que leer.

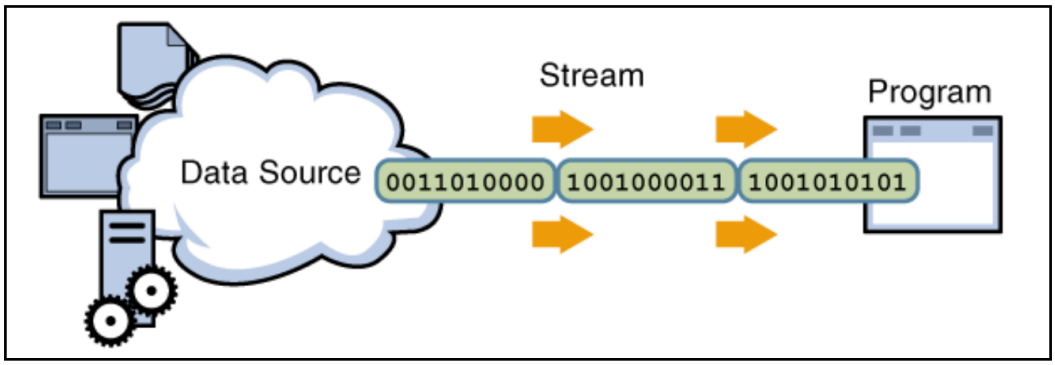

Figura 1.1: Esquema general del flujo de archivos.

Buffering

Para leer elementos en bloque se requiere de un buffer (o memoria temporal) que permita almacenar los caracteres hasta que se cumpla una condición, un salto de línea por ejemplo.

- **BufferedReader** es un buffer para leer líneas de caracteres y almacenarlos en objetos String.

- `BufferedReader` no puede leer directamente de un archivo, si no que requiere de un objeto.

- El método **readLine()** permite leer una cadena de caracteres completa hasta encontrar la marca de fin de línea.

- El método `readLine()` no regresa `null` cuando no puede leer más líneas.

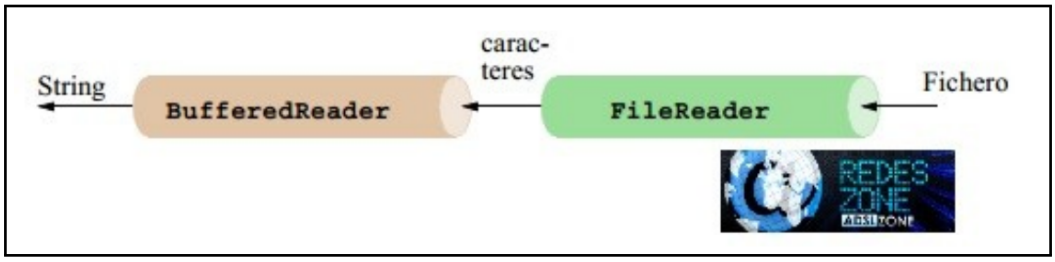

Figura 1.2: Esquema detallado del flujo de archivos.

Los elementos leídos por `FileReader` y/o `BufferedReader` deben almacenarse en algún objeto.

- **Estructura estática (String):** Antes de leer el archivo se requiere conocer el númeor de elementos a leer (caracteres o líneas). Si no se conocen, primero se deben contar los elementos y luego almacenarlos.

```
String[] lines = new String[ELEMENTS_COUNT];
```

- **Estructura dinámica (Listas enlazadas):** Al leer un elemento, se va adicionando a la estructura, qué va creciendo conforme se adicionan elementos.

```
List<String> lines = new ArrayList<>();
```

Código 1.1: Clase auxiliar para funciones con archivos.

```
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class FileHelper {

   public static List<String> readFile(String fileName) {

      List<String> lines = new ArrayList<>();

      try ( //Flujo de caracteres
            FileReader stream = new FileReader(fileName);
            //Buffer de Strings
            BufferedReader reader = new BufferedReader(stream)) {

         String line = reader.readLine();
         while (line != null) { //Si line == null se alcanzó el final del archivo
            lines.add(line);
            line = reader.readLine();
         }

      } catch (FileNotFoundException ex) {
         System.err.println("Archivo no encontrado");
         System.err.println("Detalles: " + ex.getMessage());
      } catch (IOException ex) {
         System.err.println("Error con el flujo del archivo");
         System.err.println("Detalles: " + ex.getMessage());
      }

      return lines;
   }
}
```

