ESCUELA POLITÉCNICA
SUPERIOR DE CÓRDOBA
Universidad de Córdoba

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

# SimAS 3.0 descendente predictivo. Simulador de analizadores sintácticos descendentes predictivos.

## Manual de Código

Autor
**D. Antonio Llamas García**

Director
**Prof. Dr. Nicolás Luis Fernández García**

**Septiembre, 2025**

UNIVERSIDAD D CORDOBA

# Índice general

**1. Introducción** **1**

1.1. Propósito del documento . . . . . 1

1.2. Descripción general del proyecto . . . . . 1

1.3. Características principales . . . . . 2

1.3.1. Interfaz de usuario moderna . . . . . 2

1.3.2. Algoritmos de análisis sintáctico . . . . . 2

1.3.3. Arquitectura modular . . . . . 2

1.4. Alcance del manual . . . . . 2

1.5. Convenciones utilizadas . . . . . 3

1.6. Estructura del documento . . . . . 3

1.7. Referencias técnicas . . . . . 3

**I Documentación externa** **5**

**2. Recursos y Requisitos de desarrollo y ejecución** **7**

2.1. Introducción . . . . . 7

2.2. Requisitos del sistema . . . . . 7

2.2.1. Requisitos mínimos de hardware . . . . . 7

2.2.2. Requisitos de software . . . . . 7

2.2.2.1. Java Runtime Environment (JRE) . . . . . 7

2.2.2.2. Sistemas operativos soportados . . . . . 8

2.3. Entorno de desarrollo . . . . . 8

2.3.1. Java Development Kit (JDK) . . . . . 8

2.3.2. JavaFX SDK . . . . . 8

2.3.3. Herramientas de construcción . . . . . 8

2.4. Arquitectura del sistema . . . . . 8
2.4.1. Visión general . . . . . 8
2.4.2. Patrones de diseño utilizados . . . . . 9
2.4.2.1. Modelo-Vista-Controlador (MVC) . . . . . 9
2.4.2.2. Observer Pattern . . . . . 9
2.4.2.3. Factory Pattern . . . . . 9
2.5. Organización modular . . . . . 9
2.5.1. Estructura de paquetes . . . . . 9
2.5.2. Dependencias entre módulos . . . . . 10
2.6. Recursos de compilación . . . . . 10
2.6.1. Scripts de construcción . . . . . 10
2.6.1.1. build.sh (Unix/Linux/macOS) . . . . . 10
2.6.1.2. build.bat (Windows) . . . . . 10
2.6.1.3. create-standalone-app.sh . . . . . 11
2.6.2. Configuración de compilación . . . . . 11

**II Documentación interna 13**

**3. Código de la aplicación 15**

3.1. Introducción . . . . . 15
3.2. Clases principales del paquete bienvenida . . . . . 15
3.2.1. Bienvenida.java . . . . . 15
3.2.2. MenuPrincipal.java . . . . . 16
3.3. Clases principales del paquete gramatica . . . . . 18
3.3.1. Gramatica.java . . . . . 18
3.3.2. Simbolo.java . . . . . 19
3.3.3. Terminal.java . . . . . 20
3.3.4. NoTerminal.java . . . . . 20
3.4. Algoritmos de análisis sintáctico . . . . . 21
3.4.1. Cálculo de conjuntos FIRST . . . . . 21
3.4.2. Cálculo de conjuntos FOLLOW . . . . . 22
3.5. Clases del paquete simulador . . . . . 22

3.5.1. PanelSimulacion.java . . . . . 22
3.5.2. SimulacionFinal.java . . . . . 24
3.6. Clases del paquete editor . . . . . 27
3.6.1. Editor.java . . . . . 27
3.7. Clases del paquete utils . . . . . 32
3.7.1. TabManager.java . . . . . 32
3.7.2. SecondaryWindow.java . . . . . 36
3.7.3. LanguageItem.java . . . . . 40
3.8. Patrones de diseño implementados . . . . . 41
3.8.1. Patrón MVC . . . . . 41
3.8.2. Patrón Observer . . . . . 41
3.8.3. Patrón Factory . . . . . 41
3.9. Consideraciones de rendimiento . . . . . 42
3.9.1. Gestión de memoria . . . . . 42
3.9.2. Optimizaciones implementadas . . . . . 42
3.10. Tratamiento de errores . . . . . 42
3.10.1. Validación de gramáticas . . . . . 42
3.10.2. Manejo de excepciones . . . . . 42

**4. Documentación de Paquetes** **43**
4.1. Introducción . . . . . 43
4.2. Paquete bienvenida . . . . . 43
4.2.1. Propósito . . . . . 43
4.2.2. Clases principales . . . . . 43
4.2.2.1. Bienvenida.java . . . . . 43
4.2.2.2. MenuPrincipal.java . . . . . 44
4.2.3. Dependencias . . . . . 44
4.3. Paquete editor . . . . . 44
4.3.1. Propósito . . . . . 44
4.3.2. Clases principales . . . . . 45
4.3.2.1. Editor.java . . . . . 45
4.3.2.2. EditorWindow.java . . . . . 45

4.3.2.3. PanelCreacionGramatica.java . . . . . 45
4.3.2.4. PanelCreacionGramaticaPaso1.java . . . . . 45
4.3.2.5. PanelCreacionGramaticaPaso2.java . . . . . 45
4.3.2.6. PanelCreacionGramaticaPaso3.java . . . . . 45
4.3.2.7. PanelCreacionGramaticaPaso4.java . . . . . 45
4.3.2.8. PanelProducciones.java . . . . . 46
4.3.2.9. PanelSimbolosNoTerminales.java . . . . . 46
4.3.2.10. PanelSimbolosTerminales.java . . . . . 46
4.3.3. Dependencias . . . . . 46
4.4. Paquete gramatica . . . . . 46
4.4.1. Propósito . . . . . 46
4.4.2. Clases principales . . . . . 46
4.4.2.1. Gramatica.java . . . . . 46
4.4.2.2. Simbolo.java . . . . . 47
4.4.2.3. Terminal.java . . . . . 47
4.4.2.4. NoTerminal.java . . . . . 47
4.4.2.5. Produccion.java . . . . . 47
4.4.2.6. Antecedente.java . . . . . 47
4.4.2.7. Consecuente.java . . . . . 47
4.4.2.8. TablaPredictiva.java . . . . . 47
4.4.2.9. TablaPredictivaPaso5.java . . . . . 47
4.4.2.10. FilaTablaPredictiva.java . . . . . 47
4.4.2.11. FuncionError.java . . . . . 48
4.4.3. Algoritmos implementados . . . . . 48
4.4.3.1. Cálculo de conjuntos FIRST . . . . . 48
4.4.3.2. Cálculo de conjuntos FOLLOW . . . . . 48
4.4.3.3. Generación de tabla predictiva . . . . . 48
4.4.4. Dependencias . . . . . 48
4.5. Paquete simulador . . . . . 48
4.5.1. Propósito . . . . . 48
4.5.2. Clases principales . . . . . 48
4.5.2.1. PanelSimuladorDesc.java . . . . . 48

4.5.2.2. PanelSimulacion.java . . . 49
4.5.2.3. SimulacionFinal.java . . . 49
4.5.2.4. PanelNuevaSimDescPaso*.java . . . 49
4.5.2.5. EditorCadenaEntradaController.java . . . 49
4.5.2.6. NuevaFuncionError.java . . . 49
4.5.2.7. PanelGramaticaOriginal.java . . . 49
4.5.3. Características del simulador . . . 49
4.5.4. Dependencias . . . 49
4.6. Paquete utils . . . 50
4.6.1. Propósito . . . 50
4.6.2. Clases principales . . . 50
4.6.2.1. SecondaryWindow.java . . . 50
4.6.2.2. TabManager.java . . . 50
4.6.2.3. TabPaneMonitor.java . . . 50
4.6.2.4. ActualizableTextos.java . . . 50
4.6.2.5. LanguageItem.java . . . 50
4.6.2.6. LanguageListCell.java . . . 50
4.6.3. Sistema de internacionalización . . . 50
4.6.4. Dependencias . . . 51
4.7. Paquete centroayuda . . . 51
4.7.1. Propósito . . . 51
4.7.2. Clases principales . . . 51
4.7.2.1. AcercaDe.java . . . 51
4.7.3. Recursos incluidos . . . 51
4.7.4. Dependencias . . . 51
4.8. Paquete vistas . . . 52
4.8.1. Propósito . . . 52
4.8.2. Archivos FXML principales . . . 52
4.8.3. Características de las vistas . . . 52
4.9. Resumen de dependencias . . . 52

**5. Sistema de Internacionalización** **53**

5.1. Introducción . . . . . 53
5.2. Arquitectura del sistema . . . . . 53
5.2.1. Componentes principales . . . . . 53
5.2.2. Idiomas soportados . . . . . 53
5.3. Implementación . . . . . 54
5.3.1. LanguageItem.java . . . . . 54
5.3.2. LanguageListCell.java . . . . . 56
5.3.3. ActualizableTextos.java . . . . . 57
5.4. Archivos de propiedades . . . . . 58
5.4.1. Estructura de los archivos . . . . . 58
5.4.2. Archivo en inglés . . . . . 58
5.4.3. Archivo en alemán . . . . . 59
5.5. Integración con la interfaz . . . . . 60
5.5.1. Cambio dinámico de idioma . . . . . 60
5.5.2. Actualización de textos . . . . . 60
5.6. Configuración y uso . . . . . 60
5.6.1. Inicialización del sistema . . . . . 60
5.6.2. Configuración del selector de idiomas . . . . . 61
5.7. Recursos gráficos . . . . . 61
5.7.1. Banderas de países . . . . . 61
5.7.2. Ubicación de recursos . . . . . 62
5.8. Consideraciones técnicas . . . . . 62
5.8.1. Rendimiento . . . . . 62
5.8.2. Mantenimiento . . . . . 62
5.8.3. Extensibilidad . . . . . 62
5.9. Mejores prácticas implementadas . . . . . 63
5.9.1. Separación de contenido y código . . . . . 63
5.9.2. Gestión de recursos . . . . . 63
5.9.3. Experiencia de usuario . . . . . 63
5.10. Pruebas y validación . . . . . 63
5.10.1. Verificación de traducciones . . . . . 63
5.10.2. Pruebas de interfaz . . . . . 63

**6. Compilación y Despliegue** **65**

6.1. Introducción . . . 65

6.2. Herramientas de construcción . . . 65

6.2.1. Java Development Kit (JDK) . . . 65

6.2.2. JavaFX SDK . . . 65

6.3. Scripts de construcción . . . 66

6.3.1. build.sh (Unix/Linux/macOS) . . . 66

6.3.2. build.bat (Windows) . . . 70

6.3.3. create-standalone-app.sh . . . 74

6.4. Proceso de compilación . . . 77

6.4.1. Paso 1: Preparación del entorno . . . 77

6.4.2. Paso 2: Compilación del código fuente . . . 77

6.4.3. Paso 3: Estructura de archivos generada . . . 78

6.5. Creación de ejecutables nativos . . . 78

6.5.1. Usando jpackage . . . 78

6.5.1.1. macOS . . . 79

6.5.1.2. Windows . . . 79

6.5.1.3. Linux . . . 79

6.5.2. Aplicación independiente para macOS . . . 79

6.6. Configuración del manifest . . . 80

6.6.1. MANIFEST.MF . . . 80

6.7. Distribución . . . 80

6.7.1. Requisitos para usuarios finales . . . 80

6.7.2. Instrucciones de instalación . . . 81

6.7.2.1. Para desarrolladores . . . 81

6.7.2.2. Para usuarios finales . . . 81

6.8. Solución de problemas . . . 81

6.8.1. Errores comunes . . . 81

6.8.1.1. Error: `JavaFX runtime components are missing` . . . 81

6.8.1.2. Error: `Main class not found` . . . 81

6.8.1.3. Error: `Permission denied` . . . 81

6.8.2. Verificación de la instalación . . . 82

6.9. Optimizaciones de rendimiento . . . . . 82
6.9.1. Configuración de JVM . . . . . 82
6.9.2. Reducción del tamaño . . . . . 82
6.10. Automatización con CI/CD . . . . . 82
6.10.1. GitHub Actions . . . . . 82
6.11. Consideraciones de seguridad . . . . . 83
6.11.1. Firma de código . . . . . 83
6.11.2. Verificación de dependencias . . . . . 83

**Bibliografía** **83**

# Índice de figuras

2.1. Arquitectura en capas de SimAS 3.0 . . . . . 9
2.2. Dependencias entre módulos de SimAS 3.0 . . . . . 10

# Capítulo 1

# Introducción

## 1.1. Propósito del documento

Este manual de código tiene como objetivo proporcionar una documentación técnica completa y detallada de la aplicación **SimAS 3.0** (Simulador de Análisis Sintáctico), desarrollada como Trabajo de Fin de Grado en Ingeniería Informática en la Universidad de Córdoba.

El documento está dirigido a desarrolladores, mantenedores del software y cualquier persona que necesite comprender la estructura interna, el funcionamiento y los algoritmos implementados en la aplicación.

## 1.2. Descripción general del proyecto

SimAS 3.0 es una aplicación educativa desarrollada en Java que implementa un simulador de analizadores sintácticos descendentes predictivos. La aplicación permite a los usuarios:

- Crear y editar gramáticas libres de contexto
- Generar automáticamente tablas de análisis predictivo
- Simular el proceso de análisis sintáctico descendente
- Visualizar el árbol de derivación resultante
- Gestionar funciones de error personalizadas

## 1.3. Características principales

### 1.3.1. Interfaz de usuario moderna

La aplicación utiliza JavaFX 17 para proporcionar una interfaz de usuario moderna e intuitiva, con:

- Diseño responsivo y adaptable
- Sistema de pestañas para múltiples proyectos
- Interfaz completamente internacionalizada
- Atajos de teclado para operaciones frecuentes

### 1.3.2. Algoritmos de análisis sintáctico

Implementa algoritmos estándar de análisis sintáctico descendente predictivo:

- Construcción de conjuntos FIRST y FOLLOW
- Generación de tablas de análisis predictivo
- Algoritmo de análisis LL(1)
- Detección y manejo de conflictos

### 1.3.3. Arquitectura modular

El sistema está diseñado con una arquitectura modular que separa claramente:

- Lógica de negocio (modelo de gramáticas)
- Interfaz de usuario (controladores y vistas)
- Algoritmos de análisis (simulador)
- Utilidades y servicios auxiliares

## 1.4. Alcance del manual

Este manual cubre los siguientes aspectos de la aplicación:

1. **Arquitectura del sistema**: Diseño general y organización de componentes
2. **Documentación de paquetes**: Descripción detallada de cada paquete Java

3. **Clases principales**: Documentación de las clases más importantes
4. **Algoritmos implementados**: Explicación de los algoritmos de análisis sintáctico
5. **Interfaz de usuario**: Documentación de la capa de presentación
6. **Sistema de internacionalización**: Gestión de múltiples idiomas
7. **Compilación y despliegue**: Proceso de construcción de la aplicación

## 1.5. Convenciones utilizadas

A lo largo de este manual se utilizarán las siguientes convenciones:

- **Código Java**: Se mostrará con sintaxis resaltada y numeración de líneas
- **Nombres de clases**: Se escribirán en formato `ClaseNombre`
- **Nombres de métodos**: Se escribirán en formato `metodoNombre()`
- **Paquetes**: Se escribirán en formato `paquete.subpaquete`
- **Archivos**: Se escribirán en formato `archivo.extensión`

## 1.6. Estructura del documento

El manual está organizado en las siguientes secciones principales:

**Parte I - Documentación Externa** Contiene información sobre recursos, requisitos y arquitectura general del sistema.

**Parte II - Documentación Interna** Incluye la documentación detallada del código fuente, algoritmos y componentes internos.

