Resumen y Reescritura de JavaFX y Bibliotecas

Bibliotecas adicionales en JavaFX

JavaFX no solo proporciona una base sólida para el desarrollo de interfaces gráficas modernas en Java, sino que también permite extender sus funcionalidades a través de bibliotecas de terceros. Algunas de las más populares y útiles son:

- BootstrapFX: Esta biblioteca implementa estilos visuales inspirados en Bootstrap, el conocido framework de diseño web. Gracias a ella, los componentes visuales como botones, cajas de texto o listas se presentan con un aspecto más profesional y moderno, similar al de una página web. Es ideal cuando se busca una apariencia coherente y atractiva sin dedicar demasiado tiempo al diseño.

- ControlsFX: Amplía considerablemente el abanico de controles disponibles en JavaFX. Proporciona elementos como notificaciones, barras de tareas, validadores de formularios, autocompletados, entre otros. Es muy recomendable para quienes deseen ahorrar tiempo en el desarrollo de componentes comunes.

- FormsFX: Orientada al trabajo con formularios, esta biblioteca facilita la creación, validación y gestión de formularios con múltiples campos. Es muy útil en aplicaciones que requieren interacción continua con el usuario, como sistemas de registro o encuestas.

- FXGL: Si bien JavaFX está pensado principalmente para interfaces gráficas, FXGL va un paso más allá al permitir la creación de videojuegos. Esta biblioteca incluye utilidades específicas como manejo de físicas, lógica de colisiones, y renderizado de gráficos 2D.

- Iconli y TilesFX: Ambas se enfocan en mejorar la estética y visualización de elementos. Iconli proporciona iconos vectoriales modernos, y TilesFX permite crear paneles visuales tipo "dashboard" con indicadores como medidores, gráficos circulares y demás visualizaciones dinámicas.

- ValidatorFX: Como su nombre indica, ayuda a implementar validaciones de entrada en campos de formularios. Es especialmente útil para garantizar que los datos introducidos por el usuario cumplan ciertos requisitos antes de ser procesados.

Modelo estructural en aplicaciones JavaFX

En el desarrollo con JavaFX se recomienda utilizar el patrón MVC (Modelo - Vista - Controlador). Este enfoque divide la aplicación en tres capas claramente diferenciadas:

1. Main: Punto de entrada de la aplicación. Es el encargado de iniciar la interfaz gráfica, cargar el archivo .fxml y configurar la ventana principal. Suele extender la clase Application y redefinir el método start.

2. Controlador (Controller): Vinculado al archivo .fxml, este componente gestiona la lógica de interacción. Es donde se declaran los métodos que reaccionan a los eventos generados por los elementos visuales, como botones o cuadros de texto. Aquí se utilizan anotaciones como @FXML para enlazar elementos definidos en el archivo visual, y se implementa el método initialize() para configurar valores iniciales al cargar la escena.

3. Vista (.fxml): Archivo que describe la interfaz mediante una estructura XML. La edición puede hacerse directamente o mediante herramientas como SceneBuilder, lo que permite arrastrar y soltar componentes sin necesidad de escribir código XML manualmente.

Resumen y Reescritura de JavaFX y Bibliotecas

Elementos comunes y manejo multimedia

JavaFX dispone de numerosos componentes visuales agrupados según su funcionalidad. Algunos ejemplos son:

- Paneles de disposición (VBox, HBox, GridPane) para organizar elementos.
- Controles de entrada como TextField, TextArea, CheckBox, ComboBox.
- Controles de navegación como TabPane o Accordion.

Además, JavaFX permite integrar imágenes y sonidos fácilmente:

Para mostrar imágenes se usa ImageView y se puede cargar una imagen con:

```
protected void insertImage(ImageView imageView, String resourceName) {
    Image image = new Image(getClass().getClassLoader().getResourceAsStream(resourceName));
    imageView.setImage(image);
}
```

Para reproducir audio se utiliza la librería javafx.media, con código como:

```
protected static void insertSong(String resourceName) {
                                                    String                path                =
ControllerMainStage.class.getClassLoader().getResource(resourceName).toExternalForm();
    Media media = new Media(path);
    mediaPlayer = new MediaPlayer(media);
```

```
    mediaPlayer.setCycleCount(MediaPlayer.INDEFINITE);
    mediaPlayer.play();
}
```

Persistencia de datos con JSON

Guardar y recuperar información entre ejecuciones es una necesidad habitual. Una forma popular de hacerlo es mediante el formato JSON. En Java, con la biblioteca Gson, se puede convertir un objeto en un archivo .json o viceversa de forma sencilla:

```
public static <T> void guardarObjetoEnArchivo(String rutaArchivo, T objeto) {
    Gson gson = new Gson();
    try (FileWriter writer = new FileWriter(rutaArchivo)) {
        gson.toJson(objeto, writer);
    } catch (IOException e) {
        e.printStackTrace();
    }
}

public static <T> T cargarObjetoDesdeArchivo(String rutaArchivo, Class<T> clase) {
    Gson gson = new Gson();
    try (FileReader reader = new FileReader(rutaArchivo)) {
        return gson.fromJson(reader, clase);
    } catch (IOException e) {
        e.printStackTrace();
        return null;
```

```
    }
}
```

Gestión de registros o logs

En el contexto informático, los logs son archivos que recopilan todo tipo de eventos ocurridos en el sistema o aplicación. Permiten registrar errores, advertencias, accesos, entre otros. Son esenciales en depuración y monitorización.

Para usarlos, es necesario:

1. Añadir una biblioteca especializada.
2. Configurar un archivo .xml que determine qué tipo de eventos registrar, en qué niveles y en qué formato.

Uno de los estándares más conocidos es syslog, que ofrece un esquema estructurado para la gestión de registros. Este sistema es ampliamente utilizado en entornos empresariales para asegurar trazabilidad y auditoría.