San Jose State University

SJSU ScholarWorks

Master's Projects

Master's Theses and Graduate Research

2010

Automatic Grading of Programming Assignments

Ashlesha Patil

San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_projects

Part of the Computer Sciences Commons

Recommended Citation

Patil, Ashlesha, "Automatic Grading of Programming Assignments" (2010). *Master's Projects*. 51.
DOI: https://doi.org/10.31979/etd.vnt7-hgnd
https://scholarworks.sjsu.edu/etd_projects/51

This Master's Project is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Projects by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

AUTOMATIC GRADING OF PROGRAMMING ASSIGNMENTS

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Computer Science

by

Ashlesha Patil

May 2010

©2010

Ashlesha Patil

ALL RIGHTS RESERVED

ABSTRACT

AUTOMATIC GRADING OF PROGRAMMING ASSIGNMENTS

By Ashlesha Patil

Solving practical problems is one of the important aspects of learning programming languages. But the assessment of programming problems is not straightforward. It involves time consuming and tedious steps required to compile and test the solution. In this project, I have developed a online tool, Javabrat that allows the students and language learners to practice Java and Scala problems. Javabrat automatically assesses the user's program and provides the instant feedback to the user. The users can also contribute their own programming problems to the existing problem set. I have also developed a plugin for a learning management system, Moodle. This plugin allows the instructors to create the Java programming assignments in Moodle. The Moodle plugin also facilitates automatic grading of the Java problems.

ACKNOWLEDGMENTS

I owe a great many thanks to Dr. Horstmann for trusting me with his idea. This project would not have been possible without his guidance and support. One simply could not wish for a more knowledgeable and caring adviser.

I would also like to express my gratitude to both Dr. Jon Pearce and Dr. Chris Pollett for their suggestions and time. I also thank them for agreeing to be on the project committee.

Last but not the least, I would like to thank my husband Nikhil whose care, love and support has taken the load off my shoulder. I owe him a lot for his guidance and encouragement. This journey would not have been successful without you.

Table of Contents

1. Introduction

Solving practical problems is one of the important aspects of learning programming languages. According to the author of "Teaching programming paradigms and languages for qualitative learning" published in the proceedings of the 2nd Australasian Conference on Computer Science, "an important observation in many programming related subjects is that major learning takes place through practical programming assignments" [1]. There are several tools and web applications available that are used to test as well as grade users' programming skills. Examples are CodingBat [2] and better programmer [3]. But at the time of this writing, there is no existing online tool that assesses Scala programming problems. A limitation of the existing applications is that they have a fixed set of programming problems. As of December 2009, none of these applications allowed users to contribute to the problem set (Only recently, CodingBat has moved in this direction). One of the project goals was to develop a online grader to address the above mentioned limitations.

Computer science educators make extensive use of programming assignments. One way to enhance the outcomes of the learning process is to require students to solve practical coding problems. But the evaluation of programming assignments is not straightforward. It involves time-consuming steps for compiling answers and testing them for the various inputs. This evaluation overhead confines the number of programming assignments an instructor can assign during a course period. One way of reducing this overhead is to automate the evaluation process. In this project I have developed a system that automates the grading of Java programming assignments.

Many instructors and universities are adopting Learning Management Systems (LMS) such as Moodle [4] or Blackboard. These tools effectively lessen the course management efforts. In this project, I am extending Moodle to carry out a automatic grade evaluation. This would facilitate teachers to use one integrated LMS that provides automatic grading of the programming assignments along with the other course management aspects.

Thus, in this project work, I have addressed two different sets of users:

1. Individual programmers learning Java or Scala languages
2. Instructors who want to carry out automatic grading for the Java programming assignments.

I started the project with a existing Java grader (Labrat) and a web interface (WebLabrat). These applications were developed by Dr. Horstmann. I have developed the following two different systems addressing the above mentioned user sets respectively:

1. Javabrat: A web-based grader useful for the programmers learning Java and Scala languages. Javabrat has the following enhancements over WebLabrat:

   i. an web interface to view the collection of existing problems
   ii. assessment of Scala programs
   iii. a feature that allows users to contribute their own programming exercises to the existing problem set.
2. A Moodle plugin that facilitates instructors to grade Java assignments using Labrat.

2. Prior Work

2.1. Online Programming Tools

There are several programming tools developed as web applications, where users can exercise their programming skills by coding a solution for the given problem. Some of these tools are CodingBat, betterprogrammer, Practice-It [11]. One of the advantages of these systems is that the users get instantaneous feedback about their answers. Even if a user fails to solve a problem at the very first attempt, an accurate and detailed feedback inspires him to re-attempt the problem until he reaches the correct solution. CodingBat permits the users to practice Java and Python coding online. It also allows students to create accounts and save their work. Students can share their CodingBat accounts with their teachers. It facilitates teachers to view their student's advancement. Only recently, CodingBat has released a problem authoring feature where users can add their own programming problems to the existing set. Practice-It is another online tool developed by Mark Stepp to practice Java programs. It contains practice problems from the author's book – Building Java Programs [19]. There are also some tools to conduct online programing examinations. Roberts and Verbyla propose one such tool to securely take programming exams for Java [10].

At the time when I decided to develop Javabrat, all of the above mentioned systems had a closed set of questions. None of these applications provide an interface for the users to input their own programming problems. As mentioned above, only recently (in December 2009) some applications such as CodingBat have moved in this direction.

None of the current available tools help users to practice Scala programming problems. A website – Simply Scala [5] allows users to test different Scala language constructs, but it does not contain programming problems.

2.2. Automatic Grading of Student's Programming Assignments.

There are number of tools used by instructors or educational institutions that facilitate the automatic grading of programming assignments. Some of the examples are Marmoset [18] and WebCAT [14]. Marmoset is a framework that provides an automatic submission and testing system [12] . Marmoset evaluates students submission using public tests that are visible to students and using secret tests that are not visible to students. This system also collects a snapshot of a students' work every time they save a file. This data is useful for the researchers studying how students learn programming languages. [13] discusses the benefits of Marmoset for students, instructors and researchers

Along with the automatic grading, WebCAT uses a different approach to grade the student's code. It supports grading of the assignments where students are asked to submit their own test cases. The WebCAT site describes WebCAT as “an advanced automated grading system that can grade students on how well they test their own code”. It is a language independent tool that focuses on test driven development. Edward proposes a new approach of teaching software testing using WebCAT in [15].

In addition to the above mentioned grading tools, there are several Learning Management Tools available. As per Suleman “Sakai and Moodle are among the most popular learning management systems today” [17]. Sakai is a course based portal and provides all the standard LMS features.

Moodle is a online learning and course management tool. Moodle stands for “Modular Object-Oriented Dynamic Learning Environment”. Moodle is one of the most popular and extensively used LMS today. At the time of this writing, Moodle has around 35 million users and 50,000 registered sites. More details about Moodle can be found in [4]. In this project, I have developed a plugin for Moodle that facilitates the automatic grading of Java programming assignments.

Here are the reasons why I chose Moodle for this project :

- Moodle is open source, stable and it has a good community support.
- The Moodle installation is very straightforward and it is easy to use.
- Moodle is developed on modular architecture. Hence it is easy to extend or customize for your own needs.

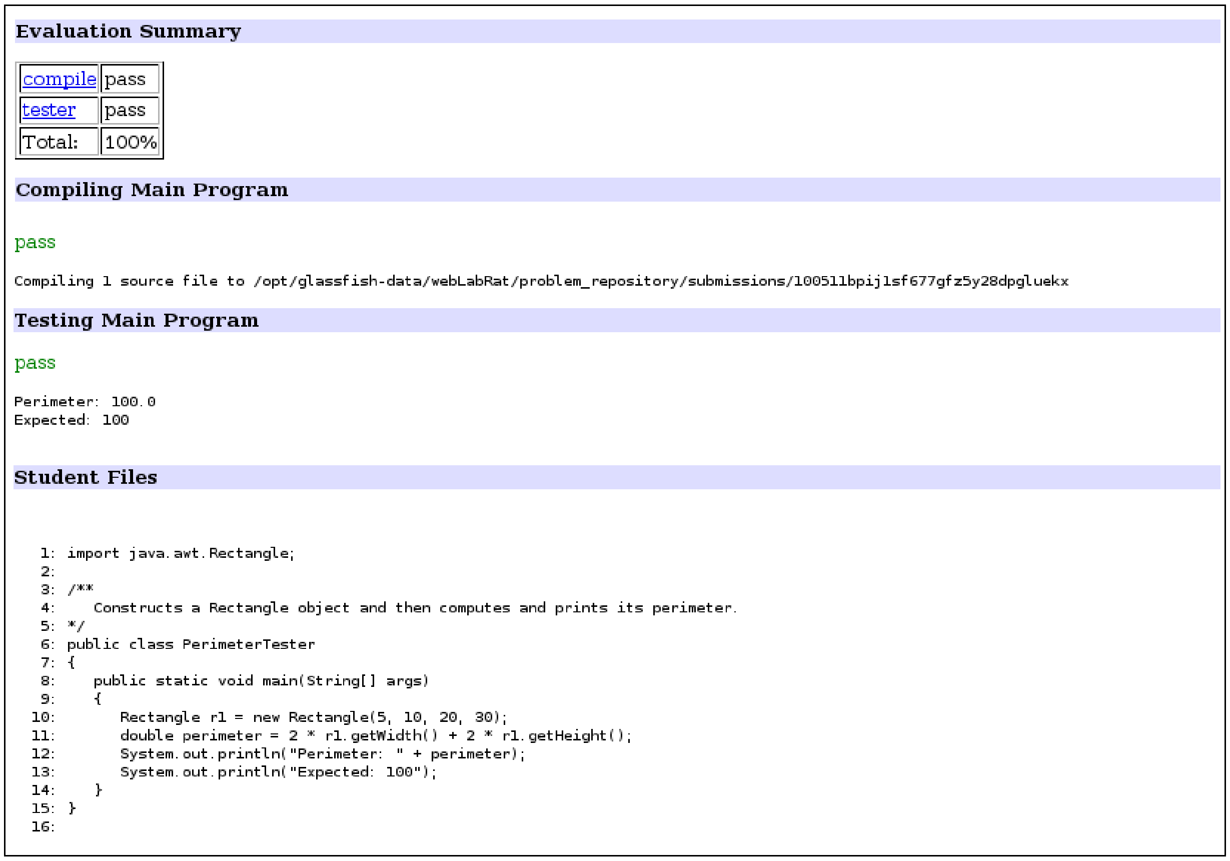
As discussed in [7], the features provided by Moodle are sufficient for a standard course. But Moodle can be further improved to provide more advanced features. Some of the possible improvements are discussed in [8]. One of the feature that Moodle lacks is the ability to generate and grade programming questions. Moodle has ten standard built-in question types. But none of these standard question types provides a way to assess programming problems.