Esta estructura permite tanto una visión general del sistema como un análisis profundo de su implementación, facilitando la comprensión y mantenimiento del código.

## 1.7. Referencias técnicas

Este manual se basa en los principios fundamentales de análisis sintáctico descritos en [1] y [2], implementados utilizando las tecnologías Java [4] y JavaFX [3]. La aplicación utiliza herramientas modernas de desarrollo como [5] para la generación de ejecutables nativos.

# Parte I

# Documentación externa

# Capítulo 2

# Recursos y Requisitos de desarrollo y ejecución

## 2.1. Introducción

Este capítulo describe los recursos necesarios para el desarrollo, compilación y ejecución de la aplicación SimAS 3.0. Se incluyen tanto los requisitos técnicos como las herramientas de desarrollo utilizadas en el proyecto.

## 2.2. Requisitos del sistema

### 2.2.1. Requisitos mínimos de hardware

Para la ejecución de SimAS 3.0 se requieren los siguientes recursos mínimos de hardware:

- **Procesador**: Intel Core i3 o equivalente AMD
- **Memoria RAM**: 4 GB (recomendado 8 GB)
- **Espacio en disco**: 500 MB para la aplicación y dependencias
- **Resolución de pantalla**: 1024x768 píxeles (recomendado 1366x768 o superior)
- **Conectividad**: No requerida para funcionamiento básico

### 2.2.2. Requisitos de software

#### 2.2.2.1. Java Runtime Environment (JRE)

- **Versión requerida**: Java 17 o superior
- **Distribución**: Oracle JDK, OpenJDK, o cualquier distribución compatible

- **Arquitectura**: Compatible con x64, ARM64 (Apple Silicon)

#### 2.2.2.2. Sistemas operativos soportados

- **Windows**: Windows 10 o superior (x64)
- **macOS**: macOS 10.15 (Catalina) o superior
- **Linux**: Distribuciones modernas con soporte para JavaFX

## 2.3. Entorno de desarrollo

### 2.3.1. Java Development Kit (JDK)

- **Versión**: JDK 17 o superior
- **Recomendado**: OpenJDK 17 LTS
- **Configuración**: Variables de entorno JAVA_HOME configuradas correctamente

### 2.3.2. JavaFX SDK

- **Versión**: JavaFX 17.0.12
- **Ubicación**: Incluido en el directorio `lib/javafx-sdk-17.0.12/`
- **Propósito**: Proporciona las librerías necesarias para la interfaz gráfica

### 2.3.3. Herramientas de construcción

- **jpackage**: Incluido en JDK 14+ para crear ejecutables nativos
- **Scripts de construcción**:
  - `build.sh` - Para sistemas Unix/Linux/macOS
  - `build.bat` - Para sistemas Windows
  - `create-standalone-app.sh` - Para crear aplicación independiente

## 2.4. Arquitectura del sistema

### 2.4.1. Visión general

SimAS 3.0 sigue una arquitectura de capas bien definida que separa las responsabilidades de cada componente:

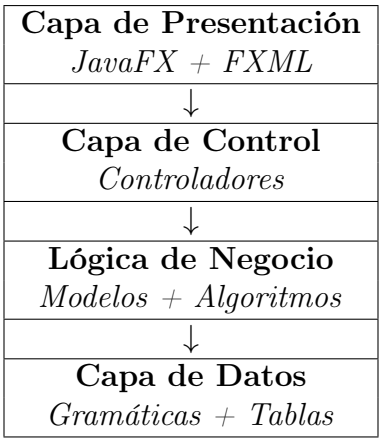


Figura 2.1: Arquitectura en capas de SimAS 3.0

### 2.4.2. Patrones de diseño utilizados

#### 2.4.2.1. Modelo-Vista-Controlador (MVC)

- **Modelo**: Clases en el paquete `gramatica` que representan las entidades del dominio
- **Vista**: Archivos FXML en el directorio `vistas`
- **Controlador**: Clases Java que manejan la lógica de presentación

#### 2.4.2.2. Observer Pattern

- Utilizado para la gestión de eventos de la interfaz de usuario
- Implementado a través de los mecanismos de JavaFX

#### 2.4.2.3. Factory Pattern

- Para la creación de diferentes tipos de símbolos (terminales, no terminales)
- En la generación de componentes de la interfaz de usuario

## 2.5. Organización modular

### 2.5.1. Estructura de paquetes

La aplicación está organizada en los siguientes paquetes principales:

`bienvenida` Gestión de la pantalla de bienvenida y menú principal

`editor` Editor de gramáticas con múltiples paneles de configuración

`gramatica` Modelo de datos y algoritmos para gramáticas libres de contexto

`simulador` Simulador de análisis sintáctico descendente predictivo

`utils` Utilidades generales, internacionalización y gestión de ventanas

`centroayuda` Sistema de ayuda y documentación integrada

`vistas` Archivos FXML que definen las interfaces de usuario

### 2.5.2. Dependencias entre módulos

| **Módulo** | **Dependencias** |
|---|---|
| bienvenida | editor, simulador, utils, gramatica |
| editor | gramatica, utils, vistas |
| simulador | gramatica, utils, vistas |
| gramatica | (independiente) |
| utils | (independiente) |
| centroayuda | utils |
| vistas | utils |

Figura 2.2: Dependencias entre módulos de SimAS 3.0

## 2.6. Recursos de compilación

### 2.6.1. Scripts de construcción

#### 2.6.1.1. build.sh (Unix/Linux/macOS)

- Compila el código fuente Java
- Empaqueta las dependencias JavaFX
- Genera el archivo JAR ejecutable
- Crea la estructura de directorios necesaria

#### 2.6.1.2. build.bat (Windows)

- Equivalente al script build.sh para sistemas Windows
- Utiliza comandos nativos de Windows
- Genera la misma estructura de archivos

#### 2.6.1.3. create-standalone-app.sh

- Crea una aplicación independiente para macOS
- Utiliza jpackage para generar un archivo .app
- Incluye todas las dependencias necesarias
- Genera un ejecutable completamente autónomo

### 2.6.2. Configuración de compilación

El proceso de compilación requiere:

- Configuración correcta de las variables de entorno Java
- Acceso a las librerías JavaFX en `lib/javafx-sdk-17.0.12/`
- Permisos de ejecución en los scripts de construcción
- Espacio suficiente en disco para archivos temporales

# Parte II

# Documentación interna

# Capítulo 3

# Código de la aplicación

## 3.1. Introducción

Este capítulo presenta una documentación detallada del código fuente de SimAS 3.0, incluyendo ejemplos de las clases más importantes, explicaciones de algoritmos clave y análisis de la implementación.

## 3.2. Clases principales del paquete bienvenida

### 3.2.1. Bienvenida.java

La clase `Bienvenida` es el punto de entrada de la aplicación y gestiona la pantalla de bienvenida.

```
1  package bienvenida;
2
3  import javafx.animation.KeyFrame;
4  import javafx.animation.Timeline;
5  import javafx.application.Application;
6  import javafx.application.Platform;
7  import javafx.fxml.FXMLLoader;
8  import javafx.scene.Scene;
9  import javafx.stage.Stage;
10 import javafx.stage.StageStyle;
11 import javafx.util.Duration;
12
13 public class Bienvenida extends Application {
14
15     @Override
16     public void start(Stage primaryStage) throws Exception {
17         FXMLLoader loader = new
            ↪  FXMLLoader(getClass().getResource("/vistas/Bienvenida.fxml"));
```

```
18          Scene scene = new Scene(loader.load());
19
20          // Configurar la ventana de bienvenida
21          primaryStage.initStyle(StageStyle.UNDECORATED);
22          primaryStage.setScene(scene);
23          primaryStage.setAlwaysOnTop(true);
24          primaryStage.setWidth(700);
25          primaryStage.setHeight(600);
26          primaryStage.setMinWidth(600);
27          primaryStage.setMinHeight(500);
28
29          // Centrar la ventana en la pantalla
30          primaryStage.centerOnScreen();
```

**Análisis del código:**

- **Líneas 1-12**: Importaciones necesarias para JavaFX y gestión de tiempo
- **Línea 13**: La clase extiende `Application`, siguiendo el patrón de JavaFX
- **Líneas 16-32**: Método `start()` que configura la ventana de bienvenida
- **Líneas 21-27**: Configuración de la ventana (sin decoraciones, siempre encima, dimensiones)
- **Líneas 35-40**: Timeline que espera 2.5 segundos antes de abrir el menú principal
- **Líneas 43-51**: Método que lanza el menú principal de forma asíncrona

### 3.2.2. MenuPrincipal.java

La clase `MenuPrincipal` es el controlador principal de la aplicación.

```
1  package bienvenida;
2
3  import editor.Editor;
4  import editor.EditorWindow;
5  import utils.SecondaryWindow;
6  import javafx.application.Application;
7  import javafx.fxml.FXML;
8  import javafx.fxml.FXMLLoader;
9  import javafx.scene.Parent;
10 import javafx.scene.Scene;
11 import javafx.scene.control.*;
12 import javafx.stage.Stage;
13 import java.awt.Desktop;
14 import java.io.File;
15 import java.io.IOException;
```

```
16 import java.util.Locale;
17 import java.util.ResourceBundle;
18 import utils.TabManager;
19 import utils.LanguageItem;
20 import utils.LanguageListCell;
21 import utils.ActualizableTextos;
22 import gramatica.Gramatica;
23 import simulador.PanelSimuladorDesc;
24 import javafx.scene.input.KeyCode;
25 import javafx.scene.input.KeyCodeCombination;
26 import javafx.scene.input.KeyCombination;
27 import java.util.Map;
28
29 public class MenuPrincipal extends Application {
30
31     @FXML private TabPane tabPane;
32     @FXML private Tab mainTab;
33     @FXML private Button btnCerrarTabs;
34     @FXML private ComboBox<LanguageItem> comboIdioma;
35     @FXML private Button btnEditor;
36     @FXML private Button btnSalir;
37     @FXML private Button btnSimulador;
38     @FXML private Button btnAyuda;
39     @FXML private Button btnTutorial;
40     @FXML private Label labelTitulo;
41     @FXML private Label labelSubtitulo;
42     @FXML private Label labelDesarrollado;
43     private Tab lastSelectedTab;
44     private ResourceBundle bundle;
45     private Locale currentLocale = new Locale("es");
46
47     @Override
48     public void start(Stage primaryStage) {
49         try {
50             // Cargar el FXML
```

**Análisis del código:**

- **Líneas 1-27**: Importaciones que incluyen JavaFX, utilidades y otros paquetes
- **Líneas 31-45**: Declaración de componentes FXML y variables de estado
- **Líneas 47-65**: Método `start()` que configura la ventana principal
- **Líneas 66-80**: Configuración de atajos de teclado para operaciones frecuentes

## 3.3. Clases principales del paquete gramatica

### 3.3.1. Gramatica.java

La clase `Gramatica` es el núcleo del sistema, implementando los algoritmos de análisis sintáctico.

```
1  package gramatica;
2
3  import com.itextpdf.text.*;
4  import com.itextpdf.text.pdf.BaseFont;
5  import com.itextpdf.text.pdf.PdfWriter;
6  import com.itextpdf.text.pdf.PdfPageEventHelper;
7  import com.itextpdf.text.pdf.ColumnText;
8  import com.itextpdf.text.pdf.PdfPTable;
9  import com.itextpdf.text.pdf.PdfPCell;
10 import com.itextpdf.text.pdf.draw.LineSeparator;
11 import javafx.beans.property.IntegerProperty;
12 import javafx.beans.property.SimpleIntegerProperty;
13 import javafx.beans.property.SimpleStringProperty;
14 import javafx.beans.property.StringProperty;
15 import javafx.collections.FXCollections;
16 import javafx.collections.ObservableList;
17 import javafx.stage.FileChooser;
18 import javafx.stage.Window;
19 import org.w3c.dom.Element;
20 import org.w3c.dom.Node;
21 import org.w3c.dom.NodeList;
22 import org.xml.sax.SAXException;
23
24 import javax.xml.parsers.DocumentBuilder;
25 import javax.xml.parsers.DocumentBuilderFactory;
26 import javax.xml.parsers.ParserConfigurationException;
27 import java.io.*;
28 import java.util.List;
29 import java.util.*;
30 import java.util.logging.Level;
31 import simulador.SimulacionFinal.HistorialPaso;
32 import java.util.logging.Logger;
33 import java.util.stream.Collectors;
34 import java.util.Enumeration;
35 import java.util.Arrays;
36 import java.util.Collections;
37
38
39 public class Gramatica {
40
41     // Propiedades para permitir el binding con la UI en JavaFX
```

```
42      private StringProperty nombre = new SimpleStringProperty();
43      private StringProperty descripcion = new SimpleStringProperty();
44      private StringProperty simbInicial = new SimpleStringProperty();
45      private StringProperty archivoFuente = new SimpleStringProperty(); //
        ↪  Nombre del archivo fuente (sin extensión)
46      private final IntegerProperty estado = new SimpleIntegerProperty();
47
48      // Colecciones de objetos de la gramática (modelo)
49      private final ObservableList<Terminal> terminales =
        ↪  FXCollections.observableArrayList();
50      private final ObservableList<NoTerminal> noTerminales =
        ↪  FXCollections.observableArrayList();
```

**Características principales:**

- Almacena símbolos terminales y no terminales
- Gestiona las producciones de la gramática
- Implementa algoritmos de cálculo de conjuntos FIRST y FOLLOW
- Genera tablas de análisis predictivo
- Valida si la gramática es LL(1)

### 3.3.2. Simbolo.java

Clase abstracta base para todos los símbolos de la gramática.

```
1  package gramatica;
2
3  import javafx.beans.property.SimpleStringProperty;
4  import javafx.beans.property.StringProperty;
5
6  /**
7   * Representa un símbolo en la gramática.
8   */
9  public class Simbolo {
10
11      private final StringProperty nombre;
12      private final StringProperty valor;
13
14      public Simbolo(String nombre, String valor) {
15          this.nombre = new SimpleStringProperty(nombre);
16          this.valor = new SimpleStringProperty(valor);
17      }
18
19      // Getter y setter para 'nombre'
```

```
20      public String getNombre() {
21          return nombre.get();
22      }
23
24      public void setNombre(String nombre) {
25          this.nombre.set(nombre);
26      }
27
28      public StringProperty nombreProperty() {
29          return nombre;
30      }
```

### 3.3.3. Terminal.java

Implementación concreta para símbolos terminales.

```
1  package gramatica;
2
3  /**
4   * Representa un símbolo terminal de la gramática.
5   */
6  public class Terminal extends Simbolo {
7
8      public Terminal(String nombre, String valor) {
9          super(nombre, valor);
10     }
11
12     @Override
13     public String toString() {
14         return getNombre(); // Usa el método de la clase padre
15     }
16 }
```

### 3.3.4. NoTerminal.java

Implementación concreta para símbolos no terminales.

```
1  package gramatica;
2
3  import javafx.collections.FXCollections;
4  import javafx.collections.ObservableList;
5
6  /**
7   * Representa un símbolo no terminal de la gramática.
8   */
9  public class NoTerminal extends Simbolo {
```

```
10
11      private boolean simboloInicial;
12      private ObservableList<Terminal> primeros;
13      private ObservableList<Terminal> siguientes;
14
15      public NoTerminal(String nombre, String valor) {
16          super(nombre, valor);
17          // Inicializamos las listas como ObservableList
18          this.primeros = FXCollections.observableArrayList();
19          this.siguientes = FXCollections.observableArrayList();
20      }
21
22      public boolean getSimboloInicial() {
23          return simboloInicial;
24      }
25
```

