Chair of Software Engineering

A web-based IDE for Java

Software Engineering Laboratory

By: Marcel Bertsch

Supervised by: Christian Estler

Dr. Martin Nordio

Prof. Dr. Bertrand Meyer

Student Number: 09-928-896

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

Content

1 Introduction

1.1 Motivation

Software development is teamwork. Therefore we need good tools which allow us to cooperate in an easy and pleasant way. Cloudstudio is such a tool; it provides not only code sharing features but also a complete IDE to develop projects. What I like most at Cloudstudio is that it is browser-based, so no additional software needs to be installed; hence compatibility is not an issue. The cloud is growing more important, local data storage will probably be less common in the future. Google Docs is a good example of how people can work together in the cloud, so why not also work with code in that way?

1.2 Goal

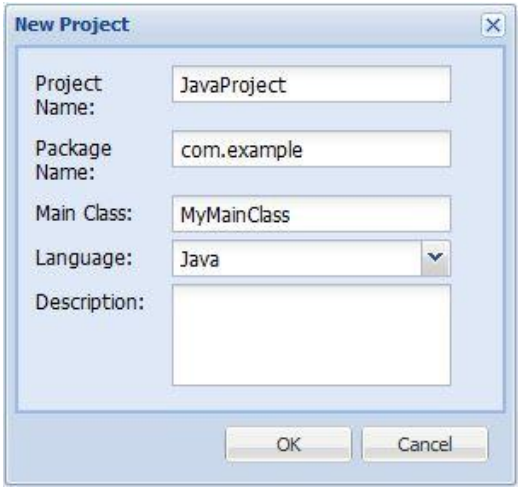
Goal of this project is extending Cloudstudio to also provide an IDE for Java development. Java is ideal because it is frequently used and well established. It also stands for platform independence and therefore correlates well with Cloudstudio. Starting point is an IDE which is completely designed for Eiffel development. The Java version should have a look and feel similar to the Eiffel version but also be comfortable for Java programmers not used to EiffelStudio.

2 Project Structure

2.1 Creation

In the New Project Dialog two new fields appear when selecting Java as a language:

- Package Name
- Main Class



The user can fill in one, both or none.

- If both fields are used, a class containing the main method gets created in a package, both named as given by the user.
- If only a package name is filled in, Cloudstudio will generate an empty package.
- If the user provides a class name but not a package name, the system will put it into a default package.
- If neither is given the project will be initially empty

For the RPC call the Eiffel version is reused. The package and main class name get appended to the project's name separated by spaces since all these parameters cannot contain any. The package name will always be attached and is called "(default)" if the user has not entered something different. This makes it possible to distinguish between package and main class name. On the server side this information gets extracted if the project's language is Java and the project accordingly created.

2.2 Database

The database adds the following entries for a new project:

Table Name	*Content*
`project`	– The project's name – A generated id – The owner's id – The language – The project description
`documentfolders`	– A folder for the source code named 'Src' with a generated id, the project's id and '0' as the parent folder's id which means the folder is at the root – Another folder with the package name having the 'Src' folder as the parent folder (if no package name is given, a '(default)' folder gets inserted)
`source_files`	– The main class file (if provided) with `folder_id` pointing to the package in `documentfolders`

2.3 Initial Main Class Content

Source file content is stored in the same way as used for Eiffel files, making it possible to reuse all kinds of functionalities for source code management. For a detailed description have a look at the Cloud Control[1] project by Alexandru Dima and Alejandro García.

Package and class names are inserted making the initial content of the main class look like this:

```
1  package com.example;
2
3  public class MyMainClass {
4
5      public static void main (String[] args) {
6
7      }
8
9  }
```

3 IDE

3.1 Syntax Highlighter

Syntax highlighting is realized in the new Eiffel class `EVSH_JAVA` inheriting from `EV_SYNTAX_HIGHLIGHTER`. When opening a file, the editor checks its ending. If it is ".java", an instance of `EVSH_JAVA` is created. Keywords, types and separators are taken from the book "Java in a Nutshell"[2].

Known types	boolean, char, byte, short, int, long, float, double

[1] http://cloudstudio.origo.ethz.ch/wiki/cloudcontrol
[2] Java in a Nutshell 5th Edition by David Flanagan

Keywords	abstract, assert, boolean, break, byte, case, catch, char, class, const, continue, default, do, double, else, enum, extends, false, final, finally, float, for, goto, if, implements, import, instanceof, int, interface, long, native, new, null, package, private, protected, public, return, short, static, strictfp, super, switch, synchronized, this, throw, throws, transient, true, try, void, volatile, while
Separators	( ) { } [ ] < > : ; , . @
String/Character delimiter	‘ “
Escape character	\

3.2 Package Explorer

3.2.1 Structure

In comparison to the Eiffel IDE, both the folders and the files must be loaded when opening a project. This makes the `updateClusters` method in class `IdeGroupPresenter` a bit more complicated. Since all the server database requests are asynchronous, making two separated ones would not lead to a deterministic result. The solution is combining them in a single request which returns two lists, one with folders and the other with files. Both are packed into a hash map to be sent to the client. To work with document folders in the code and database two new classes have been added:

