Online Java Compilation on the Moodle

ITEC810 Project Report

This project report presents an online Java compilation plug-in for the Moodle Learning Management System The system allows students to compile and execute Java programs directly through the Moodle Learning Management System

Kartik (41563638)
6/5/2009

Supervisor:
Matt Bower

Acknowledgement

My most sincere thanks go to my advisor, Dr. Matthew Bower. I thank him for providing me opportunity to work in the area of online compilation. I thank him for his guidance, encouragement and support during the development of this project. He has been helping me to improve my English communication and writing skills.

I am indebted to Prof Robert Dale for his advices and inspiration about project management, time management and presentation.

I also would like to express my eternal gratitude to my parents, my wife Komal and bahuchar mata for their everlasting love and support.

Table of Contents

Figure Index

1 Abstract

This project report presents an online Java compilation plug-in for the Moodle Learning Management System (Moodle Community, n. d.). The system allows students to compile and execute Java programs directly through the Moodle Learning Management System so that they can concentrate on the programming concepts rather than learning to operate new technologies. This feature allows students to do Java programming anywhere, anytime using just web interface. The system also provides error diagnostics on the compilation and execution errors. System also provides log details for better understanding of errors. Each attempt of compiling Java program generates time statistics and gets stored in database. Teachers can view statistics generated by system and analyze students programming behavior. The technologies and Java APIs related to the online compilation processes are discussed in detail. The document also highlights security problems related to the servers and their resolutions. Development outcomes are presented and opportunities for future work are discussed.

2 Introduction

Most universities across the world are incorporating web-based Learning Management Systems into their courses. Moodle (Moodle Community, n. d.) is has been rated as the most popular Learning Management System (Kathy D.Munoz, Joan Van Duzer, 15 February, 2005) and comes with a number of features for creating forums, organizing quizzes, giving choices to students, conduct surveys, uploading assignments, holding chat sessions, creating workshops and much more. This project is developed using the Moodle which requires some understanding of Moodle (Moodle Community, n. d) Learning Management System (LMS). Moodle is software that produces internet based courses and web sites. This project is design to support a social constructionist framework of education. Moodle promotes learner involvement. The Moodle can be installed on any computer which supports PHP. It also supports SQL type that means different databases which allows organizations to choose from wide range of databases. The word Moodle was originally an acronym for Modular Object-Oriented Dynamic Learning Environment, which is mostly useful to programmers and education theorists. Moodle provides three main features Site management, User management and Course management. Importantly, Moodle is an open source system and as such allows universities to customize the Moodle server for their usage.

One problem that students experience when first learning computer programming is cognitive overload caused by not only having to learn programming concepts but also how to operate an Integrated Development Environment (IDE). As well, teachers experience problems coordinating coursework tasks in terms of specifying tasks, collecting student responses, analyzing student work and providing feedback. The online Java compilation on the Moodle provides solution to this problem. As Moodle offers many features which were mention earlier in this section. These features are integrated with different modules. Among all these modules, the question type module provides functionality which can be integrated with online compilation. The question type module of the Moodle allows creating all familiar forms of assessment including true-false, multiple choices, short answer, matching questions, random questions, numerical questions, embedded answer questions with descriptive text and graphics. The following figure 1 and 2 depicts Moodle question type module and new quiz creation.

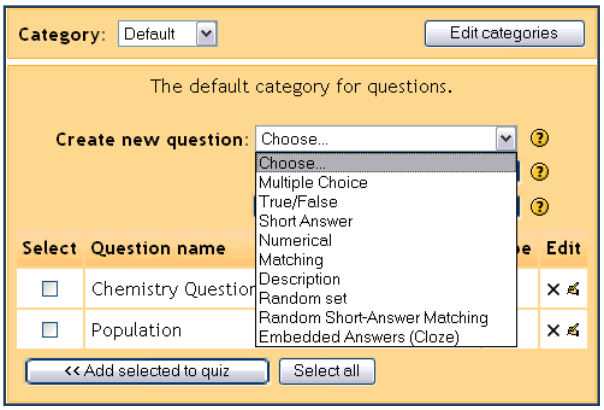


Figure 1 Question type selection screen

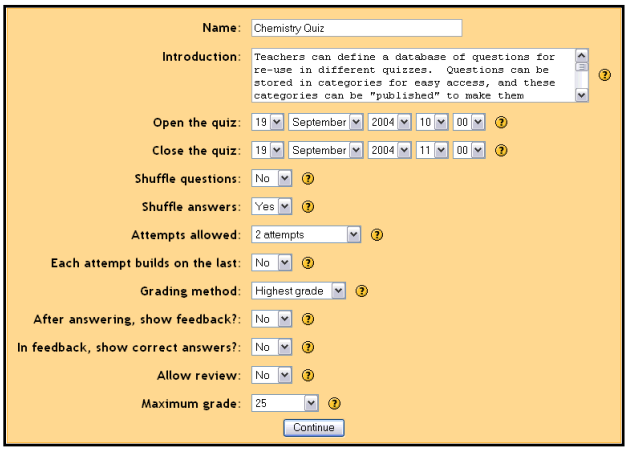


Figure 2 Creating new Quiz

This document reported on the development of a new Moodle JavaOnline question type plug-in which provides the facility to compile Java programs online. This means that students do not have to install and configure different compilers on their machines. As well, the plug-in reduces the overhead of evaluating answers and provides error statistics to teachers which can help them to identify more common mistakes made by students. The plug-in is developed considering security aspects and runs independently from the Moodle server. The proposed feature of web-based compilation on Moodle server adds up new functionality.

This document reports on the design of the Java Question type plug-in and explains the rationale behind the design. After briefly discussing related work, the overall design of the module is outlined in terms of the way in which the PHP/Java Bridge (Jost Bökemeier, n. d.) is used to communicate between the Moodle server and Java compiler. The approach to information processing is explained and security issues are identified. The overall outcomes of the project are presented and areas for future research and development are proposed.

3 Related Work

This section provides detailed information from resources and literature relating to the development of a web-based compilation module on the Moodle (Moodle Community, n. d.) server. With rapid development in information and communication technology, features offered by online learning environment has changed over. Learning Management System like Blackboard (Blackboard Inc, 2009), Moodle (Moodle Community, n. d.) has good popularity in market and offers great E-learning solutions.

In last three decades, a number of programming languages have evolved with different features and capability. Among all these languages, Java (Sun Microsystems Inc, 2009) is one of the programming languages which accepted by almost all development community. According to Douglas Kramer (1996), the reason for its popularity is that the Java programming language is a pure objected-oriented language that uses a Java Virtual Machine (JVM) to accomplish cross platform applications. Compared to other functional or structural programming languages learning Java can be hampered by the fact that to run even the simplest program the learner has to define a public class and method with a certain signature. The purpose of web-based compilation is to provide basic knowledge and better understanding of programming language without needing to at the same time an IDE and within a structured sequence of learning

activities. There has been major amount of work done in this area using Interpreters that inform how student's programs may be processed prior to compilation which defines setup logical grammar for online compilation but none of these Interpreters are integrated with Learning Management Systems. There are few other ways by which we can compile and run Java program online using JVM. These are APIs provided by Java itself but again these have security tradeoff which is discussed in later section.

3.1 Interpreters

There are number of Java Interpreters developed including JOSH (Stephan Diehl & Claudia Bieg, 2008), MiniJava (E. Roberts, 2001), DynamicJava (S. Hillion, n.d.) and BeanJava (P. Niemeyer, n.d.). All these interpreters preserve original semantics of Java and allow access to all features and API of Java but they differ extremely from Java Interpreter. This can help learner to understand primitive values, variables, expressions, assignments and control-flow statements before even knowing about classes and methods. Development of interpreters required logical or low level programming knowledge as we are putting new rules and definitions. Two such interpreters discussed over here are: JOSH (Stephan Diehl & Claudia Bieg, 2008) and DrJava (E.Allen, R.cartwright and B. Stoler, n.d.)

3.1.1 JOSH and JOSH-online

JOSH (Stephan Diehl & Claudia Bieg, 2008) is a stand-alone Java interpreter. It enable learner to evaluate expressions, to execute simple statements, to declare variable and to define methods without the need to define classes. The term code fragment is defined by this interpreter and it is a sequence of one or more simple code fragments. JOSH starts evaluating or executing as soon as user enters a code fragment. So when user presses the return key it checks whether the input so far is code fragment or the prefix of code fragment. If input buffer is not code fragment then as a result syntax error is reported and input buffer is emptied.

JOSH-online (Stephan Diehl & Claudia Bieg, 2008) is same JOSH stand-alone Java interpreter but it is setup on a server so that users don't have to setup the JDK class path and other configuration files. This facilitates access to JOSH such that everyone with a Java-enable web browser can use it.