2.3. Labrat

Labrat automates the assessment of Java programming problems. It was authored by Dr. Horstmann to support his text book [20]. At an abstract level, Labrat compiles the user's program (or solution) and runs the program against test cases. Based upon the compilation output and test cases results, Labrat generates a report that provides feedback to the user. Labrat has two modes of operations – code completion and full program grading. In the code completion mode, users are provided with a partial solution and they are supposed to complete it in order to solve the problem. The full program mode requires users to provide the complete solution for the problem. Labrat needs the following inputs to generate the report :

1. mode of the operation (code completion or full program)
2. path of the directory containing user's answer files
3. path of the directory containing the metadata of the problem
4. path of the directory containing the grader metadata such as test case specifications

Labrat generates the evaluation report in two forms – HTML and text. A typical HTML Labrat report consists of a evaluation summary, an compilation output, an tester output and the student's files. Figure 1 shows a sample Labrat report. Our web application Javabrat internally calls Labrat to assess Java programming problems.



Evaluation Summary

compile	pass
tester	pass
Total:	100%

Compiling Main Program

pass

```
Compiling 1 source file to /opt/glassfish-data/webLabRat/problem_repository/submissions/100511bpij1sf677gfz5y28dpgluekx
```

Testing Main Program

pass

```
Perimeter: 100.0
Expected: 100
```

Student Files

```
import java.awt.Rectangle;

/**
   Constructs a Rectangle object and then computes and prints its perimeter.
*/
public class PerimeterTester
{
   public static void main(String[] args)
   {
      Rectangle r1 = new Rectangle(5, 10, 20, 30);
      double perimeter = 2 * r1.getWidth() + 2 * r1.getHeight();
      System.out.println("Perimeter: " + perimeter);
      System.out.println("Expected: 100");
   }
}

```

Figure 1: Labrat report

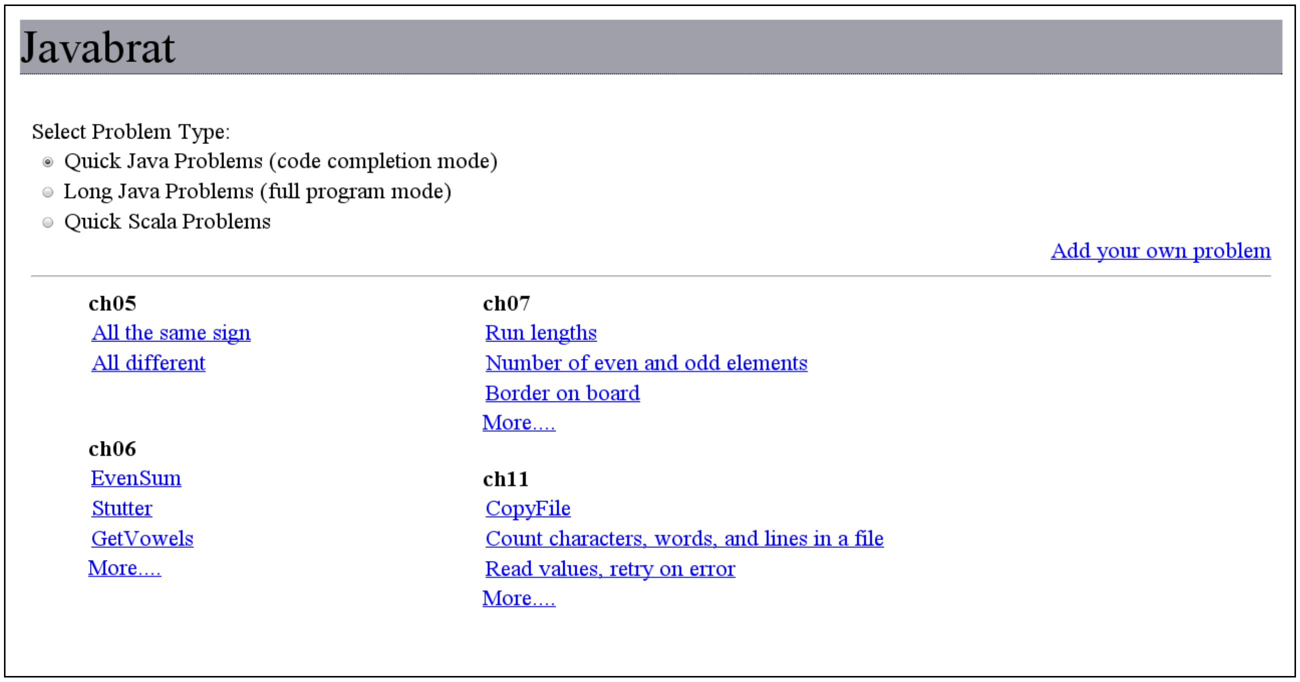
3. Javabrat

Javabrat provides an web interface for our system. It has been developed using Java Server Faces. Javabrat ties together different parts of the system and provides an integrated view. At its most basic level, Javabrat displays the programming problem, accepts user solution and shows the feedback to the user. The following section describes the Javabrat work flow.

3.1. Javabrat Work Flow:

3.1.1. View Problem

When a user opens the Javabrat welcome page , he can see a list of problems categorized under different sections. The user can choose to view Quick Java problems (code completion mode), Long Java Problems (full program mode) or Quick Scala Problems (Scala code completion mode). By default Javabrat shows Java Quick Problems. Figure 2 shows the list of problems for the default selection. When the user selects Long Java problems or Quick Scala problems, Javabrat fetches the corresponding problem list and refreshes the page with the new list. The problems are further divided into chapters. They can also be divided based upon their complexity level, language constructs or any other criteria. The following figure shows some of the Javabrat questions from Dr. Horstmann's problem repository. He has divided the Java problems based upon his book chapters.


Javabrat

Select Problem Type:
- [✓ Quick Java Problems (code completion mode)]
- Long Java Problems (full program mode)
- Quick Scala Problems

Add your own problem

ch05
All the same sign
All different

ch06
EvenSum
Stutter
GetVowels
More....

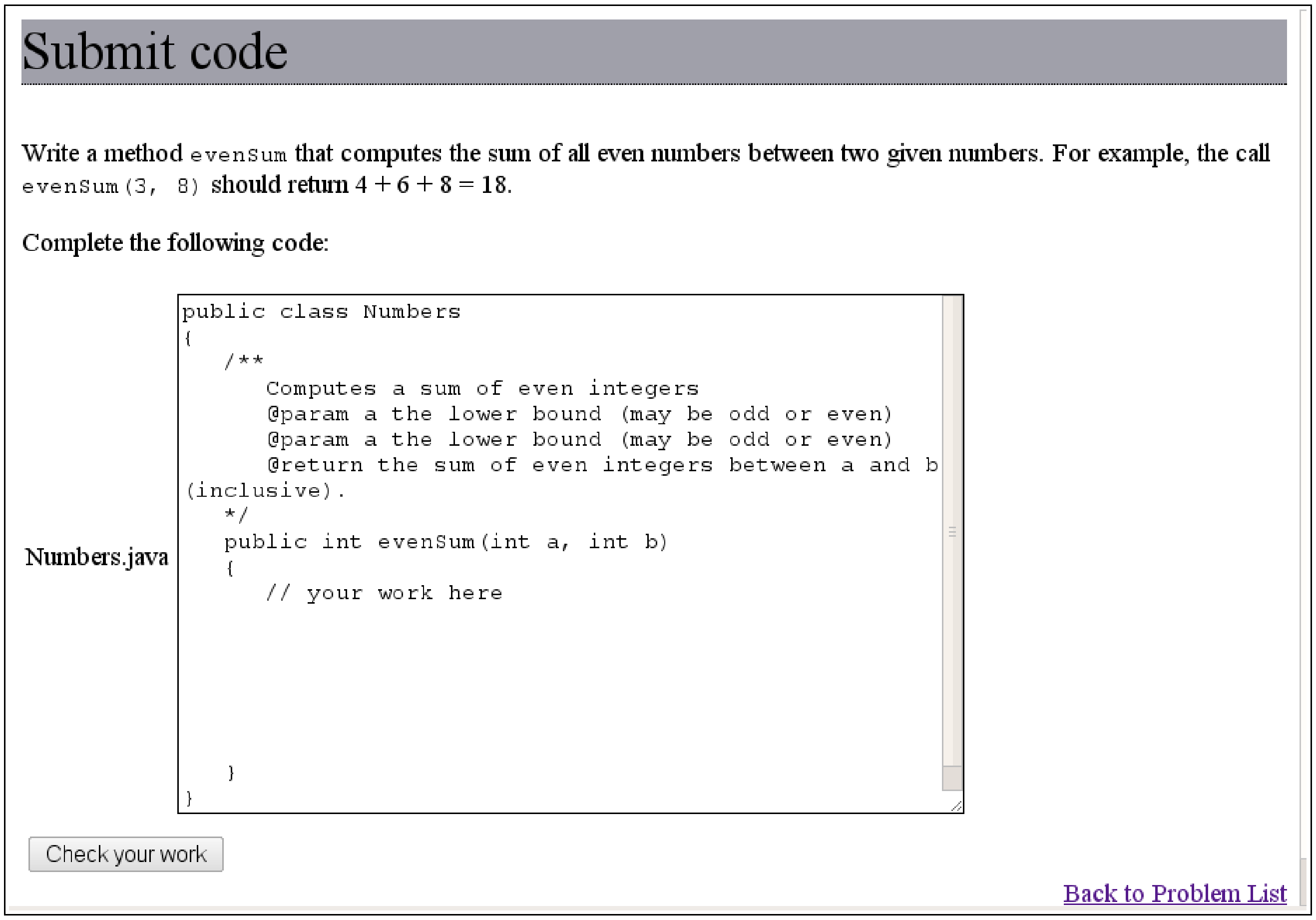
ch07
Run lengths
Number of even and odd elements
Border on board
More....

ch11
CopyFile
Count characters, words, and lines in a file
Read values, retry on error
More....

Figure 2: Javabrat snapshot

On the front page (the one shown in Figure 2), at the most three problems are shown for each chapter. If a user wants to see more problems from a particular chapter, he can click on the "More..." link. The problem repository author can determine which three problems should be displayed on the front page. He can mention his favorite problems in the favorite.properties file kept under each chapter in the problem repository.

When a user clicks on a particular problem link, Javabrat displays the page where user can actually attempt the problem. Figure 3 shows the page of the "Even Sum" problem. The problem page provides a problem description and a textarea to accept the user's code. If the problem is one of the quick types, then the text area also contains the code provided as partial solution. The user is supposed to fill in his solution in this text area and click "Check your work" to view the Javabrat feedback.



Submit code

Write a method `evenSum` that computes the sum of all even numbers between two given numbers. For example, the call `evenSum(3, 8)` should return 4 + 6 + 8 = 18.

Complete the following code:

Numbers.java

```
public class Numbers
{
   /**
      Computes a sum of even integers
      @param a the lower bound (may be odd or even)
      @param a the lower bound (may be odd or even)
      @return the sum of even integers between a and b
(inclusive).
   */
   public int evenSum(int a, int b)
   {
      // your work here






   }
}
```

Figure 3: Javabrat problem page

3.1.2. Check Problem

When a user completes his answer, he can get it evaluated by clicking “Check your work”. At the server side, Javabrat bundles the user's solution and other meta information into a temperately created directory. If the problem is any of the Java types, it asks Labrat to grade the problem. If the problem is of Scala type, Javabrat asks Scalabrat to assess the problem. Once the assessment is done, Javabrat picks up the HTML report file generated by Labrat or Scalabrat and sends it back as the response.

