MP3: Testing

1 Introduction

Now you have gotten more familiar with the Jsoup project! But how effective are the tests for Jsoup and how can we help improve the tests for Jsoup?

In this MP, you are expected to evaluate and improve the testing effectiveness for the real-world Jsoup project. Note that if you have skipped MP1, please first follow MP1 to clone the Jsoup version we use for this class via `git clone https://github.com/CS427-UIUC/jsoup`; if you already have done MP1, you can reuse the same Jsoup copy you had.

2 Instructions

Before starting with the actual tasks, you need to learn how to collect the code coverage for the Jsoup project. You probably have already noticed that the `pom.xml` file of our Jsoup version includes the following plugin information:

```
<plugin>
    <groupId>org.openclover</groupId>
    <artifactId>clover-maven-plugin</artifactId>
    <version>4.4.1</version>
</plugin>
```

This plugin is used to run Clover (https://openclover.org/), a state-of-the-art code coverage collection tool for Java and Groovy. Besides line and branch coverage, it further collects over 20 code metrics. It not only shows the untested areas of your application but also combines coverage and other code metrics to find the most risky code. More importantly, the resulting HTML coverage reports show a detailed list of tests covering each line of your code.

You can invoke Clover to collect code coverage of the Jsoup tests directly using the following command:

```
mvn clean clover:setup test clover:aggregate clover:clover
```

This first cleans the previous build, executes `clover:setup` to instrument all the code files at the source code level, runs the test, and finally aggregates and presents the coverage reports. You can find the coverage report under the `target/site/clover` directory. You can simply click the HTML file named `index.html` to view the coverage for all your code files. The left panel presents the project overview and you can navigate to the detailed report for any code file. In each detailed report of a specific file, Clover presents the number of times each line has been covered next to the line number. When you click the coverage number (marked in green bars) for a code line, Clover will further show a detailed list of tests covering this specific line.

2.1 Task 1: Manual Test Generation

Congratulations for successfully obtaining code coverage, now you are ready for the actual tasks! Your first task is to identify the uncovered code lines for class `org.jsoup.parser.HtmlTreeBuilder`, and then design new tests to achieve 100% coverage for method `parseFragment(...)`. Hint: You can use Clover to inspect the other tests that cover similar lines in the method (by clicking the green bar left of the code line), and then try to design similar tests.

Please create a new package named `org.jsoup.mytests` under the test directory (i.e., create the directory `src/test/java/org/jsoup/mytests`). Inside the package, create a Java file named `ManualTest.java` to include your manually written tests.

Make sure that your tests in `ManualTest.java` can be successfully executed via "mvn test" and achieve the desired coverage. Additionally, make sure that your `ManualTest.java` file declares the correct package, `org.jsoup.mytests`, at the top of the file.

2.2 Task 2: Automated Test Generation

How was your experience for the first task? Are you already bored with manual test generation? No worries! You are now expected to explore automated test generation for the second task! More specifically, your second task is to identify the uncovered code lines for class `org.jsoup.nodes.Element`, and then automatically generate new tests. The new tests should achieve a minimum of 97% coverage for this class when running together with the original developer tests. You should run Randoop (https://randoop.github.io/randoop/manual/index.html#getting_randoop), a state-of-the-art automated test generation tool, on class `org.jsoup.nodes.Element`. Please go through the Randoop tutorial carefully, and the suggested command to generate tests is:

```
java -classpath "myclasspath:${RANDOOP_JAR}" randoop.main.Main gentests
     --testclass=org.jsoup.nodes.Element --output-limit=500
```

(Of course, you need to replace `myclasspath` with your complete classpath).

After generating the tests, create a Java file named `AutomatedTest.java` inside the package `org.jsoup.mytests` to include your generated tests. Make sure that your tests in `AutomatedTest.java` can be successfully executed via "mvn test" and achieve the desired coverage. Additionally, make sure that your `AutomatedTest.java` file declares the correct package, `org.jsoup.mytests`, at the top of the file.

Common Problems

- If you are using a Windows machine, you may need to use ";" for your classpath as the path separator instead of ":".
- For the test generation phase, you need to add the `target/classes` directory (i.e., the directory containing the compiled bytecode generated from the source code) to your classpath so that Randoop has access to the compiled bytecode of Jsoup for test generation.
- If you see something like the following error, you need to recompile the `jsoup` project to run Randoop. The reason of such an error is that the Clover coverage collection tool (applied earlier to solve Task 1) modified the compiled bytecode for Jsoup to collect code coverage (e.g., adding access to `com_atlassian_clover.TestNameSniffer`), and you want to recompile it to get a fresh copy of the compiled bytecode for Jsoup before proceeding.

  ```
  Could not load class org.jsoup.nodes.Element:
  Unable to load class "org.jsoup.nodes.Element" due to exception:
  java.lang.ClassNotFoundException:
  com_atlassian_clover.TestNameSniffer
  ```

3 Deliverables

You are expected to upload a zip file including only the `ManualTest.java` and `AutomatedTest.java` files you created (**no directories** please). The name of the zip file should be your NetID. In other words, the hierarchy of your submission zip file should be like this (replace `YourNetID` with your NetID):

```
YourNetID.zip
|-- ManualTest.java
|-- AutomatedTest.java
```

To create such a zip file, you can run the following command in your terminal:

```
zip YourNetID.zip ManualTest.java AutomatedTest.java
```

The autograder will copy your zip file into the `src/test/java/org/jsoup/mytests` directory, unzip it, and execute the tests. Failing to follow this structure may result in losing all the points.

Warning: you may lose all your points if you violate the following rules:

- Please make sure that your zip file is named with your NetID and only includes the specified files: **DO NOT include any directories in the zip file**.
- Please **DO NOT use any absolute paths** in your solution since your absolute paths will not match our grading machines. Also, please **only use "/"** as the file separator in your relative paths (if needed for this MP) since Java will usually make the correct conversions if you use the Unix file separator "/" in relative paths (for both Windows and Unix-style systems).
- Please **DO NOT fork any assignment repo** from our GitHub organization or share your solution online.
- Please make sure that you **follow the detailed instructions** to use the specified packages/-names. The autograder will only look for the specified paths; thus you may lose all points if you fail to use the same paths (even if you can generate all required tests).

4 Grading rubric

Overall score (5pt)

- Manual tests (3pt)
  - If you achieve 100% coverage for method `parseFragment(...)`, you get 3pt
  - Else if you achieve 95% coverage, you get 2pt
  - Else if you achieve 90% coverage, you get 1pt
  - Else you get 0pt
- Automated tests (2pt)
  - If you achieve 97% coverage for class `org.jsoup.nodes.Element`, you get 2pt
  - Else if you achieve 95%, you get 1pt
  - Else you get 0pt