The way in which JOSH processes a code fragment is shown in Figure 3 (Stephan Diehl & Claudia Bieg, 2008). It first reads user input and parses that input to

generate code by compiling using the JVM to store its state in temporary file. It repeats process until it encounters an exit point. During this process if it encounters any compilation or parse error then it shows it to user. If the user has finished then it checks previous state put it along with current state and compile it. On successful compilation, it executes the program and returns the result to the user.

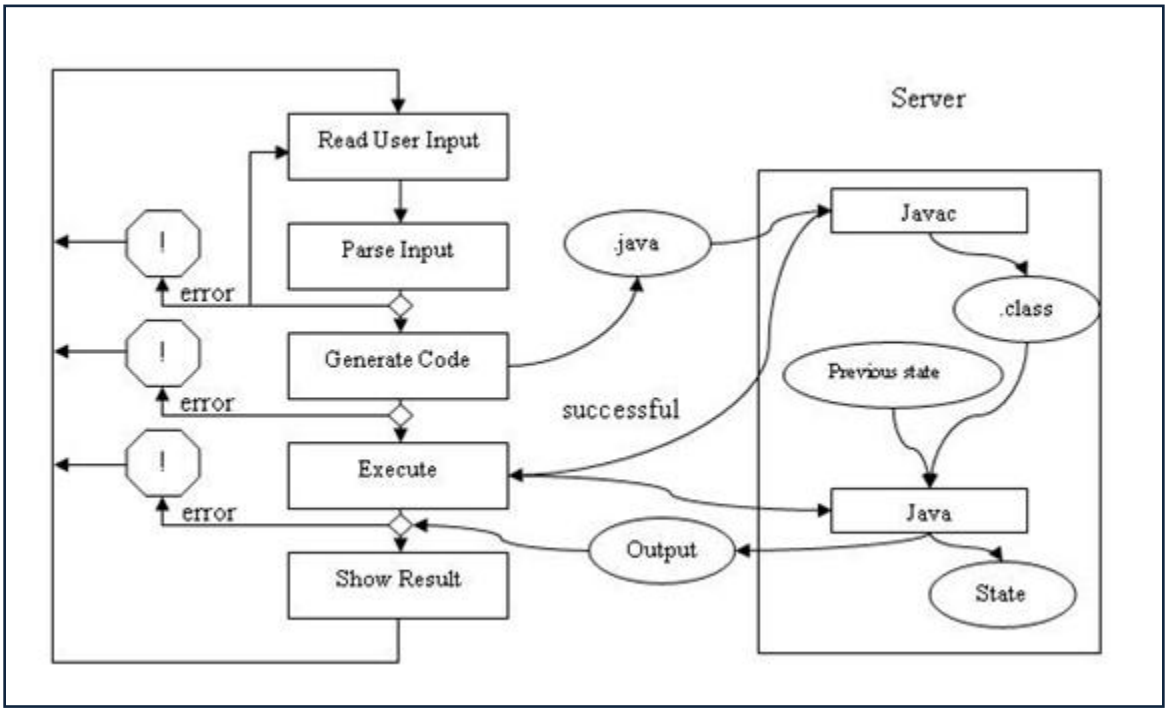


Figure 3: Code fragment in the JOSH

In the JOSH-online applet the user enters code fragments in a text area. After clicking a button the code fragments is sent to the server, the server parses the fragment. If it is complete the fragment is compiled in a similar way as before. If the compilation was successful, the client dynamically loads the newly generated class from the server and executes its main method. Each output produced by the compiler is stored as different state in server. When user presses the return key it combines all states and execute.

The JOSH approach can be summarized as follows: First compile code fragments, and then execute them externally or by dynamic class loading while preserving state.

3.1.2 DrJava

DrJava (E.Allen, R.cartwright and B. Stoler, n.d.) is a light weight educational programming environment for Java. It enables students to focus on designing their programs than learning how to use the environment. The environment provides a simple interface based on a ``read-eval-print loop''. It enables a programmer to develop, test, and debug Java programs in an interactive and incremental fashion. DrJava supports a transparent programming interface. This interface consists of window with two panes:

An interactions pane, where the student can input Java expressions and statements and immediately see their results.
A definitions pane, where the student can enter and edit class definitions with support for brace matching, syntax highlighting and automatic indents.

These two panes are linked by integrated compiler that compiles the classes in the definition pane for use in interaction pane. DrJava is a Java programming environment that uses DynamicJava(S. Hillion, n.d.) within its interaction window. For classes defined in other windows it calls the Java compiler.

JOSH and JOSH-online are basically front-ends to whatever Java compiler you want to use. Compared to DynamicJava and thus DrJava it executes code fragments more than 100 times faster[1]. JOSH has no restrictions with respect to the APIs that can be used and it is possible to use the Java debugger in combination with JOSH.

[1] For example the loop for (int i=0; i<10000000;i++); took 3 seconds in JOSH and 297 seconds in DrJava

4 Justification of the Approach Used

Online compilers for languages like C, C++, PHP and Ruby are available but all of these compilers have limitations and they are not design to integrate with Learning Management systems. Considering Java programming language, there are number of Java Interpreters developed including JOSH (Stephan Diehl & Claudia Bieg, 2008), MiniJava (E. Roberts, 2001), DynamicJa-va (S. Hillion, n.d.) and BeanJava (P. Niemeyer, n.d.) which discussed in the 3.1 section. These Interpreters are developed in early days to emphasize on the same problem as online compilation but these approaches define rule engines and complex implementation. User programs pass through this rule engines and evaluate it. However, compilers and interpreters have different set of operations and today programming languages like Java provides rich sets of APIs using that online compilation problem can be simplified.

Until recently no Learning Management System provided a complete solution for online Java compilation. In the past few months the Junit question type (Süreç Özcan, January 2009) plug-in was released for the Moodle server, which compiles Java programs online. This plug-in uses the exec() method to compile and execute Java Programs. However, there are limitations associated with Junit's use of the exec() method, such as the limited size of the output buffer and a range of security issues. The main purpose of exec() method is to execute commands over the shell and return back a result of execution but compilation of Java programs is different. Passing small Java programs to exec() method may compile properly in some cases but not all. For instance a Java program which does extensive string operations may throw exceptions such as ArrayIndexOutOfBoundException, NullPointerException and in most of cases the exec() method shows blank output or possibly throws a stack overflow exception. This is because of the limited buffer size. In terms of security issues, the Junit depends on the Java policy file setup by the administrator and if setup incorrectly can impose security vulnerabilities. As well since the Junit plug-in must call the exec() method on the same server as the Moodle server there is the potential that executed code may not only crash the plug-in but also the entire Moodle server.

There are other approaches like Runtime.exec() on Java and exec() on PHP, that can be used to compile and execute Java programs. These approaches are simpler but do not provide a complete solution to online Java compilation. Some of the issues related with exec() are; it does not provided detail compilation statistics, the programs need to be compiled on the same machine, with larger

error statistics it may throw statckoverflow exceptions and other unexpected results.

The approach adopted in the JavaOnline question type utilizes an entirely different design in order to the allow compilation of Java programs using Java API's and on a different server. This provides a more secure and robust approach to online compilation of Java programs. It runs on the same JVM (Java Virtual Machine) as the current program which reduces the overhead of starting a new JVM. As well, the complier keeps running on the server so when more source files are compiled the compiler does not need to be reinvoked each time the way it would if using exec() (Roedy Green, February, 2006). The JavaCompiler works with any plat-form, whereas platforms need to be individually configured to use the exec() approach. The idea of extending applications via compiling and loading Java extensions is not new, and several existing frameworks such as JSP (Java Server Pages) already support this capability.

5 Technology Used

The JavaOnline question type plug-in is implemented using the API's of Java Standard Edition 6 (Sun Microsystems Inc, 2009c). New API's available in Java 6 make it possible for programs to compile source code and these programs can reside on a remote server other than the Moodle server. The PHP/Java Bridge provides a communication channel between the Moodle server and the Java web-server, as shown in Figure 4.

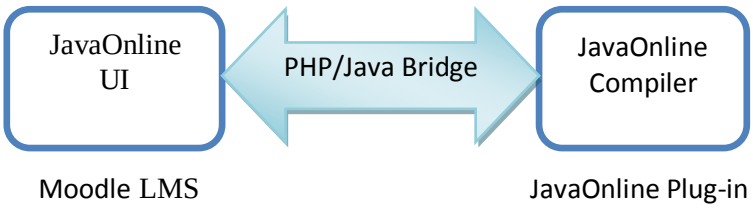


Figure 4: Structure of JavaOnline question type

5.1 JavaOnline User Interface

Moodle Learning Management System is designed and developed in PHP. Moodle has well defined and organized structures for each of its modules. There are different types of questions, like multiple-choice questions, essay questions, description question and numerical questions. Each of these questions extends default question class from Moodle questionlib file. This library file provides basic data structure and data model that each of the question should implement. This library file provide functionality such as creating answer field, creating edit form fields, creation of user sessions, resume user sessions, compare user session, storing user response, delete question type, storing and retrieving responses from database and so on. These are referred to as "question types" in the Moodle Learning Management System. Developers can use Moodle development templates that are available on Moodle web-site to develop new plug-in for the Moodle sub-system.