Aplicación para desplegar el contenido de un archivo

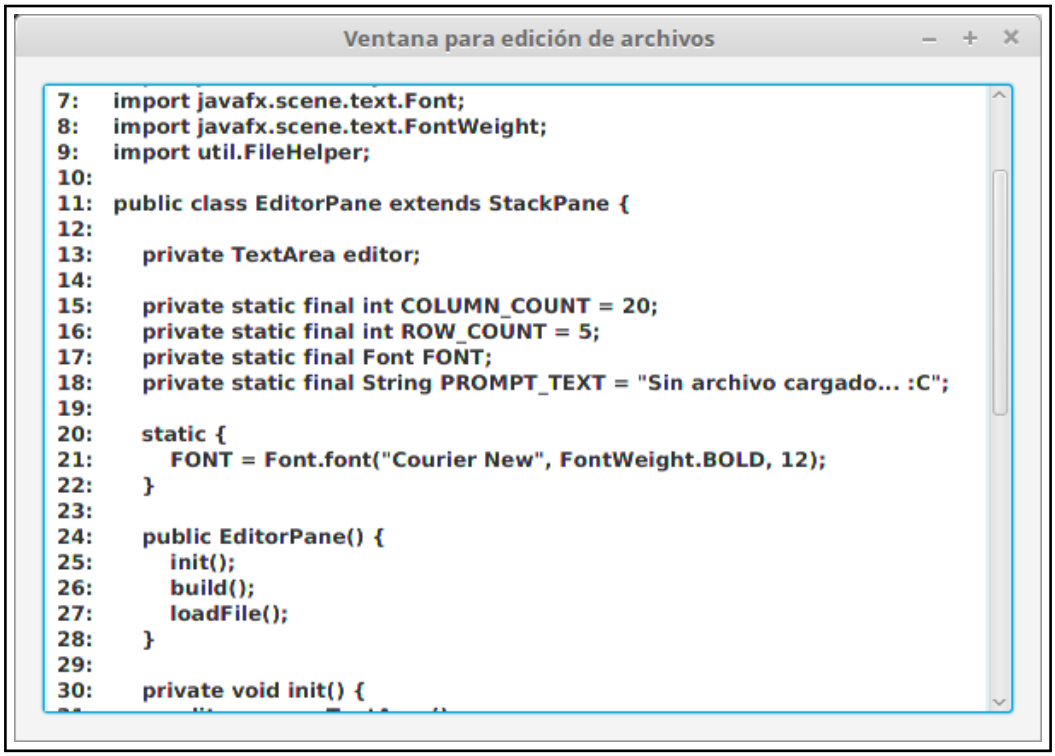

Figura 1.3: Área de texto que muestra el código leído desde un archivo

Código 1.2: Aplicación del editor que lee un archivo.

```
public class Main extends Application {

   @Override
   public void start(Stage primaryStage) {
      Parent root = new EditorPane();
      primaryStage.setTitle("Ventana para edición de archivos");
      primaryStage.setScene(new Scene(root, 600, 400));
      primaryStage.show();
   }

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

Código 1.3: Clase EditorPane.

```
import java.util.List;
import javafx.geometry.Insets;
import javafx.scene.control.TextArea;
import javafx.scene.layout.StackPane;
import javafx.scene.text.Font;
import javafx.scene.text.FontWeight;
import util.FileHelper;

public class EditorPane extends StackPane {

   private TextArea editor;

   private static final int COLUMN_COUNT = 20;
   private static final int ROW_COUNT = 5;
   private static final Font FONT;
   private static final String PROMPT_TEXT = "Sin archivo cargado... :C";

   static {
      FONT = Font.font("Courier New", FontWeight.BOLD, 12);
   }

   public EditorPane() {
      init();
      build();
      loadFile();
   }

   private void init() {
      editor = new TextArea();
      setPadding(new Insets(16));
   }

   private void build() {
      editor.setPrefColumnCount(COLUMN_COUNT);
      editor.setPrefRowCount(ROW_COUNT);
      editor.setFont(FONT);
      editor.setPromptText(PROMPT_TEXT);
      getChildren().add(editor);
   }

   private void loadFile() {
      String file = "./src/code5/EditorPane.java";

      List<String> content = FileHelper.readFile(file);

      int i = 1;