## 3.4. Algoritmos de análisis sintáctico

### 3.4.1. Cálculo de conjuntos FIRST

El algoritmo para calcular los conjuntos FIRST se implementa en la clase `Gramatica`:

```
1  public void calcularFirst() {
2      // Inicializar conjuntos FIRST
3      for (NoTerminal noTerminal : noTerminales) {
4          first.put(noTerminal, new HashSet<>());
5      }
6
7      boolean cambio = true;
8      while (cambio) {
9          cambio = false;
10         for (Produccion produccion : producciones) {
11             NoTerminal A = produccion.getAntecedente().getNoTerminal();
12             List<Simbolo> alfa =
    produccion.getConsecuente().getSimbolos();
13
14             // FIRST(A) = FIRST(A) U FIRST(alfa)
15             Set<Terminal> firstAlfa = calcularFirstCadena(alfa);
16             if (first.get(A).addAll(firstAlfa)) {
17                 cambio = true;
18             }
19         }
20     }
21 }
```

Listing 3.1: Algoritmo de cálculo de conjuntos FIRST

### 3.4.2. Cálculo de conjuntos FOLLOW

El algoritmo para calcular los conjuntos FOLLOW:

```
1  public void calcularFollow() {
2      // Inicializar conjuntos FOLLOW
3      for (NoTerminal noTerminal : noTerminales) {
4          follow.put(noTerminal, new HashSet<>());
5      }
6
7      // FOLLOW(S) = {$} donde S es el símbolo inicial
8      follow.get(simboloInicial).add(new Terminal("$"));
9
10     boolean cambio = true;
11     while (cambio) {
12         cambio = false;
13         for (Produccion produccion : producciones) {
14             NoTerminal A = produccion.getAntecedente().getNoTerminal();
15             List<Simbolo> alfa =
       produccion.getConsecuente().getSimbolos();
16
17             for (int i = 0; i < alfa.size(); i++) {
18                 if (alfa.get(i) instanceof NoTerminal) {
19                     NoTerminal B = (NoTerminal) alfa.get(i);
20                     List<Simbolo> beta = alfa.subList(i + 1,
       alfa.size());
21
22                     Set<Terminal> firstBeta = calcularFirstCadena(beta);
23                     if (firstBeta.contains(new Terminal("epsilon"))) {
24                         firstBeta.remove(new Terminal("epsilon"));
25                         if (follow.get(B).addAll(follow.get(A))) {
26                             cambio = true;
27                         }
28                     }
29                     if (follow.get(B).addAll(firstBeta)) {
30                         cambio = true;
31                     }
32                 }
33             }
34         }
35     }
36 }
```

Listing 3.2: Algoritmo de cálculo de conjuntos FOLLOW

## 3.5. Clases del paquete simulador

### 3.5.1. PanelSimulacion.java

Panel principal que gestiona la interfaz del simulador.

```
1  package simulador;
2
```

```
3  import javafx.geometry.Insets;
4  import javafx.scene.control.*;
5  import javafx.scene.layout.*;
6  import javafx.scene.text.Font;
7  import javafx.scene.text.FontWeight;
8  import gramatica.*;
9  import java.util.*;
10 import javafx.fxml.FXML;
11 import javafx.fxml.FXMLLoader;
12 import javafx.collections.FXCollections;
13 import javafx.collections.ObservableList;
14 import java.io.IOException;
15 import javafx.stage.Stage;
16 import utils.SecondaryWindow;
17 import java.util.ResourceBundle;
18
19 public class PanelSimulacion extends VBox {
20     @FXML private VBox root;
21     @FXML private TextField inputField;
22     @FXML private Button buttonSimular;
23     @FXML private TextArea outputArea;
24     @FXML private TableView<String> pilaTableView;
25     @FXML private TableView<String> entradaTableView;
26     @FXML private Label estadoLabel;
27
28     private Gramatica gramatica;
29     private TablaPredictivaPaso5 tablaPredictiva;
30     private List<FuncionError> funcionesError;
31     private String entrada;
32
33     // Componentes de la UI
34     private TextArea areaEntrada;
35     private TextArea areaPila;
36     private TextArea areaSalida;
37     private TreeView<String> arbolDerivacion;
38     private TextField campoEntrada;  // Campo para ingresar la cadena a
       ↪  analizar
39     private Button buttonIniciar;
40     private Button buttonSiguiente;
```

**Características principales:**

- Gestiona la interfaz de usuario del simulador
- Mantiene el estado de la simulación (pila, entrada, posición)
- Coordina la visualización de resultados
- Implementa controles de navegación paso a paso

### 3.5.2. SimulacionFinal.java

La clase `SimulacionFinal` es el componente principal del simulador final que ejecuta el análisis sintáctico paso a paso.

```
1  package simulador;
2
3  import gramatica.Gramatica;
4  import gramatica.TablaPredictivaPaso5;
5  import gramatica.FuncionError;
6  import javafx.fxml.FXML;
7  import javafx.fxml.FXMLLoader;
8  import javafx.scene.Parent;
9  import javafx.scene.control.*;
10 import javafx.scene.layout.BorderPane;
11 import javafx.stage.Modality;
12 import javafx.stage.Stage;
13 import javafx.stage.FileChooser;
14 import javafx.scene.Scene;
15 import javafx.scene.layout.FlowPane;
16 import javafx.geometry.Insets;
17 import javafx.geometry.Pos;
18 import java.io.IOException;
19 import javafx.scene.layout.HBox;
20 import javafx.scene.layout.Priority;
21 import javafx.scene.layout.VBox;
22 import javafx.collections.FXCollections;
23 import javafx.collections.ObservableList;
24 import java.util.Stack;
25 import java.util.Arrays;
26 import java.util.List;
27 import javafx.beans.property.SimpleStringProperty;
28 import java.util.ArrayList;
29 import javafx.scene.control.Tab;
30 import javafx.scene.control.TabPane;
31 import java.io.File;
32 import utils.ActualizableTextos;
33 import utils.TabManager;
34 import java.util.ResourceBundle;
35 import javafx.application.Platform;
36 import java.util.Map;
37 import java.util.HashMap;
38 import javafx.scene.control.Label;
39
40 public class SimulacionFinal extends BorderPane implements
   ↪ ActualizableTextos {
41     @FXML private TextField campoEntrada;
42     @FXML private Button btnIniciar;
43     @FXML private Button btnPaso;
```

```
44      @FXML private Button btnFinal;
45      @FXML private Button btnRetroceso;
46      @FXML private Button btnInicio;
47      // Las áreas de texto individuales se han eliminado, ahora solo
        ↪  usamos la tabla de historial
48      @FXML private Button btnEditarCadena;
49      @FXML private Button btnDerivacion;
50      @FXML private Button btnArbol;
51      @FXML private Button btnGenerarInforme;
52      @FXML private Label labelTitulo;
53      @FXML private TableView<HistorialPaso> tablaHistorial;
54      @FXML private TableColumn<HistorialPaso, String> colPaso;
55      @FXML private TableColumn<HistorialPaso, String> colPila;
56      @FXML private TableColumn<HistorialPaso, String> colEntrada;
57      @FXML private TableColumn<HistorialPaso, String> colAccion;
58      // Labels para internacionalización
59      @FXML private Label labelEntrada;
60      @FXML private Label labelHistorial;
61
62      private Gramatica gramatica;
63      private TablaPredictivaPaso5 tablaPredictiva;
64      private TabPane tabPane;
65      private ResourceBundle bundle;
66
67      // Variables para el informe PDF (copiadas del paso 6)
68      private List<FuncionError> funcionesError;
69
70      // Estado de la simulación
71      private Stack<String> pilaSimulacion;
72      private List<String> entradaSimulacion;
73      private int pasoActual;
74      private boolean simulacionEnCurso = false;
75      private ObservableList<HistorialPaso> historialObservable =
        ↪  FXCollections.observableArrayList();
76      // Lista para almacenar los estados anteriores
77      private List<EstadoSimulacion> estadosAnteriores = new ArrayList<>();
78      // Flag para saber si ya se ha realizado al menos un paso
79      private boolean seHaRealizadoAlMenosUnPaso = false;
80      // Flag para saber si estamos en un estado final (aceptación o error)
81      private boolean estadoFinalAlcanzado = false;
82
83      private String simuladorPadreId;
84      private String grupoId;
85      private int numeroGrupo;
86      private int numeroInstancia = 1;
87      public String simulacionId;
88
89      // Referencias a pestañas hijas activas
90      private Tab derivacionTab;
```

```
91      private Tab arbolTab;
92
93      // Clase para almacenar el estado de la simulación
94      private static class EstadoSimulacion {
95          Stack<String> pila;
96          List<String> entrada;
97
98          public EstadoSimulacion(Stack<String> pila, List<String> entrada,
            ↪  String accion) {
99              this.pila = new Stack<>();
100             this.pila.addAll(pila);
```

**Características principales:**

- **Análisis sintáctico completo:** Implementa el algoritmo completo de análisis sintáctico descendente predictivo
- **Simulación paso a paso:** Permite ejecutar el análisis sintáctico paso por paso con navegación completa (avance, retroceso, ir al inicio/final)
- **Visualización del estado:** Muestra en tiempo real el estado de la pila, entrada restante y acción realizada
- **Generación de derivaciones:** Crea automáticamente la derivación izquierda del análisis
- **Visualización de árboles sintácticos:** Genera representaciones gráficas del árbol sintáctico usando Graphviz
- **Estados de simulación:** Maneja estados de aceptación, error y finalización
- **Historial completo:** Mantiene un historial detallado de todos los pasos realizados
- **Internacionalización:** Soporte completo para múltiples idiomas
- **Generación de informes:** Crea informes PDF profesionales con el resultado del análisis

**Algoritmo principal de simulación:**

```
1  private void avanzarPaso() {
2      if (!simulacionEnCurso) return;
3      if (pilaSimulacion.isEmpty() || entradaSimulacion.isEmpty()) return;
4
5      // Guardar estado actual antes de modificarlo
6      estadosAnteriores.add(new EstadoSimulacion(pilaSimulacion,
       entradaSimulacion, ""));
7
8      // Marcar que se ha realizado al menos un paso
9      seHaRealizadoAlMenosUnPaso = true;
10
11     String cimaPila = pilaSimulacion.peek();
12     String simboloEntrada = entradaSimulacion.get(0);
```

```
13      String accionRealizada = "";
14
15      // Caso de aceptación
16      if (cimaPila.equals("$") && simboloEntrada.equals("$")) {
17          accionRealizada =
    bundle.getString("simulacionfinal.accion.aceptar");
18          simulacionEnCurso = false;
19          estadoFinalAlcanzado = true;
20          btnPaso.setDisable(true);
21          btnFinal.setDisable(true);
22          actualizarEstadoBotonInforme();
23          pasoActual++;
24          agregarPasoHistorial(accionRealizada);
25          actualizarVista();
26          actualizarPestañasHijas();
27          return;
28      }
29
30      // Si son iguales y terminales, consumir
31      if (cimaPila.equals(simboloEntrada)) {
32          pilaSimulacion.pop();
33          entradaSimulacion.remove(0);
34          accionRealizada =
    bundle.getString("simulacionfinal.accion.emparejar");
35      } else {
36          // Buscar producción o función de error en la tabla predictiva
37          String accion = buscarAccionTabla(cimaPila, simboloEntrada);
38          if (accion == null || accion.isEmpty()) {
39              accionRealizada =
    bundle.getString("simulacionfinal.accion.error");
40              simulacionEnCurso = false;
41              estadoFinalAlcanzado = true;
42              // ... manejo de error
43          } else {
44              // Procesar producción o función de error
45              // ... lógica de expansión
46          }
47      }
48      pasoActual++;
49      agregarPasoHistorial(accionRealizada);
50      actualizarVista();
51      actualizarPestañasHijas();
52  }
```

Listing 3.3: Algoritmo principal de simulación paso a paso

## 3.6. Clases del paquete editor

### 3.6.1. Editor.java

La clase `Editor` es el componente principal para la creación y edición de gramáticas en SimAS 3.0.

```
1  package editor;
```

```
2
3  import bienvenida.MenuPrincipal;
4  import simulador.PanelSimuladorDesc;
5
6  import gramatica.Gramatica;
7  import javafx.collections.FXCollections;
8  import javafx.collections.ObservableList;
9  import javafx.fxml.FXML;
10 import javafx.fxml.FXMLLoader;
11 import javafx.scene.Parent;
12 import javafx.scene.control.*;
13 import javafx.scene.control.Alert.AlertType;
14 import javafx.scene.layout.BorderPane;
15 import javafx.scene.layout.GridPane;
16 import javafx.scene.layout.Priority;
17 import javafx.scene.layout.VBox;
18 import javafx.stage.Stage;
19 import javafx.stage.FileChooser;
20
21 import java.io.IOException;
22 import java.io.File;
23 import java.util.ArrayList;
24 import java.util.List;
25 import java.util.Map;
26 import java.util.Optional;
27 import java.util.ResourceBundle;
28 import utils.TabManager;
29 import utils.ActualizableTextos;
30
31 public class Editor extends VBox implements ActualizableTextos {
32
33     // Modelo
34     private Gramatica gramatica = crearGramatica();
35
36     // Dependencias del sistema
37     public TabPane tabPane;
38     public MenuPrincipal menuPane;
39
40     // Componentes inyectados desde el FXML
41     @FXML private BorderPane rootPane;
42     @FXML private Label labelTitulo;
43     @FXML private Button btnAnadir;
44     @FXML private Button btnAbrir;
45     @FXML private Button btnGuardar;
46     @FXML private Button btnEditar;
47     @FXML private Button btnEliminar;
48     @FXML private Button btnValidar;
49     @FXML private Button btnInforme;
50     @FXML private Button btnSimular;
```

```
51      @FXML private Button btnSalir;
52      @FXML private TextField txtNombre;
53      @FXML private TextField txtAreaDesc;
54      @FXML private TextField txtSimInicial;
55      @FXML private ListView<String> listNoTerminales;
56      @FXML private ListView<String> listTerminales;
57      @FXML private ListView<String> listProducciones;
58      @FXML private Label labelPanelTitulo;
59      @FXML private Label labelNombre;
60      @FXML private Label labelSimboloInicial;
61      @FXML private Label labelDescripcion;
62      @FXML private Label labelNoTerminales;
63      @FXML private Label labelTerminales;
64      @FXML private Label labelProducciones;
65
66      private ResourceBundle bundle;
67
68      // Sistema de identificación para relaciones padre-hijo
69      private String editorId;
70      private static int contadorEditores = 0;
71      private boolean listenerConfigured = false;
72
73      // ============================
74      // CONSTRUCTORES
75      // ============================
76
77      /**
78       * Constructor vacio (requerido por JavaFX para cargar FXML).
79       * ATENCION: Se deben asignar `tabPane` y `menuPane` despues de la
          ↪  carga.
80       */
81      public Editor() {
82          this.gramatica = new Gramatica();
83          this.editorId = "editor_" + System.currentTimeMillis() + "_" +
             ↪  (++contadorEditores);
84          cargarFXML();
85          configurarRelacionesPadreHijo();
86      }
87
88      /**
89       * Constructor con TabPane y MenuPrincipal para uso manual.
90       */
91      public Editor(TabPane tabPane, MenuPrincipal menuPane) {
92          this.tabPane = tabPane;
93          this.menuPane = menuPane;
94          this.gramatica = new Gramatica();
95          this.editorId = "editor_" + System.currentTimeMillis() + "_" +
             ↪  (++contadorEditores);
96          cargarFXML();
```

```
97          configurarRelacionesPadreHijo();
98      }
99
100     /**
101      * Constructor con gramatica preexistente.
102      */
103     public Editor(TabPane tabPane, Gramatica gramatica, MenuPrincipal
        ↪ menuPane) {
104         this.tabPane = tabPane;
105         this.menuPane = menuPane;
106         this.gramatica = gramatica;
107         this.editorId = "editor_" + System.currentTimeMillis() + "_" +
            ↪ (++contadorEditores);
108         cargarFXML();
109         configurarRelacionesPadreHijo();
110     }
111
112     /**
113      * Constructor con TabPane, MenuPrincipal y ResourceBundle.
114      */
115     public Editor(TabPane tabPane, MenuPrincipal menuPane, ResourceBundle
        ↪ bundle) {
116         this.tabPane = tabPane;
117         this.menuPane = menuPane;
118         this.gramatica = new Gramatica();
119         this.bundle = bundle;
120         this.editorId = "editor_" + System.currentTimeMillis() + "_" +
            ↪ (++contadorEditores);
121         cargarFXML();
122         configurarRelacionesPadreHijo();
123     }
124
125     // ==========================
126     // MÉTODOS DE INICIALIZACIÓN
127     // ==========================
128
129     /**
130      * Configura las relaciones padre-hijo para cerrar pestanas hijas
        ↪ cuando se cierre el editor.
131      */
132     public void configurarRelacionesPadreHijo() {
133         if (tabPane != null && !listenerConfigured) {
134             // Configurar listener para detectar cuando se cierran
                ↪ pestañas
135
                ↪ tabPane.getTabs().addListener((javafx.collections.ListChangeListener.C
                ↪ extends Tab> change) -> {
136                 while (change.next()) {
137                     if (change.wasRemoved()) {
```

```
                        for (Tab tab : change.getRemoved()) {
                            if (tab.getContent() == this &&
                            ↪  tab.getUserData() != null) {
                                String elementId =
                                ↪  tab.getUserData().toString();
                                // Cerrar las pestañas hijas
                                TabManager.closeChildTabs(tabPane,
                                ↪  elementId);
                                // Forzar renumeración de grupos

                                ↪  TabManager.reasignarNumerosGruposGramatica(tabPane
                            }
                        }
                    }
                }
            });