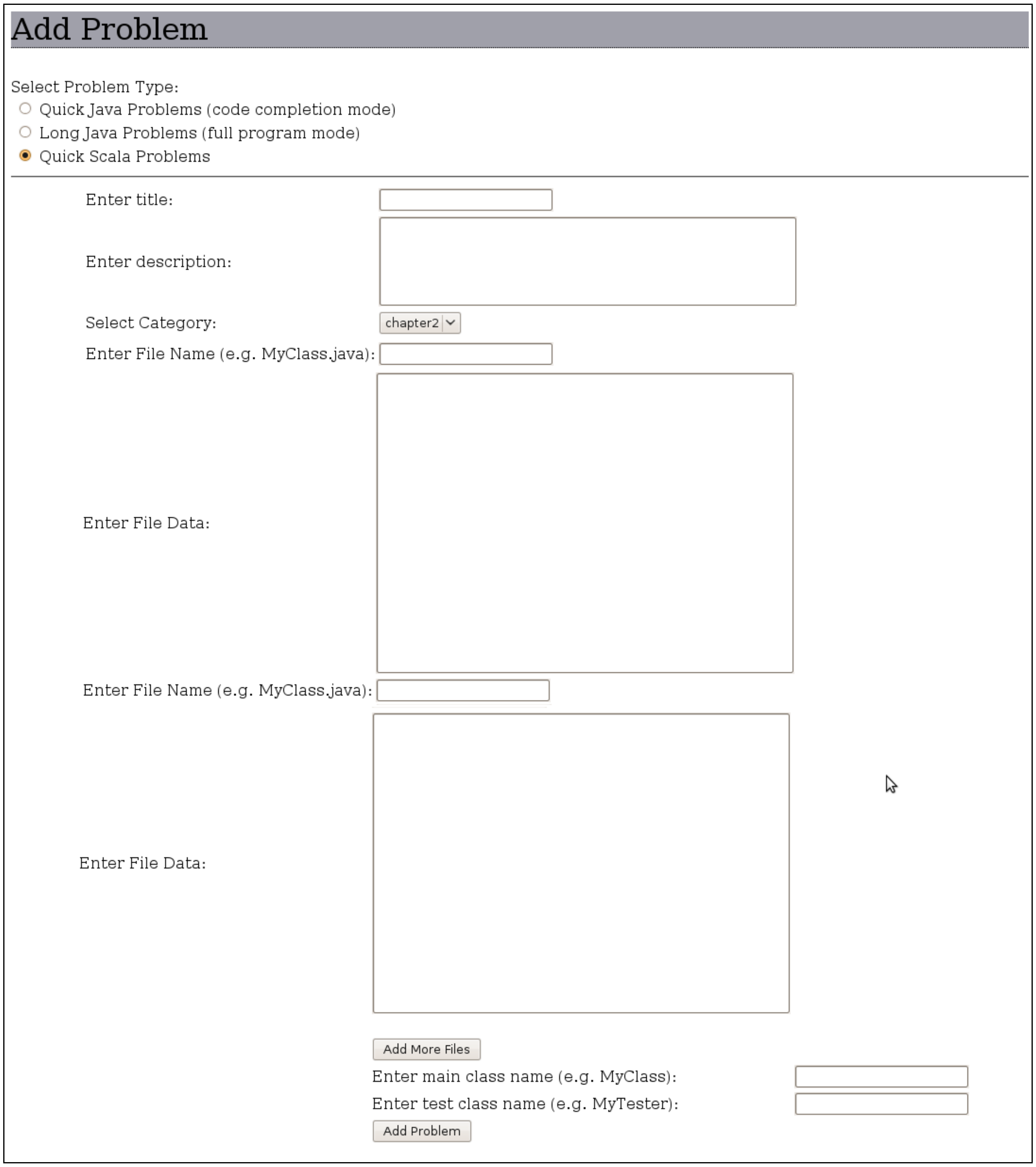
3.1.3. Add Problem

Users can contribute their own problems to the Javabrat problem repository. Once added the new problems are treated as normal Javabrat problems and visible to all other users. This feature is especially useful for instructors who want to create their own programming problems and make them available to the students for practice. An instructor can add his own problems to the Javabrat repository and ask students to practice them.

When the user clicks on the “Add you own problem” link, a blank problem is created. The user can select the problem type he wants to add. By default the problem to be added is considered as a quick Java problem. Figure 4 shows a Javabrat page to add a Scala problem.

The problem author has to fill in the problem details in the provided form and click on the “Add Problem” button. I have summarized these form fields for all the three types of problems as

1. **Problem title:** Title shown in the problem list
2. **Problem description:** A description that states and explains the problem and also provides guideline if any to solve the problem.
3. **Problem category:** The user can put his problem in one of the existing categories or can choose to put it in the “Other” category.
4. **Problem contents (only for quick problem type):** The problem author can provide a partial solution to the problem when it is one of the quick problem types
5. **Tests:** The author should provide the test cases to evaluate the accuracy of the user's solution. The test cases format differs for the Java and Scala problems. Section 5.1.3 describes the test case format for the Scala problems



Add Problem

Select Problem Type:

- ○ Quick Java Problems (code completion mode)
- ○ Long Java Problems (full program mode)
- ● Quick Scala Problems

Enter title:	
Enter description:	
Select Category:	chapter2
Enter File Name (e.g. MyClass.java):	
Enter File Data:	
Enter File Name (e.g. MyClass.java):	
Enter File Data:	
Enter main class name (e.g. MyClass):	
Enter test class name (e.g. MyTester):	

Figure 4: Javabrat Add Problem page

3.2. Implementation Details

Javabrat has been built on the already existing web interface developed by Dr. Horstmann. It was then called WebLabrat. Over the time WebLabrat has been evolved to accommodate the new features. I have contributed the following modifications or features to this web interface.

3.2.1. Migration to JSF 2.0

WebLabrat was developed using JSF 1.2. I converted the application to JSF 2.0. I have used the following features of JSF 2.0

3.2.1.1 Templates with Facelets

With Facelets templates, one can create a common layout for a group of pages. It provides a common place to apply styles and decide on the look and feel of the application. I have divided the layout into two logical areas – header and contents. I have used this composition for all the pages of Javabrat. For example, the following code is from the page that shows the problem details to the user. The page uses a composition defined in master_layout and inserts the actual content files in the corresponding logical group. In the listing 3-1 `ui` defines the Facelets namespace.

```
<ui:composition template="/templates/master_layout.xhtml">
  <ui:define name="header">
    <ui:include src="/sections/codecomp/submitCode/header.xhtml"/>
  </ui:define>
  <ui:define name="contents">
    <ui:include src="/sections/codecomp/submitCode/contents.xhtml"/>
  </ui:define>
</ui:composition>
```

Listing 3-1: Template with Facelets

3.2.1.2 Easier Navigation

The prior version of JSF (JSF 1.2) requires navigation rules to be specified in the configuration file faces-config.xml. But with JSF 2.0, navigation is easier and does not require to specify the navigation rules in the configuration file. Instead, it implicitly navigates to the page by appending `.xhtml` to the return value of a method or to the action attribute of a UI component.

3.2.1.3 Managend Bean Annotation

JSF 2.0 introduces annotations to use with the Java objects. It eliminates the need to configure managed beans in the config file. When a class is annotated with the `@ManagedBean` annotation, it becomes a manged bean at runtime. The scope of the managed bean is specified using the `@RequestScoped`, `@SessionScoped` or `@ApplicationScoped` annotations. I have used this feature to define all the managed beans.

3.2.2. Integration with Scala Problems

I have extended Javabrat to support Scala problems. To accomplish this I have created a `ScalaCodeCompletionProblem` class that extends from Javabrat's `Problem` class. This class is responsible for loading a Scala problem. When a user clicks on any Scala problem, the `load` method of `ScalaCodeCompletionProblem` class is called. The `load` method reads all the meta data of the problem stored in the problem repository and populates the problem object. This class is also responsible to call the Scalabrat application to grade Scala problems. It overloads the `commandArgs` method of the `Problem` class to bundle all the arguments required to run the Scala grader.

3.2.3. Integrated View of All the Problems

Initially, to view a problem through WebLabrat, the user has to provide the problem identifier with the request URL as a query parameter. For example, to view the Even Sum problem shown in the Figure 3, a user has to request following URL specifying the problem identifier as id parameter.

http://helsinki.cs.sjsu.edu:8080/javabrat/check.jsf?id=ch06/c06_exp_6_102

It was inconvenient for the users who did not know the exact problem identifier. To eliminate this problem, I added a functionality where the user can choose the problem type and see the list of all the problems under a specific category. They can choose to attempt any problem from the list. This page is shown in Figure 2.

To develop this functionality, I have used a session scoped bean called `problemList.` This bean reads all the problems of the selected type from the problem repository. Every problem directory contains a properties file specifying the problem meta data such as name and type of the problem. The `ReadProblems` method of `problemList` bean peeks into all the problem directories and reads the properties file. It populates the `problemList` bean with the problem details. To display the problem list, I have used `dataTable` tag of JSF. The following code snippet shows the `dataTable` code that displays the problem list.

```
<h:dataTable value="#{problemList.chapters}" var="chapter"
    style="margin-left:1cm;margin-right:7cm">
  <h:column>
    <h:outputText value="Prablems from #{chapter.level}"
         style="font-weight:bold"/>
    <hr />
    <h:dataTable value="#{chapter.problems}" var="problem">
      <h:column >
        <h:link value="#{problem.title}"
            outcome="../check.jsf?id=#{problem.id}"/>
      </h:column>
    </h:dataTable>
  </h:column>
</h:dataTable>
```

Listing 3-2: JSF data table to show the problem list

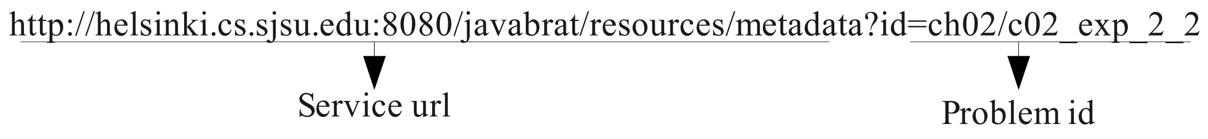
The outer `datatable` tag iterates through `chapters` object of `problemList` bean. The inner `dataTable` tag iterates through the list of problems stored in the each `chapter` object and displays each problem as a link. When the user clicks on this link, `check.jsf` page is requested with the current problem's id. The `check.jsf` page displays the question details of the requested problem.

