ECE 720 Assignment Hands-on Project 1

Assignment 1 due on February 11, 2022, 11:59 PM MST.

Late Submission Policy: You have **3** free late days. Afterwards, **10%** off per day late.

1 Introduction

Unit testing is a part of software testing process. The purpose of writing a unit test is to test the smallest piece of code that can be logically isolated in a system. Writing unit tests is often the responsibility of the devloper. Existing tools like JUnit could provide user-friendly framework for manually writing unit tests, however, these turns out to be tedious and time-consuming. Randoop [1] is one of the tools that can automatically generate unit tests. Although automated test generation is efficient, it is still important to understand both the strengths and weaknesses of the tools you are using.

In this assignment, you're expected to finish some small tasks about setting up Randoop for automated testing.

2 Resources

- Unit testing
- The Linux command line for beginners
- Java Download Page
- Randoop Project Page
- Randoop User Manual
- Jacoco Command Line Interface User Manual

3 Development Tool

3.1 Terminal

All given code in this assignment is based on Command Line Tool. If you're not familiar with how to use Terminal under Linux or Mac OS environment, please refer to the tutorial in Resources Section.

3.2 IDE

VS Code [2], Atom [3] and Sublime [4] are good choices. You can also use IDEs like IntelliJ IDEA and Eclipse if you're familiar with them.

4 Warm up

In warm up section, you are going to fix an error in a toy Java class called `MyInteger`. Briefly introduced, this task will guide you running the Randoop test generation tool for `MyInteger` class.

[1]https://randoop.github.io/randoop/
[2]https://code.visualstudio.com
[3]https://atom.io
[4]https://www.sublimetext.com

4.1 Environment setup

- Make sure you have Java installed on your computer and the JVM version is Java 8 or 11, and also have JAVA_HOME Environment Variable set correctly. To test whether you have installed JAVA correctly:

  ```
  java -version # The version of JAVA should appear on the screen
  ```

- This project is required to be running under Unix OS. If you are using Windows, you need to be very careful about the path.

- Download the zip file from e-class and unzip it (**Please make sure the path doesn't include any folder whose name containing white space, otherwise it might trigger an error in Randoop. To fix this, e.g., replace** /ECE 720/Project1 **with** /ECE720/Project1).

- The structure of Project1 is shown below. All required jar files are included in */lib/ folder.

```
   Project1
 |__MyInteger
 |  |__MyInteger.java MyInteger
 |  |__lib
 |     |__*.jar lib
 |__joda
    |__lib
       |__*.jar lib
```

4.2 (4 points) Discover a bug

1. Make sure to read the **Generating tests for java.util.Collections** example provided by Randoop User Manual first.

2. Before you start, your current working directory should be /Project1/MyInteger

3. Please first compile the example code:

   ```
   javac MyInteger.java
   ```

4. Use randoop-all-4.2.7.jar file in /MyInteger/lib folder to generate test. If you do not know what the Java classpath is, please spend a moment to learn about it instead of directly copy the code provided below. Briefly, it tells the JVM where to find classes. When you run Randoop, the JVM will need to load and run classes from Randoop itself, and also classes from the application under test (and any libraries it uses). Be sure that the classpath contains both.

   ```
   java -classpath ./:./lib/randoop-all-4.2.7.jar randoop.main.Main \
   gentests --testclass=MyInteger --time-limit=30
   ```

   After running the above code, Randoop will create test suites: one that reveals errors that are already in the program, and one that creates regression tests. The fact that Randoop generated the error-revealing tests means that it discovered a faulty behavior. You may find that all methods in ErrorTest*.java file violate the same contract:

   ```
   Contract failed: equals-hashcode on * and *
   ```

4.3 (6 points) Fix the bug

1. If we look at the class (`Project1/MyInteger/MyInteger.java`) more closely than before, we can find that the equals method is incorrectly defined

    ```
    public boolean equals(Object other) {
        if (other instanceof MyInteger) {
            return true;
        }
            return false;
    }
    ```

    and there is no `hashCode()` method for `other instanceof MyInteger`.

2. Now we need to fix the equals function. See following code:

    ```
    public boolean equals(Object obj) {
        if (obj instanceof MyInteger) {
            // TODO
        }
        return false;
    }
    ```

    Try to fill the TODO part by yourself. Think about how to correctly covert the other instances to a MyIntger instance. Then compile MyInteger.java file and run Randoop Test again. If you write the equals function correctly, you will find that the Error-reveailing tests are all gone.