The User Interface of JavaOnline is developed using PHP and core libraries provided by the Moodle environment. Each of the question type modules in the Moodle uses the same folder structure. The root folder for each module has the same name as question type. Under this folder, it has db folder that contains information about schema for that module and lang folder which contains

information about languages. Other than that it also contains at least two most important files; edit_MODULENAME_form.php and question-type.php. These two files represent user interface for JavaOnline question type. The Folder structure of JavaOnline is shown in Figure 5 below.

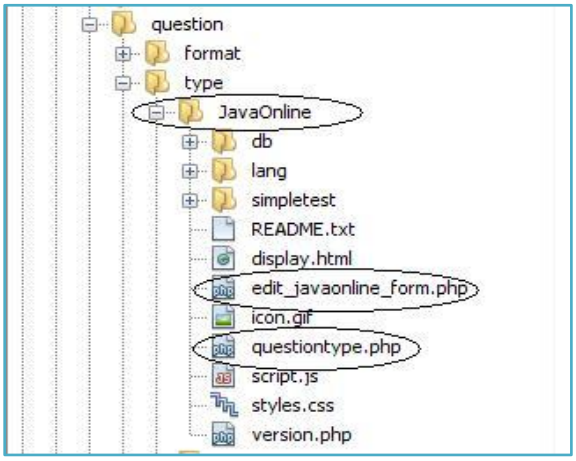


Figure 5 JavaOnline question type on the Moodle server

5.1.1 Teacher View

JavaOnline provides two different views one is for teaches and another for students, as is the case for all question types in the Moodle. The teacher view allows teachers to create new JavaOnline questions by putting information about question. Teachers can put information including Program name, expected output and logic that gives idea to students (optional). Teachers can also specify number of attempts that each student can use. This information generates unique question id for JavaOnline Question type and get stored in the Moodle database. The JavaOnline Question type uses Moodle internal database to store data. There is no need of creating new schema structure to store data. The JavaOnline Teacher view is shown in Figure 6 below.

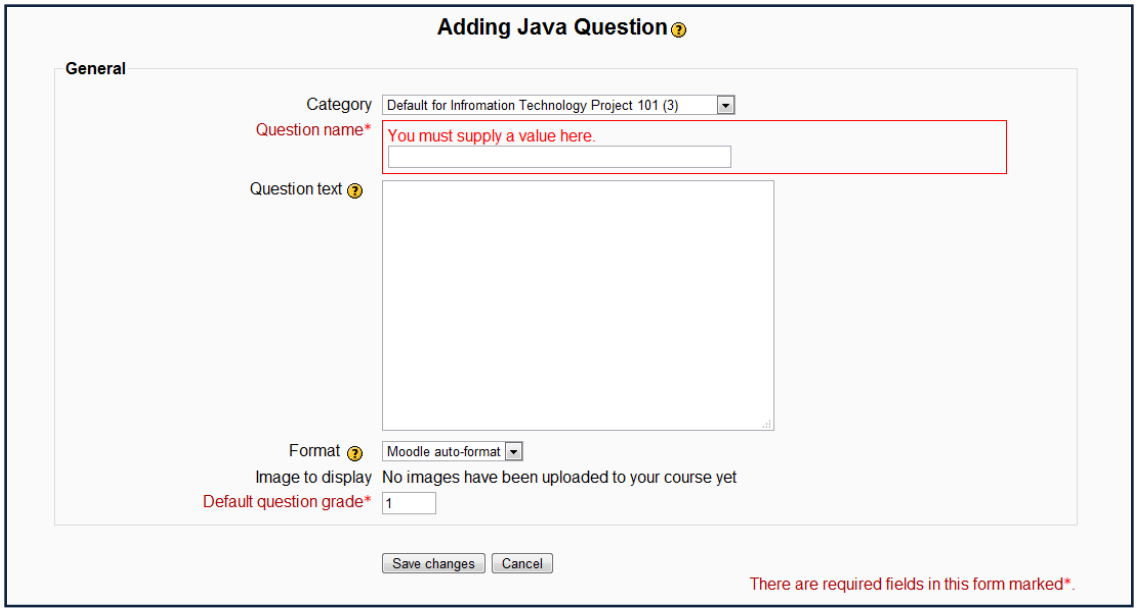


Figure 6: Teacher View

5.1.2 Student View

In the student view, students are provided with text area and submit buttons. Students should enter Java program in the text box and click on submit button which actually send program to compiler and compile it on the server. The output of the compiled program returns back with log details. The log provides better understanding of errors. If students try to compile code which may be violating security on the server, then log detail returns values like permission denied. The Output window shows compilation and execution output on successful compilation only.

There are 3 more buttons available the first button allows students to store their program without changing attempt. These means user can save their current session can retrieve back whenever they want. Another button submits program and starts new session for the student. And last button allows them to submit the program and return to main screen. The JavaOnline student view is shown in Figure 7 below.

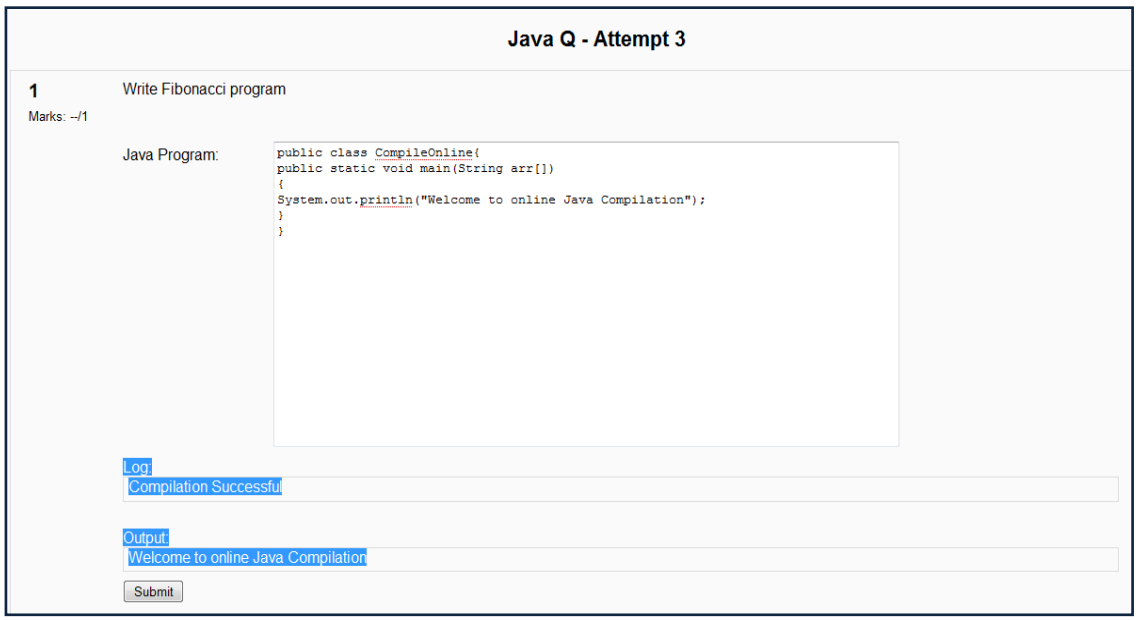
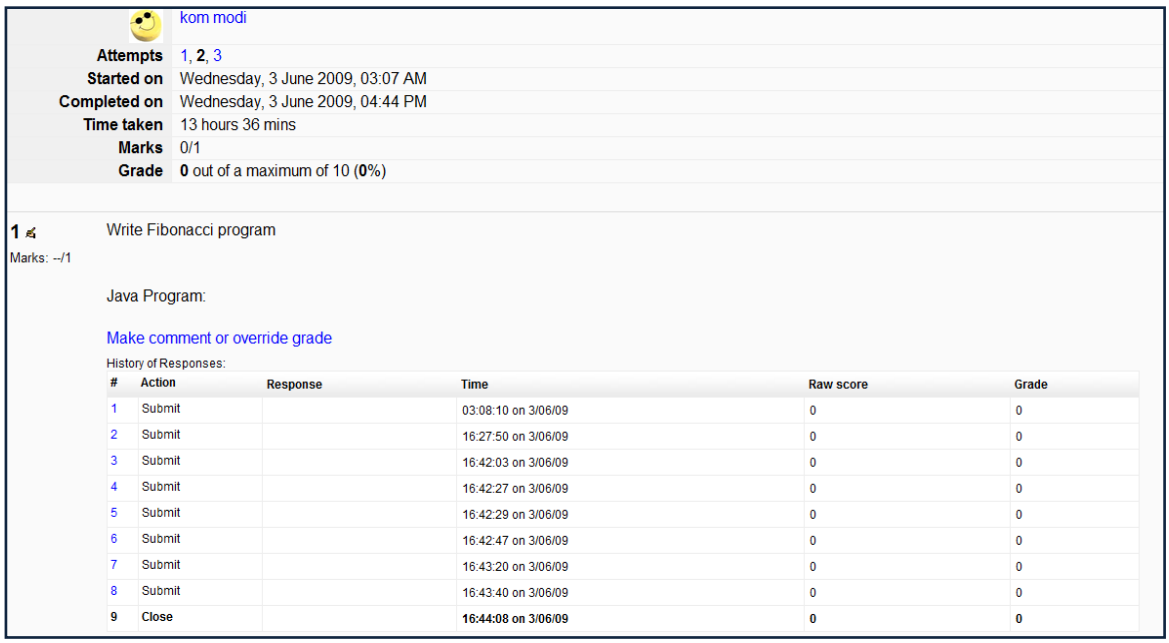


Figure 7: Student View

5.1.3 Session Creation and Data storing

Each click on the submit button compile the code on the server and the statistics about compilation gets stored in the database. Teachers can see the numbers of clicks on the compile button by each student. The result of compilation is not getting store in the current version of JavaOnline but these will be available in the future. Teacher can also specify number of attempts which is different than the number of times user tries to compile. Each attempt allows student to start question again with the new session.
If teacher has selected adaptive mode during quiz setup then students would be allowed to compile only once and students are allowed to compile up to maximum numbers of attempts set by teachers. If teacher has set unlimited attempts then student can compile any number of times.