```

**Características principales:**

- **Interfaz completa de edición:** Proporciona una interfaz completa para crear y editar gramáticas
- **Validación integrada:** Incluye validación automática de gramáticas con mensajes de error detallados
- **Generación de informes:** Crea informes PDF profesionales de las gramáticas
- **Sistema de pestañas jerárquico:** Gestiona relaciones padre-hijo entre pestañas
- **Internacionalización:** Soporte completo para múltiples idiomas
- **Integración con simulador:** Permite lanzar simulaciones directamente desde el editor
- **Guardado y carga:** Soporta guardar y cargar gramáticas desde archivos
- **Estados dinámicos:** Actualiza automáticamente el estado de los botones según la gramática actual

**Constructor principal:**

```
public Editor(TabPane tabPane, MenuPrincipal menuPane) {
    this.tabPane = tabPane;
    this.menuPane = menuPane;
    this.gramatica = new Gramatica();
    this.editorId = "editor_" + System.currentTimeMillis() + "_" +
    (++contadorEditores);
    cargarFXML();
    configurarRelacionesPadreHijo();
}
```

Listing 3.4: Constructor principal del Editor

## 3.7. Clases del paquete utils

### 3.7.1. TabManager.java

La clase `TabManager` es el núcleo del sistema de gestión de pestañas de SimAS 3.0, implementando un sistema complejo de pestañas jerárquicas.

```
1  package utils;
2
3  import javafx.scene.control.Tab;
4  import javafx.scene.control.TabPane;
5  import java.util.*;
6  import javafx.scene.Node;
7  import javafx.stage.Stage;
8  import javafx.application.Platform;
9  import javafx.stage.Window;
10 import javafx.scene.Scene;
11
12 public class TabManager {
13     private static final Map<TabPane, Map<Class<?>, Tab>> tabInstances =
       ↪  new HashMap<>();
14     private static final Map<TabPane, Map<String, List<Tab>>>
       ↪  parentChildRelations = new HashMap<>();
15     private static final Map<TabPane, Map<String, String>>
       ↪  elementoToGrupo = new HashMap<>(); // Mapea editorId/simuladorId
       ↪  -> grupoId
16     private static final Map<TabPane, Map<String, Integer>>
       ↪  gruposGramatica = new HashMap<>(); // Mapea grupoId ->
       ↪  numeroGrupo
17     private static final Map<TabPane, java.util.ResourceBundle>
       ↪  resourceBundles = new HashMap<>();
18
19     // Contador global para generar IDs únicos de grupo
20     public static int contadorGrupos = 0;
21
22     public static Tab getOrCreateTab(TabPane tabPane, Class<?> tabType,
       ↪  String title, Object content) {
23         return getOrCreateTab(tabPane, tabType, title, content, null,
           ↪  null);
24     }
25
26     public static Tab getOrCreateTab(TabPane tabPane, Class<?> tabType,
       ↪  String title, Object content, String parentId, String childId) {
27
28         // Inicializar el mapa para este TabPane si no existe
29         tabInstances.computeIfAbsent(tabPane, k -> new HashMap<>());
30         parentChildRelations.computeIfAbsent(tabPane, k -> new
           ↪  HashMap<>());
```

```
31          elementoToGrupo.computeIfAbsent(tabPane, k -> new HashMap<>());
32          gruposGramatica.computeIfAbsent(tabPane, k -> new HashMap<>());
33
34          // Si es una pestaña hija, verificar que estamos en la ventana
            ↪  correcta del padre
35          if (parentId != null) {
36              // Buscar la ventana que contiene el padre
37              TabPane correctTabPane = null;
38              Window currentWindow = tabPane.getScene().getWindow();
39
40              // Primero verificar la ventana actual
41              for (Tab tab : tabPane.getTabs()) {
42                  if (tab.getUserData() != null &&
                    ↪  tab.getUserData().toString().equals(parentId)) {
43                      correctTabPane = tabPane;
44                      break;
45                  }
46              }
47
48              // Si no está en la ventana actual, buscar en otras ventanas
49              if (correctTabPane == null) {
50                  for (Window window : Window.getWindows()) {
51                      if (window instanceof Stage && window !=
                        ↪  currentWindow) {
52                          Scene scene = ((Stage) window).getScene();
53                          if (scene != null) {
54                              for (Node node :
                                ↪  scene.getRoot().lookupAll(".tab-pane")) {
55                                  if (node instanceof TabPane) {
56                                      TabPane otherTabPane = (TabPane)
                                        ↪  node;
57                                      for (Tab tab :
                                        ↪  otherTabPane.getTabs()) {
58                                          if (tab.getUserData() != null &&
59
                                                ↪  tab.getUserData().toString().equals(pa
                                                ↪  {
60                                              // Encontramos el padre en
                                                ↪  otra ventana
61                                              return
                                                ↪  getOrCreateTab(otherTabPane,
                                                ↪  tabType, title, content,
                                                ↪  parentId, childId);
62                                          }
63                                      }
64                                  }
65                              }
66                          }
67                      }
```

```
68                  }
69              }
70          }
71
72          // Obtener el mapa de pestañas para este TabPane
73          Map<Class<?>, Tab> paneTabs = tabInstances.get(tabPane);
74
75          // Para editores, simuladores independientes y pestañas hijas de
            ↪  editores/simuladores, permitir múltiples instancias (no usar
            ↪  caché)
76          boolean isChildOfEditor = (parentId != null &&
            ↪  parentId.startsWith("editor_")) ||
77                                    (childId != null &&
                                      ↪  (childId.contains("editor_") ||
78
                                                                      ↪  childId.contains("creacion_")
                                                                      ↪  ||
79
                                                                      ↪  childId.startsWith("terminal
                                                                      ↪  ||
80
                                                                      ↪  childId.startsWith("no_termi
                                                                      ↪  ||
81
                                                                      ↪  childId.startsWith("producci
82
83          boolean isChildOfSimulator = (parentId != null &&
            ↪  parentId.startsWith("simulador_")) ||
84                                        (childId != null &&
                                          ↪  (childId.startsWith("gramatica_simulador_")
                                          ↪  ||
85
                                                                          ↪  childId.startsWith("funci
                                                                          ↪  ||
86
                                                                          ↪  childId.startsWith("deriva
                                                                          ↪  ||
87
                                                                          ↪  childId.startsWith("arbol_
88
89          boolean isSimuladorIndependiente = isSimuladorType(tabType) &&
            ↪  parentId != null && childId == null;
90
91          if (!isEditorType(tabType) && !isChildOfEditor &&
            ↪  !isChildOfSimulator && !isSimuladorIndependiente) {
92              // Solo usar caché para pestañas que realmente deben ser
                ↪  únicas globalmente
93              if (paneTabs.containsKey(tabType)) {
94                  Tab existingTab = paneTabs.get(tabType);
```

```
95                  if (tabPane.getTabs().contains(existingTab)) {
96                      tabPane.getSelectionModel().select(existingTab);
97                      return existingTab;
98                  } else {
99                      // Si la pestaña existe en el mapa pero no en el
                        ↪  TabPane, eliminarla del mapa
100                     paneTabs.remove(tabType);
```

**Características principales:**

- **Gestión de pestañas jerárquica:** Sistema padre-hijo para organizar pestañas relacionadas
- **Agrupación de elementos:** Agrupa editores y simuladores relacionados por número de grupo
- **Movimiento entre ventanas:** Permite arrastrar y soltar pestañas entre ventanas
- **Renovación automática:** Reasigna números de grupo cuando cambian las pestañas
- **Internacionalización:** Soporte para textos en múltiples idiomas
- **Instancias múltiples:** Permite múltiples instancias de editores y simuladores
- **Menús contextuales:** Gestiona menús contextuales personalizados para pestañas
- **Persistencia de estado:** Mantiene el estado de grupos y relaciones entre sesiones

**Método principal de creación de pestañas:**

```
1  public static Tab getOrCreateTab(TabPane tabPane, Class<?> tabType,
       String title,
2                                   Object content, String parentId, String
       childId) {
3
4      // Inicializar mapas para este TabPane
5      tabInstances.computeIfAbsent(tabPane, k -> new HashMap<>());
6      parentChildRelations.computeIfAbsent(tabPane, k -> new HashMap<>());
7      elementoToGrupo.computeIfAbsent(tabPane, k -> new HashMap<>());
8      gruposGramatica.computeIfAbsent(tabPane, k -> new HashMap<>());
9
10     // Lógica para determinar si usar caché o crear nueva instancia
11     boolean isChildOfEditor = (parentId != null &&
       parentId.startsWith("editor_")) ||
12                               (childId != null &&
       (childId.contains("editor_") ||
13
       childId.contains("creacion_") ||
14
       childId.startsWith("terminales_")));
15
16     boolean isChildOfSimulator = (parentId != null &&
       parentId.startsWith("simulador_")) ||
```

```
17                                  (childId != null &&
    (childId.startsWith("gramatica_simulador_") ||
18
    childId.startsWith("funciones_error_simulador_") ||
19
    childId.startsWith("derivacion_") ||
20
    childId.startsWith("arbol_")));
21
22      // Crear nueva pestaña según las reglas definidas
23      Tab newTab = new Tab(title, nodeContent);
24      newTab.setClosable(true);
25
26      // Establecer userData para identificar relaciones padre-hijo
27      if (childId != null) {
28          newTab.setUserData(childId);
29      } else if (parentId != null) {
30          newTab.setUserData(parentId);
31      }
32
33      // Configurar listeners y relaciones
34      configurarListenersYPadreHijo(newTab, tabPane, parentId, childId);
35
36      return newTab;
37  }
```

Listing 3.5: Método principal getOrCreateTab

### 3.7.2. SecondaryWindow.java

La clase `SecondaryWindow` implementa el sistema de ventanas secundarias de SimAS 3.0.

```
1  package utils;
2
3  import javafx.scene.Scene;
4  import javafx.scene.control.Tab;
5  import javafx.scene.control.TabPane;
6  import javafx.scene.layout.BorderPane;
7  import javafx.stage.Stage;
8  import javafx.scene.input.KeyCode;
9  import javafx.scene.input.KeyCodeCombination;
10 import javafx.scene.input.KeyCombination;
11 import javafx.scene.input.TransferMode;
12 import java.util.ResourceBundle;
13 import java.util.ArrayList;
14 import java.util.List;
15 import java.util.Map;
16 import java.util.concurrent.ConcurrentHashMap;
17 import javafx.application.Platform;
18 import java.util.Set;
```

```
19 import java.util.HashSet;
20 import java.util.Comparator;
21 import gramatica.Gramatica;
22 import simulador.PanelSimuladorDesc;
23 import java.awt.Desktop;
24 import java.io.File;
25 import java.io.IOException;
26 import javafx.scene.control.Alert;
27 import javafx.scene.control.ButtonType;
28 import javafx.scene.control.TextArea;
29 import editor.EditorWindow;
30 import editor.Editor;
31
32 public class SecondaryWindow extends EditorWindow {
33
34     private static final Map<String, SecondaryWindow> activeWindows = new
       ↪  ConcurrentHashMap<>();
35     private final String windowId;
36     private final TabPane localTabPane;
37     private final Stage stage;
38     private ResourceBundle bundle;
39     private int windowNumber;
40
41     static {
42     }
43
44     /**
45      * Obtiene una copia del mapa de ventanas secundarias activas.
46      * @return Un mapa con las ventanas secundarias activas, donde la
       ↪  clave es el ID de la ventana
47      */
48     public static Map<String, SecondaryWindow> getActiveWindows() {
49         // Limpiar ventanas que ya no están visibles
50         activeWindows.entrySet().removeIf(entry -> {
51             SecondaryWindow window = entry.getValue();
52             if (window == null || window.getStage() == null ||
                ↪  !window.getStage().isShowing()) {
53                 return true;
54             }
55             return false;
56         });
57
58         // Devolver una copia del mapa para evitar modificaciones
           ↪  concurrentes
59         return new ConcurrentHashMap<>(activeWindows);
60     }
61
62     public SecondaryWindow(ResourceBundle bundle, String baseTitle) {
63         super(null); // No inicializar la ventana en la clase padre
```

```
64
65          this.bundle = bundle;
66
67          windowNumber = getNextAvailableNumber();
68          windowId = "SecondaryWindow-" + windowNumber;
69          activeWindows.put(windowId, this);
70
71          // Crear un nuevo TabPane local para esta ventana
72          localTabPane = new TabPane();
73          configureTabPane();
74
75          // Crear un contenedor raíz para aplicar el fondo
76          BorderPane rootContainer = new BorderPane();
77          rootContainer.setCenter(localTabPane);
78
79          // Configurar la ventana
80          stage = new Stage();
81          Scene scene = new Scene(rootContainer);
82          stage.setScene(scene);
83
84          // Configurar el título con el número de ventana
85          updateWindowTitle(baseTitle);
86
87          // Configurar el tamaño de la ventana
88          stage.setWidth(800);
89          stage.setHeight(900);
90          stage.setMinWidth(600);
91          stage.setMinHeight(700);
92
93          // Aplicar estilos CSS si existen
94          try {
95
                ↪  scene.getStylesheets().add(getClass().getResource("/vistas/styles2.css
96          } catch (Exception e) {
97              e.printStackTrace();
98          }
99
100         // Configurar los atajos de teclado específicos para esta ventana
101         configureKeyboardShortcuts(stage, scene);
102
103         // Configurar el manejo de arrastre
104         configureDragAndDrop();
105
106         stage.setOnCloseRequest(event -> {
107             if (localTabPane != null) {
108                 // Desregistrar del monitor antes de cerrar
109
                    ↪  TabPaneMonitor.getInstance().desregistrarTabPane(localTabPane);
110
```

```
111                 // Cerrar pestañas localmente
112                 for (Tab tab : new ArrayList<>(localTabPane.getTabs())) {
113                     localTabPane.getTabs().remove(tab);
114                 }
115             }
116             activeWindows.remove(windowId);
117             reorderWindowNumbers();
118         });
119     }
120 
121     private void configureTabPane() {
122         localTabPane.setTabDragPolicy(TabPane.TabDragPolicy.REORDER);
123 
124         // ESTABLECER RESOURCEBUNDLE EN TABMANAGER PARA ESTA VENTANA
125         TabManager.setResourceBundle(localTabPane, bundle);
126 
127         TabManager.configurarMenuContextual(localTabPane, bundle);
128 
129         // Registrar este TabPane en el monitor para supervisión continua
130         TabPaneMonitor.getInstance().registrarTabPane(localTabPane,
            ↪ "VentanaSecundaria-" + windowNumber);
131 
132 
            ↪ localTabPane.getTabs().addListener((javafx.collections.ListChangeListener<
            ↪ change -> {
133             while (change.next()) {
134                 if (change.wasAdded()) {
135                     for (Tab tab : change.getAddedSubList()) {
136                         updateTabPaneReferences(tab);
137                     }
138                 }
139                 if (change.wasRemoved()) {
140                     if (localTabPane.getTabs().isEmpty()) {
141                         Platform.runLater(() -> stage.close());
142                     }
143 
                        ↪ TabManager.reasignarNumerosGruposGramatica(localTabPane);
144                 }
145             }
146         });
147     }
148 
149     private void configureKeyboardShortcuts(Stage stage, Scene scene) {
150         // Cerrar pestaña actual (Cmd/Ctrl + W)
```

