Case Western Reserve University

Final Project Report

JEdu

Shiqing Gao

Yihe Guo

Melody Li

Yue Shu

Jiaqi Yang

Roxanne Yang

December 5, 2019

Date	Version Number	Note
December 5	V 1.0	Initial Release Documentation
December 5	V 1.1	Team Reviewed and Signoff

1. Background and Objectives

1.1 Background

Major Java IDEs, like most of the other IDEs, are developed for professional programmers and have specific requirements for project structures. When using such IDEs, programmers need to meet these project structure requirements or they will encounter compiling and runtime errors. Although certain project structures allow the programmers to manage their projects conveniently, they are not instinctive for most students taking intro-level programming classes.

DrJava is a Java programming environment favored by professors teaching introductory Java courses. Over many semesters, thousands of students picked up DrJava as their first Java programming environment. They benefited from its powerful interactive functionalities such as the interaction pane, the auto JUnit testing component, the integrated JDB debugger, as well as the JavaDoc generator. The need for this project arises from the fact that while JDK continues to release new updates, DrJava no longer supports the latest version of JDK and will soon become outdated due to the lack of maintenance. As a solution, we have found an alternative environment currently under proper maintenance, jEdit.

jEdit is a Java text editor open to external plugin development. It is also friendly to beginner programmers since it provides a simple and self-explanatory user interface. The major obstacle for its plugin development is that presently all necessary dependencies, such as the compiler and debugger plugins for Java to function in jEdit are corrupted. We aim to explore the feasibility of replacing DrJava with jEdit by recreating the essential features of DrJava.

1.2 Objectives, Success Criteria, Measures and Results

No	Component	Success Criteria	Measure	Priority	Results
1	JUnit Plugin	The plugin is developed as a functional testing plugin with a consistent GUI and does not require the presence of a project.	The plugin should allow the users to create test files and input their test cases formatted in the JUnit standard.	1	Completed
			The plugin features two displays. One will show each test with a fail/pass indicator, the other one will give the important error messages for the tests that fail.	1	Completed
			The plugin should be able to test all of the other Java files that are saved under the same directory as the test file.	1	Completed
2	Interaction Pane	The interaction pane component should	The interaction pane should take basic Java expressions and statements as	1	Completed

		feature the same behavior as that of DrJava's interaction pane.	inputs, then interpret and evaluate; the result should be displayed.		
			The interaction pane should provide compiler and runtime error messages based on user input and status of the code.	1	Completed
			The interaction pane should support basic operations such as input history trace back, user Java class file autoloading, and output history log.	2	Completed
3	JDB Plugin	The JDB plugin should support some basic functionalities of a standard Java debugger.	The JDB plugin should no longer contains any deprecated jEdit library usage.	1	Completed
			Developers should now be able to build the project with the config files we provide and continue with future development.	2	Completed

1.3 Features

1.3.1 Major Features

These are the features that we think are the most important and can likely be completed within a semester.

FE-1	JUnit plugin: allows running customized JUnit tests and displaying easy-to-understand test outputs (pass/fail + error reports) without the need of a project structure.
FE-2	Interaction pane plugin: takes basic Java expressions, such as (1+1), class instantiations, method calls, and package imports as inputs. The expressions will then be interpreted and evaluated, and the result will be displayed.

1.3.2 Stretch Features

These are the less important features that we would like to complete if we have time left.

SE-1	JDB plugin: provide standard debugging operations: setting breakpoints, step into, step out, step over, resume, restart, etc. The plugin should also allow the users to investigate the values of parameters before and after the line is executed and trace the method calls.

2. Deliverable Report

2.1 Interaction Pane Plugin

We have been able to achieve our goal for the interaction pane. We successfully implemented a Read - Eval - Print Loop based on the JShell library. The interaction pane plugin is fully functional, it allows the user input standard java commands, compile and evaluate, and finally print out the update value of the variable if there is one. There is an inbuilt functionality to display meaningful values of a variable. For example, the human-readable value of a list is displayed instead of the object address. The interaction pane also displays a full set of compiler errors in case the input is invalid.