3.3. Javabrat as a Web Service

I have exposed Javabrat's Java problem assessment service as a web service using the REST architecture. I have also created a separate web service to fetch the problem details from the Javabrat problem repository. My Moodle plugin uses these services to generate the question and to grade the student's response. These web services can be used with any LMS by parsing their outcome. The following sections explain these two services in detail:

3.3.1. Metadata

This service serves the problem metadata to the client. While requesting this service, the client should send the id of a problem for which he wants to receive information. This problem information (metadata) contains all the information required to generate a Javabrat type problem. It includes the problem title, the problem description, the problem type and a few other fields. Here is an example of the metadata service URL:



http://helsinki.cs.sjsu.edu:8080/javabrat/resources/metadata?id=ch02/c02_exp_2_2

The metadata service returns the problem metadata in XML format. An example response for the above URL is shown below:

```
<?xml version='1.0'?>
<question>
  <name>PerimeterTester</name>
  <type>fp</type>
    <description>
      <p>Write a PerimeterTester program that constructs a Rectangle object and
          then computes and prints its perimeter. Use the getWidth and getHeight
          methods. Also print the expected answer.</p>
    </description>
  <filenames>
    <file>PerimeterTester.java</file>
    <data></data>
  </filenames>
  <okToAddFiles>false</okToAddFiles>
</question>
```

Listing 3-3: Response from Metadata service

3.3.2. Check

This web service accepts the user's solution and returns the Javabrat evaluation report. The call to the check service should be a POST request as it involves sending the user's solution to the web service. The check service should be invoked with the following parameters:

1. id= id of the attempted problem
2. name= name of the Java file that contains the solution
3. data=contents of the solution file

There can be multiple name and data parameters based on the number of files required for the solution. The Check service stores all the user data in a temporary directory and asks the Javabrat's `check` method to generate the report in HTML format. It then sends back this report to the caller. The following section describes the implementation details for above two web services

3.3.3. Implementation Details

3.3.3.1 REST

REST stands for "Representational State Transfer"and it is an architecture to develop web services. I chose REST because it is:

- **light-weight**: The data that is being transferred to and from our web service is simple and small. REST provides an easy way to transfer the data between the client and the server by using HTTP protocol and it is suitable for the exchange of short data.
- **requires only a thin client**: A REST service is invoked with an HTTP request. It is very simple to construct such a request and does not require any client side library.
- **easy to build:** REST is easy to understand and develop. Java provides support for writing RESTful services.

3.3.3.2 JAX-RS

To implement the REST services, I have used JAX-RS. JAX-RS provides APIs to implement RESTful web services in Java. With JAX-RS standards and annotations, a simple Java object can be exposed as a web service. The following code snippet shows the metadata web service:

```
@Path(value="metadata")
public class Metadata {
  …
  …
  @GET
  @Produces("text/plain")
  public String getMessage(@Context ServletContext context,
        @QueryParam("id") String id, @Context HttpServletRequest request) {
    String repositoryPath =
    context.getInitParameter("com.horstmann.labrat.repositoryPath");
```

```
        Problem problem = Submission.loadProblem(id, repositoryPath);
        String servername = request.getServerName();
        String contextName = request.getContextPath();
        StringBuffer sb = request.getRequestURL();
        …
        …
        return getXml(problem, url);
      }
  }
```

Listing 3-4:Metadata service

As shown in the above code snippet, the class `Metadata` is a simple Java class that acts as a web resource. The method `getMessage` loads the specified problem and passes this problem to the `getXml` method. The `getXml` method extracts the problem metadata from the problem object and generates a xml document as shown in the listing 3-4.

Here is the description of some of the annotations used in the above code:

- `@Path`: a relative URI for which the `Check` class will serve the request
- `@GET`: declares that the `getMessage` method will serve the HTTP GET requests
- `@Produces`: mime type of the response created by this method

I have used a similar approach to develop the check service.

3.4. Problem Repository Structure

Each problem in the problem repository is represented by a directory in the local file system. The two grader applications, Scalabrat and Labrat work on the problems stored in the files with a standard directory structure. A problem contributor should adhere to this problem structure while adding a new problem. In a whole, the Javabrat problem repository also follows a directory structure to arrange Java and Scala problems. At the topmost level, I have divided the problems in the following three categories:

1. Quick Java Problems (code completion mode – cc)
   contains the Java problems where partial solution is provided to the user.
2. Long Java Problems (full program mode – fp)
   contains the Java problems where user has to solve the problem from scratch.
3. Quick the Scala Problems (scala code completion mode - scala_cc)
   contains Scala problems with their partial solutions.

At the level below this topmost categorization, the problems are grouped according to chapters. At this level it is also possible to group the problems based on their type or any other criteria. Here is the screen-shot of a sample problem repository:

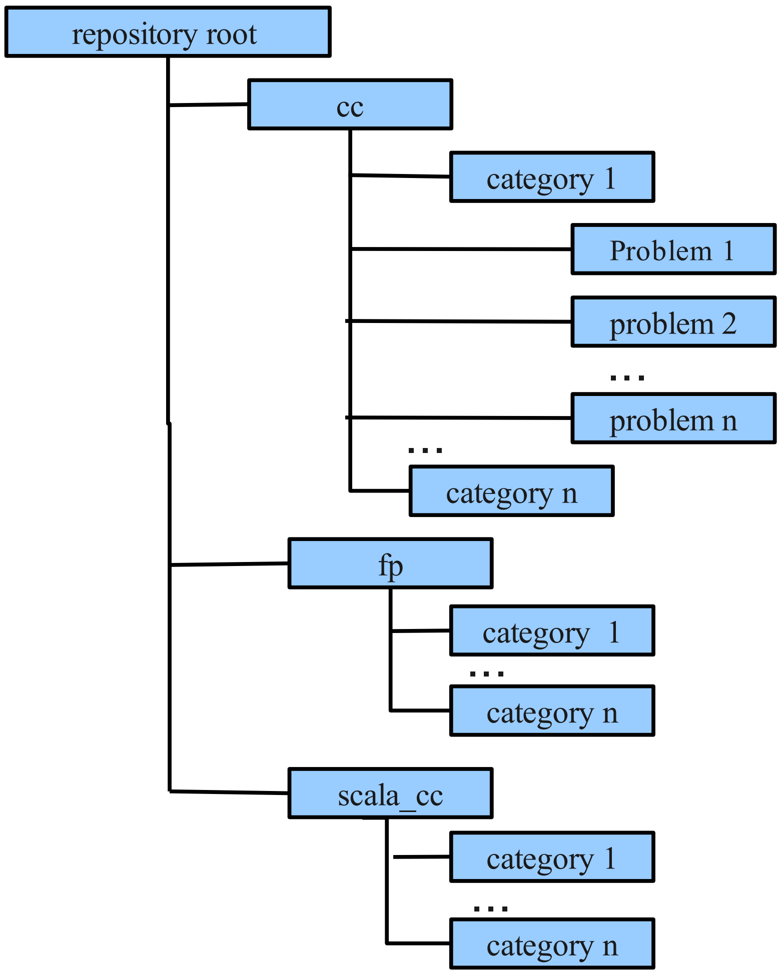


Figure 5: Problem repository structure

4. Moodle Plugin

This section describes in detail the Moodle plugin that I have developed as a part of this project. Our system is comprised of the Moodle plugin, Javabrat and other sub modules. It provides a setup for the instructors, students and language learners for taking or assigning programming questions. The following diagram shows the high level overview of our system .