**Características principales:**

- **Ventanas múltiples:** Permite crear múltiples ventanas secundarias independientes

- **Arrastre de pestañas:** Soporta arrastrar y soltar pestañas entre ventanas
- **Atajos de teclado:** Implementa atajos específicos para ventanas secundarias
- **Numeración automática:** Asigna números automáticamente a las ventanas
- **Internacionalización:** Soporte completo para múltiples idiomas
- **Integración con TabManager:** Funciona perfectamente con el sistema de pestañas
- **Cierre inteligente:** Gestiona el cierre automático cuando no hay pestañas
- **Estados persistentes:** Mantiene el estado de las ventanas activas

**Constructor principal:**

```
1  public SecondaryWindow(ResourceBundle bundle, String baseTitle) {
2      super(null); // No inicializar la ventana en la clase padre
3
4      this.bundle = bundle;
5
6      windowNumber = getNextAvailableNumber();
7      windowId = "SecondaryWindow-" + windowNumber;
8      activeWindows.put(windowId, this);
9
10     // Crear un nuevo TabPane local para esta ventana
11     localTabPane = new TabPane();
12     configureTabPane();
13
14     // Configurar la ventana
15     stage = new Stage();
16     Scene scene = new Scene(rootContainer);
17     stage.setScene(scene);
18
19     // Configurar el título con el número de ventana
20     updateWindowTitle(baseTitle);
21 }
```

Listing 3.6: Constructor de SecondaryWindow

### 3.7.3. LanguageItem.java

Representa un elemento de idioma en el sistema de internacionalización.

```
1  package utils;
2
3  import javafx.scene.image.Image;
4  import javafx.scene.image.ImageView;
5
6  /**
7   * Clase que representa un idioma con su bandera y nombre
8   * para ser usado en el ComboBox de selección de idioma
```

```
 */
public class LanguageItem {
    private final String name;
    private final String locale;
    private final String flagPath;
    private final ImageView flagImageView;

    public LanguageItem(String name, String locale, String flagPath) {
        this.name = name;
        this.locale = locale;
        this.flagPath = flagPath;

        // Crear el ImageView para la bandera
        ImageView tempImageView;
        try {
            Image flagImage = new
            ↪  Image(getClass().getResourceAsStream("/resources/" +
            ↪  flagPath));
            tempImageView = new ImageView(flagImage);
```

## 3.8. Patrones de diseño implementados

### 3.8.1. Patrón MVC

La aplicación implementa claramente el patrón Modelo-Vista-Controlador:

- **Modelo**: Clases en el paquete `gramatica`
- **Vista**: Archivos FXML en el directorio `vistas`
- **Controlador**: Clases Java que manejan la lógica de presentación

### 3.8.2. Patrón Observer

Utilizado para la gestión de eventos de la interfaz de usuario a través de los mecanismos de JavaFX.

### 3.8.3. Patrón Factory

Implementado para la creación de diferentes tipos de símbolos y componentes de la interfaz.

## 3.9. Consideraciones de rendimiento

### 3.9.1. Gestión de memoria

- Uso de `HashSet` para conjuntos FIRST y FOLLOW para acceso O(1)
- Implementación eficiente de algoritmos de análisis sintáctico
- Gestión adecuada de recursos JavaFX

### 3.9.2. Optimizaciones implementadas

- Cálculo incremental de conjuntos FIRST y FOLLOW
- Cache de resultados de análisis
- Lazy loading de componentes de interfaz

## 3.10. Tratamiento de errores

### 3.10.1. Validación de gramáticas

La aplicación implementa validaciones exhaustivas:

- Verificación de gramáticas bien formadas
- Detección de conflictos LL(1)
- Validación de símbolos y producciones

### 3.10.2. Manejo de excepciones

- Uso de excepciones específicas para diferentes tipos de errores
- Mensajes de error informativos para el usuario
- Recuperación graceful de errores

# Capítulo 4

# Documentación de Paquetes

## 4.1. Introducción

Este capítulo proporciona una documentación detallada de cada paquete Java que constituye la aplicación SimAS 3.0. Para cada paquete se incluye una descripción de su propósito, las clases que contiene y las relaciones con otros paquetes.

## 4.2. Paquete bienvenida

### 4.2.1. Propósito

El paquete `bienvenida` se encarga de la gestión de la pantalla de bienvenida y el menú principal de la aplicación. Es el punto de entrada de la aplicación y coordina la navegación hacia los diferentes módulos.

### 4.2.2. Clases principales

#### 4.2.2.1. Bienvenida.java

Clase principal que gestiona la pantalla de bienvenida de la aplicación.

- **Herencia**: Extiende `Application` de JavaFX
- **Responsabilidades**:
  - Mostrar la pantalla de bienvenida con información del proyecto
  - Gestionar la transición automática al menú principal
  - Configurar el estilo y comportamiento de la ventana de bienvenida
- **Métodos principales**:
  - `start(Stage primaryStage)`: Inicializa la ventana de bienvenida

- `abrirMenuPrincipal()`: Lanza el menú principal después del tiempo de espera

#### 4.2.2.2. MenuPrincipal.java

Controlador principal que gestiona el menú principal de la aplicación y coordina la navegación.

- **Herencia**: Extiende `Application` de JavaFX
- **Responsabilidades**:
  - Gestionar la interfaz del menú principal
  - Coordinar la apertura de diferentes módulos (editor, simulador, ayuda)
  - Gestionar el sistema de pestañas para múltiples proyectos
  - Implementar la internacionalización de la interfaz
  - Configurar atajos de teclado para operaciones frecuentes
- **Métodos principales**:
  - `onBtnEditorAction()`: Abre el editor de gramáticas
  - `onBtnSimuladorAction()`: Abre el simulador
  - `onBtnAyudaAction()`: Abre el sistema de ayuda
  - `cambiarIdioma()`: Cambia el idioma de la interfaz

### 4.2.3. Dependencias

- `editor.Editor`: Para abrir el editor de gramáticas
- `simulador.PanelSimuladorDesc`: Para abrir el simulador
- `utils.*`: Para gestión de ventanas, internacionalización y tabs
- `gramatica.Gramatica`: Para manejo de datos de gramáticas

## 4.3. Paquete editor

### 4.3.1. Propósito

El paquete `editor` implementa el editor de gramáticas, permitiendo a los usuarios crear, modificar y validar gramáticas libres de contexto de manera visual e intuitiva.

### 4.3.2. Clases principales

#### 4.3.2.1. Editor.java

Clase principal del editor que coordina todos los paneles de edición.

- **Responsabilidades**:
  - Coordinar los diferentes paneles de edición
  - Gestionar el flujo de trabajo de creación de gramáticas
  - Validar la gramática en cada paso del proceso
  - Gestionar la persistencia de datos

#### 4.3.2.2. EditorWindow.java

Ventana principal del editor que contiene todos los paneles de edición.

- **Responsabilidades**:
  - Gestionar la interfaz de usuario del editor
  - Coordinar la navegación entre paneles
  - Gestionar el estado de la gramática en edición

#### 4.3.2.3. PanelCreacionGramatica.java

Panel base que define la estructura común para todos los paneles de creación.

#### 4.3.2.4. PanelCreacionGramaticaPaso1.java

Panel para la definición del nombre y descripción de la gramática.

#### 4.3.2.5. PanelCreacionGramaticaPaso2.java

Panel para la definición de símbolos terminales y no terminales de la gramática.

#### 4.3.2.6. PanelCreacionGramaticaPaso3.java

Panel para la definición de producciones de la gramática.

#### 4.3.2.7. PanelCreacionGramaticaPaso4.java

Panel para la definición del símbolo inicial y validación de la gramática.

#### 4.3.2.8. PanelProducciones.java

Panel especializado para la gestión de producciones con funcionalidades avanzadas.

#### 4.3.2.9. PanelSimbolosNoTerminales.java

Panel especializado para la gestión de símbolos no terminales.

#### 4.3.2.10. PanelSimbolosTerminales.java

Panel especializado para la gestión de símbolos terminales.

### 4.3.3. Dependencias

- `gramatica.*`: Para el modelo de datos de gramáticas
- `utils.*`: Para utilidades de interfaz y gestión de ventanas
- `vistas.*`: Para las definiciones FXML de la interfaz

## 4.4. Paquete gramatica

### 4.4.1. Propósito

El paquete `gramatica` contiene el modelo de datos y los algoritmos fundamentales para el manejo de gramáticas libres de contexto y la generación de tablas de análisis predictivo.

### 4.4.2. Clases principales

#### 4.4.2.1. Gramatica.java

Clase central que representa una gramática libre de contexto completa.

- **Responsabilidades**:
  - Almacenar todos los componentes de una gramática
  - Implementar algoritmos de análisis sintáctico
  - Generar tablas de análisis predictivo
  - Validar la gramática y detectar conflictos
- **Métodos principales**:

- `generarTablaPredictiva()`: Genera la tabla de análisis predictivo
- `calcularFirst()`: Calcula los conjuntos FIRST
- `calcularFollow()`: Calcula los conjuntos FOLLOW
- `esLL1()`: Verifica si la gramática es LL(1)

#### 4.4.2.2. Simbolo.java

Clase abstracta base para todos los símbolos de la gramática.

#### 4.4.2.3. Terminal.java

Representa un símbolo terminal de la gramática.

#### 4.4.2.4. NoTerminal.java

Representa un símbolo no terminal de la gramática.

#### 4.4.2.5. Produccion.java

Representa una producción de la gramática con su antecedente y consecuente.

#### 4.4.2.6. Antecedente.java

Representa el lado izquierdo de una producción (no terminal).

#### 4.4.2.7. Consecuente.java

Representa el lado derecho de una producción (secuencia de símbolos).

#### 4.4.2.8. TablaPredictiva.java

Clase base para la representación de tablas de análisis predictivo.

#### 4.4.2.9. TablaPredictivaPaso5.java

Implementación específica de la tabla predictiva con funcionalidades avanzadas.

#### 4.4.2.10. FilaTablaPredictiva.java

Representa una fila de la tabla predictiva.

#### 4.4.2.11. FuncionError.java

Representa una función de error personalizada para el análisis.

### 4.4.3. Algoritmos implementados

#### 4.4.3.1. Cálculo de conjuntos FIRST

El algoritmo calcula para cada símbolo no terminal el conjunto de símbolos terminales que pueden aparecer al inicio de las cadenas derivadas.

#### 4.4.3.2. Cálculo de conjuntos FOLLOW

El algoritmo calcula para cada símbolo no terminal el conjunto de símbolos terminales que pueden aparecer inmediatamente después de él en alguna derivación.

#### 4.4.3.3. Generación de tabla predictiva

Utiliza los conjuntos FIRST y FOLLOW para construir la tabla de análisis predictivo LL(1).

### 4.4.4. Dependencias

Este paquete es independiente y no tiene dependencias externas, sirviendo como base para otros paquetes.

## 4.5. Paquete simulador

### 4.5.1. Propósito

El paquete `simulador` implementa el simulador de análisis sintáctico descendente predictivo, permitiendo a los usuarios simular el proceso de análisis paso a paso.

### 4.5.2. Clases principales

#### 4.5.2.1. PanelSimuladorDesc.java

Panel principal del simulador que coordina todos los componentes de simulación.

#### 4.5.2.2. PanelSimulacion.java

Panel que gestiona la interfaz de usuario del simulador y el estado de la simulación.

#### 4.5.2.3. SimulacionFinal.java

Clase que implementa la lógica de simulación del análisis sintáctico.

#### 4.5.2.4. PanelNuevaSimDescPaso*.java

Conjunto de paneles para la configuración paso a paso de una nueva simulación.

#### 4.5.2.5. EditorCadenaEntradaController.java

Controlador para la edición de cadenas de entrada a analizar.

#### 4.5.2.6. NuevaFuncionError.java

Panel para la creación de nuevas funciones de error personalizadas.

#### 4.5.2.7. PanelGramaticaOriginal.java

Panel que muestra la gramática original utilizada en la simulación.

### 4.5.3. Características del simulador

- Simulación paso a paso del análisis sintáctico
- Visualización de la pila de análisis
- Visualización del estado de la entrada
- Generación del árbol de derivación
- Manejo de errores con funciones personalizadas
- Interfaz intuitiva con controles de navegación

### 4.5.4. Dependencias

- `gramatica.*`: Para acceso a gramáticas y tablas predictivas
- `utils.*`: Para utilidades de interfaz
- `vistas.*`: Para las definiciones FXML

## 4.6. Paquete utils

### 4.6.1. Propósito

El paquete `utils` contiene utilidades generales, servicios de internacionalización y herramientas de gestión de ventanas que son utilizadas por toda la aplicación.

### 4.6.2. Clases principales

#### 4.6.2.1. SecondaryWindow.java

Utilidad para la gestión de ventanas secundarias de la aplicación.

#### 4.6.2.2. TabManager.java

Gestiona el sistema de pestañas para múltiples proyectos simultáneos.

#### 4.6.2.3. TabPaneMonitor.java

Monitor que gestiona el estado y comportamiento de las pestañas.

#### 4.6.2.4. ActualizableTextos.java

Interfaz para componentes que pueden actualizar sus textos según el idioma seleccionado.

#### 4.6.2.5. LanguageItem.java

Representa un elemento de idioma en el sistema de internacionalización.

#### 4.6.2.6. LanguageListCell.java

Celda personalizada para la visualización de idiomas en listas.