The GUI of interaction pane is similar to shell, which also scrolling back to the historical input with up and down buttons. The major challenge we faced was the usage of the JShell library. Since the JShell library is not commonly used for any other applications except for java editors, there was not a lot of guidance on how each method in the library is used. We had to spend an enormous amount of time on reading Oracle documentation and digging into other open source editors' code base for examples. Another challenge we faced was building the plugin according to jEdit's requirements. Since the jEdit plugin development manual was written in the early two thousands, and when the plugin is not accepted, it does not show us the reason why it does not work. We had to solve this issue by looking at the source code of many other jEdit plugin and compare their confirguation files.

The source code of the interaction pane plugin could be found in `./InteractionPane`. The project is structured as an ant project, so in order to test the plugin should be built to a `jar` file and installed on jEdit by the plugin manager. Once we publish the plugin officially to the jEdit community as an open-source project, users should be able to get download the plugin from http://plugins.jedit.org/.

Figure 1-3 are some screenshots of the usage of our interaction pane plugin.

Figure 1. Declaration of for loop and primitive variables; Figure 2. Package import and invoke of Java libraries; Figure 3. Some invalid input

2.2 Progress of JUnit Plugin

Since the writing of Progress Report 2, we were able to achieve all the goals for the JUnit plugin: removing the project structure dependency and making the UI more friendly.

Like we showed in the demo, the old JUnit plugin required a project structure. While that is standard and therefore acceptable for more advanced development, it doesn't fit the needs of our target users: students learning Java for the first time. We identified the part in the workflow that interacts with the Project Viewer plugin, removed the interaction, and replaced the logic with a simple "select folder" operation, which we also created a UI method for, as can be seen in Figure 1.

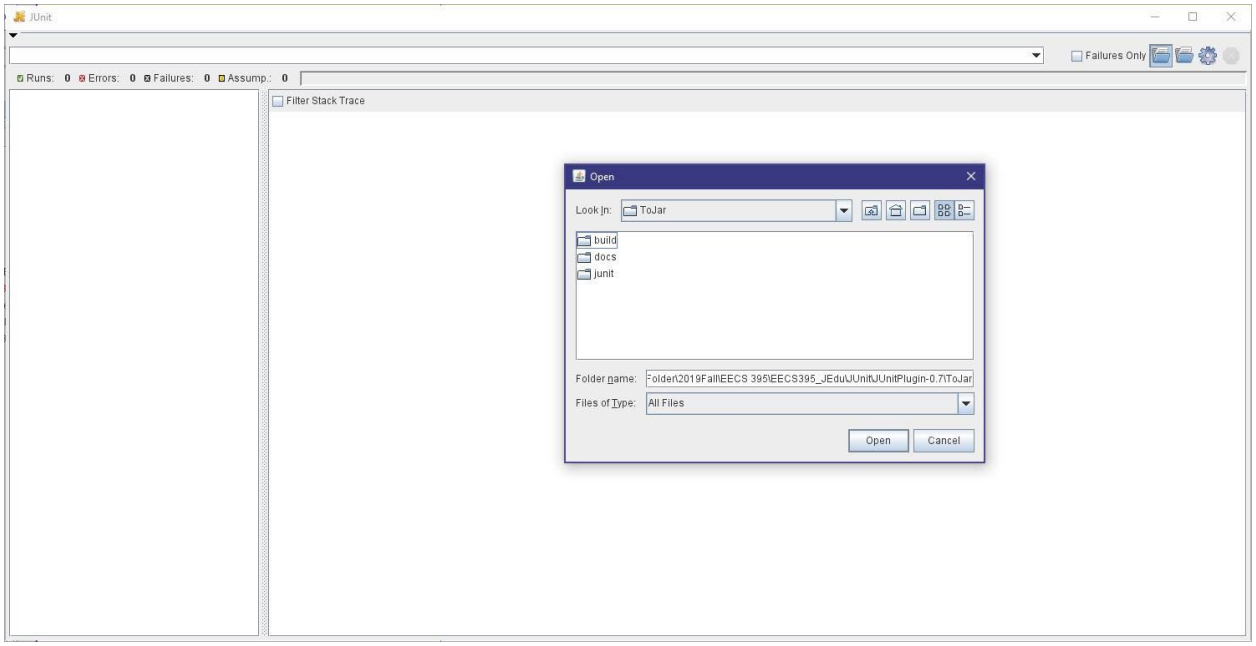

Figure 1. The UI component for folder selection

Additionally, we also made purely-UI changes to the JUnit plugin. We removed the buttons that provided less important functionalities and reorganized some of the components so that users find them more intuitive, and therefore can use the plugin with more ease.