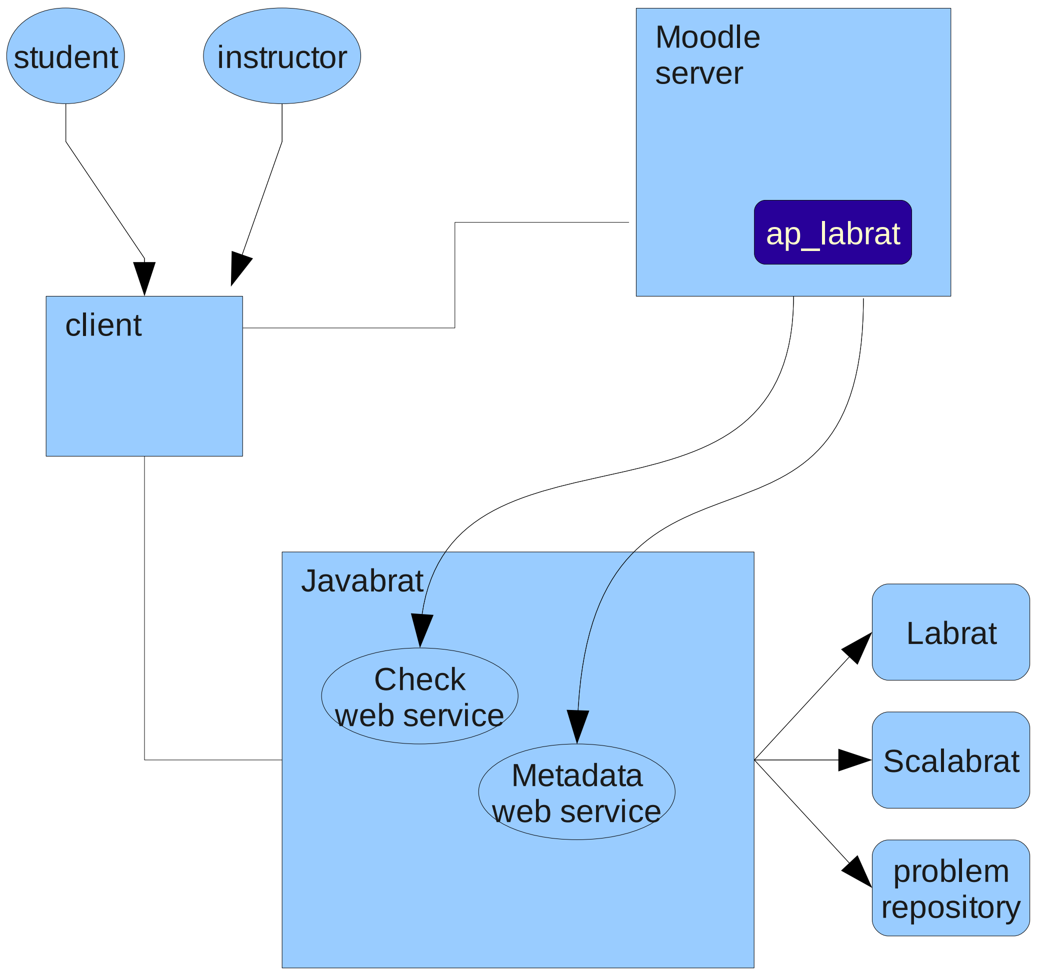


Figure 6: System Overview

4.1. Moodle Workflow

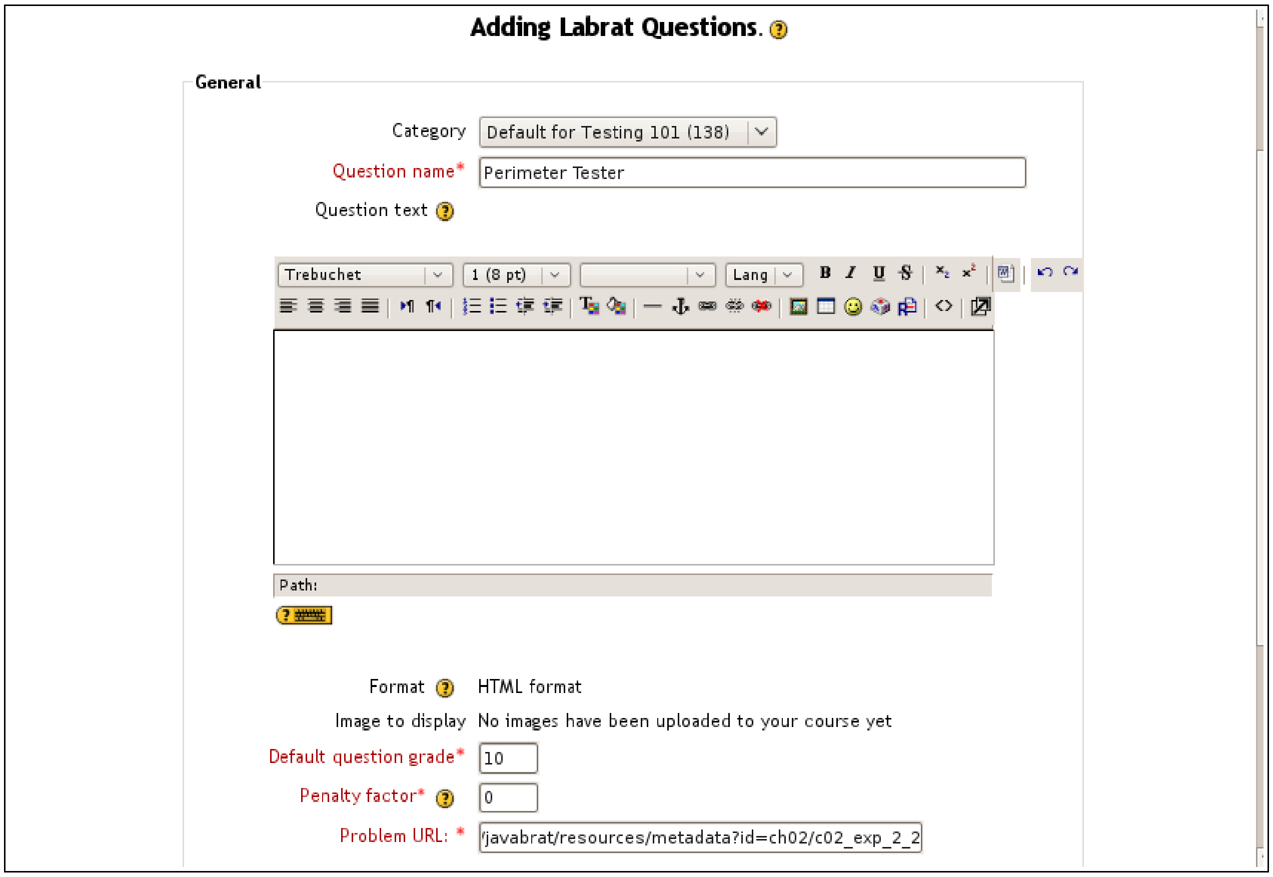
As shown in the Figure 6, students and instructors connect to Moodle through a web browser. Our plugin ap_labrat resides in Moodle. This plugin allows the instructors to create Java programming questions. The instructors can create a quiz comprised of these questions and assign it as a homework to the students. The instructors create a programming question in Moodle by providing a question URL. The question URL has been discussed in more detail in Section 3.3. The actual questions reside in the problem repository and the assessment logic resides in Javabrat. When a student logs in to Moodle to take the quiz, ap_labrat fetches the question details such as the problem description and the names of the Java classes from Javabrat and formats the question. To accomplish this task, ap_labrat calls the metadata web service. When a student submits his answers, ap_labrat asks Javabrat to evaluate the student's submission. Javabrat serves this request with its check web service. The check service assesses the student's response and returns a report based on the assessment to Moodle. Moodle stores this report in its database. The Instructor or grader can later view this report while generating grades for the students. In the remaining sections of this report, instead of using two separate terms for students and/or language learners, I have used a single and more generic term – users.

The Moodle architecture is based on a modular platform and is highly customizable. Moodle provides many features that can be extended or customized by writing a plugin. For example, the report generation can be customized by developing a report plugin. As mentioned in the introduction section, we wanted Moodle to handle programming assignments and also grade them automatically. After studying the available plugin types for Moodle, I decided to create a question type plugin.

In Moodle, questions are supposed to be used with quizzes. Questions are stored in the question bank and can be reused. A quiz can contain any number of questions. The quiz can be assigned to students as a homework assignment. The following section describes question type plugin that I have developed.

4.2. Labrat Type Question

Our plugin introduces a new type of question in Moodle – Labrat questions. Labrat questions are similar to the Javabrat questions. In fact, as discussed in Section 3.3, our plugin make use of Javabrat web service to generate the question. The following snapshot shows a Moodle page for creating a Labrat type question.



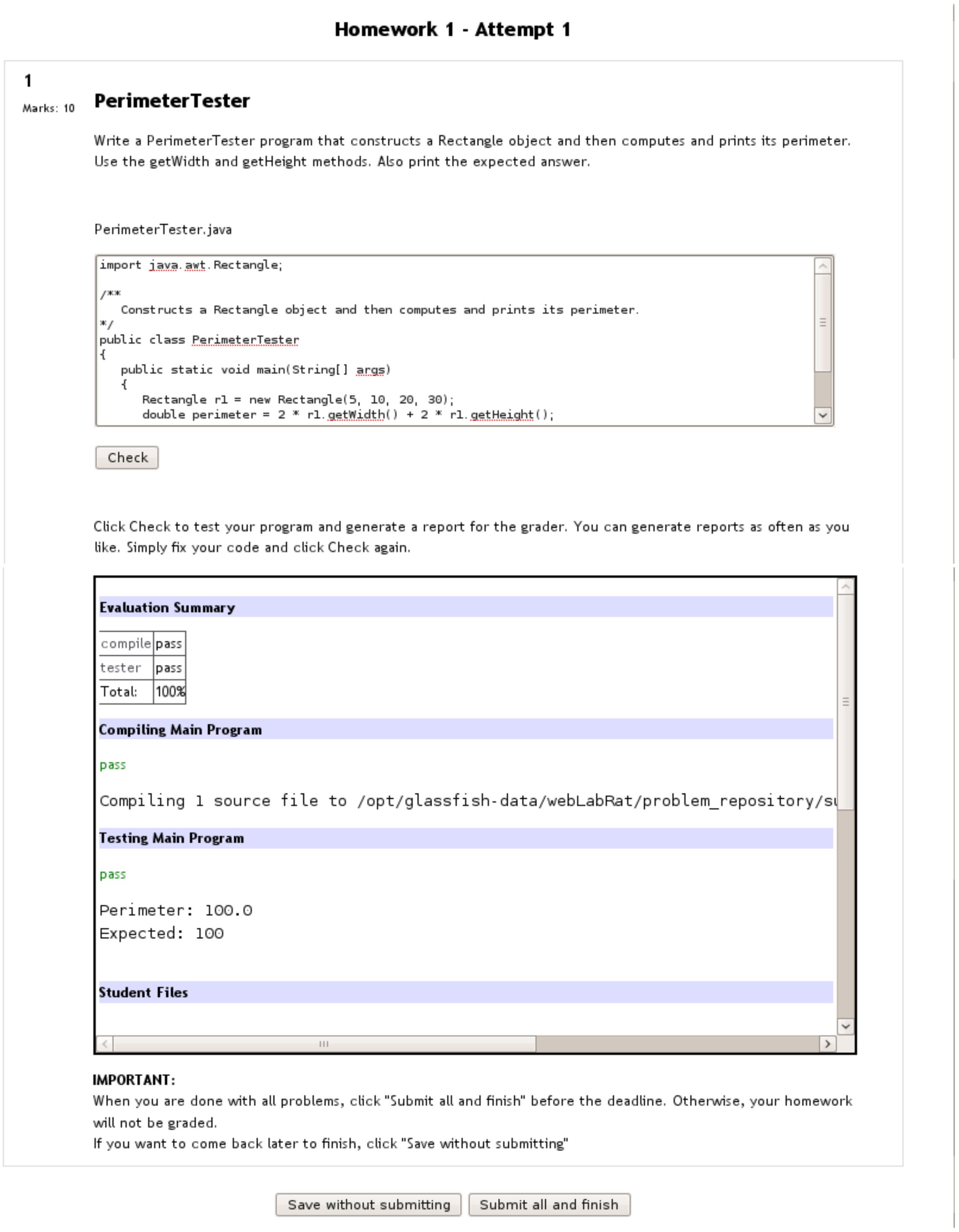
Adding Labrat Questions.

General

Category	Default for Testing 101 (138)
Question name*	Perimeter Tester
Question text	
Format	HTML format
Image to display	No images have been uploaded to your course yet
Default question grade*	10
Penalty factor*	0
Problem URL: *	/javabrat/resources/metadata?id=ch02/c02_exp_2_2

Figure 7: Creating Labrat type question

As shown in the above figure, the instructor should provide a problem URL while creating a Labrat type question. This problem URL should invoke the metadata web service. Moodle can create only those Labrat questions which are already present in the Javabrat problem repository. If the instructor wants to create his own question, then he should first add the question details to the repository and then provide it's identifier as the value of the id parameter to the problem URL. After creating a Labrat question, the instructor can use this question in a quiz. Once the quiz is set, the students can view the questions by attempting the quiz. The following figure shows the student's view of a Labrat question. When a student clicks Check, the evaluation report is displayed on the same page.



Homework 1 - Attempt 1

1

Marks: 10

PerimeterTester

Write a PerimeterTester program that constructs a Rectangle object and then computes and prints its perimeter. Use the getWidth and getHeight methods. Also print the expected answer.

PerimeterTester.java

```
import java.awt.Rectangle;

/**
   Constructs a Rectangle object and then computes and prints its perimeter.
*/
public class PerimeterTester
{
   public static void main(String[] args)
   {
      Rectangle r1 = new Rectangle(5, 10, 20, 30);
      double perimeter = 2 * r1.getWidth() + 2 * r1.getHeight();
```

Click Check to test your program and generate a report for the grader. You can generate reports as often as you like. Simply fix your code and click Check again.

Evaluation Summary

compile	pass
tester	pass
Total:	100%

Compiling Main Program

pass

```
Compiling 1 source file to /opt/glassfish-data/webLabRat/problem_repository/su
```

Testing Main Program

pass

```
Perimeter: 100.0
Expected: 100
```

Student Files

IMPORTANT:
When you are done with all problems, click "Submit all and finish" before the deadline. Otherwise, your homework will not be graded.
If you want to come back later to finish, click "Save without submitting"

Figure 8: Moodle Labrat question

The students are supposed to generate the Javabrat report before they submit their answers. Moodle stores this Javabrat report into the database. To generate the Javabrat report, the student should press the "Check" button. The Javabrat report is displayed in the text panel provided below the "Check" button. Students can generate the report as many times as they want. When students are happy with their reports for all the questions, they can press the "Submit all and finish" button.

4.3. Implementation Details

Moodle is open source software and it is written in PHP. It can support SQL type databases (I used Moodle with PostgreSQL). Moodle provides a template [9] to develop a question type plugin. I have built the Labrat question type based on this plugin. The following sections describe it's implementation details.

