ESCUELA POLITÉCNICA NACIONAL

FACULTAD DE INGENIERÍA DE SISTEMAS

INGENIERÍA DE SOFTWARE

Asignatura:
Construcción y evolución del software

Integrantes:
Anguaya Otavalo Angel Manuel
Arrobo Pinto Julio Cesar
Cofre Santo Juan Fernando
Parra Sarasti Jhordy Oliver

Grupo: 1

Contenido

1 Introducción

1.1 Propósito del documento

El propósito de este documento es describir de manera detallada el entorno de desarrollo, las herramientas, dependencias, configuraciones y procedimientos necesarios para la correcta instalación, ejecución y mantenimiento de la aplicación de traducción de texto a Braille. Además, proporciona lineamientos para la organización del proyecto bajo el patrón arquitectónico MVC, el uso de JavaFX, el flujo de trabajo con GitHub y la generación de documentación mediante Javadoc. Su objetivo es servir como una guía técnica clara y completa para desarrolladores actuales y futuros que participen en la construcción o evolución del sistema.

2 Herramientas de desarrollo

- Lenguaje de programación: Java (OpenJDK 25.0.1)
- Entorno de desarrollo integrado (IDE): Visual Studio Code
- Sistema de control de versiones: GitHub
- Herramienta de construcción y gestión de dependencias: Maven

3 Librerías y dependencias

Javafx-controls: Biblioteca que contiene la mayor parte de los componentes de la interfaz de usuario (UI). Incluye todos los widgets o controles estándar que un usuario puede ver e interactuar, como botones, etiquetas (labels), campos de texto (TextAreas), y layoutsm (contenedores).

Javafx-fxml: biblioteca que implementa el lenguaje FXML, el cual es un lenguaje basado en XML que se utiliza para definir la estructura de la interfaz de usuario separada de la lógica de Java.

Maven-compiler-plugin: Plugin estándar de Maven para compilar el código fuente de Java. Su función principal es tomar los archivos .java del proyecto (Modelo, Vista, Controlador) y traducirlos a archivos de bytecode

Javafx-maven-plugin: Plugin especializado para proyectos JavaFX. Su rol principal es facilitar la ejecución del proyecto en el entorno de desarrollo y, crucialmente, el empaquetado de la aplicación

4 Configuración del proyecto

4.1 Estructura de carpetas

Estructura principal del código siguiendo el patrón MVC.

- src\main\java\com\traductor\controller: contiene la lógica de manejo de eventos: TraductorController.
- src\main\java\com\traductor\model: contiene la lógica del negocio: TraductorBraille, DiccionarioBraille, SimboloBraille, ITraductor.
- src\main\java\com\traductor\view: Contiene la interfaz gráfica: InterfazTraductor.

4.2 Configuración de JavaFX

El proyecto consiste en el desarrollo de una aplicación capaz de convertir textos en español a su representación en Braille. Para su implementación se utilizará el patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual permitirá separar la lógica de negocio de la interfaz gráfica, facilitando la mantenibilidad, escalabilidad y evolución del sistema. La interfaz de usuario será desarrollada utilizando JavaFX, aprovechando sus capacidades para construir aplicaciones de escritorio modernas y estructuradas.

4.3 Configuración de la herramienta de construcción

```
<properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>17</maven.compiler.source>
    <maven.compiler.target>17</maven.compiler.target>
    <javafx.version>21.0.1</javafx.version>
</properties>

<dependencies>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-controls</artifactId>
        <version>${javafx.version}</version>
    </dependency>
    <dependency>
        <groupId>org.openjfx</groupId>
        <artifactId>javafx-fxml</artifactId>
        <version>${javafx.version}</version>
    </dependency>
</dependencies>
```

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.11.0</version>
            <configuration>
                <source>17</source>
                <target>17</target>
            </configuration>
        </plugin>
        <plugin>
            <groupId>org.openjfx</groupId>
            <artifactId>javafx-maven-plugin</artifactId>
            <version>0.0.8</version>
            <configuration>
                <mainClass>com.traductor.view.InterfazTraductor</mainClass>
            </configuration>
        </plugin>
    </plugins>
</build>
```

5 Flujo de ramas de GitHub

- **feature/traductor**
  Propósito: Desarrollo e implementación de la lógica del Modelo del traductor: TraductorBraille, DiccionarioBraille, SimboloBraille y ITraductor.
  Regla: Debe contener las reglas de negocio para la transcripción de español a braille, no debe incluir código de la interfaz.
- **feature/interfaz**
  Propósito: Desarrollo de la Vista y la lógica de los controladores de la interfaz JavaFX.
  Regla: Implementar los componentes que permiten al usuario ingresar texto y ver la traducción en Braille, y también debe incluir las clases InterfazTraductor y TraductorController.
- **Main**
  Propósito: Contiene el código estable de la última versión de producción.
  Regla: El merge a Main solo se permite desde la rama develop, no se permite el desarrollo directo.
- **documentación**
  Propósito: Manejar actualizaciones a la documentación del proyecto.
  Regla: Usada para subir documentación formal sin afectar directamente el código fuente de la aplicación.

6 Procedimiento de instalación

Secuencia que se debe que seguir para clonar el repositorio y ejecutar la aplicación por primera vez.

1. **Instalar prerrequisitos**
   Instalar JDK de java versión 25.0.1
   Instalar Git
   Instalar el IDE de Visual Studio Code
2. **Clonar repositorio**
   Ejecutar el siguiente comando de git desde la terminal en donde se desea almacenar el repositorio.
   git clone –repositorio
3. **Configurar el proyecto en el IDE**
   Abrir el proyecto en el IDE, asegurando que el IDE reconozca el archivo de construcción Maven
4. **Descargar dependencias**
   Ejecutar el comando de construcción para descargar todas las librerías necesarias: mvn clean install
5. **Ejecutar la aplicación**
   Ejecutar usando el siguiente comando de construcción:
   Primero: mvn clean compile
   Segundo: mvn javafx:run

7 Javadoc

Se utilizo la herramienta de Javadoc para generar documentación legible directamente desde el código fuente, la cual se debe aplicar a todas las clases, interfaces, métodos públicos y atributos importantes.

7.1 Etiquetas claves a documentar:

@param: Se usa para describir un parámetro de un método o constructor. Debe incluir el nombre del parámetro y una descripción de su uso.

@return: Describe el valor de retorno de un método.

@throws: Indica las excepciones que puede lanzar un método y proporciona una descripción de cada una.

@see: Proporciona enlaces a otras clases, métodos o paquetes relacionados.

@since: Se utiliza en JavaDoc para indicar desde qué versión de una clase, método o campo está disponible en la API.