The biggest difficulty within the development for the JUnit plugin was packaging the plugin. We have summarized a workflow (included in both the source code and Progress Report 2) for future developers. The source code could be found under `./JUnit/JUnitPlugin-0.7`. As mentioned in the previous design document, a majority of the source code for JUnit plugin is based on the legacy code contributed by some other open-source developer, Andre Kaplan, Denis Koryavov, Eric Le Lay, and many others.

2.3 Progress of JDB Plugin

We successfully restored the configuration files for JavaDebugger's source code and replaced the outdated jEdit dependencies. Since there are still major dependencies that need to be replaced, the estimated time to complete the update for the JavaDebugger is longer than this semester. We hope to put out a usable plugin for jEdit as soon as possible as DrJava is no longer fit for new programmers. Thus we have completed the interaction pane feature as stated above.

The current source code we have is majorly based on Krishna Prakash's open source project JDebugger and could be found under `./JDB/debugger`. We have cleaned up most of the outdated dependencies as indicated above, yet the compiler could still give errors due to other corrupted libraries.

3. Future Work

There are still a few steps left before we finally wrap up the plugin bundle and publish it to the jEdit developer community as an open-source project. We will implement the interpreter for bash commands in the interaction pane so that the users will be able to make some bash operations such as `cd`, `ls`, etc. We are also planning on adding more GUI features to the interaction pane such as a clear button to delete all the cached contents, a refresh button to reload the java file, etc. We will also continue working on the Java debugger plugin by replacing all the outdated dependencies until it can fully functionate. By which time, we hope that our work will be able to contribute to the success of more Java beginner programmers, just like how DrJava did.

Appendix A Developer Guide

This part does not count toward our final report. Instead, it should be considered as a survival guide for future jEdit plugin developers who wish to continue working on the JavaDebugger plugin.

Getting Started

In order to use jEdit and use its various plugins, install the newest version of jEdit from the main website: http://www.jedit.org/ and install to desired location

There are two ways to install a jEdit plugin:

1. Open jEdit and in the Menu bar select Plugins -> Plugin Manager -> Install and search for the desired plugin
2. Download plugin source code from the plugin list page: http://plugins.jedit.org/list.php and drop the jar file found in the source code folder to the `[jEdit source directory]/jars` folder. Then go to jEdit menu bar -> Plugins -> Plugin Manager -> Manage and check the box for desired plugins

To use the plugin, go to Menu bar -> Plugins and look for corresponding plugin name in the drop down list

Configure the SDK version of jEdit

jEdit does not provide a way of confirguring the java SDK used. In order to use SDK's higher than version 8 there are two options:

1. If you are using Windows, manually change the setup for jEdit (contributed by: jrs40: http://community.jedit.org/?q=node/view/37943):
    a. Install the Windows jEdit distribution as normal
    b. Insure that the path `:\ProgramData\Oracle\Java\javapath` is in the PATH
    c. Add the jEdit directory to the PATH
    d. In the jEdit directory enter the following Batch File and name it `"jEdit.cmd": @echo off set JAVA_PATH=:\ProgramData\Oracle\Java\javapath` set `JEDIT_HOME=[Path to the jEdit distribution] start`

```
"jEdit" /b %JAVA_PATH%\javaw.exe -Xmx192M -jar
%JEDIT_HOME%\jedit.jar -reuseview %*
```

e. Add the following keys to the registry:

```
[HKEY_CLASSES_ROOT\*\shell\Open with jEdit]
"Icon"="[Path to the jEdit distribution]unins000.exe"
[HKEY_CLASSES_ROOT\*\shell\Open with jEdit\Command]
@=":\\ProgramData\\Oracle\\Java\\javapath\\javaw.exe
-Xmx192M -jar [Path to the jEdit
distribution]\\jedit.jar -reuseview \"%L\""
```

f. Modify the target of the shortcuts to Emacs the following string: `:\ProgramData\Oracle\Java\javapath\javaw.exe -Xmx192M -jar [Path to the jEdit distribution]\jedit.jar -reuseview.` Also change the icon of the shortcuts to the one contained in the jEdit.exe launcher executable

g. Rename the offending java.exe to java .xex

2. If you are using Mac:
    a. First download the latest java SDK jar and include it as jEdit library
    b. In the configuration(more below) file set the java SDK as a required dependency