### 4.6.3. Sistema de internacionalización

El paquete incluye archivos de propiedades para múltiples idiomas:

- `messages_es.properties`: Textos en español
- `messages_en.properties`: Textos en inglés

- `messages_de.properties`: Textos en alemán
- `messages_fr.properties`: Textos en francés
- `messages_ja.properties`: Textos en japonés
- `messages_pt.properties`: Textos en portugués

### 4.6.4. Dependencias

Este paquete es independiente y proporciona servicios a otros paquetes.

## 4.7. Paquete centroayuda

### 4.7.1. Propósito

El paquete `centroayuda` implementa el sistema de ayuda integrado de la aplicación, incluyendo documentación, tutoriales y información sobre el proyecto.

### 4.7.2. Clases principales

#### 4.7.2.1. AcercaDe.java

Ventana que muestra información sobre la aplicación, desarrolladores y versión.

### 4.7.3. Recursos incluidos

- `ayuda.html`: Documentación principal de ayuda
- `SimAS.html`: Información específica sobre la aplicación
- `Tema_*.pdf`: Documentos temáticos sobre análisis sintáctico
- `imagenes/`: Recursos gráficos para la documentación

### 4.7.4. Dependencias

- `utils.*`: Para utilidades de interfaz

## 4.8. Paquete vistas

### 4.8.1. Propósito

El paquete `vistas` contiene todos los archivos FXML que definen las interfaces de usuario de la aplicación, siguiendo el patrón de separación de vista y lógica.

### 4.8.2. Archivos FXML principales

- `Bienvenida.fxml`: Pantalla de bienvenida
- `MenuPrincipal.fxml`: Menú principal de la aplicación
- `Editor.fxml`: Interfaz principal del editor
- `PanelCreacionGramaticaPaso*.fxml`: Paneles de creación de gramáticas
- `PanelSimulacion.fxml`: Interfaz del simulador
- `SimulacionFinal.fxml`: Interfaz de simulación final
- `styles2.css`: Estilos CSS para la aplicación

### 4.8.3. Características de las vistas

- Diseño responsivo y adaptable
- Uso de estilos CSS para personalización
- Integración con el sistema de internacionalización
- Separación clara entre presentación y lógica

## 4.9. Resumen de dependencias

La siguiente tabla resume las dependencias principales entre paquetes:

| **Paquete** | **Dependencias principales** |
|---|---|
| bienvenida | editor, simulador, utils, gramatica |
| editor | gramatica, utils, vistas |
| simulador | gramatica, utils, vistas |
| gramatica | (independiente) |
| utils | (independiente) |
| centroayuda | utils |
| vistas | utils |

Tabla 4.1: Dependencias entre paquetes de SimAS 3.0

# Capítulo 5

# Sistema de Internacionalización

## 5.1. Introducción

SimAS 3.0 implementa un sistema completo de internacionalización (i18n) que permite a la aplicación adaptarse a diferentes idiomas y regiones. Este capítulo documenta la implementación, configuración y uso del sistema de internacionalización.

## 5.2. Arquitectura del sistema

### 5.2.1. Componentes principales

El sistema de internacionalización está compuesto por los siguientes elementos:

- **Archivos de propiedades**: Contienen las traducciones para cada idioma
- **LanguageItem**: Representa un idioma disponible
- **LanguageListCell**: Celda personalizada para mostrar idiomas
- **ActualizableTextos**: Interfaz para componentes que pueden actualizar sus textos
- **ResourceBundle**: Mecanismo de Java para cargar recursos localizados

### 5.2.2. Idiomas soportados

La aplicación soporta los siguientes idiomas:

| Código | Idioma | Archivo |
|---|---|---|
| es | Español | messages_es.properties |
| en | English | messages_en.properties |
| de | Deutsch | messages_de.properties |
| fr | Français | messages_fr.properties |
| ja | Japanese | messages_ja.properties |
| pt | Português | messages_pt.properties |

Tabla 5.1: Idiomas soportados en SimAS 3.0

## 5.3. Implementación

### 5.3.1. LanguageItem.java

La clase `LanguageItem` representa un idioma disponible en el sistema:

```
1  package utils;
2
3  import javafx.scene.image.Image;
4  import javafx.scene.image.ImageView;
5
6  /**
7   * Clase que representa un idioma con su bandera y nombre
8   * para ser usado en el ComboBox de selección de idioma
9   */
10 public class LanguageItem {
11     private final String name;
12     private final String locale;
13     private final String flagPath;
14     private final ImageView flagImageView;
15
16     public LanguageItem(String name, String locale, String flagPath) {
17         this.name = name;
18         this.locale = locale;
19         this.flagPath = flagPath;
20
21         // Crear el ImageView para la bandera
22         ImageView tempImageView;
23         try {
24             Image flagImage = new
                ↪  Image(getClass().getResourceAsStream("/resources/" +
                ↪  flagPath));
25             tempImageView = new ImageView(flagImage);
26             tempImageView.setFitHeight(16);
27             tempImageView.setFitWidth(24);
28             tempImageView.setPreserveRatio(true);
```

```
29          } catch (Exception e) {
30              // Si no se puede cargar la imagen, crear un ImageView vacío
31              tempImageView = new ImageView();
32              System.err.println("No se pudo cargar la bandera para " +
                ↪  name + ": " + e.getMessage());
33          }
34          this.flagImageView = tempImageView;
35      }
36
37      public String getName() {
38          return name;
39      }
40
41      public String getLocale() {
42          return locale;
43      }
44
45      public String getFlagPath() {
46          return flagPath;
47      }
48
49      public ImageView getFlagImageView() {
50          return flagImageView;
51      }
52
53      @Override
54      public String toString() {
55          return name;
56      }
57
58      @Override
59      public boolean equals(Object obj) {
60          if (this == obj) return true;
61          if (obj == null || getClass() != obj.getClass()) return false;
62          LanguageItem that = (LanguageItem) obj;
63          return locale.equals(that.locale);
64      }
65
66      @Override
67      public int hashCode() {
68          return locale.hashCode();
69      }
70  }
```

**Características principales:**

- **name**: Nombre del idioma en su idioma nativo
- **code**: Código ISO del idioma

- **flag**: Bandera representativa del país/región
- **locale**: Objeto Locale de Java para la localización

### 5.3.2. LanguageListCell.java

Celda personalizada para mostrar idiomas en listas desplegables:

```
1  package utils;
2
3  import javafx.scene.control.ListCell;
4  import javafx.scene.layout.HBox;
5  import javafx.scene.layout.Priority;
6  import javafx.scene.paint.Color;
7  import javafx.scene.control.Label;
8  import javafx.scene.image.ImageView;
9
10 /**
11  * Celda personalizada para mostrar idiomas con banderas en el ComboBox
12  */
13 public class LanguageListCell extends ListCell<LanguageItem> {
14
15     private final HBox content;
16     private final Label textLabel;
17
18     public LanguageListCell() {
19         content = new HBox();
20         content.setSpacing(8);
21
22         textLabel = new Label();
23         textLabel.setTextFill(Color.WHITE); // Establecer el color del
           ↪  texto en blanco
24         textLabel.setStyle("-fx-text-fill: white;"); // CSS adicional
           ↪  para asegurar el color blanco
25         HBox.setHgrow(textLabel, Priority.ALWAYS);
26
27         content.getChildren().addAll(textLabel);
28     }
29
30     @Override
31     protected void updateItem(LanguageItem item, boolean empty) {
32         super.updateItem(item, empty);
33
34         if (empty || item == null) {
35             setGraphic(null);
36             setText(null);
37         } else {
38             // Limpiar contenido anterior
```

```
39              content.getChildren().clear();
40
41              // Crear una nueva instancia del ImageView para evitar
                ↪  problemas de referencia
42              ImageView flagView = new
                ↪  ImageView(item.getFlagImageView().getImage());
43              flagView.setFitHeight(16);
44              flagView.setFitWidth(24);
45              flagView.setPreserveRatio(true);
46
47              // Agregar la bandera
48              content.getChildren().add(flagView);
49
50              // Agregar el texto del idioma
51              textLabel.setText(item.getName());
52              content.getChildren().add(textLabel);
53
54              // Asegurar que el contenido se mantenga visible
55              setGraphic(content);
56              setText(null);
57
58              // Forzar la actualización del layout
59              content.requestLayout();
60          }
61      }
62  }
```

**Funcionalidades:**

- Muestra la bandera del país/región
- Muestra el nombre del idioma
- Formato visual atractivo
- Integración con JavaFX

### 5.3.3. ActualizableTextos.java

Interfaz que permite a los componentes actualizar sus textos dinámicamente:

```
1  package utils;
2  import java.util.ResourceBundle;
3
4  public interface ActualizableTextos {
5      void actualizarTextos(ResourceBundle bundle);
6  }
```

## 5.4. Archivos de propiedades

### 5.4.1. Estructura de los archivos

Los archivos de propiedades siguen el formato estándar de Java:

```
# SimAS 3.0 - Mensajes en español
# Archivo: messages_es.properties

# Menú principal
menu.title=SimAS 3.0
menu.subtitle=Simulador de Análisis Sintáctico
menu.editor=Editor de Gramáticas
menu.simulator=Simulador Descendente
menu.help=Manual de Usuario
menu.exit=Salir

# Editor
editor.title=Editor de Gramáticas
editor.step1=Términos No Terminales
editor.step2=Términos Terminales
editor.step3=Producciones
editor.step4=Tabla Predictiva

# Simulador
simulator.title=Simulador Descendente
simulator.input=Cadena de Entrada
simulator.start=Iniciar Simulación
simulator.next=Siguiente Paso
simulator.reset=Reiniciar

# Mensajes de error
error.invalid.grammar=Gramática inválida
error.conflict=Conflicto detectado en la gramática
error.syntax=Error de sintaxis
```

Listing 5.1: Ejemplo de archivo de propiedades

### 5.4.2. Archivo en inglés

```
# Main Menu
btn.editor=Editor
btn.simulador=Descendant Simulator
btn.ayuda=User Manual
btn.tutorial=Interactive Tutorial
btn.salir=Exit
btn.cerrar=Close all tabs

# Labels
label.entrada=Input string
label.idioma=Language:
label.titulo=SimAS 3.0
```

```
label.subtitulo=Syntax Analysis Simulator
label.desarrollado=Developed by Antonio

# Titles
title.menu=Menu
title.simulador=Syntax Analysis Simulator
title.cerrar.pestanas=Close tabs

# Messages
msg.confirmar.cerrar=Are you sure you want to close all tabs except the
 ↪ main one?
msg.confirmar.salir=Are you sure you want to exit?
msg.error.manual=Could not open the user manual
msg.error.tutorial=Could not open the tutorial
msg.error.escritorio=Desktop functionality is not supported on this
 ↪ system
msg.error.archivo=File not found

# Tooltips
tooltip.cerrar=Close all tabs except the main one
```

### 5.4.3. Archivo en alemán

```
# Hauptmenü
btn.editor=Editor
btn.simulador=Absteigender Simulator
btn.ayuda=Benutzerhandbuch
btn.tutorial=Interaktives Tutorial
btn.salir=Beenden
btn.cerrar=Alle Tabs schließen

# Beschriftungen
label.entrada=Eingabezeichenkette
label.idioma=Sprache:
label.titulo=SimAS 3.0
label.subtitulo=Syntaxanalyse-Simulator
label.desarrollado=Entwickelt von Antonio

# Titel
title.menu=Menü
title.simulador=Syntaxanalyse-Simulator
title.cerrar.pestanas=Tabs schließen

# Nachrichten
msg.confirmar.cerrar=Sind Sie sicher, dass Sie alle Tabs außer dem
 ↪ Haupttab schließen möchten?
msg.confirmar.salir=Sind Sie sicher, dass Sie beenden möchten?
msg.error.manual=Benutzerhandbuch konnte nicht geöffnet werden
```

```
msg.error.tutorial=Tutorial konnte nicht geöffnet werden
msg.error.escritorio=Desktop-Funktionalität wird auf diesem System nicht
 ↪  unterstützt
msg.error.archivo=Datei nicht gefunden

# Tooltips
tooltip.cerrar=Alle Tabs außer dem Haupttab schließen
```

## 5.5. Integración con la interfaz

### 5.5.1. Cambio dinámico de idioma

El sistema permite cambiar el idioma de la aplicación en tiempo de ejecución:

```
public void cambiarIdioma(LanguageItem idiomaSeleccionado) {
    currentLocale = idiomaSeleccionado.getLocale();
    bundle = ResourceBundle.getBundle("utils.messages", currentLocale);

    // Actualizar textos de la interfaz
    actualizarTextos();

    // Notificar a otros componentes
    notificarCambioIdioma();
}
```

Listing 5.2: Método para cambiar idioma

### 5.5.2. Actualización de textos

Los componentes que implementan `ActualizableTextos` pueden actualizar sus textos:

```
@Override
public void actualizarTextos(ResourceBundle bundle) {
    labelTitulo.setText(bundle.getString("menu.title"));
    labelSubtitulo.setText(bundle.getString("menu.subtitle"));
    btnEditor.setText(bundle.getString("menu.editor"));
    btnSimulador.setText(bundle.getString("menu.simulator"));
    btnAyuda.setText(bundle.getString("menu.help"));
    btnSalir.setText(bundle.getString("menu.exit"));
}
```

Listing 5.3: Implementación de actualización de textos

## 5.6. Configuración y uso

### 5.6.1. Inicialización del sistema

El sistema se inicializa al arrancar la aplicación:

```
1  private void inicializarInternacionalizacion() {
2      // Cargar idioma por defecto (español)
3      currentLocale = new Locale("es");
4      bundle = ResourceBundle.getBundle("utils.messages", currentLocale);
5
6      // Configurar combo de idiomas
7      configurarComboIdiomas();
8
9      // Actualizar textos iniciales
10     actualizarTextos();
11 }
```

Listing 5.4: Inicialización del sistema de i18n

### 5.6.2. Configuración del selector de idiomas

```
1  private void configurarComboIdiomas() {
2      List<LanguageItem> idiomas = Arrays.asList(
3          new LanguageItem("Espanol", "es", "espana.png"),
4          new LanguageItem("English", "en", "england.png"),
5          new LanguageItem("Deutsch", "de", "alemania.png"),
6          new LanguageItem("Francais", "fr", "francia.png"),
7          new LanguageItem("Japanese", "ja", "japon.png"),
8          new LanguageItem("Portugues", "pt", "portugal.png")
9      );
10
11     comboIdioma.setItems(FXCollections.observableArrayList(idiomas));
12     comboIdioma.setCellFactory(listView -> new LanguageListCell());
13     comboIdioma.setButtonCell(new LanguageListCell());
14
15     // Seleccionar idioma actual
16     comboIdioma.getSelectionModel().select(
17         idiomas.stream()
18                 .filter(idioma ->
    idioma.getCode().equals(currentLocale.getLanguage()))
19                 .findFirst()
20                 .orElse(idiomas.get(0))
21     );
22 }
```

Listing 5.5: Configuración del selector de idiomas

## 5.7. Recursos gráficos

### 5.7.1. Banderas de países

El sistema utiliza imágenes de banderas para representar visualmente cada idioma:

- `espana.png`: Bandera de España
- `england.png`: Bandera de Inglaterra

- `alemania.png`: Bandera de Alemania
- `francia.png`: Bandera de Francia
- `japon.png`: Bandera de Japón
- `portugal.png`: Bandera de Portugal

### 5.7.2. Ubicación de recursos

Las banderas se encuentran en el directorio `src/resources/` y se cargan como recursos de la aplicación.