Class Name	*Functionalities*
`DocumentFoldersTable`	– Inserting a new folder into the database – Get a folder from the DB by its id – Get all folders of a project – Delete a folder and all subfolders/-files if wanted
`DocumentFolderModel`	– Represents a document folder as in the DB – Contains the folder’s name, its id, the parent’s id and the project id

First the folders get added to the `TreeStore` object which is used for the package explorer. Top-level folders (those with parent id “0”) are added straightaway, sub folders not until their parent is in the structure. Added folders get deleted from the list and the process is repeated until the list is empty. A hash map keeps track of added folders by mapping their ids to the corresponding `ModelData` in the tree. If the parent folder is named “Src”, the package icon is used, a normal folder icon otherwise. In a second step the source files are added.

3.2.2 Adding & Deleting

Adding and deleting packages and source files can be done by right-clicking on the parent folder/package for adding and on the element itself for deleting. The context menu always contains the two options “New…” and “Delete…” The action when clicked depends on the element.

Element	*New…*	*Delete…*
Project Folder	Create a new top-level folder	Not supported
Src Folder	Create a new package	Not supported
Package	Add a new file to the package	Delete the package and all files in it
Source File	Not supported	Delete the file
Other Folders	Not supported yet, might be used for library import in the future	

4 Compilation and Execution

4.1 Compiler

Java compilation starts in the same way as it was implemented for Eiffel projects. When the user starts the RPC by clicking the compile button, all the project's source files are read from the database and copied to the file system at location *C:/Cloudstudio/[projectID]/[userID]*. Now the class `JavaCompiler` uses java.lang.Runtime and java.lang.Process to create a new folder for the compiled project at *C:/Cloudstudio/[projectID]/[userID]/compiled* and starts javac to compile the source files. If javac exits with code 0, the user is informed that the compilation was successful. If the exit code is not equal to 0, javac's error output is sent to the client console. Unlike as in the Eiffel version, these return messages are not processed on the client side and are printed as sent by the server.

```
// Compile project to path/compiled
String compileCommand = "javac -g -d " + path + "/compiled" + " *.java";
process = runtime.exec(compileCommand, null, new File(path));
BufferedReader errorStream = new BufferedReader(new InputStreamReader(process.getErrorStream()));
String line, errors = "";
while((line=errorStream.readLine()) != null) {
    errors += line + "\n";
}
errorStream.close();
int exitValue = process.waitFor();
```

Note that the JDK binary folder must be in the server's path for this to work.

4.2 Running a Project

Due to security concerns it is not possible to run projects on the server. The workaround is an executable JAR archive of the project, generated by the server and sent to the user. When the user clicks the run button, a dialog appears where the name of the main class (including package) must be entered. This information is required to determine the entry point of the program and will be put into the manifest file in the archive which makes the JAR executable.

The callback of the run RPC opens a new window connecting to getJarServlet mapped to `GetJarService`. The URL is provided by the callback and contains all information required (projected and userID) to find the JAR in the file system which then is sent to the client where the browser's standard download dialog appears. As soon as the download is completed, the JAR file on the server gets deleted and further requests with the same URL will receive a file not found error. To execute the JAR, the user can simply double-click it or type "*java –jar [JarName].jar*" into the command shell.

cloudstudio/WEB-INF/web.xml

```
<servlet>
    <servlet-name>getJarServlet</servlet-name>
    <servlet-class>com.cloudstudio.server.compiler.GetJarService</servlet-class>
</servlet>
<servlet-mapping>
    <servlet-name>getJarServlet</servlet-name>
    <url-pattern>/getJar</url-pattern>
</servlet-mapping>
```

com.cloudstudio.server.compiler.JavaCompiler.runJavaProject

```
// Create manifest
String manifestCommand = "cmd /c echo Main-Class: " + mainClass + " > manifest.txt";
process = runtime.exec(manifestCommand, null, new File(path));
if (process.waitFor() != 0)
    return "error";

// Create JAR
String JARCommand = "jar cvfm " + projectName + ".jar " + path + "/manifest.txt *";
process = runtime.exec(JARCommand, null, new File(path + "/compiled"));
if (process.waitFor() != 0)
    return "error";
```

com.cloudstudio.server.compiler.GetJarService.doGet

```
// Send file
resp.setContentType("application/java-archive");
resp.setHeader("Content-Disposition", "attachment;filename="+fileName);
FileInputStream fileIn = new FileInputStream(file);
ServletOutputStream servletOut = resp.getOutputStream();
byte[] buffer = new byte[4096];
int bytes;
while ((bytes=fileIn.read(buffer)) > 0) {
    servletOut.write(buffer, 0, bytes);
}
```

5 Conclusion

Programmers now have a web-based IDE where they can develop Java projects together. They are able to add and delete files and packages, write code, compile it and download an executable version of their project. There are still a lot of things to do and improve; this project is to be seen as a framework for further development steps on Cloudstudio.