3. Take a screenshot to the equals function you wrote and attach it to your project report file.

4.4 (3 points) Test coverage

Test coverage is a measurement (in percentage) of the degree to which the source code of a program is executed when a particular test suite is run. To evaluate the automated generated tests' coverage, you will use Jacoco Command Line Interface to generate a report. You will find the detailed information about the test coverage in this report.

1. You need to compile the generated test files and put all output class file to test-classes folder.

    ```
    mkdir test-classes

    javac -classpath .:./lib/junit-4.13.2.jar:./lib/hamcrest-core.jar \
    RegressionTest*.java -d test-classes
    ```

2. Now we've successfully compiled the Randoop test files, the next is to generate the coverage analysis.

    ```
    java -javaagent:./lib/jacocoagent.jar \
    -cp ./:./lib/junit-4.13.2.jar:./lib/hamcrest-core-1.3.jar:classes:test-classes \
    org.junit.runner.JUnitCore RegressionTest


    java -jar ./lib/jacococli.jar report jacoco.exec --classfiles ./MyInteger.class \
    --html report
    ```

After running the above code, you will find index.html file under `/Project1/MyInteger/report/` directory. You can open `index.html` with a browser and find detailed code coverage information.

3. Take a screenshot to the test coverage report and attach it to your project report file.

5 Automated generated tests analysis

In this section, you are going to use Randoop to generate tests for Joda-Time DateTime class. Then analyze any potential flaky tests existed in Error-revealing tests. Flaky tests are software tests that exhibit a seemingly random outcome (pass or fail) when run against the same, identical code [EPCB19].

5.1 (4 points) Generate Joda-Time Datetime tests

Using Randoop to generate tests for Joda-Time DateTime class. `DateTime` **is a class in** `Joda-Time` **package.** All required `jar` files are at `/Project1/joda/lib/`. **Please write the commands you used to the project report file**.

Hint: View Joda-Time DateTime documentation and Randoop User Manual to find how to set Joda-Time DateTime as test-class.

5.2 (3 points) Flaky tests

How many error-revealing tests were generated? If there are any, determine whether these tests really reveal the errors, or they are flaky tests. If you think any flaky tests existed, explain why they are flaky and how to fix the flaky.

6 Submission Guidelines

You need to submit this assignment as a zip file (.zip) containing your code and project report file on eClass. The zip file name should be '[First name]_[Student ID]_asg1.zip'. Please keep the exact same file structure as the following. For example,

```
zhijie_1234567_asg1.zip
├── MyInteger
│   ├── MyInteger.java MyInteger
│   ├── lib
│   │   └── *.jar
│   └── test-classes
├── joda
│   ├── lib
│   │   └── *.jar
│   └── test-classes
└── report.pdf
```

References

[EPCB19] Moritz Eck, Fabio Palomba, Marco Castelluccio, and Alberto Bacchelli. Understanding flaky tests: The developer's perspective. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2019, page 830–840, New York, NY, USA, 2019. Association for Computing Machinery.

ACADEMIC INTEGRITY

Students at the University of Alberta must read and follow, in its entirety, the

Code of Student Behaviour

Failure to know the code is not an acceptable excuse for breaking the code.

The University of Alberta is committed to the highest standards of academic integrity and honesty. Students are expected to be familiar with these standards regarding academic honesty and to uphold the policies of the University in this respect. Students are particularly urged to familiarize themselves with the provisions of the Code of Student Behaviour (on the University Governance website) and avoid any behaviour which could potentially result in suspicions of cheating, plagiarism, misrepresentation of facts and/or participation in an offence. Academic dishonesty is a serious offence and can result in suspension or expulsion from the University.

Engineering students studying in the province of Alberta should also follow the

Code of Ethics

by The Association of Professional Engineers and Geoscientists of Alberta (APEGA).

The Code of Student Behaviour should not be too hard to follow. Listen to your instructor, be a good person, and do your own work, as this will lead you toward a path to success. Failure to follow the code can result in a grade of ‘F’ for the course, a transcript remark, suspension, and even expulsion from the university.

"Integrity is doing the right thing, even when no one is watching"
C. S. Lewis

Engineering at Alberta