In the JavaOnline question type, students' responses are stored in $state->responses. Each response is in the form of a string array that holds the Java program written by the student. The JavaOnline student session details are shown in Figure 8 below.



kom modi

Attempts	1, **2**, 3
Started on	Wednesday, 3 June 2009, 03:07 AM
Completed on	Wednesday, 3 June 2009, 04:44 PM
Time taken	13 hours 36 mins
Marks	0/1
Grade	**0** out of a maximum of 10 (**0**%)

1 Write Fibonacci program

Marks: --/1

Java Program:

Make comment or override grade

History of Responses:

#	Action	Response	Time	Raw score	Grade
1	Submit		03:08:10 on 3/06/09	0	0
2	Submit		16:27:50 on 3/06/09	0	0
3	Submit		16:42:03 on 3/06/09	0	0
4	Submit		16:42:27 on 3/06/09	0	0
5	Submit		16:42:29 on 3/06/09	0	0
6	Submit		16:42:47 on 3/06/09	0	0
7	Submit		16:43:20 on 3/06/09	0	0
8	Submit		16:43:40 on 3/06/09	0	0
9	**Close**		**16:44:08 on 3/06/09**	**0**	**0**

Figure 8: Student Session details

5.2 JavaOnline Compiler

The JavaOnline compiler is another component of JavaOnline developed using Java SE 6 (Sun Microsystems Inc, 2009c). This component runs independent from the Moodle server. It is a JAR file which is deployed on the web server. On the start up of the web server, this Jar file gets initialized and checks the path detail where students programs get saved.

Java SE 6 (Sun Microsystems Inc, 2009c) is the current major release of the Java SE platform. In this release they have introduced new package javax.tool that offers interfaces for retrieving and compiling files passed as strings. Thus clients can locate and run Java compilers from within a program. The package also provides other interfaces that generate diagnostics and override file access in order to support the compilation process (described below). Figure 9 shows javax.tool interface.

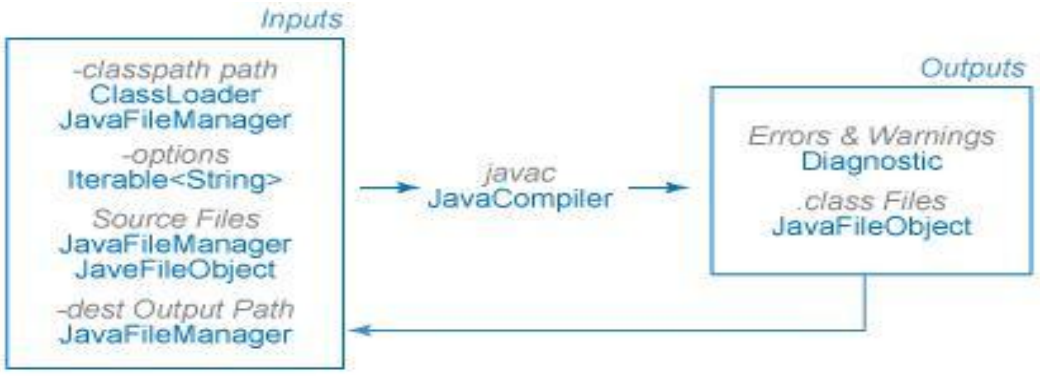


Figure 9 javax.tool Interface

5.2.1 JavaCompiler Interface

The new JavaCompiler interface of the tool package invokes Java programming language compilers from programs. It is JSR 199 specification released in final form (Peter von der Ahé, 3 April, 2007). The interface can create Java classes by compiling valid Java source, bypassing the previous need to validate byte code, or understand new object models of classes, methods, statements and expressions (David J.Biesack, Dec, 2007). It simplifies and standardizes the compilation process by providing one supported mechanism for code generation and loading that is not limited to passing a single file of source code. The JavaCompiler relies on two services: DiagnosticListener and JavaFileManager to generate diagnostics during compilation. The following example shows creation of JavaCompiler object.

```
/*Initialize Java Compiler Class */
JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();
```

5.2.2 Diagnostic Interface

The Diagnostic Interface usually reports a problem at a specific position in a specific file. It supplies error details like the line number, column number, start position, end position, error message, source of error and kind of error. It also provides details about warning and information messages generated during compilation. The diagnostic listener which is part of diagnostic interface that listen to error messages and it will be written to default output by means of System.err. There is another class called diagnostic collector that has java bean properties which get initialized when diagnostic listener finds errors. The following example shows creation of the DiagnosticCollector.

```
/* Create Diagnostic Collector to store compilation Errors */
DiagnosticCollector<JavaFileObject> diagnostics = new DiagnosticCollector
<JavaFileObject>();
```

5.2.3 FileObject Interface

The FileObject Interface is an abstraction tool. In this context file means abstraction of regular files and other sources of data. For example, a file object can be used to represent regular files, memory cache or data in database. All methods in this interface might also throw a securityException if security rules are violated. The StandardJavaFileManager interface serve two purposes: it reduces the overhead of scanning files and reading Java archive (JAR) files. The following example (Sun Microsystems Inc, 2008) shows a coding pattern.

```
/* Initialize File Manger to supply path of Java file*/
StandardJavaFileManager fileManager = compiler.getStandardFileManager(
file,diagnostics,  null, null);

compiler.getTask(null, fileManager, diagnostics, null, null, null).call();

 for (Diagnostic diagnostic : diagnostics.getDiagnostics())
 System.out.format("Error on line %d in %d%n",
                        diagnostic.getLineNumber()
                        diagnostic.getSource().toUri());

 fileManager.close();
```

5.2.4 File Execution

On successful program compilation, Java compiler generates byte code in the form of class file. This file contains executable code in the form of instruction that Java virtual machine executes. The JavaOnline plug-in then uses Runtime.exec() method to execute. On some of the platforms, Runtime.exec() throws stackoverflow exception because of limited output buffer size. To avoid this situation, JavaOnline itself provide input and output buffers. On the execution, output gets stored in these buffers and passed back to the user interface. On the execution error, the JavaOnline fetches errors from System.err to buffer and send back to User Interface. The following example shows creation of new process using Runtime.exec() method.

```
Runtime rt = Runtime.getRuntime();
Process proc = rt.exec("java classfilename");
InputStream stderr = proc.getErrorStream();
InputStreamReader isr = new InputStreamReader(stderr);
BufferedReader br = new BufferedReader(isr);
String line = null;
while ( (line = br.readLine()) != null)
System.out.println(line);
int exitVal = proc.waitFor();
System.out.println("Process exitValue: " + exitVal);
```

5.3 PHP/Java Bridge

As JavaOnline question type plug-in developed using Java and PHP language, communication between these two different languages was major concern. This problem is solved using PHP/Java Bridge (Jost Bökemeier, n. d.). PHP/Java Bridge is an implementation of a streaming XML-based network protocol, which can be used to connect a native script engine with the Java virtual machine. This API works under Java and PHP enable servers. The following example shows the important library file that needs to be included in the PHP file to create the Java objects using PHP.

```
require_once("http://localhost:8080/Java/java/Java.inc")
```

The following examples show creation of Java objects, methods and a session from PHP.

```
/* invoke java.lang.System.getProperties() */
$pro=java("java.lang.System")->getProperties();

/* convert the result object into a PHP array */
$array = java_values($props);
foreach($array as $k=>$v) {
echo "$k=>$v"; echo "<br>\n";
}

/* Create Java session */
$session = java_session();
```

6 Information Processing

For the purposes of this paper, information processing describes information that is passed from the JavaOnline user interface to the JavaOnline compiler where the Java programs get compiled. Figure 10 and Figure 13 gives overview of information processing for the JavaOnline question type. After initiating the session for the JavaOnline question type, student is provided with submit button on their screen. When student first time initiate JavaOnline question, it create new session and attempts gets initialized with one in the database. The values associated with this session are question id, allowable attempts, date/time of last attempt, last response data, feedback detail if provided by teachers and much more. Each new attempt changes these values in the database.

