JabRef - Initial Setup & Challenges Faced

Devang Shetty (a1894311) & Harish Babu (a1908761)

Introduction

In this report, I'll describe how we set up JabRef, an open-source reference management software, on our local machines. The journey involved some challenges, particularly with Java Development Kit (JDK) conflicts and other dependency issues, but we managed to overcome them. Here's a step-by-step account of our experience.

Prerequisites

Before diving into the setup, we ensured we had the following tools:

1. **Java Development Kit (JDK) 21**: JabRef requires JDK 21.
2. **Git**: To clone the JabRef repository from GitHub.
3. **Gradle**: Helpful for building the project. The project includes a Gradle Wrapper, so we didn't necessarily need to install Gradle separately.

Learning Git

Before starting with the JabRef setup, we had to get comfortable with Git. Here's how we learned it:

1. **Understanding Basics**: We learned about repositories, commits, branches, and merges, practicing these concepts by creating our own repositories and performing various operations.
2. **Essential Commands**: We familiarize ourselves with commands such as `git clone`, `git status`, `git add`, `git commit`, `git push`, and `git pull`. This helped us track changes, stage them, commit them, and sync with remote repositories.
3. **Collaborative Work**: We practiced collaborating on shared repositories, handling merge conflicts, and submitting pull requests on GitHub. This practice was invaluable for working on JabRef.

Learning Gradle

To efficiently build and manage the JabRef project, we had to familiarize ourselves with Gradle. Here are the main points we focused on:

1. **Understanding Gradle Basics**: We started by learning the fundamental concepts of Gradle, such as tasks, build scripts, and the project structure. This helped us understand how Gradle automates the building, testing, and deployment of projects.

2. **Using the Gradle Wrapper**: We learned to use the Gradle Wrapper (`./gradlew`), which is included in the JabRef project. This ensures that we use a consistent Gradle version, avoiding compatibility issues. We practiced commands like `./gradlew build` to compile the project, `./gradlew test` to run tests, and `./gradlew clean` to remove build artifacts.
3. **Managing Dependencies**: We explored how Gradle handles dependencies through the `build.gradle` file. This involved adding, updating, and resolving dependencies from repositories like Maven Central. We also learned to troubleshoot dependency conflicts and ensure that all required libraries are correctly included in the build.

Challenges Faced

Throughout the setup process, we encountered several challenges and learned how to address them effectively:

1. **JDK Conflicts**: We encountered conflicts due to different versions of the JDK installed across our team members' machines. This caused inconsistencies and build failures during the setup process.
2. **Dependency Issues**: Some team members faced dependency issues due to outdated local repositories, which resulted in missing or incompatible libraries.
3. **IDE Integration**: Importing the project into IntelliJ IDEA and Eclipse required additional configuration steps. Setting the correct JDK path and updating necessary plugins were crucial.
4. **Incomplete Builds**: One team member experienced incomplete builds caused by earlier repository issues. These issues left the build in a partially completed state.
5. **Interactions for Documentations:** Finding the relevant Jabref documentations for the issues that we were facing was not always easy.

Resolve Strategy

1. **JDK Conflicts:** Standardize on a specific JDK version for the entire team. Ensure everyone installs the same version and updates their environment variables accordingly.

```
# For Mac

export JAVA_HOME=/path/to/jdk

export PATH=$JAVA_HOME/bin:$PATH
```

```
# For Windows

set JAVA_HOME=C:\path\to\jdk

set PATH=%JAVA_HOME%\bin;%PATH%
```

2. **Dependency Issues:** Clear local repository caches and configure build tools to fetch the latest versions from remote repositories. Use a dependency management tool like Maven or Gradle.

```
# For Gradle

gradle clean build --refresh-dependencies
```

3. **IDE Integration:** Provide a detailed setup guide for configuring the IDEs, including setting the correct JDK path and updating necessary plugins. Create project-specific configuration files.

```
<!-- IntelliJ IDEA: .idea/misc.xml -->

<component name="ProjectRootManager" version="2"
languageLevel="JDK_1_8" default="false"
project-jdk-name="1.8" project-jdk-type="JavaSDK">

  ...

</component>
```

4. **Incomplete Builds:** Perform a clean build after resolving repository issues to ensure all dependencies are correctly installed and the build completes successfully.

```
# For Gradle

gradle clean build
```

5. **Interactions for Documentations:** Finding the relevant JabRef documentation for the issues we faced was not always easy. To resolve this, we turned to the JabRef Gitter community for help. The community managers were the only ones who responded consistently, but we often had to wait for solutions since they were managing multiple requests. We made sure to ask detailed questions and follow up politely if we didn't get a quick response.