CSIT 5100 (Spring 2020) Assignment 1

Object-Oriented Programming and Testing

Instructor: Prof. S. C. Cheung

Department of Computer Science and Engineering
The Hong Kong University of Science and Technology

Due Dates: Stage 1 submission (23:50, March 17, 2020); Stage 2 submission (23:50, March 31, 2020)

1. Objective and Requirement

The objective of this assignment is to practice unit testing of object-oriented programs under the JUnit 5 framework. You need to construct a suite of JUnit test cases to help reveal potential faults in an open-source calendar system. The test suite should be adequate to achieve high statement and branch coverage.

2. Assignment Description

JUnit is an open-source testing framework to automate unit testing of Java programs (http://www.junit.org/). A JUnit *test case* is a Java class with a set of test methods. Each test method invokes some methods of the class/object under test (CUT), and uses assertions to examine if the CUT behaves as expected. A *test suite* comprises a set of test cases, which collectively achieve high statement and branch coverage.

The Java program that you need to test in this assignment implements a multi-user calendar manager system, called *MCM*. *MCM* supports basic calendar functionalities to help users schedule daily appointments. It also provides a user system supporting multi-user collaboration. MCM consists of a server and a client. You can only use the client to log into the system after the server is started. ***You only need to construct test cases for the client. Only the test cases or test suites for the client code will be graded.*** We also provide the source code for the server for your reference, you can use it to test the client and have a better understanding of the whole *MCM* program. The assignment resources contain three zip files.

- The "*exe*" folder contains bytecode files for both the server and the client for MCM, namely "*Server.jar*" and *"Client.jar"*, together with two input files needed to start the server. If you have Java Runtime Environment (JRE) installed on your computer, you should be able to run the program and have a tryout. README.txt contains the steps to run *MCM*. Please read it before you have a try.

- The *"MCM_Server.zip"* contains an Eclipse project of *MCMServer*. You may run MCMServer directly using a line command *"java -jar Server.jar Input_MCMServer.txt"* under the *exe* directory. The Eclipse project of MCMServer is provided for your reference. You do not necessarily need to import this project to Eclipse for this assignment. **Note that you need to run MCMServer before MCMClient**. If you get an error "Registry Creation at port 1099 failed" when running the above line command, it is likely that the port 1099 is being occupied by another running process, likely an earlier MCMServer process. For Windows users, you may check it by "netstat –an". The input_MCMServer.txt file configures the user information of registered users for this MCM application.

- The "*MCM_Client.zip*" contains an Eclipse project of the client of *MCM*. You need to import this project to Eclipse and write JUnit tests for MCMClient. You may import the project using the import facility under the File menu and then select MCM_Client.zip as the archive file. **Note that MCMServer must be running in order for MCMClient to function**.

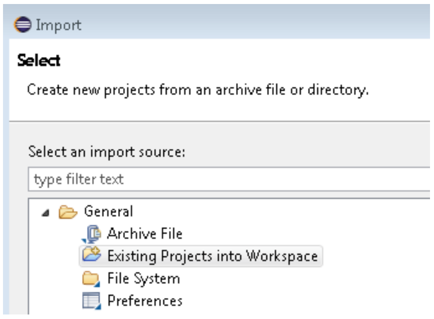
- The "*Doc.pdf*" is a document explaining the major functionality of *MCM*. This document can help you better understand the project and may help you construct test cases.

You need to use *EclEmma* to measure the statement and branch coverage achieved by your test suite. If you find any infeasible statements or branches (e.g., dead code or predicates), please list them in the final report of this assignment, and provide proper justification. You may explain in the report if these infeasible statements or branches reveal any bugs. Note that you are not required to develop a test suite that achieves 100% statement and branch coverage because this is not possible in most cases due to the existence of infeasible statements or branches.

2.1 Quick Start

- We will use Java SE 13 and Eclipse IDE (2019-12) for grading. You can download the latest Eclipse IDE for Java developers (2019-12) from http://www.eclipse.org/downloads/. By default, it is bundled with JUnit 5. Please install Java SE 13 before installing Eclipse.

- You can follow the instructions below to import the source code into an Eclipse project.


Import

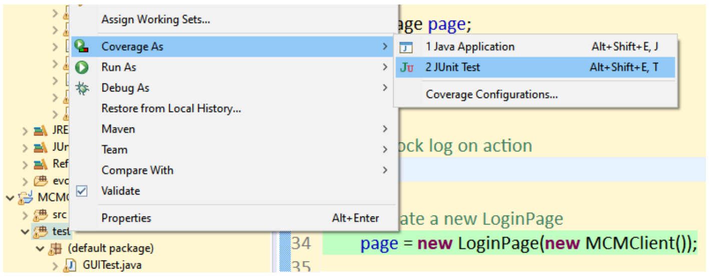
Select

Create new projects from an archive file or directory.