There are few configuration needed to run JavaOnline plug-in on the server. These configurations allow communication between JavaOnline user interface and JavaOnline compiler. Once JavaOnline compiler gets deployed on the server, administrator needs to configure properties such as Java program storage path with read, write and executable permission and unique id scheme to store program for each student. On the Moodle user Interface, administrator needs to configure properties such as server name and port number.

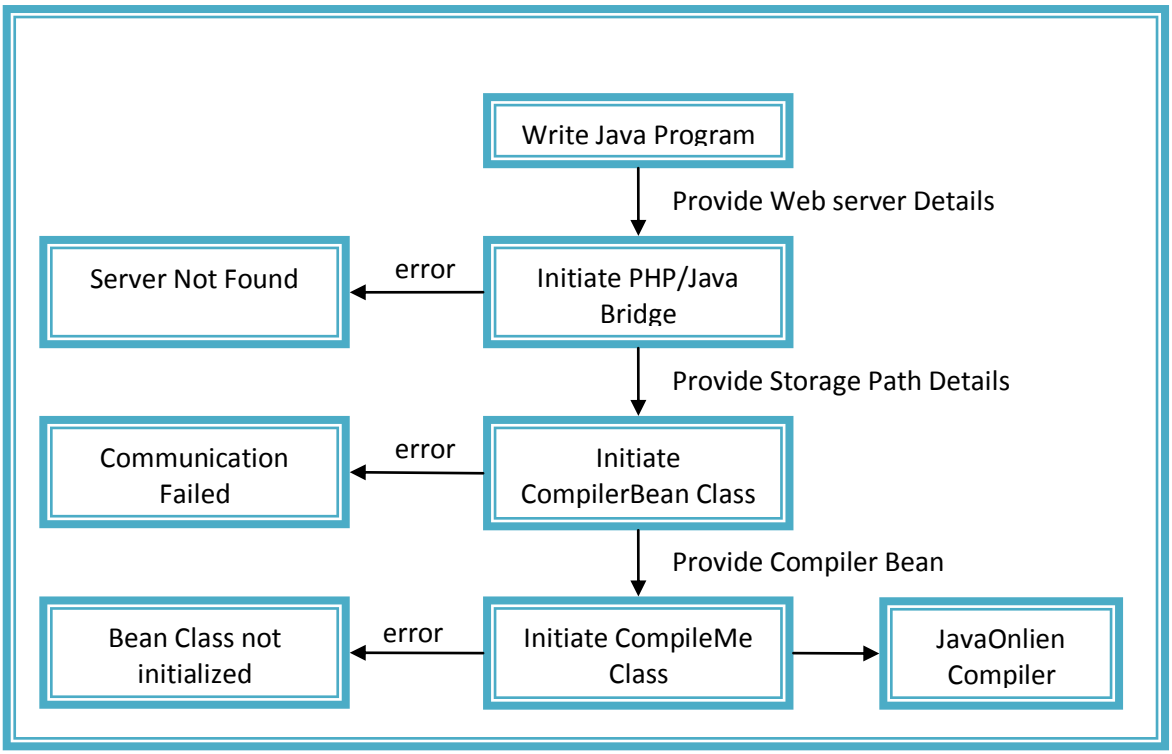


Figure 10: Information Processing in the JavaOnline User Interface

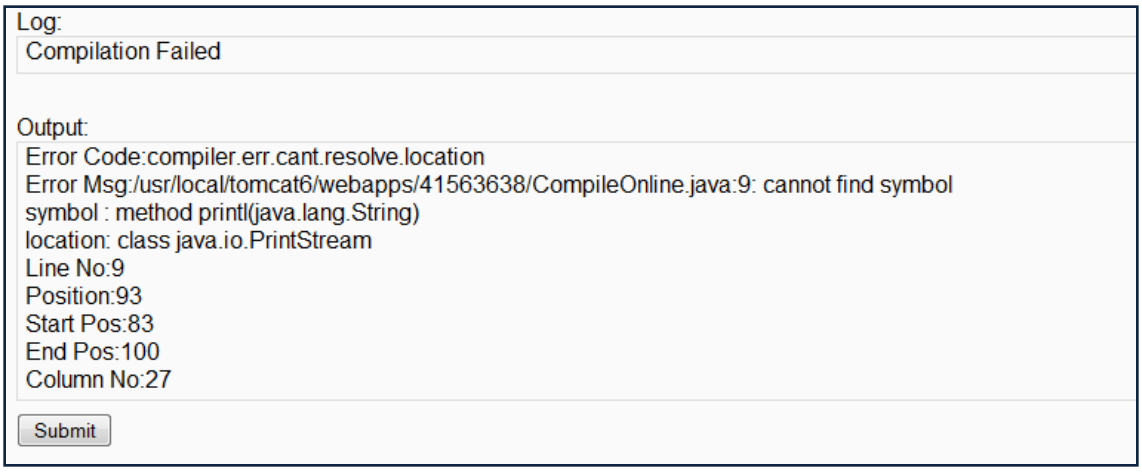


Figure 11: Compilation Exception

When students compile their Java program using submit button on the user interface, communication with JavaOnline compiler is initiated using the PHP/Java Bridge. This starts new session or restores session if student choose to continue with older session. The information present in state->responses gets converted into a Java string object from the PHP array. There is CompilerBean class which contains getter and setter methods for properties such as the user id, user name, number of attempts, the current attempt number and the Java string object. The instance of CompilerBean is created to pass information to the server. This object also fetches details of storage path setup by the administrator. Once object gets loaded with necessary information, it is then passed to JavaOnline compiler. Note that the Java string object is nothing but the Java program. Figure 10 shows information processing in the JavaOnline user interface.

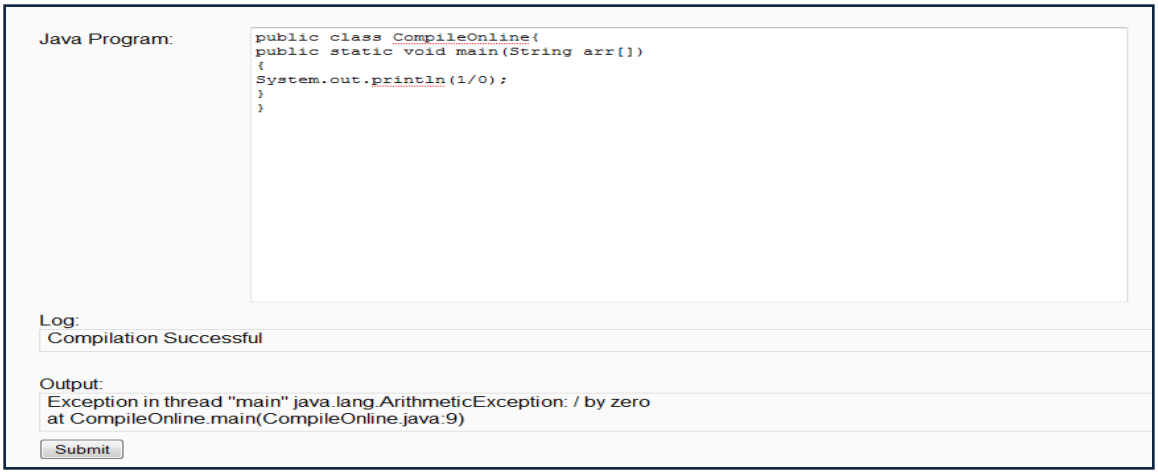


Figure 12: Execution Error

The CompileMe function calls the filesystem method which manages the directory structure on the JavaOnline compiler. It uses the path defined by the administrator as the root directory. It creates directories for each user by using their user id under the root directory. Whenever students compile their Java programs the CompileMe function checks for proper file structure and generate files under that directory only. This function search for public class defined in the student program using regular expression. If search result failed, then it returns compilation error and log window displays class not found message. If student have more than one public class in the file then also compilation failed and log window displays more than one public class found message. On successful retrieval of class name it stores file under specified directory structure using the same name. If student do not have permission or administrator has blocked that student then on compilation, JavaOnline user interface shows permission denied in the system log window.

On successful creation of file structure, CompileMe() function uses the Java APIs to compile the student's Java program, and generates error diagnostics on a compilation error. On unsuccessful compilation, it passes back diagnostics to the user interface and displays them on the screen. On successful compilation, the diagnostics object has a null value which enables execution status true on the server. This also generate class file on the same folder. This function also sets details of compilation like information or warnings generated by a compilation to the CompilerBean. If Compilation has any error then it sets parameters such as line number, start position, end position, type of error and error source of CompilerBean and passed back to the JavaOnline user interface where these details gets displayed. The compilation and execution output is shown in the Figure 11 and Figure 12 respectively.