4.3.1. Database Table

To store the question details, the Labrat question plugin creates a table `question_ap_labrat` in the Moodle database. This table contains following columns:

- id: primary key of the table
- question id: foreign key of the table that corresponds to the primary key of the `question` table of the Moolde database.
- url: to store the problem URL.

Moodle provides a abstraction layer to interact with the underlying database. This layer is called XMLDB. It facilitates the Moodle development to be database independent. With XMLDB, the developers create their database structures in a XML format irrespective of the actual database being used. This file should be named as `install.xml` and should be kept under the `plugin_name/db` directory. To create the `question_ap_labrat` table I have used the XMLDB specifications.

4.3.2. Question Editing Form

The question editing form is shown in the Figure 7. It displays the question details to be filled or edited by the problem author. This form extends from Moodle's `question_edit_form`. Moodle's standard form class provides easy APIs to add or remove HTML elements. While designing the form shown in the Figure 7, I have used these APIs to create the HTML elements. Listing 4-1 shows a code snippet from `question_edit_ap_labrat_form.php` file

```
class question_edit_ap_labrat_form extends question_edit_form {
  function definition_inner(&$mform) {
    $mform->removeElement('generalfeedback');
    $mform->addElement('text', 'url', 'Problem URL: ','size=40');
    $mform->addRule('url', null, 'required', null, 'client');
    …
    }
    …
  }
```

Listing 4-1:Labrat question form

As shown in the above code snippet, to add a text field for the problem URL, I have used the `addElement` method of `mForm`.

4.3.3. Question Type Class

Once the question form is developed, I had to write a question type class that can display this form and handle the grading of the solution. Our question type class `ap_labrat_qtype` extends from Moodle's `default_questiontype` class. The question type class is written in `questiontype.php` file. This class is responsible for the following tasks

4.3.3.1 Displaying Question to Students

To display the question, the question type class first retrieves the problem URL from the Moodle database. Then it calls the Javabrat metadata service and collects the problem details. Here is the code snippet that performs the web service call and extracts the problem details

```
$url = urlencode($question->options->url);
$xml = simplexml_load_file($url);
$title =  $xml->xpath('//name');
$type = $xml->xpath('//type');
$description = $xml->xpath('//description');
$filenames = $xml->xpath('//filenames');
```

Listing 4-2:Call to Metadata service

After extracting the problem metadata, PHP code generates the HTML page shown in the Figure 8.

4.3.3.2 Handling Student's Response

When a student clicks on the “Check” button, the question type class, bundles the student's response in a post request and asks the check service to generate it's evaluation report. I have used the PHP `fopen` function to invoke the check web service.

```
$params = array('http' => array('method' => 'POST', 'content' => $data ));
  $ctx = stream_context_create($params);
  $fp = fopen($url, 'r', false, $ctx);
```

Listing 4-3:Call to Check Service

In the above code, `$param` is an associative array that contains the specifications for the context stream. This array contains another array specifying the following HTTP header parameters:

- method: contains the type of HTTP request method. For the above request I have used POST method.
- content: contains the data to be sent with this request. `$data` is an array containing student's answers as name value pairs

The function `stream_context_create` creates a stream for the HTTP protocol as specified in `$params` array.

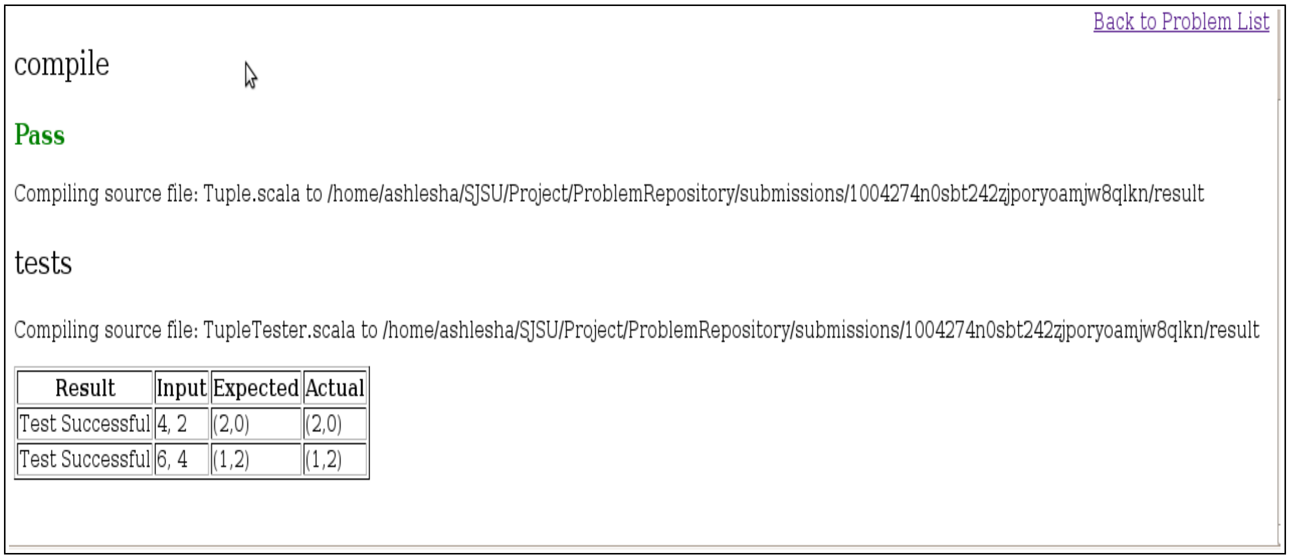
To call the check service I have used the `fopen` function. Here are the parameters passed to the `fopen` function:

- $url: url of the Javabrat's Check service
- r: open url with read mode
- false: do not search this url in the include_path
- $ctx: context stream containing the HTTP request parameters

The return value, `$fp` points to the file containing the evaluation report generated by Javabrat's Check service. The question type class further extracts this report and displays the report as shown in Figure 8.

5. Scalabrat

The primary responsibility of Scalabrat is to assess the Scala programing problems. Scalabrat has been developed in the Java language. I have integrated Scalabrat with the Javabrat web interface. Javabrat internally calls Scalabrat when a problem to be assessed is a Scala problem. Scalabrat can also be used as a standalone application. It compiles the user's solution with the Scala compiler. If the compilation succeeds, Scalabrat runs a few predefined test cases on the user's code to assess it's correctness. Scalabrat generates a report based upon the compiler output and the test cases results. Figure 9 shows a snapshot of a sample evaluation report generated by Scalabrat


Back to Problem List

compile

Pass

Compiling source file: Tuple.scala to /home/ashlesha/SJSU/Project/ProblemRepository/submissions/1004274n0sbt242zjporyoamjw8qlkn/result

tests

Compiling source file: TupleTester.scala to /home/ashlesha/SJSU/Project/ProblemRepository/submissions/1004274n0sbt242zjporyoamjw8qlkn/result

Result	Input	Expected	Actual
Test Successful	4, 2	(2,0)	(2,0)
Test Successful	6, 4	(1,2)	(1,2)

Figure 9 : Scalabrat Report

5.1. Technical Details of Scalabrat

5.1.1. Ant Listener Mechanism

Report generation was one of the important part of the Scalabrat development. As shown in the Figure 9, a Scalabrat report contains the following segments:

- the compilation result
- the messages generated by the Scala compiler
- the results of the test cases

I have used Ant to compile the user's code and to run the test cases. I have specified these tasks as the targets in the Ant's build file. The reason I chose Ant was that it's listener mechanism provides a easy way to capture the messages generated during the build process. The build process here comprises of the compiling user's code and running the test cases.

I have written an Ant build file `codecomp.xml` that contains the Ant targets for compiling and running the submitted answer. Scalabrat copies this build file into the directory where the student's submission is kept and fires the Ant build. Scalabrat invokes the build using the API's provided by the Ant's `Project` class. Here is a snippet of the `codecomp.xml` showing some of the important targets:

```
<target name="compile" depends="prepare">
  <mkdir dir="${result.dir}" />
  <myscalac srcdir="${submission.dir}" classpathref="scala.classpath"
     scalacdebugging="true" failonerror="true" destdir="${result.dir}">
    <exclude name="*Tester.scala" />
  </myscalac>
</target>

<target name="tests" depends="compile">
  <myscalac srcdir="${submission.dir}" scalacdebugging="true"
     destdir="${result.dir}">
    <classpath>
      <path refid="scala.classpath" />
      <path location="/home/ashlesha/SJSU/Project/svnSource/JavaBrat/scalabrat/
          bin" />
    </classpath>
    <include name="${testclass}.scala" />
   </myscalac>
   <java classname="scala.tools.nsc.MainGenericRunner" fork="true">
     <classpath>
        <path refid="scala.classpath" />
        <path location="${result.dir}" />
        <path location="/home/ashlesha/SJSU/Project/svnSource/JavaBrat/
            scalabrat/bin" />
     </classpath>
       <arg value="${testclass}" />
    </java>
 </target>
```

Listing 5-1: Ant build file - codecom.xml

As shown in the Listing 5-1, the first target, `compile`, compiles all the Scala files from the submission directory, except the test classes. The second target, `tests`, compiles and runs the test cases. Ant provides a built-in listener mechanism. Ant listeners are alerted when the certain events occur. For example, a listener is alerted when a build start event occurs. These events provide useful information about the target. For example, a target finished event can be used to check if the target was finished successfully or not. A listener should be register in order to get informed when an event is generated. The following code show how a listener can be registered to an Ant project.

```
import org.apache.tools.ant.Project;

 public class AntRunner {
   …
   …
   Project project = new Project();
   ReportBuilder reportBuilder = new ReportBuilder();
   project.addBuildListener(reportBuilder);
   …
   …

 }
```

Listing 5-2: Registering listeners to Ant project

The ReportBuilder class mentioned in the above code is the listener class that implements Ant's BuildListener interface.

The BuildListener interface declares the callback methods for each event generated by Ant. For example, when a target finishes its execution, an event occurs and the targetFinished method of the registered BuildListener is notified. Here is the targetFinished method of the ReportBuilder listener that checks if the compile and tests targets executed successfully or not.

```
public void targetFinished(BuildEvent event){
  String msg;
  Target target = event.getTarget();
  if (!currentDetail.targetName.equals(target.getName()))
    return;
  if (target.getName().equals("prepare"))
    return;
  currentDetail.exception = event.getException();
  msg = event.getMessage();
  if (msg != null)
    currentDetail.message.append(event.getMessage());
  if (event.getException() != null)
    currentDetail.failed = true;
    details.add(currentDetail);
}
```

Listing 5-3: Sample code for Ant listener

The above listener checks whether any message is generated by the event. If this message can provide useful feedback to the user, then it is added to the report during report generation phase. If the event has generated any exceptions, then the current target is marked as failed target. I have also used a few other Ant events like build started, target started, message logged, build finished to collect the information to generate the report.