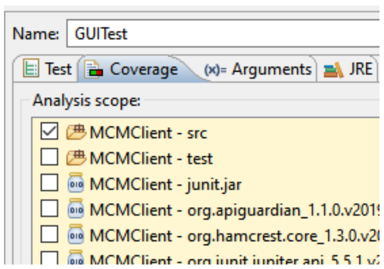
Select an import source:

type filter text

1. Select "*Import…*" under "*File*" menu to import "*Existing Projects into Workspace*".
2. Browse *MCM_Client.zip* after selecting "*Select archive File*".
3. Click "*Finish*" to import the *MCM* project.



```
age page;
ock log on action
ate a new LoginPage
page = new LoginPage(new MCMClient());
```


Name: GUITest

Analysis scope:

- [x] MCMClient - src
- [ ] MCMClient - test
- [ ] MCMClient - junit.jar
- [ ] MCMClient - org.apiguardian_1.1.0.v201
- [ ] MCMClient - org.hamcrest.core_1.3.0.v2
- [ ] MCMClient - org.junit.jupiter.api_5.5.1.v2

4. Test cases and the test suite should be put into the "***test***" folder under the *default* package. ***We will run JUnit Test against the "test" folder when grading your submission as shown in the diagram above. Test cases in other folders will not be graded***. ***We only count the statement and branch coverage of code in the "src" folder of MCMClient. You should***

not make any changes to the src folder. You may confine the coverage measurement to the "src" folder by configuring the Coverage configurations under "Coverage As -> Coverage Configurations…".

- To construct a JUnit test case, you can choose menu *File → New → JUnit Test Case*. Besides invoking the methods under test and writing assertions, you should also properly implement the routine methods such as "*setup()*" and "*teardown()*" when necessary. ***Please note that the statements covered by a test case that checks the test results using meaningless assertions such as "assertTrue(true)" will not be counted.*** We provide an example for your reference in the appendix. The imported MCMClient project should have included the bundled JUnit 5 library. If not, you may include the JUnit library to the MCMClient project by right clicking the MCMClient project and then select "Build Path -> Add Libraries…" followed by "JUnit" and select "JUnit 5". After that, you will see the JUnit 5 library added to the MCMClient project.

- MCM client is a GUI program involving a number of GUI widgets and corresponding action handlers. In case you may be confused about how to write JUnit test for GUI widget, we provide an example test case in the MCM client project. It may give you some hints on how to invoke the registered handlers of a GUI widget. Please refer to *"GUITest.java"* in test folder of MCM client project to learn more details.

- *EclEmma* should be bundled with Eclipse; otherwise, you can directly search "*EclEmma*" in Eclipse Marketplace and install it. You may use other code coverage tools, but the grading will be based on the coverage results provided by *EclEmma*. For the usage of *EclEmma*, please refer to its online user guide (http://www.eclemma.org/userdoc/index.html). You are highly encouraged to go through the following two topics:

  1. Launching in Coverage Mode
  2. Using the Coverage View

2.2 Submission Requirement

- **Stage 1.** By the end of this stage, your test suite should achieve at least 50% statement and 10% branch coverage. You need to submit your test cases at Canvas by the deadline.

- **Stage 2.** You can further improve the coverage of your test suite in this stage. After completion, you are required to submit the improved test suite with a brief report. In the report, you should clearly present
  1. A few sentences describing the basic structure of the system under test.
  2. The statement and branch coverage your test suite achieved.
  3. How you construct the test cases to achieve high coverage with affordable effort. If you applied some technique to automate the test generation process, please also state the basic idea of your technique. For example, you may randomly generate a large number of test cases for a method, and add the test case that increases coverage into the test suite.
  4. The infeasible program statements, if any. It is generally impossible to find all. Just try your best.

When submitting your assignment, you should zip (1) the exported archive of MCMClient project, which includes the source code of both the original MCMClient program and your JUnit tests, and (2) assignment report (stage 2 only) into a single zip file with name in the form "*yourname_studentId_assignment1.zip*". The report should be in MS Word (docx) format.

3. Marking Scheme

The following three factors contribute to your score of this assignment.

- Statement Coverage (35%); Branch Coverage (35%). Higher coverage leads to higher score.
- Assignment Report (10%). Well written reports with some interesting and concrete discussion about your finding will lead to higher score. Discussion can sometimes be more concrete if it is accompanied by concrete examples from the code.
- Successful submission of Stages 1 satisfying the requirement by the deadline (20%). Note that at stage 2, you may modify the test cases submitted in stage 1.

Please do make sure your submission conforms to all the above requirements. Otherwise, you may lose marks (up to 10%) if your submission does not conform to the requirements.

4. Further References

- JUnit User Guide: https://junit.org/junit5/docs/current/user-guide/
- JUnit Javadoc: https://junit.org/junit5/docs/current/api/
- EclEmma Installation: http://www.eclemma.org/installation.html
- EclEmma User Guide: http://www.eclemma.org/userdoc/index.html

Appendix: An Example Test Case

The following unit test case can be used to test class Appointment of *MCM*. The class Appointment is used to store the information for a calendar appointment. The following test case is designed to test the functionality of constructing an appointment object and adding it to the hash table storing all the existing appointments. To demonstrate how to write test cases for graphical user interfaces, we have also prepared an example GUITest.java in the MCMClient project for your reference.

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import mcm.Appointment;
import mcm.User;
import java.util.Date;

class AppointmentTest {
	private Date start, stop;
	private String description;
	private User user;

	@BeforeEach
	void setUp() throws Exception {
		start = new Date();
		stop = new Date();
		description = "CSIT5100";
		user = new User("Jim", "jim@world.com");
	}

	@AfterEach
	void tearDown() throws Exception {
		start = null;
		stop = null;
		description = null;
		user = null;
	}

	@Test
	void test() {
		Appointment appointment = new Appointment(start, stop, description, user);
		Appointment.add(appointment);
		Appointment foundAppointment = Appointment.find(start, user);

		assertTrue(appointment.equals(foundAppointment));
		assertTrue(appointment.getStartTime().equals(start));
		assertTrue(appointment.getEndTime().equals(stop));
		assertTrue(appointment.getDescription().equals(description));
		assertTrue(appointment.getInitiator().equals(user));
	}
}
```