      for (String line : content) {
         editor.appendText(i + ":\t" + line + "\n");
         i++;
      }
   }
}
```

Clases para leer y dibujar un polígono

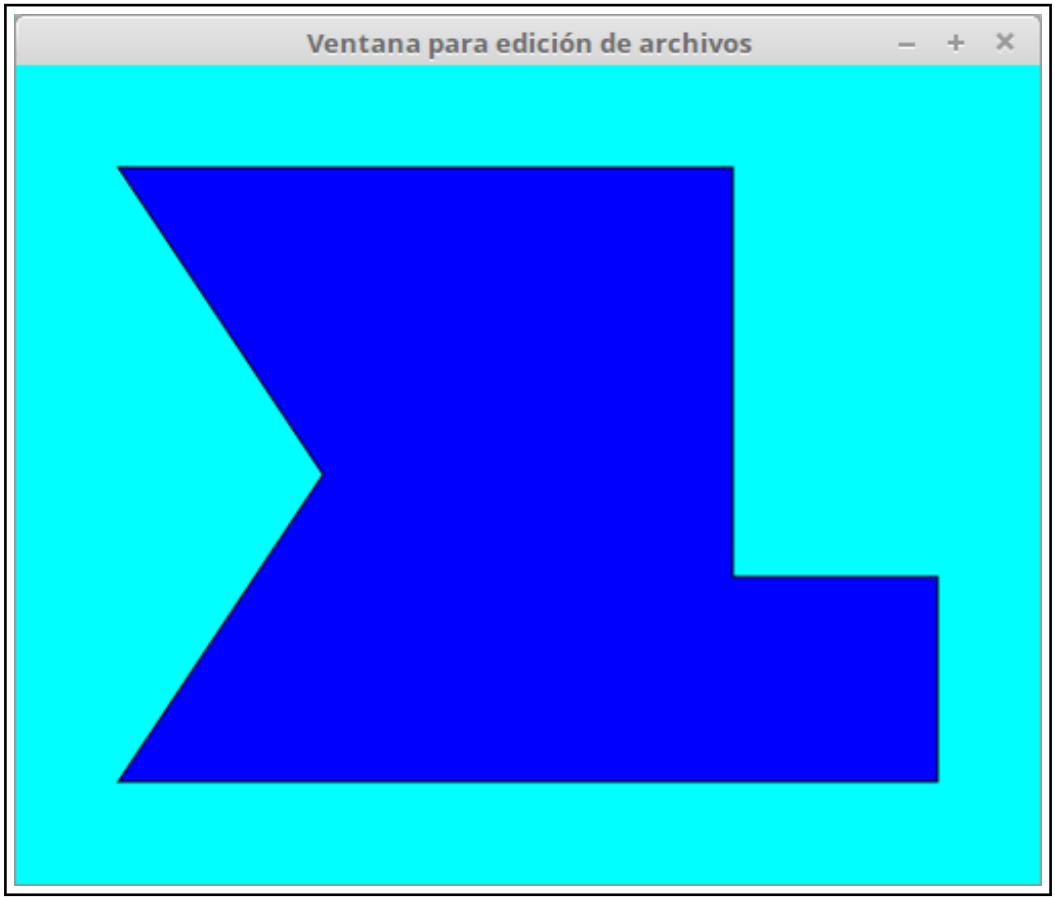

Figura 1.4: Poligono dibujado con coordenadas leídas desde un archivo

Código 1.4: Aplicación para dibujar figuras un polígono usando un archivo.

```
public class Main extends Application {

   @Override
   public void start(Stage primaryStage) {
      Parent root = new PolygonPane();
      primaryStage.setTitle("Dibujo de un polígono");
      primaryStage.setScene(new Scene(root));
      primaryStage.show();
   }

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

Código 1.5: Clase PolygonPane.

```
import java.util.List;
import javafx.geometry.Insets;
import javafx.scene.canvas.Canvas;
import javafx.scene.canvas.GraphicsContext;
import javafx.scene.layout.Background;
import javafx.scene.layout.BackgroundFill;
import javafx.scene.layout.CornerRadii;
import javafx.scene.layout.StackPane;
import javafx.scene.paint.Color;
import util.FileHelper;

public class PolygonPane extends StackPane {

   private double[] x;
   private double[] y;

   private final Canvas canvas;

   public PolygonPane() {
      canvas = new Canvas(500, 400);
      init();
      loadFile();
      draw();
   }

   private void init() {
      setBackground(new Background(
            new BackgroundFill(Color.CYAN, CornerRadii.EMPTY, Insets.EMPTY)));
      getChildren().add(canvas);
   }

   private void draw() {
      if (x != null) {
         GraphicsContext gc = canvas.getGraphicsContext2D();
         gc.setFill(Color.BLUE);
         gc.fillPolygon(x, y, x.length);
         gc.setStroke(Color.BLACK);
         gc.strokePolygon(x, y, x.length);
      }
   }

   private void loadFile() {
      String file = "./src/code6/points.txt";

      List<String> lines = FileHelper.readFile(file);

      int pointsCount = lines.size();
      x = new double[pointsCount];
      y = new double[pointsCount];

      for (int i = 0; i < pointsCount; i++) {
         String linea = lines.get(i);
         String[] coord = linea.split(",");
         x[i] = Double.parseDouble(coord[0]);
         y[i] = Double.parseDouble(coord[1]);
      }
   }
}
```