## 5.8. Consideraciones técnicas

### 5.8.1. Rendimiento

- Los archivos de propiedades se cargan una sola vez al inicializar
- El cambio de idioma es instantáneo
- No hay impacto significativo en el rendimiento

### 5.8.2. Mantenimiento

- Fácil adición de nuevos idiomas
- Estructura clara y organizada
- Separación entre código y textos

### 5.8.3. Extensibilidad

El sistema está diseñado para ser fácilmente extensible:

- Agregar nuevos idiomas solo requiere crear un archivo de propiedades
- Agregar nuevos textos requiere actualizar todos los archivos de propiedades
- La interfaz se adapta automáticamente a nuevos idiomas

## 5.9. Mejores prácticas implementadas

### 5.9.1. Separación de contenido y código

- Todos los textos visibles están en archivos de propiedades
- No hay cadenas hardcodeadas en el código
- Fácil localización por parte de traductores

### 5.9.2. Gestión de recursos

- Uso eficiente de ResourceBundle
- Carga lazy de recursos gráficos
- Gestión adecuada de memoria

### 5.9.3. Experiencia de usuario

- Cambio de idioma inmediato
- Interfaz intuitiva para selección de idioma
- Persistencia de la preferencia de idioma

## 5.10. Pruebas y validación

### 5.10.1. Verificación de traducciones

- Validación de completitud de traducciones
- Verificación de formato de archivos de propiedades
- Pruebas de carga de recursos

### 5.10.2. Pruebas de interfaz

- Verificación de adaptación de textos largos
- Pruebas de cambio dinámico de idioma
- Validación de recursos gráficos

# Capítulo 6

# Compilación y Despliegue

## 6.1. Introducción

Este capítulo describe el proceso completo de compilación, empaquetado y despliegue de la aplicación SimAS 3.0. Se incluyen los scripts de construcción, configuración de dependencias y procedimientos para generar ejecutables nativos.

## 6.2. Herramientas de construcción

### 6.2.1. Java Development Kit (JDK)

**Requisitos:**

- JDK 17 o superior
- Herramienta `jpackage` (incluida desde JDK 14+)
- Variables de entorno configuradas correctamente

**Verificación de instalación:**

```
1 java -version
2 javac -version
3 jpackage --version
```

Listing 6.1: Verificación del JDK

### 6.2.2. JavaFX SDK

**Versión utilizada:** JavaFX 17.0.12

**Ubicación:** `lib/javafx-sdk-17.0.12/`

**Componentes incluidos:**

- `lib/`: Librerías JavaFX
- `bin/`: Herramientas de JavaFX
- `legal/`: Licencias y documentación legal

## 6.3. Scripts de construcción

### 6.3.1. build.sh (Unix/Linux/macOS)

Script principal para sistemas Unix-like:

```
#!/bin/bash

# Script de build para SimAS 3.0
# Crea ejecutables multiplataforma usando jpackage

set -e

# Colores para output
RED='\033[0;31m'
GREEN='\033[0;32m'
YELLOW='\033[1;33m'
NC='\033[0m' # No Color

# Configuración
APP_NAME="SimAS"
APP_VERSION="3.0"
MAIN_CLASS="bienvenida.Bienvenida"
JAVAFX_PATH="./lib/javafx-sdk-17.0.12"
OUTPUT_DIR="./dist"
BUILD_DIR="./build"

echo -e "${GREEN}=== SimAS 3.0 Build Script ===${NC}"

# Verificar que Java 17+ esté instalado
if ! command -v java &> /dev/null; then
    echo -e "${RED}Error: Java no está instalado${NC}"
    exit 1
fi

JAVA_VERSION=$(java -version 2>&1 | head -n 1 | cut -d'"' -f2 | cut -d'.'
↪  -f1)
if [ "$JAVA_VERSION" -lt 17 ]; then
    echo -e "${RED}Error: Se requiere Java 17 o superior. Versión actual:
    ↪  $JAVA_VERSION${NC}"
    exit 1
```

```
fi

echo -e "${GREEN}Java version: $(java -version 2>&1 | head -n 1)${NC}"

# Verificar que jpackage esté disponible
if ! command -v jpackage &> /dev/null; then
    echo -e "${RED}Error: jpackage no está disponible. Asegúrate de tener
    ↪  Java 14+ instalado${NC}"
    exit 1
fi

# Crear directorios
mkdir -p "$BUILD_DIR"
mkdir -p "$OUTPUT_DIR"

echo -e "${YELLOW}Compilando aplicación...${NC}"

# Compilar con módulos JavaFX pero sin iText como módulo
javac --module-path "$JAVAFX_PATH/lib" \
      --add-modules javafx.controls,javafx.fxml \
      -cp "$JAVAFX_PATH/lib/itextpdf-5.5.13.3.jar" \
      -d "$BUILD_DIR" \
      $(find src -name "*.java")

echo -e "${GREEN}Compilación completada${NC}"

# Copiar recursos
echo -e "${YELLOW}Copiando recursos...${NC}"
cp -r src/vistas "$BUILD_DIR/"
cp -r src/resources "$BUILD_DIR/"
cp -r src/centroayuda/ayuda.html "$BUILD_DIR/"
cp -r src/centroayuda/SimAS.html "$BUILD_DIR/"
cp -r src/centroayuda/*.pdf "$BUILD_DIR/"
cp -r src/centroayuda/imagenes "$BUILD_DIR/"
cp -r src/utils/*.properties "$BUILD_DIR/"

# Copiar iText JAR al directorio de build
echo -e "${YELLOW}Copiando dependencias...${NC}"
cp "$JAVAFX_PATH/lib/itextpdf-5.5.13.3.jar" "$BUILD_DIR/"

# Crear JAR ejecutable
echo -e "${YELLOW}Creando JAR ejecutable...${NC}"
cd "$BUILD_DIR"

# Crear MANIFEST.MF
cat > MANIFEST.MF << EOF
Manifest-Version: 1.0
Main-Class: $MAIN_CLASS
Class-Path: itextpdf-5.5.13.3.jar
```

```
EOF

# Crear JAR con MANIFEST
jar cfm SimAS.jar MANIFEST.MF *
cd - > /dev/null

# Detectar el sistema operativo
OS=$(uname -s)
case "$OS" in
    Darwin*)    PLATFORM="mac";;
    Linux*)     PLATFORM="linux";;
    MINGW*|CYGWIN*|MSYS*) PLATFORM="windows";;
    *)          echo -e "${RED}Sistema operativo no soportado: $OS${NC}"; exit 1;;
esac

echo -e "${YELLOW}Creando ejecutable para $PLATFORM...${NC}"

# Crear el ejecutable usando jpackage
jpackage \
    --input "$BUILD_DIR" \
    --name "$APP_NAME" \
    --main-jar SimAS.jar \
    --main-class "$MAIN_CLASS" \
    --module-path "$JAVAFX_PATH/lib" \
    --add-modules javafx.controls,javafx.fxml \
    --type app-image \
    --dest "$OUTPUT_DIR" \
    --app-version "$APP_VERSION" \
    --vendor "Universidad de Córdoba" \
    --description "Simulador de Autómatas y Sintaxis 3.0" \
    --icon "src/resources/logo.png" \
    --java-options "--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED" \
    --java-options "--add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED" \
    --java-options "--add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED" \
    --java-options "-cp $JAVAFX_PATH/lib/itextpdf-5.5.13.3.jar"

echo -e "${GREEN}¡Ejecutable creado exitosamente!${NC}"
echo -e "${GREEN}Ubicación: $OUTPUT_DIR/$APP_NAME${NC}"

# Crear instalador si es posible
echo -e "${YELLOW}Creando instalador...${NC}"

case "$PLATFORM" in
            "mac")
            jpackage \
                --input "$BUILD_DIR" \
```

```
128                 --name "$APP_NAME" \
129                 --main-jar SimAS.jar \
130                 --main-class "$MAIN_CLASS" \
131                 --module-path "$JAVAFX_PATH/lib" \
132                 --add-modules javafx.controls,javafx.fxml \
133                 --type dmg \
134                 --dest "$OUTPUT_DIR" \
135                 --app-version "$APP_VERSION" \
136                 --vendor "Universidad de Córdoba" \
137                 --description "Simulador de Autómatas y Sintaxis 3.0" \
138                 --icon "src/resources/logo.png" \
139                 --java-options
                    ↪ "--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED"
                    ↪ \
140                 --java-options
                    ↪ "--add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED"
                    ↪ \
141                 --java-options
                    ↪ "--add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED" \
142                 --java-options "-cp
                    ↪ $JAVAFX_PATH/lib/itextpdf-5.5.13.3.jar"
143         ;;
144     "linux")
145         jpackage \
146             --input "$BUILD_DIR" \
147             --name "$APP_NAME" \
148             --main-jar SimAS.jar \
149             --main-class "$MAIN_CLASS" \
150             --module-path "$JAVAFX_PATH/lib" \
151             --add-modules javafx.controls,javafx.fxml \
152             --type deb \
153             --dest "$OUTPUT_DIR" \
154             --app-version "$APP_VERSION" \
155             --vendor "Universidad de Córdoba" \
156             --description "Simulador de Autómatas y Sintaxis 3.0" \
157             --icon "src/resources/logo.png" \
158             --java-options
                ↪ "--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED" \
159             --java-options
                ↪ "--add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED"
                ↪ \
160             --java-options
                ↪ "--add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED" \
161             --java-options "-cp $JAVAFX_PATH/lib/itextpdf-5.5.13.3.jar"
162         ;;
163     "windows")
164         jpackage \
165             --input "$BUILD_DIR" \
166             --name "$APP_NAME" \
```

```
167             --main-jar SimAS.jar \
168             --main-class "$MAIN_CLASS" \
169             --module-path "$JAVAFX_PATH/lib" \
170             --add-modules javafx.controls,javafx.fxml \
171             --type exe \
172             --dest "$OUTPUT_DIR" \
173             --app-version "$APP_VERSION" \
174             --vendor "Universidad de Córdoba" \
175             --description "Simulador de Autómatas y Sintaxis 3.0" \
176             --icon "src/resources/logo.png" \
177             --java-options
                ↪  "--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED" \
178             --java-options
                ↪  "--add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED"
                ↪  \
179             --java-options
                ↪  "--add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED" \
180             --java-options "-cp $JAVAFX_PATH/lib/itextpdf-5.5.13.3.jar"
181         ;;
182 esac
183
184 echo -e "${GREEN}¡Instalador creado exitosamente!${NC}"
185 echo -e "${GREEN}=== Build completado ===${NC}"
186 echo -e "${YELLOW}Archivos generados en: $OUTPUT_DIR${NC}"
```

**Funcionalidades:**

- **Líneas 1-10**: Configuración de variables y verificación de dependencias
- **Líneas 12-15**: Creación de directorios necesarios
- **Líneas 17-25**: Compilación del código fuente Java
- **Líneas 27-35**: Copia de recursos y dependencias
- **Líneas 37-45**: Creación del archivo JAR ejecutable
- **Líneas 47-50**: Generación del manifest con información de la aplicación

### 6.3.2. build.bat (Windows)

Script equivalente para sistemas Windows:

```
1 @echo off
2 setlocal enabledelayedexpansion
3
4 REM Script de build para SimAS 3.0 en Windows
5 REM Crea ejecutables usando jpackage
```

```
6
7   REM Configuración
8   set APP_NAME=SimAS
9   set APP_VERSION=3.0
10  set MAIN_CLASS=bienvenida.Bienvenida
11  set JAVAFX_PATH=.\lib\javafx-sdk-17.0.12
12  set OUTPUT_DIR=.\dist
13  set BUILD_DIR=.\build
14
15  echo === SimAS 3.0 Build Script ===
16
17  REM Verificar que Java esté instalado
18  java -version >nul 2>&1
19  if errorlevel 1 (
20      echo Error: Java no está instalado
21      exit /b 1
22  )
23
24  REM Verificar versión de Java
25  for /f "tokens=3" %%g in ('java -version 2^>^&1 ^| findstr /i "version"')
    ↪  do (
26      set JAVA_VERSION=%%g
27      set JAVA_VERSION=!JAVA_VERSION:"=!
28      for /f "tokens=1 delims=." %%v in ("!JAVA_VERSION!") do set
        ↪  JAVA_MAJOR=%%v
29  )
30
31  if !JAVA_MAJOR! LSS 17 (
32      echo Error: Se requiere Java 17 o superior. Versión actual:
        ↪  !JAVA_VERSION!
33      exit /b 1
34  )
35
36  echo Java version: !JAVA_VERSION!
37
38  REM Verificar que jpackage esté disponible
39  jpackage --help >nul 2>&1
40  if errorlevel 1 (
41      echo Error: jpackage no está disponible. Asegúrate de tener Java 14+
        ↪  instalado
42      exit /b 1
43  )
44
45  REM Crear directorios
46  if not exist "!BUILD_DIR!" mkdir "!BUILD_DIR!"
47  if not exist "!OUTPUT_DIR!" mkdir "!OUTPUT_DIR!"
48
49  echo Compilando aplicación...
50
```

```
REM Compilar sin módulos (enfoque tradicional)
javac -cp "!JAVAFX_PATH!\lib\*;!JAVAFX_PATH!\lib\itextpdf-5.5.13.3.jar" ^
      -d "!BUILD_DIR!" ^
      src\bienvenida\*.java ^
      src\editor\*.java ^
      src\simulador\*.java ^
      src\utils\*.java ^
      src\gramatica\*.java ^
      src\centroayuda\*.java

if errorlevel 1 (
    echo Error en la compilación
    exit /b 1
)

echo Compilación completada

REM Copiar recursos
echo Copiando recursos...
xcopy /E /I /Y src\vistas "!BUILD_DIR!\vistas" >nul
xcopy /E /I /Y src\resources "!BUILD_DIR!\resources" >nul
copy /Y src\centroayuda\ayuda.html "!BUILD_DIR!\" >nul
copy /Y src\centroayuda\SimAS.html "!BUILD_DIR!\" >nul
copy /Y src\centroayuda\*.pdf "!BUILD_DIR!\" >nul
xcopy /E /I /Y src\centroayuda\imagenes "!BUILD_DIR!\imagenes" >nul
copy /Y src\utils\*.properties "!BUILD_DIR!\" >nul

REM Copiar iText JAR al directorio de build
echo Copiando dependencias...
copy "!JAVAFX_PATH!\lib\itextpdf-5.5.13.3.jar" "!BUILD_DIR!\" >nul

REM Crear JAR ejecutable
echo Creando JAR ejecutable...
cd "!BUILD_DIR!"

REM Crear MANIFEST.MF
echo Manifest-Version: 1.0 > MANIFEST.MF
echo Main-Class: !MAIN_CLASS! >> MANIFEST.MF
echo Class-Path: itextpdf-5.5.13.3.jar >> MANIFEST.MF

REM Crear JAR con MANIFEST
jar cfm SimAS.jar MANIFEST.MF *
cd /d "%~dp0"

echo Creando ejecutable para Windows...

REM Crear el ejecutable usando jpackage (sin módulos)
jpackage ^
    --input "!BUILD_DIR!" ^
```

```
100     --name "!APP_NAME!" ^
101     --main-jar SimAS.jar ^
102     --main-class "!MAIN_CLASS!" ^
103     --type app-image ^
104     --dest "!OUTPUT_DIR!" ^
105     --app-version "!APP_VERSION!" ^
106     --vendor "Universidad de Córdoba" ^
107     --description "Simulador de Autómatas y Sintaxis 3.0" ^
108     --icon "src\resources\logo.png" ^
109     --java-options "--module-path !JAVAFX_PATH!\lib" ^
110     --java-options "--add-modules javafx.controls,javafx.fxml" ^
111     --java-options "--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED"
        ↪ ^
112     --java-options
        ↪ "--add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED" ^
113     --java-options "--add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED" ^
114     --java-options "-cp !JAVAFX_PATH!\lib\itextpdf-5.5.13.3.jar"
115
116 if errorlevel 1 (
117     echo Error al crear el ejecutable
118     exit /b 1
119 )
120
121 echo ¡Ejecutable creado exitosamente!
122 echo Ubicación: !OUTPUT_DIR!\!APP_NAME!
123
124 REM Crear instalador EXE
125 echo Creando instalador...
126 jpackage ^
127     --input "!BUILD_DIR!" ^
128     --name "!APP_NAME!" ^
129     --main-jar SimAS.jar ^
130     --main-class "!MAIN_CLASS!" ^
131     --type exe ^
132     --dest "!OUTPUT_DIR!" ^
133     --app-version "!APP_VERSION!" ^
134     --vendor "Universidad de Córdoba" ^
135     --description "Simulador de Autómatas y Sintaxis 3.0" ^
136     --icon "src\resources\logo.png" ^
137     --java-options "--module-path !JAVAFX_PATH!\lib" ^
138     --java-options "--add-modules javafx.controls,javafx.fxml" ^
139     --java-options "--add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED"
        ↪ ^
140     --java-options
        ↪ "--add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED" ^
141     --java-options "--add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED" ^
142     --java-options "-cp !JAVAFX_PATH!\lib\itextpdf-5.5.13.3.jar"
143
144 if errorlevel 1 (
```

```
    echo Error al crear el instalador
    exit /b 1
)

echo ¡Instalador creado exitosamente!
echo === Build completado ===
echo Archivos generados en: !OUTPUT_DIR!

pause
```