If execution status is true then program automatically calls ExecuteMe() function which intern execute compiled java file. The output of execution gets store in the output stream buffer. The function also waits for execution of output for specific period of time otherwise it throws interrupt exception and returns error message back to Java user interface. On successful execution, it returns back output of execution and display it in the output window. The figure 13 displays information processing in the JavaOnline compiler.

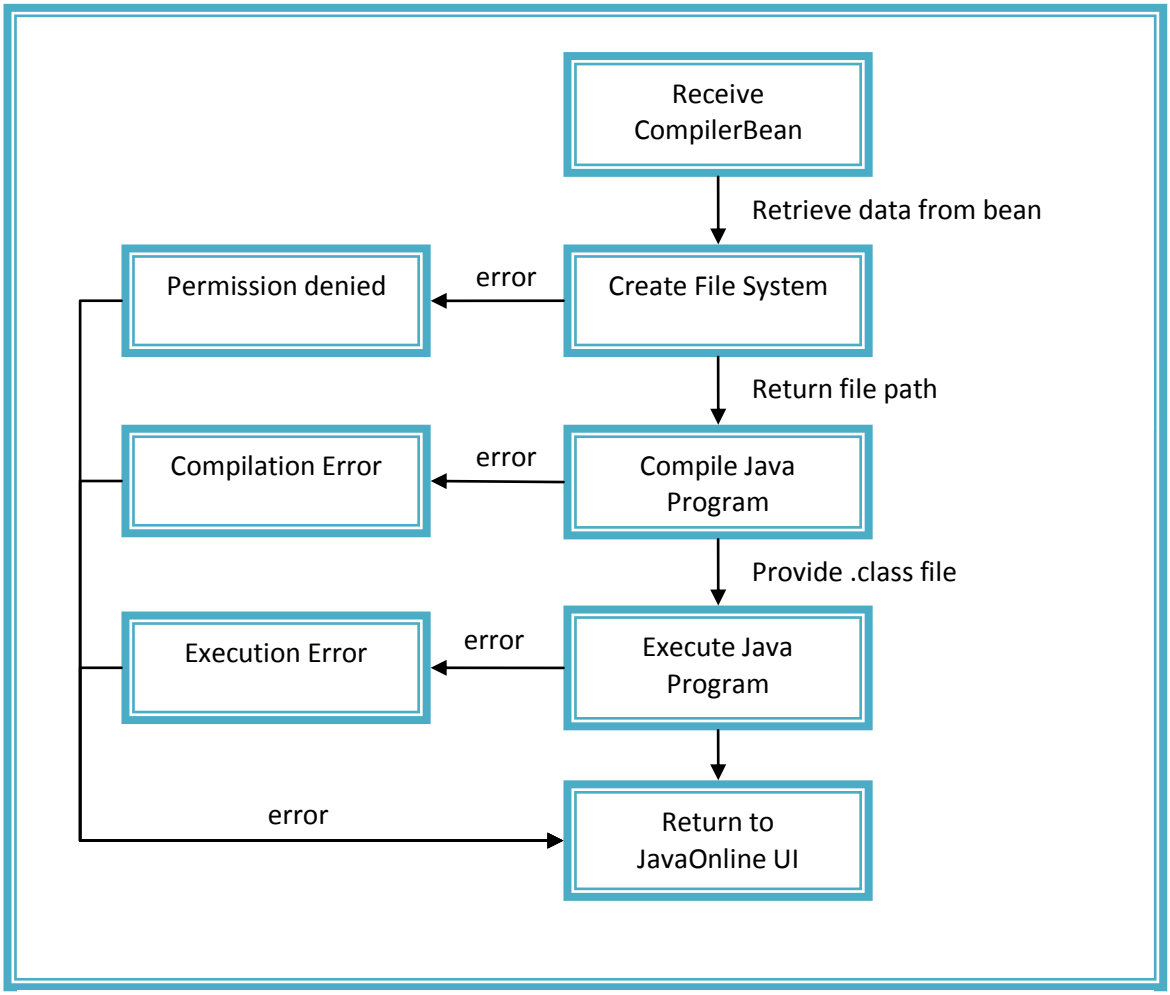


Figure 13: Information processing in the JavaOnline Compiler

If students have used all attempts then user interface disables access to JavaOnline question. For instance, consider the scenario, where teacher has set the number of attempts to be two. If the student has compiled program correctly that automatically execute compiled program and display result on the user interface. In the second attempt compilation fails which delete previously compiled file from server and display message compilation failed on the user interface. Each of these Java program gets stored in the database.

7 Security

Security of a server is major concern when considering online compilation. Online compilation on the Moodle server means allowing users to compile and run Java programs on the server. A Java program may be written by a beginner or someone who is experts in the Java programming. Both scenarios have a potential threat to the server. In the case of expert students, he/she may try to program using different Java API's which may not be configured or protected in the security policies setup by the Administrator. In the case of normal students, they may not be aware of exception handling or memory management. For an example consider the situation where teacher has asked for a program which does the simple file operation of copying content of one file to another file. In this case if student put extra loop in their program which just keeps making files on the server then the server will run out of memory and eventually crash.

Till now all discussion has considered single users but the Moodle is a web based which means that multiple users can access it at the same time and as such may create problems related to resource sharing. If there are hundreds of users try to compile Java programs that require database or file resources or may be accessing shared memory space. In this case the outcome of this compilation may be deadlock if someone holds a resource and is not releasing it for other students. Thus process management and thread management need to be considered during configuration of the server. This might be one of the reasons that none of the online learning systems have implemented web-based compilation.

The security issues related with the servers are discussed in detail on the Java Security Overview (Sun Microsystems Inc, 2005) and Moodle Administrator Document (2009). There are few techniques which minimize threats on the servers and help to achieve goals of online compilation. Configuration of the Moodle server will play an important role here. Administrator needs to isolate the Moodle server and web-based compilation server so that if web-based compilation server crashes, it does not effect on other activities of the Moodle server. Web-based compilation should be configured in such a way that for each user gets restricted amount of memory and resources. There should be an active agent which should throw interrupts after specific time so that if user has not finished compilation within specific time period then it should forcefully terminated and this way deadlocks and other resource intensive operation can be avoided.

More security would be provided with gate keeper which checks every input and output of compilation submitted by student before giving control back to the student. This gate keeper should analyze Java program by checking the header files used in the program. If user tries to include header files which do system specific operation such as java security manager, java IO then program should restrict the student. But Implementation requires deep analysis of Java API's. If a program is asking or setting any system specific changes then those changes should be avoided and the connection should be terminated. Syntax checks and some error handling should be done by the analyzer to reduce the load on the server. The light weight analyzer could also be implemented using a Java script for the client side validation process. The other activity that analyzer should perform is to check the number of clicks on the compile button and avoid it if unnecessary.

The JavaOnline plug-in always checks for size of directory allowed for each student before compiling. If this size reaches to maximum allowed size then they get error message and JavaOnline plug-in disables submit buttons. In the JavaOnline plug-in, students are only allowed to access their directory. They do not have access outside their directory. Java provide security framework that uses policy files. This policy files contain information related to access of API's available to the users. This framework allows administrator to configure server with limited access of Java API's. This policy file kept on the web server and gets initialize on the start up of the web server. An example of the policy file is shown below.

```
//example of policy file
grant codeBase "file://loction/JavaOnline.jar" {
permission java.util.PropertyPermission "user.dir", "read";
permission java.io.FilePermission "${us-er.dir}${/}", "read,write,delete";
};
```

8 Installation

This section provides installation details of JavaOnline plug-in for Moodle LMS.

Basic requirements:

- Deployed Moodle server v 1.8
- Web server (tomcat)
- Java S.E 6.0
- PHP/Java Bridge war

JavaOnline plug-in has two different part

- JavaOnline.jar
- Javaonline folder

Installation steps:

- Download PHP/Java Bridge war file from http://php-java-bridge.sourceforge.net/pjb/index.php
- Copy PHP/Java Bridge in the web server
- Open war file and add JavaOnline.jar file in the WEB-INF/lib folder
- Deploy PHP/Java Bridge file and start server
- Open questiontype.php file from javaonline folder
- Edit variable named SERVER with web server name and port number with full context path e.g. SERVER = 'localhost:8080/JavaBridge/'
- Edit variable FILEPATH with path where Java file gets stored on the web server
- Save file
- Copy javaonline folder in to the "<Moodle installation directory>\ question\type\" direcotory
- Start Moodle server

Troubleshooting

- If compilation displays blank screen that means web server is not running or it is not configured properly. Try testing web server and check server and port number details.
- PHP/Java Bridge needs PHP as well Java on the installed server to function.