5.1.2. Compilation Process

The Scala distribution provides several Ant tasks to build a Scala project. Initially I decided to use the standard scalac task to compile the user's code. The scalac task compiles the code but it fails to deliver the messages generated during the compilation phase to the Ant listeners. I wanted to capture these messages as they are important for the Scalabrat report generation. When I debugged the Scala compiler code, I found out that the standard Scala compiler uses the ConsoleReporter class to handle the compiler output. The ConsoleReporter class displays the messages produced by the compiler on the console but it does not save them. As a result, it fails to deliver these messages to the Ant listener.

To overcome this limitation, I have created a new class Myscalac that extends the Scala compiler class Scalac. I have also created my own reporter class that extends the Scala compiler's `Reporter` class. My reporter class is called as `AshleshaReporter`. This reporter overrides the `info0` method of the inner class `info`. The new reporter 's `info` method properly formats the compilation output and stores it in a string. `Myscalac` and `AshleshaReporter` deliver the compilation messages to the Ant listeners.

5.1.3. Test Cases

Scalabrat uses the test cases to determine if the user's program functions correctly. If the compilation target executes successfully then as shown in the listing 5-1, the Ant builder runs the `tests` target. For better evaluation, a program should be tested against varied input data. The test cases should be simple and easy to write so that the problem author is motivated to write more test cases. I have accomplished this by providing a base class for all the test cases. This class is called as `TestBase` class. All the test cases provided for the Scalabrat problems should extend this test base and simply call its `test` method with certain parameters. The `TestBase` class and its `test` method is described in more detail in the next paragraph.

In Scala, functions are considered as objects. Hence they can be passed to another function as an argument or can be assigned to a variable. I have used this feature in the `TestBase` class. The `test` method of the `TestBase` class accepts the method to be tested from the user's code and invokes this method dynamically irrespective of its type. As shown in the listing 5-4, the `test` method accepts the function to be tested as its first parameter. The notation f: A=>R means a function that accepts a parameter of type A and returns a value of type R. The following section helps to understand this notation in more detail.

Scala has inbuilt support for genericity. As per [6], “Genericity is the ability to write code parametrized by types”. Scala allows you to define generic methods. The `test` method defined in the `TestBase` class is a generic method that accepts three parameters. The first parameter `f`, is of type function that accepts a parameter of type A and returns a value of type R. The second parameter, `arg` is of type A and the third parameter `expected` is of type R. A and R are the generic types that can accept any value. At runtime, they are replaced with the types specified for that method call. For example, the following call specifies that for this test call, A and R values are of type Int.

```
  test[int, int](square, 2, 4)
```

Because of this feature, I could use a single `test` method to test methods without caring about the actual types of the expected or returned values. Here is the code snippet from the `TestBase` class:

```
class TestBase{

  def main(args:Array[String]){
  }

  def test[A, R](f: A=>R, arg: A, expected: R){
    var result = f(arg);
      if(result == expected)
        println("Test Successful");
      else
        println("Test failed");
        println(toString(arg));
        println(toString(expected));
        println(toString(result));
}
```

Listing 5-4: Code snippet from `TestBase` class

In the above code, the `test` method calls the function `f` with a input value `arg`. Then it checks if the result received from this function is as the expected result. If the result is as expected it prints test successful otherwise prints test failed.

The above `test` method is useful only to test the functions that accept a single input parameter. But the user code can have functions that accepts more than one input parameter. To address this problem I have overridden the `test` method for multiple input parameters. I have provided ten `test` methods that accepts up to ten input parameters. For example, the following `test` method can test a function that accepts four input parameters:

```
def test[A1, A2, A3, A4, R](f:(A1, A2, A3, A4)=>R, arg1:A1, arg2:A2, arg3:A3,
    arg4:A4, expected:R){
  …
  …
  …
}
```

Listing 5-5: Code snippet from `TestBase` class

Here is one example of how the actual test case class looks like

```
import com.ashlesha.scalabrat.TestBase;

object TupleTester extends TestBase{
  test[Int, Int, Tuple2[Int, Int]](Tuple.divmod, 4, 2,
      new Tuple2[Int, Int](2, 0))
  test[Int, Int, Tuple2[Int, Int]](Tuple.divmod, 6, 4,
      new Tuple2[Int, Int](1, 2))
}
```

Listing 5-6: Sample code for test cases

The code above calls the `test` method of the `TestBase` class to test function `divmod` of the `Tuple` class. The input values for the first test are 4 and 2 and the expected output is a `Tuple` with values (2, 0).

6. Evaluation

This section contains the evaluation of the system based on the actual statistics. Dr. Horstmann has used Moodle with the Labrat plugin in his CS 46 course. He has been using it from the start of the semester. But unfortunately, early versions did not contain enough information to associate submissions with individual students. For this report, I started collecting data that contained student IDs only recently. For this evaluation, I have collected the data from each request of Javabrat's check web service. A call to the check web service corresponds to an attempt by a student for a question in the homework. I have analyzed the available data with respect to the following aspects:

6.1. Draft Submission

Dr. Horstmann uses a draft submission policy in his homeworks. He provides the questions of the next homework with the current homework and asks the students to submit a draft of the solution for these questions. Draft submissions are intended to make students think about the assignment early. For the draft submissions, students are supposed to provide a compilable class with the method stubs and Javadoc comments. Students get bonus points for the successful draft submission. The following statistics shows how students responded for the draft questions for Homework 18 to 20.

No Homework	Question Id	Number of students submitted draft	Number of students did not submit draft	Number of students received full grade for draft questions	Number of student s received full grade and who had submitted draft	Difficulty level of the question
17	7_13	14	15	20	11	Difficult
17	7_15	13	16	15	8	Difficult
18	8_1	22	7	18	18	Easy
18	8_2	21	8	15	15	Easy
19	8_7	18	11	9	9	Medium
19	8_8	17	12	7	7	Difficult
20	9_4	21	7	18	17	Easy
20	9_8	18	11	20	18	Easy
20	9_9	16	13	12	11	Difficult

Table 1: Draft submission statistics

From the above table it is evident that the students are likely to get full credit for those questions for which hey have submitted the draft. For all the questions except questions 7_13 and 7_15, more than 80% of the students who got full grade had submitted the draft. The above statistics shows that the draft questions are helpful for students. Dr. Horstmann could assign more draft questions without any extra overhead because Moodle plugin automates the grading process.

6.2. Automatic Grading Vs Manual Grading

In this section, I have provided an analysis of automatically graded submissions by Javabrat with respect to manual grading. Out of available data, I randomly selected 20 submissions for which Javabrat had given full credit to the students. I manually graded these submissions and compared them with the Javabrat report. Out of these 20 questions, 10 are easy to medium type of questions and 10 are difficult questions. All of the 20 submissions that I manually assessed, contained the valid answers for the given problems. On the whole, Javabrat did a good job in the assessment of these submissions. But there are some areas for which a manual grader would have done a better assessment. These areas are:

6.2.1. Javadoc Comments

Many students had not provided proper Javadoc comments in their submissions. Majority of the students had skipped Javadoc comments for :

1. Java classes
2. return values

But Javabrat could not detect these mistakes and gave full credit for the submission.

One way to solve this problem is to run the static code analysis tools such as Checkstyle before the automatic grader generates the final grade. The automatic grader should consider the output generated by code analysis tool while calculating the grade.

6.2.2. Standard Coding Practices

Some of the submissions that I manually graded had not properly followed the coding conventions. For example, the variable names were not relevant and getter and setter method conventions were not properly followed. The following code snippet shows a non-standard coding practices for which the automatic grader did not penalize the students:

```
/**
   * gets a values
   * @return the value of this coin
   */
  public double getValue(){
    return howM;
   }
```

In the above example, variable name `howM` is not a relevant name and does not help the reader to understand what value does it hold. As per the getter method conventions, the get method name should have been `getHowM` instead of `getValue`.

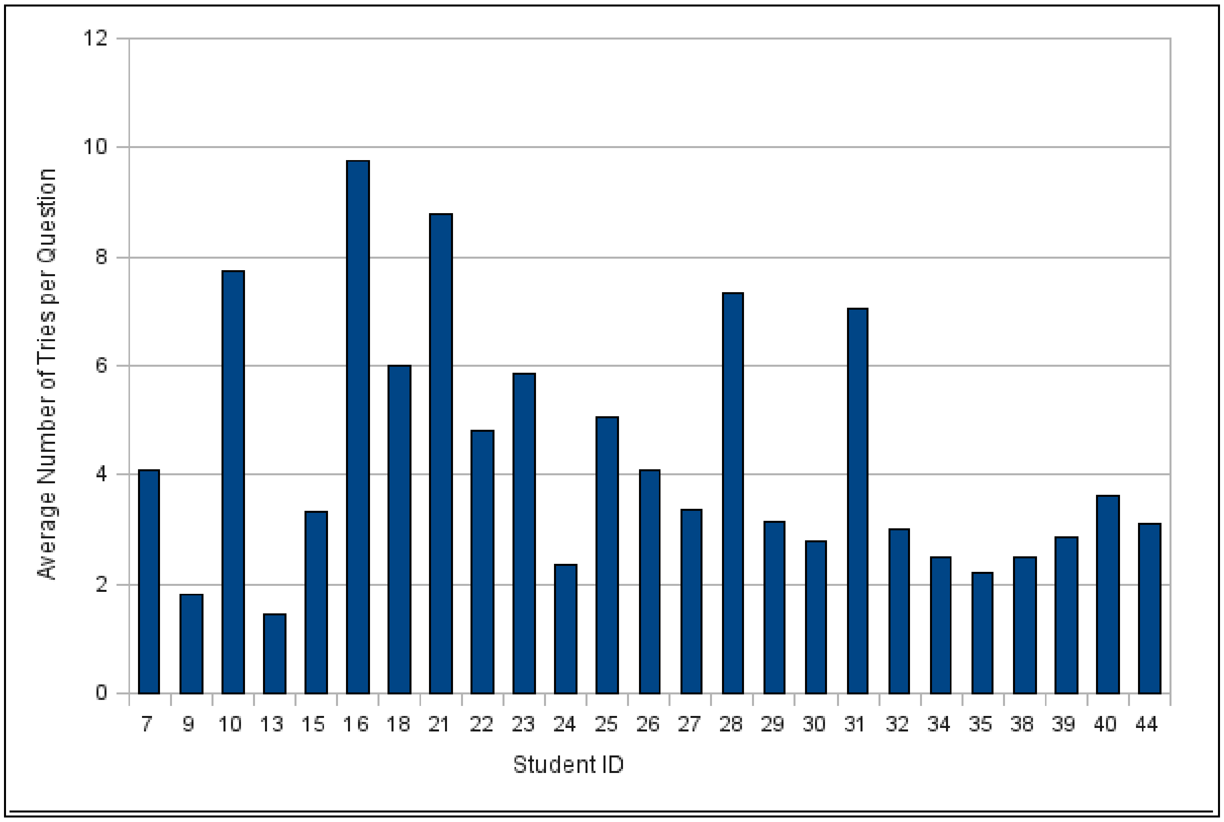
Some of the submissions had the poor code organization that can lead to the lack of readability and maintainability. Some of the submissions had very long functions ignoring the modular approach. For example, one student had written the `main` function that contained about 80% of the total code. The automatic grader ignored these violations of coding standards and the gave full credit.