To Develop a Plugin

For an overview of the development process, go to jEdit Menu bar -> Help -> jEdit Help -> Contents -> jEdit 5.5 User's Guide -> Writing Plugin and read the corresponding documentations written by the jEdit developers

jEdit checks in a plugin jar for a class with a name ending with Plugin.class, such as [PluginNamePlugin.class]. This class is known as a plugin core class and must extend jEdit's abstract EditPlugin class.

In order for a project to be recognized as jEdit plugin project, the following configuration files are necessary:

1. [project_name].props: contains metadata in human-readable string, the following properties are required for the source to be recognized as a plugin:
    a. `plugin.project_name.activate= defer`
    b. `plugin.project_name.usePluginHome= true`
    c. `plugin.project_name.name= [project_name]`
    d. `plugin.project_name.author= [author_name]`
    e. `plugin.project_name.version= [version_number]`

```
f. plugin.project_name.docs= [project_name].html
g. plugin.project_name.depend.0= [required_jEdit_version]
h. plugin.project_name.depend.1=
   [required_depenency_version]
i. ...
j. plugin.project_name.description= [A demo jEdit plugin
   that provides…]
k. plugin.project_name.longdescription=
   [description].html
```

2. Dockables.xml: contains the entry of the application:

```
<?xml version="1.0"?>
<!DOCTYPE DOCKABLES SYSTEM "dockables.dtd">
<DOCKABLES>
  <DOCKABLE NAME="[project_name]">
    new [constructor for UI here]( [view] );
  </DOCKABLE>
</DOCKABLES>
```

Here are some configuration files that might be needed if certain features are desirable:

1. Actions.xml: define procedures that can be bound to a menu item
2. Services.xml: one plugin can work with other plugins and avoid a bidirectional build-dependency
3. Build.xml: build files for ant build, which is the common way of building jEdit plugins

To Continue JavaDebugger Plugin

JavaDebugger is no longer in the install list of jEdit plugins, but the source code could be found here: http://plugins.jedit.org/plugins/?JavaDebugger

JavaDebugger source code has the following structures:

1. Core folder: This folder contains the core functionalities the debugger uses, this class is mostly based on standard Oracle's sun.jdi and java.io library and is currently usable

2. Event folder: This folder handles event such as breakpoint event and pass the request to the backend core. The event classes are also usable

3. GUI folder: This folder contains multiple classes that builds up the UI for JavaDebugger. All of JavaDebuuger's UI are made with java swing, which is currently usable but might be outdated soon.

4. Options folder: This folder contains some UI classes that are lying more in the front.

5. Plugin class: This is where the major corruption happens. The classes in the plugin folder heavily relies on the JavaCore plugin which does not function properly anymore due to its age. The JavaCore plugin was designed to manage classpaths and source paths for other java-centric plugins. The JavaCore plugin has been removed from the plugin list of jEdit's main website, but the source code could be downloaded here: http://plugins.jedit.org/plugins/?JavaCor We have not found currently replacement for the JavaCore package, but with in jEdit, the EditBus and EBMessage class are also used to pass information between jEdit and the plugin. These two classes could potentially replace JavaCore as new functions have been added to jEdit since JavaCore and JavaDebugger are developed.

6. Configuration files: The configuration files are crucial for JavaDebugger to properly build. Upon changing the source code, make sure to change the corresponding configuration files. More information about the configuration files can be found above or in the jEdit help's page

JavaDebugger was an IntelliJ ant project, but since the project was made so long ago, it does not seem to fit into the current IntelliJ ant project framework. We have been able to build the project with ant from the command line, and have been debugging the source according to javac's compiling output. Remove the output configuration from the build.xml file inorder to avoid downloading extra software during the debugging stage.