**Diferencias con build.sh:**

- Sintaxis de comandos de Windows
- Variables de entorno específicas de Windows
- Rutas con separadores de Windows

### 6.3.3. create-standalone-app.sh

Script para crear aplicación independiente en macOS:

```
#!/bin/bash

# Script para crear una aplicación completamente independiente

echo "=== Creando aplicación independiente ==="

APP_NAME="SimAS"
OUTPUT_DIR="./dist-standalone"
BUILD_DIR="./build"

if [ ! -d "$BUILD_DIR" ]; then
    echo "ERROR: No se encontró el directorio de build"
    echo "Ejecuta primero: ./build-with-natives.sh"
    exit 1
fi

# Crear directorio de salida
mkdir -p "$OUTPUT_DIR"

echo "SUCCESS: Directorio de salida creado"

# Crear estructura de la aplicación
APP_PATH="$OUTPUT_DIR/$APP_NAME.app"
mkdir -p "$APP_PATH/Contents/MacOS"
mkdir -p "$APP_PATH/Contents/Java"
mkdir -p "$APP_PATH/Contents/Resources"
```

```
27
28 echo "SUCCESS: Estructura de aplicación creada"
29
30 # Copiar JAR principal
31 cp "$BUILD_DIR/SimAS.jar" "$APP_PATH/Contents/Java/"
32
33 # Copiar todas las librerías de JavaFX
34 echo "Copiando librerías de JavaFX..."
35 cp lib/javafx-sdk-17.0.12/lib/*.jar "$APP_PATH/Contents/Java/"
36 cp lib/javafx-sdk-17.0.12/lib/*.dylib "$APP_PATH/Contents/Java/"
37
38 # Crear script de lanzamiento
39 echo "Creando script de lanzamiento..."
40 cat > "$APP_PATH/Contents/MacOS/$APP_NAME" << 'EOF'
41 #!/bin/bash
42
43 # Script de lanzamiento para SimAS
44 APP_DIR="$(dirname "$0")/.."
45 JAR_PATH="$APP_DIR/Java/SimAS.jar"
46 JAVA_LIB_PATH="$APP_DIR/Java"
47
48 # Construir classpath
49 CLASSPATH="$JAR_PATH"
50 for jar in "$JAVA_LIB_PATH"/*.jar; do
51     if [ -f "$jar" ]; then
52         CLASSPATH="$CLASSPATH:$jar"
53     fi
54 done
55
56 # Ejecutar con JavaFX modules
57 exec java \
58     --module-path "$JAVA_LIB_PATH" \
59     --add-modules javafx.controls,javafx.fxml \
60     --add-opens=javafx.graphics/javafx.scene=ALL-UNNAMED \
61     --add-opens=javafx.controls/javafx.scene.control=ALL-UNNAMED \
62     --add-opens=javafx.fxml/javafx.fxml=ALL-UNNAMED \
63     -cp "$CLASSPATH" \
64     -Djava.library.path="$JAVA_LIB_PATH" \
65     bienvenida.Bienvenida
66 EOF
67
68 # Hacer el script ejecutable
69 chmod +x "$APP_PATH/Contents/MacOS/$APP_NAME"
70
71 # Crear Info.plist
72 echo "Creando Info.plist..."
73 cat > "$APP_PATH/Contents/Info.plist" << EOF
74 <?xml version="1.0" ?>
```

```
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
 ↪  "https://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
 <dict>
  <key>LSMinimumSystemVersion</key>
  <string>10.11</string>
  <key>CFBundleDevelopmentRegion</key>
  <string>English</string>
  <key>CFBundleAllowMixedLocalizations</key>
  <true/>
  <key>CFBundleExecutable</key>
  <string>$APP_NAME</string>
  <key>CFBundleIconFile</key>
  <string>$APP_NAME.icns</string>
  <key>CFBundleIdentifier</key>
  <string>com.simas.app</string>
  <key>CFBundleInfoDictionaryVersion</key>
  <string>6.0</string>
  <key>CFBundleName</key>
  <string>$APP_NAME</string>
  <key>CFBundlePackageType</key>
  <string>APPL</string>
  <key>CFBundleShortVersionString</key>
  <string>3.0</string>
  <key>CFBundleSignature</key>
  <string>????</string>
  <key>LSApplicationCategoryType</key>
  <string>public.app-category.utilities</string>
  <key>CFBundleVersion</key>
  <string>3.0</string>
  <key>NSHumanReadableCopyright</key>
  <string>Copyright (C) 2025 Universidad de Córdoba</string>
  <key>NSHighResolutionCapable</key>
  <string>true</string>
 </dict>
</plist>
EOF

echo "SUCCESS: Aplicación independiente creada en $APP_PATH"

# Probar la aplicación
echo "Probando la aplicación..."
"$APP_PATH/Contents/MacOS/$APP_NAME" &
APP_PID=$!

sleep 5

if kill -0 $APP_PID 2>/dev/null; then
    echo "SUCCESS: ¡La aplicación se está ejecutando correctamente!"
```

```
123     kill $APP_PID
124 else
125     echo "ERROR: La aplicación no se pudo ejecutar"
126 fi
127
128 echo "=== Aplicación independiente creada ==="
129 echo "Ubicación: $APP_PATH"
130 echo "Puedes hacer doble clic en $APP_PATH para ejecutar la aplicación"
```

**Características principales:**

- Utiliza `jpackage` para crear aplicación nativa
- Incluye todas las dependencias JavaFX
- Genera archivo `.app` ejecutable
- Configuración específica para macOS

## 6.4. Proceso de compilación

### 6.4.1. Paso 1: Preparación del entorno

1. Verificar instalación de JDK 17+
2. Configurar variables de entorno:

```
1     export JAVA_HOME=/path/to/jdk17
2     export PATH=$JAVA_HOME/bin:$PATH
3
```

3. Verificar acceso a JavaFX SDK
4. Dar permisos de ejecución a scripts:

```
1     chmod +x build.sh
2     chmod +x create-standalone-app.sh
3
```

### 6.4.2. Paso 2: Compilación del código fuente

```
1 # Para sistemas Unix/Linux/macOS
2 ./build.sh
3
4 # Para sistemas Windows
5 build.bat
```

Listing 6.2: Compilación del proyecto

**Proceso interno:**

1. Creación de directorio `build/`
2. Compilación de archivos Java a bytecode
3. Copia de recursos (imágenes, FXML, propiedades)
4. Copia de librerías JavaFX
5. Generación del archivo JAR

### 6.4.3. Paso 3: Estructura de archivos generada

Después de la compilación, se genera la siguiente estructura:

```
build/
|-- SimAS.jar                    # Archivo JAR principal
|-- lib/                         # Librerías JavaFX
|   |-- javafx.controls.jar
|   |-- javafx.fxml.jar
|   |-- ...
|-- resources/                   # Recursos de la aplicación
|   |-- logo.png
|   |-- icons/
|   |-- ...
|-- vistas/                      # Archivos FXML
|   |-- MenuPrincipal.fxml
|   |-- Bienvenida.fxml
|   |-- ...
|-- utils/                       # Archivos de propiedades
|   |-- messages_es.properties
|   |-- messages_en.properties
|   |-- ...
|-- MANIFEST.MF                  # Manifest del JAR
```

Listing 6.3: Estructura del directorio build

## 6.5. Creación de ejecutables nativos

### 6.5.1. Usando jpackage

**Comando básico:**

```
jpackage --input build \
         --main-jar SimAS.jar \
         --main-class bienvenida.Bienvenida \
         --name SimAS \
         --app-version 3.0 \
         --vendor "Antonio Llamas García" \
         --description "Simulador de Análisis Sintáctico" \
         --dest dist
```

Listing 6.4: Creación de ejecutable con jpackage

**Parámetros específicos por plataforma:**

#### 6.5.1.1. macOS

```
jpackage --input build \
         --main-jar SimAS.jar \
         --main-class bienvenida.Bienvenida \
         --name SimAS \
         --type dmg \
         --app-version 3.0 \
         --mac-package-identifier com.simas.app \
         --mac-package-name "SimAS 3.0" \
         --dest dist
```

Listing 6.5: Configuración para macOS

#### 6.5.1.2. Windows

```
jpackage --input build \
         --main-jar SimAS.jar \
         --main-class bienvenida.Bienvenida \
         --name SimAS \
         --type msi \
         --app-version 3.0 \
         --win-dir-chooser \
         --win-menu \
         --win-shortcut \
         --dest dist
```

Listing 6.6: Configuración para Windows

#### 6.5.1.3. Linux

```
jpackage --input build \
         --main-jar SimAS.jar \
         --main-class bienvenida.Bienvenida \
         --name SimAS \
         --type deb \
         --app-version 3.0 \
         --linux-shortcut \
         --dest dist
```

Listing 6.7: Configuración para Linux

### 6.5.2. Aplicación independiente para macOS

El script `create-standalone-app.sh` crea una aplicación completamente independiente:

```
./create-standalone-app.sh
```

Listing 6.8: Creación de aplicación independiente

**Resultado:**

- Archivo `SimAS.app` en `dist-standalone/`
- Aplicación completamente autónoma
- No requiere Java instalado en el sistema
- Incluye todas las dependencias necesarias

## 6.6. Configuración del manifest

### 6.6.1. MANIFEST.MF

El archivo manifest contiene metadatos importantes:

```
Manifest-Version: 1.0
Main-Class: bienvenida.Bienvenida
Class-Path: itextpdf-5.5.13.3.jar
```

**Atributos principales:**

- `Main-Class`: Clase principal de la aplicación
- `Class-Path`: Ruta a las librerías JavaFX
- `Implementation-Title`: Nombre de la aplicación
- `Implementation-Version`: Versión de la aplicación
- `Implementation-Vendor`: Desarrollador

## 6.7. Distribución

### 6.7.1. Requisitos para usuarios finales

**Opción 1: JAR ejecutable**

- Java Runtime Environment (JRE) 17+
- JavaFX Runtime (incluido en JRE 11+ o por separado)

**Opción 2: Aplicación nativa**

- No requiere Java instalado
- Ejecutable nativo del sistema operativo
- Mayor tamaño de descarga

### 6.7.2. Instrucciones de instalación

#### 6.7.2.1. Para desarrolladores

1. Clonar el repositorio
2. Ejecutar `./build.sh` o `build.bat`
3. Ejecutar `java -jar build/SimAS.jar`

#### 6.7.2.2. Para usuarios finales

1. Descargar la aplicación desde el repositorio
2. Ejecutar `./dist-standalone/SimAS.app` (macOS)
3. O ejecutar `java -jar SimAS.jar` (requiere Java)

## 6.8. Solución de problemas

### 6.8.1. Errores comunes

#### 6.8.1.1. Error: `JavaFX runtime components are missing`

**Causa:** JavaFX no está disponible en el classpath **Solución:** Asegurar que las librerías JavaFX estén en `lib/`

#### 6.8.1.2. Error: `Main class not found`

**Causa:** La clase principal no está especificada correctamente **Solución:** Verificar el manifest y la estructura de paquetes

#### 6.8.1.3. Error: `Permission denied`

**Causa:** Scripts sin permisos de ejecución **Solución:** `chmod +x build.sh create-standalone-ap`

### 6.8.2. Verificación de la instalación

```
# Verificar que la aplicación se ejecuta
java -jar build/SimAS.jar

# Verificar estructura de archivos
ls -la build/

# Verificar manifest
unzip -p build/SimAS.jar META-INF/MANIFEST.MF
```

Listing 6.9: Verificación de la instalación

## 6.9. Optimizaciones de rendimiento

### 6.9.1. Configuración de JVM

**Parámetros recomendados:**

```
java -Xmx512m -Xms256m -jar SimAS.jar
```

Listing 6.10: Parámetros JVM optimizados

### 6.9.2. Reducción del tamaño

- Uso de ProGuard para ofuscación y minificación
- Eliminación de dependencias no utilizadas
- Compresión de recursos

## 6.10. Automatización con CI/CD

### 6.10.1. GitHub Actions

Ejemplo de workflow para compilación automática:

```
name: Build SimAS
on: [push, pull_request]

jobs:
  build:
    runs-on: ${{ matrix.os }}
    strategy:
      matrix:
        os: [ubuntu-latest, windows-latest, macos-latest]

    steps:
    - uses: actions/checkout@v2
```

```
13
14      - name: Set up JDK 17
15        uses: actions/setup-java@v2
16        with:
17          java-version: '17'
18          distribution: 'adopt'
19
20      - name: Build application
21        run: |
22          chmod +x build.sh
23          ./build.sh
24
25      - name: Create native executable
26        run: |
27          chmod +x create-standalone-app.sh
28          ./create-standalone-app.sh
```

Listing 6.11: Workflow de GitHub Actions

## 6.11. Consideraciones de seguridad

### 6.11.1. Firma de código

Para distribuir la aplicación de forma segura:

- Firmar el JAR con certificado digital
- Firmar ejecutables nativos
- Verificar integridad de archivos

### 6.11.2. Verificación de dependencias

- Verificar checksums de librerías JavaFX
- Actualizar dependencias regularmente
- Usar versiones estables y soportadas

# Bibliografía


[1] A. V. Aho, M. S. Lam, R. Sethi y J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. 2nd. Addison-Wesley, 2006. ISBN: 978-0321486813.

[2] J. E. Hopcroft, R. Motwani y J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. 3rd. Addison-Wesley, 2006. ISBN: 978-0321455369.

[3] OpenJFX Project. *JavaFX - Open Source Java Client Platform*. [En Línea. Última consulta: 15-09-2024]. URL: https://openjfx.io/.

[4] Oracle Corporation. *Java Platform, Standard Edition*. [En Línea. Última consulta: 15-09-2024]. URL: https://www.oracle.com/java/.

[5] Oracle Corporation. *jpackage - Java Platform, Standard Edition Tools Reference*. [En Línea. Última consulta: 15-09-2024]. URL: https://docs.oracle.com/en/java/javase/17/docs/specs/man/jpackage.html.