From the above facts it is clear that though automatic grading reduces the grading efforts, it has certain limitations and it can not totally replace the manual grading. Labrat uses a intermediate approach to handle this limitation. Labrat includes the student answers in the evaluation report along with the automatically calculated grades. The instructor or the grader can have a look at these answers before finalizing the students' grades. They can manually analyze the students' solutions with respect to the above mentioned factors and provide the right feedback and grades to the students. This approach is also useful to verify that a student has actually solved the problem and he has not cheated by tweaking the code for the few test cases.

6.3. Students' Understanding of the System

The goal of this analysis is to find out whether the students are using the grading system as intended. Students should not use the Moodle plugin as a code development tool. Students should prepare and test their answers outside Moodle in an IDE. When they are satisfied about their answers only then they should paste the answer into Moodle. I have provided the “Check” button just to give the students a chance to rectify the minor mistakes in their final submission. For example, using “Check” button, the students can verify that they have not made any mistake while copy pasting the answer in to the Moodle. The students should not use the “Check” button to verify that their answer compiles successfully or the test cases run as expected. To impose this desired behavior, I have made the editing of solutions tedious in Moodle. To accomplish this, I reduce the the size of the textarea (where students are supposed to provide answer) once the students press the “Check” button.

In this analysis, I found out that some of the students are using the system as intended but some of the students perform many attempts before submitting the final solution. The following graph shows the average number of attempts made by the students for a subset of the questions they have attempted.



Listing 6-1: Average attempts per students

The above graph shows the statistics of 25 students from Dr. Horstmann's CS 46 class. On an average 14 students made 4 attempts before submitting the final solution. Remaining 11 students had more that 4 average attempts. 2 students with IDs 16 and 21 had average number of attempts as high as 9.75 and 8.78 respectively.

To find out why some of the students required such a high number of attempts, I analyzed their each attempt in the chronological order for a subset of the problems. I found out that they were not using the system as intended. Here are some of the most common reasons why the students required multiple attempts:

- The students started with non-compiling code.
- The students introduced major code changes in the consecutive attempts.
- The students resubmitted the assignment by adding Javadoc comments after they got full credit

Thus, the students who have very high average attempts were using Moodle as a code development tool. It implies that they were not using the efficient development tools (IDEs) and instead relying on Moodle to check their solution during the development phase. They were using a tedious and a time consuming way to solve the problem. One way to stop the students from using Moodle as an IDE is to impose a restriction on the number of times a student can check his answer in Moodle.

6.4. The Moodle Survey

With the help of Dr. Horstmann, I conducted a survey for Moodle plugin in the CS 46 class. Out of 25 students 14 students participated in the survey. Here are the survey questions and summary of the responses we received from students

Q1. Please compare taking assignments through Moodle with the traditional way of submitting solutions by email. (1=Much prefer Moodle, 5=much prefer email submission)

number of students answered 1= 8
number of students answered 2= 2
number of students answered 3= 1
number of students answered 4= 2
number of students answered 5= 1

Q2. Please compare old Moodle where you have to see the question from Javabrat and copy paste the report back to Moodle with the new one that shows the report in Moodle itself (1=Much prefer copy/paste, 5=much prefer integrated report feature)

number of students answered 1= 1
number of students answered 2= 2
number of students answered 3= 2
number of students answered 4= 4
number of students answered 5= 5

Q3. The draft submissions are intended to make you think about the assignments early. How useful do you find this feature? (1=Draft feature very useful, 5 = Draft feature not useful at all)

number of students answered 1= 5
number of students answered 2= 2
number of students answered 3= 5
number of students answered 4= 2
number of students answered 5= 0

Q4. What do you submit for the draft question most of the time?

number of students who submit complete solution = 5
number of students who submit class template and method stub = 1
number of students who submit class template and method stub with partial solution = 8
number of students who submit who do not attempt draft questions = 0

Q5. How do you rate the Moodle interface? (1=very clear, 5=very confusing)

number of students answered 1= 1
number of students answered 2= 7
number of students answered 3= 6
number of students answered 4= 0
number of students answered 5= 0

7. Future Work

The developed system can be further enhanced by providing support for the following features:

7.1. Moodle Question Import Feature

With the current Moodle Labrat plugin, the instructors have to manually create each question. This job is tedious if the number of questions to be created are more . Moodle provides a question import feature that facilitates importing questions in bulk into the question bank. Moodle supports number of formats for importing questions. With the Moodle XML format, instructors can specify the problem information in an XML document and import multiple questions into Moodle. Currently Labrat plugin does not support the import feature. The system can be further improved by providing support for importing Labrat type questions.

7.2. Scala Problem Support from Moodle

Currently, Labrat plugin allows only to create the Java programming questions in Moodle. Javabrat's web services used by Moodle plugin provide support only for the Java questions. This plugin and web services can be extended to provide support for the Scala programming problems within Moodle.

7.3. IDE Plugin for Javabrat Assessment

Ideally while working on Moodle programming quizzes, the students should use their IDE to develop the solution and paste their final solution into the Moodle form. They should use the check button only to rectify the minor mistakes. But while evaluating the project uses, I found out that many students use the Moodle form as an IDE and use the check button until they develop their final solution. If a student's IDE can generate the Javabrat report, it might help the student to anticipate his actual Moodle report. With this enhancement, students can take advantage of their IDE and also see the final report. This can be achieved by developing a IDE plugin that produces the Javabrat report based on the student's current code.

8. Conclusion

In this project, I have developed a system that reduces the efforts required to assess the programming assignment from the instructor's perspective. At the same time I have focused on providing a simple and useful tool that allows the students or the language learners to practice more programming problems. I have used the existing system Moodle, to reuse already available features for the course management. I have also analyzed the students' responses to evaluate the system usage.

The work was challenging because the Moodle extension framework does not have a sound base. Initially, I thought that as Moodle is based on a modular design, extending it would be a straightforward task. I found it difficult because PHP does not encourage abstraction very well and the Moodle extension points are very poorly designed. One may say that it is a good example of how not to design the extension framework.

Moodle is very incompetent in handling the character entities of the different text formats. The core code of Moodle does not take care of this issue. Every module uses a different approach and it is not well documented. I had to spent a lot of time figuring out why the plugin is behaving weird with the input text. For example, why is it eating space or the text after every “< “character.

Apart from Moodle, I faced few challenges while developing the Javabrat web services. I was trying hard to use already developed blackened infrastructure of Javabrat for the web services. But I found out that fundamentally it was not possible to create a web service for the POST request with the existing JSF application. JSF framework assumes that any POST request is a reaction from a prior JSF page. With the JSF 2.0 it is possible to handle the GET request parameters, but it does not work with the POST request. To overcome this limitation, I used JAX-RS to expose the Javabrat methods as the web services.

The work was ambitious because I wanted to create a system that is intuitive and clear enough for the students to attempt online assignments without putting much effort to understand the system.

Dr. Horstmann has been using Moodle with Labrat plugin for his CS46 course for one semester. Automatic grading of the programming assignments has certain limitations, but it is certainly effective to reduce the evaluation efforts.

9. References

[1] Maheshwari, P. 1996. Teaching programming paradigms and languages for qualitative learning. In Proceedings of the 2nd Australasian Conference on Computer Science Education (The Univ. of Melbourne, Australia). ACSE '97, vol. 2. ACM, New York, NY, 3239. DOI= http://doi.acm.org/10.1145/299359.299365

[2] CodingBat, Java practice problems.Web site: http://codingbat.com/

[3] better programmer. Web site: http://www.betterprogrammer.com/

[4] Moodle. Web site: http://moodle.org/

[5] Simply Scala, Web site: http://www.simplyscala.com/

[6] Scala, Web site http://www.scala-lang.org/

[7] Corbera, F., Gutiérrez, E., Ramos, J., Romero, S., and Trenas, M. A. 2008. Development of a new MOODLE module for a basic course on computer architecture. In *Proceedings of the 13th Annual Conference on innovation and Technology in Computer Science Education* (Madrid, Spain, June 30 - July 02, 2008). ITiCSE '08. ACM, New York, NY, 349-349. DOI= http://doi.acm.org/10.1145/1384271.1384391

[8] Rößling, G. and Kothe, A. 2009. Extending moodle to better support computing education. In *Proceedings of the 14th Annual ACM SIGCSE Conference on innovation and Technology in Computer Science Education* (Paris, France, July 06 - 09, 2009). ITiCSE '09. ACM, New York, NY, 146-150. DOI= http://doi.acm.org/10.1145/1562877.1562926

[9] Moodle plugin template source code http://moodle.cvs.sourceforge.net/moodle/contrib/plugins/question/type/TEMPLATE/

[10] Roberts, G. H. and Verbyla, J. L. 2003. An online programming assessment tool. In *Proceedings of the Fifth Australasian Conference on Computing Education - Volume 20* (Adelaide, Australia). T. Greening and R. Lister, Eds. Conferences in Research and Practice in Information Technology Series, vol. 140. Australian Computer Society, Darlinghurst, Australia, 69-75.

[11] Practice-It. Web site: http://webster.cs.washington.edu:8080/practiceit/

[12] Spacco, J., Hovemeyer, D., Pugh, W., Emad, F., Hollingsworth, J. K., and Padua-Perez, N. 2006. Experiences with marmoset: designing and using an advanced submission and testing system for programming courses. In *Proceedings of the 11th Annual SIGCSE Conference on innovation and Technology in Computer Science Education* (Bologna, Italy, June 26 - 28, 2006). ITICSE '06. ACM, New York, NY, 13-17. DOI= http://doi.acm.org/10.1145/1140124.1140131

[13] Spacco, J., Pugh, W., Ayewah, N., and Hovemeyer, D. 2006. The Marmoset project: an

automated snapshot, submission, and testing system. In *Companion To the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications* (Portland, Oregon, USA, October 22 - 26, 2006). OOPSLA '06. ACM, New York, NY, 669-670. DOI= http://doi.acm.org/10.1145/1176617.1176665

[14] Web-CAT home page. Web site: http://web-cat.cs.vt.edu/WCWiki

[15] Edwards, S. H. 2003. Teaching software testing: automatic grading meets test-first coding. In *Companion of the 18th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications* (Anaheim, CA, USA, October 26 - 30, 2003). OOPSLA '03. ACM, New York, NY, 318-319. DOI= http://doi.acm.org/10.1145/949344.949431

[16] Sakai project. Web site: http://sakaiproject.org/

[17] Suleman, H. 2008. Automatic marking with Sakai. In *Proceedings of the 2008 Annual Research Conference of the South African institute of Computer Scientists and information Technologists on IT Research in Developing Countries: Riding the Wave of Technology* (Wilderness, South Africa, October 06 - 08, 2008). SAICSIT '08, vol. 338. ACM, New York, NY, 229-236. DOI= http://doi.acm.org/10.1145/1456659.1456686

[18] Marmoset project. Web site: http://marmoset.cs.umd.edu/

[19] Reges S. and Stepp M. (March 2010). Building Java Programs: A Back to Basics Approach. Parson Education

[20] Cay Horstmann. Big Java. Wiley Plus

[21] Checkstyle. Web site: http://checkstyle.sourceforge.net/