9 Outcomes

The current version of the JavaOnline question type is capable of performing all of the functions outlined in the above design description. This includes the capacity to:

- ✓ Provide the teacher and student User interface.
- ✓ Runs independently from the Moodle server.
- ✓ Allow teachers to create a question.
- ✓ Allow students to attempt the programming problem online by performing successive compilations.
- ✓ Have error and other diagnostic messages returned to the student.
- ✓ Store the results of each attempt into the Moodle database.
- ✓ Execute the Java program.

Thus the JavaOnline plug-in achieves the requirements of the initial design specification.

The development of the JavaOnline Module uses MVC (Model-View-Controller) framework. The user interface part of the Moodle server act as View part which allows students, teachers and administrator to interact with module.

The View part implemented using PHP/Java Bridge which actually acts as communication link between two servers. It creates session and maintains it. One user exit from module it destroy the session. This is different session than Moodle session. The session is created for compilation and execution of Java program for each user and to maintain unique identity.

The Model part implements real compilation logic. It receive data, perform operation and send back to Model which then gets displayed on the user interface.

The MVC requires less maintenance and it is easy to understand for others developers to implement further. The development has also considered security

aspect related to online compilation and implemented basic security features such as checking file permission, creating and deleting file structure.

9.1 Limitations

There are some limitations about current version of JavaOnline plug-in they are:

- Students needs to do all functionality using single class only
- Teachers do not having option of comparing each compile request of students response
- The last compiled program is only getting stored in the database
- Students do not have option of compile and execute as different action
- Current version is tested with the limited resource and single user environment only

10 Future work

There are many opportunities to further enhance the JavaOnline question type plug-in. This includes:

- ✓ Allowing students to create multiple class files on the same screen
- ✓ Automatic evaluating student's code and grading
- ✓ Feature that permits teachers to create Java programming test online with time limits.

As well the conceptual design underpinning the JavaOnline plug-in could be used to create Java projects online. This allows multiple students to work in single group on a single project from anywhere, anytime without downloading and installing IDE tools to their machine. The student can see changes done by others online and also restore them back if necessary.

11 Conclusion

The JavaOnline question type plug-in for Moodle LMS presented is presented in this document. This provides a key solution for online compilation of Java source code. This plug-in enables students to compile their programs without having to configuring their machine for Java program compilation. This also allows students to do the Java programming online, anywhere, anytime. The provided detailed diagnostics and log helps students to find compilation and execution errors much easily. This plug-in can be used in the future to implement online IDE platform which would allow students to create and maintain their project online with capability of version control. It also generates detailed statistics of student's compilations that can help teachers to improve their teaching methodologies. Teachers can also restrict student's compilation by specifying limited attempts. The JavaOnline plug-in also developed using secure and robust approach to online Java compilation that overcomes limitations in the design of the JUnit question type.

12 Bibliography

Blackboard Inc, 2009, *Blackboard home*, viewed March 21, 2009, http://www.blackboard.com/.

Douglas Kramer, 1996, *The Java Platform: A white paper*, viewed March 23, 2009, http://java.sun.com/docs/white/platform/javaplatformTOC.doc.html

David J.Biesack, Dec, 2007, Create dynamic applications with Javax.tools, viewed on May 19, 2009 , http://www.ibm.com/developerworks/java/library/j-jcomp/index.html

E.Allen, R.cartwright and B. Stoler, n.d., *DrJava*, viewed March 22, 2009, http://drjava.sourceforge.net

Moodle community, n. d, Moodle.*org:open-source community-based tools for learning*, viewed March 21, 2009, http://moodle.org/

Moodle Administrator document, 2009, *Administrator document*, viewed March 26, 2009, http://docs.moodle.org/en/Administrator_documentation

Jost Bökemeier, n. d., *Php/Java Bridge*, viewed April 20, 2009, http://php-java-bridge.sourceforge.net/pjb/index.php

Kathy D.Munoz, Joan Van Duzer, 15 February, 2005, *Blackboard vs. Moodle A Comparison of Satisfaction with Online Teaching and Learning Tools*, viewed May 18, 2009, http://www.humboldt.edu/~jdv1/moodle/all.htm

P. Niemeyer, n.d. *BeanShell*, viewed March 23, 2009, http://www.beansheell.org/

Roedy Green, February, 2006, JavaCompiler: Java Glossary, viewed on May 19, 2009, http://mindprod.com/jgloss/javacompiler.html#BENEFITS

Sun Microsystems Inc, 2005a, *Java Security Overview*, viewed March 26, 2009, http://java.sun.com/developer/technicalArticles/Security/whitepaper/JS_White_Paper.pdf

Sun Microsystems Inc, 2008b, *Java Platform Standard Edition*, viewed March 25, 2009, http://java.sun.com/javase/6/docs/api/javax/tools/package-summary.html

Sun Microsystems Inc, 2009c, *Java SE 6*, viewed May, 20, 2009, http://java.sun.com/javase/6/

Süreç Özcan, January 2009, *Junit question type*, viewed April 25, 2009, http://docs.moodle.org/en/Junit_question_type

Stephan Diehl & Claudia Bieg, 2008, *A new Approach for Implementing stand-alone and web-based Interpreters for Java*, viewed March 20, 2009, http://www.st.uni-trier.de/~diehl/pubs/pppj03.pdf

S. Hillion, n.d., *DynamicaJava*, viewed March 23, 2009, http://koala.ilog.fr/djava/

Von der Ahé Peter, 3 April, 2007, *Leading-Edge Java: the Java Compiler API*, viewed on 02 June, 2009, http://www.artima.com/lejava/articles/compiler_api.html

Appendix – Java Interface Specification

1. CompilerBean.java

```
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package itec810.bean;

import java.io.File;
import java.io.Serializable;
import javax.tools.Diagnostic.Kind;

/**
 *
 * @author Kartik Modi
 * @version 1.0
 *
 */
public class CompilerBean implements Serializable{

    private String userId;
    private String javaProgram;
    private String path;
    private File file;
    private boolean success; //true,false
    private String errCode;
    private Kind type; //err,war
    private long lineNo;
    private long colNo;
    private long pos;
    private long stPos;
    private long endPos;
    private Object source;
    private String errMsg;
    private String fileName; //with out extension for class file
    private File folderPath; //user folder name
    private File classFile;  //user class file
    private String execOutput; //exeution output

    /**
     * The output of each stuedent's Execution gets stored over here
     * It assined to null if compilation failed
     * It used to retrieve execution output in the interface
     * @return
     */
    public String getExecOutput() {
        return execOutput;
    }

    /**
```

```
 * It set execution output on the succssful compilation
 * If execution has error stream it also store that as output
 * @param execOutput
 */
public void setExecOutput(String execOutput) {
   this.execOutput = execOutput;
}

/**
 * It is used to get class file detail for program
 * execution. it has no usage on the client side
 * @return
 */
public File getClassFile() {
   return classFile;
}

/**
 * This funtion sets up class file on successful compilation
 * this is attached for easy access
 * @param classFile
 */
public void setClassFile(File classFile) {
   this.classFile = classFile;
}


/**
 * The folderpath is the path where student is allowed to compile
 * and execute
 * It retrieve folder path
 * @return
 */
public File getFolderPath() {
   return folderPath;
}

/**
 * This setup student's working directory on the startup
 * It initialized on the starting of the program
 * @param folderPath
 */
public void setFolderPath(File folderPath) {
   this.folderPath = folderPath;
}


public String getFileName() {
   return fileName;
}

public void setFileName(String fileName) {
   this.fileName = fileName;
}
```

```
public String getUserId() {
   return userId;
}

public void setUserId(String userId) {
   this.userId = userId;
}



public String getJavaProgram() {
   return javaProgram;
}

public void setJavaProgram(String javaProgram) {
   this.javaProgram = javaProgram;
}

public String getPath() {
   return path;
}

public void setPath(String path) {
   this.path = path;
}


public long getColNo() {
   return colNo;
}

public void setColNo(long colNo) {
   this.colNo = colNo;
}

public long getEndPos() {
   return endPos;
}

public void setEndPos(long endPos) {
   this.endPos = endPos;
}

public boolean isSuccess() {
   return success;
}

public void setSuccess(boolean success) {
   this.success = success;
}

public String getErrCode() {
   return errCode;
}

public void setErrCode(String errCode) {
```

```
    this.errCode = errCode;
}

public String getErrMsg() {
    return errMsg;
}

public void setErrMsg(String errMsg) {
    this.errMsg = errMsg;
}

public File getFile() {
    return file;
}

public void setFile(File file) {
    this.file = file;
}

public long getLineNo() {
    return lineNo;
}

public void setLineNo(long lineNo) {
    this.lineNo = lineNo;
}

public long getPos() {
    return pos;
}

public void setPos(long pos) {
    this.pos = pos;
}

public Object getSource() {
    return source;
}

public void setSource(Object source) {
    this.source = source;
}

public long getStPos() {
    return stPos;
}

public void setStPos(long stPos) {
    this.stPos = stPos;
}

public Kind getType() {
    return type;
}

public void setType(Kind type) {
```

```
        this.type = type;
    }


}
```

2. CompileMe.java

```
package itec810.impl;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.util.Arrays;
import javax.tools.Diagnostic;
import javax.tools.DiagnosticCollector;
import javax.tools.JavaCompiler;
import javax.tools.JavaFileObject;
import javax.tools.StandardJavaFileManager;
import javax.tools.ToolProvider;
import itec810.bean.CompilerBean;

/**
 *
 * @author Kartik Modi
 */
public class CompileMe {

    CompilerBean mycb;
    String err;

    public String getErr() {
        return err;
    }

    public CompileMe(CompilerBean cb) {
        this.mycb = cb;
    }

    public boolean strToFile() {
        boolean flag = false;
```

```
    if (mycb.getFile() == null) {
        err = "Bean File is null";
        return false;
    }
    try {
        FileWriter fw = new FileWriter(mycb.getFile());
        BufferedWriter bw = new BufferedWriter(fw);
        char[] prg = mycb.getJavaProgram().toCharArray();
        int len = mycb.getJavaProgram().length();
        int i = 0;
        int t = 0;
        while (len != i) {
            if (prg[i] == '{') {
                t++;
            }
            if (prg[i] == '}') {
                t--;
            }

            if (prg[i] == '{' || prg[i] == '}' || prg[i] == ';' || prg[i] == '\n') {

                bw.write(prg[i] + "\n");
                int tab = t;
                while (tab != 0) {
                    bw.write("\t");
                    tab--;
                }
            } else {
                bw.write(prg[i]);
            }
            i++;
        }
        // bw.write(mycb.getJavaProgram());
        bw.close();
        flag = true;

    } catch (Exception e) {
        e.printStackTrace();
    }
    return flag;
}

public boolean setUserPath() {
    boolean flag = false;
```

```
if (mycb.getPath() == null || mycb.getPath().equals("")) {
   err = "Bean Path is null";
   return false;
}

try {
   File f = new File(mycb.getPath());
   if (!f.exists() || f.isFile()) {
      err = "Please provide valid path";
      return false;
   }

   if (!f.canExecute() || !f.canRead() || !f.canWrite()) {
      err = "Do not have enough permission for specified path";
      return false;
   }

   if (mycb.getUserId() == null || mycb.getUserId().equals("")) {
      err = "User Id not present";
      return false;
   }

   File uf = new File(f.getAbsolutePath() + "/" + mycb.getUserId());
   System.out.println("File path:" + uf.getAbsolutePath());

   if (uf.isFile()) {
      err = "File present with same folder name";
      return false;
   }

   if (!uf.exists()) {
      boolean b = uf.mkdir();
      if (!b) {
         err = "User Folder creation failed";
         return false;
      }
   }

   mycb.setFolderPath(uf);

   String fileName = retriveFileName().trim();
   if (fileName == null || fileName.equals("err")) {
      System.out.println(err);
```

```
            return false;
        }
        mycb.setFileName(fileName);

        File javaFile = new File(uf.getAbsolutePath() + "/" + fileName + ".java");
        System.out.println("JavaFile:" + javaFile.getAbsolutePath());

        if (javaFile.isDirectory()) {
            err = "Name already exists as directory";
            return false;
        }

        if (javaFile.exists()) {
            boolean b = javaFile.delete();
            if (!b) {
                err = "Failded to delete";
                return false;

            }
        }

        boolean b = javaFile.createNewFile();
        if (!b) {
            err = "Java File creation failed";
            return false;
        }
        mycb.setFile(javaFile);
        flag = true;


    } catch (Exception e) {
        e.printStackTrace();
    }
    return flag;
}

private String retriveFileName() {

    int start = 0, end = 0, i = 0;
    String str = null;
    try {
        while (true) {
            start = mycb.getJavaProgram().indexOf("class", i);
            end = mycb.getJavaProgram().indexOf("{", start);
```

```
            if (start == -1 || end == -1) {
               err = "Class name not found";
               return "err";
            }


            str = mycb.getJavaProgram().substring(start + 5, end).trim();
            System.out.println("Java program name:" + str);

            int checkStart = mycb.getJavaProgram().indexOf("class", end);
            if (checkStart != -1) {
               int checkEnd = mycb.getJavaProgram().indexOf("{", checkStart);
               if (checkEnd != -1) {
                  String str2 = mycb.getJavaProgram().substring(checkStart + 5,
checkEnd).trim();
                  if (str2 != null && str.matches("[a-zA-Z_][a-zA-Z0-9_]*")) {
                     err = "keyword class found more than once in the class";
                     return "err";
                  }
               }
            }
            if (str.matches("[a-zA-Z_][a-zA-Z0-9_]*")) {
               return str;
            } else {
               i = end;
            }
         }
      } catch (Exception e) {
         e.printStackTrace();
      }

      return "err";
   }

   public boolean compileMe() {
      boolean status = false;
      try {
         JavaCompiler jc = ToolProvider.getSystemJavaCompiler();
         DiagnosticCollector<JavaFileObject> diagnostics = new
DiagnosticCollector<JavaFileObject>();
         StandardJavaFileManager sjfm = jc.getStandardFileManager(diagnostics, null,
null);
```

```
        Iterable<? extends JavaFileObject> codeObject =
sjfm.getJavaFileObjectsFromStrings(Arrays.asList(mycb.getFile().getAbsolutePath()));
        //Iterable<String> options = Arrays.asList("-d", "classes", "-sourcepath", "src");
        JavaCompiler.CompilationTask task = jc.getTask(null, sjfm, diagnostics, null, null,
codeObject);
        sjfm.close();


        boolean success = task.call();
        for (Diagnostic diagnostic : diagnostics.getDiagnostics()) {

           mycb.setErrCode(diagnostic.getCode());
           mycb.setType(diagnostic.getKind());
           mycb.setColNo(diagnostic.getColumnNumber());
           mycb.setPos(diagnostic.getPosition());
           mycb.setLineNo(diagnostic.getLineNumber());
           mycb.setStPos(diagnostic.getStartPosition());
           mycb.setEndPos(diagnostic.getEndPosition());
           mycb.setSource(diagnostic.getSource());
           mycb.setErrMsg(diagnostic.getMessage(null));

        }
        mycb.setSuccess(success);

        if (mycb.getFolderPath() == null || mycb.getFileName() == null) {
           err = "Folder Path or File Name not found";
           return false;
        }

        File classf = new File(mycb.getFolderPath() + "/" + mycb.getFileName() + ".class");
        System.out.println(classf.getAbsolutePath());

        if (!success) {
           status = deleteFile();
        } else {
           if (classf.isFile() && classf.exists()) {
              mycb.setClassFile(classf);
              status = true;
           } else {
              err = "problem in creating class file";
              status = false;
           }
        }
```

```
    } catch (Exception e) {

        e.printStackTrace();
    }
    return status;

}

public boolean deleteFile() {
    boolean status = false;
    try {
        if (mycb.getFolderPath() == null || mycb.getFileName() == null) {
            err = "Error in deleting file! file not found";
            return false;
        }

        File f = new File(mycb.getFolderPath() + "/" + mycb.getFileName() + ".class");
        System.out.println(f.getAbsolutePath());

        if (f.exists()) {
            if (f.isFile()) {
                f.deleteOnExit();
                status = true;
            } else {
                err = "file not found! it is folder";
                return false;
            }
        } else {
            status = true;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return status;
}

public boolean executeMe() {
    boolean status = false;
    try {
        /*   URL url=mycb.getFolderPath().toURI().toURL();
        System.out.println(url);
        URL[] urls= new URL[]{url};
```

```
            Class parameters[] ={String[].class};
            ClassLoader cl=new URLClassLoader(urls);

            Class c=cl.loadClass(mycb.getFileName());
            Method met=c.getMethod("main",parameters);

            String[] args={"abc","pqr"};
            Object objects[]={args};
            InputStreamReader isr = new InputStreamReader(System.in);
            BufferedReader input = new BufferedReader(isr);
            met.invoke(null,objects);

            System.out.println(input.readLine());*/
            Runtime rt = Runtime.getRuntime();
            Process proc = rt.exec("java -cp " + mycb.getFolderPath() + " " +
mycb.getFileName());
            InputStream stdOut = proc.getInputStream();
            InputStream stdErr = proc.getErrorStream();

            BufferedReader brErr = new BufferedReader(new InputStreamReader(stdErr));
            BufferedReader brOut = new BufferedReader(new InputStreamReader(stdOut));

            StringBuilder output = new StringBuilder();
            String line = null;

            while ((line = brErr.readLine()) != null) {
               output = output.append(line + "\n");
            }

            while ((line = brOut.readLine()) != null) {
               output = output.append(line + "\n");
            }

            int exitVal = proc.waitFor();
            System.out.println("Exit Value:" + exitVal);
            String mystr = output.toString();
            System.out.println(mystr);
            mycb.setExecOutput(mystr);
            status = true;
         } catch (Exception e) {
            e.printStackTrace();
         }
```

```
        return status;
    }
}
```