Testing Template Modification Description Documentation

CISC 327 Fall 2019

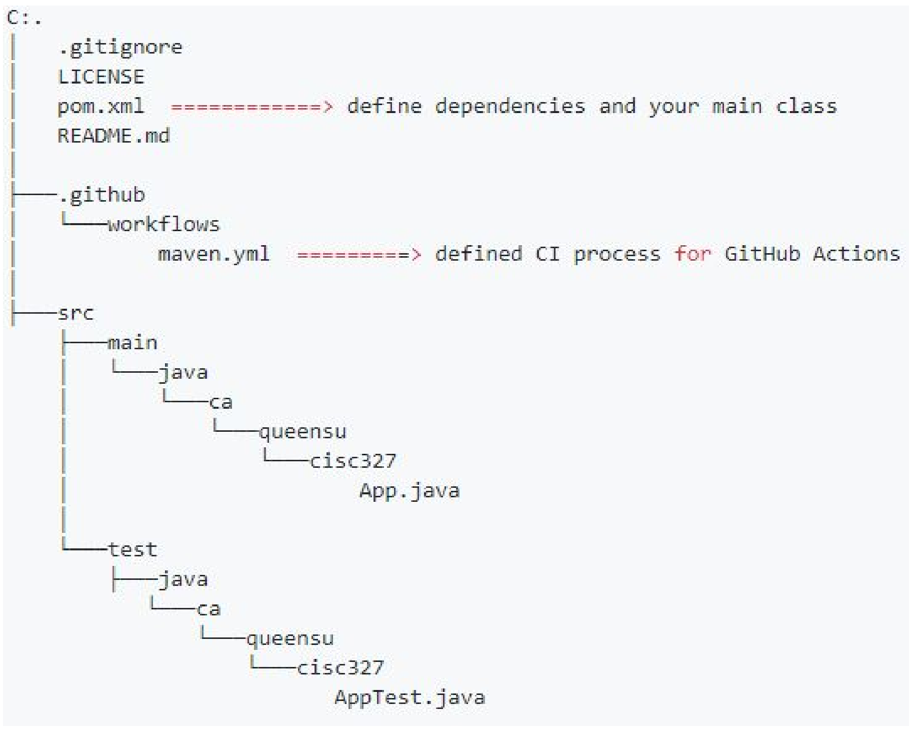
Assignment 3

Instructor: Dr. Stephen Ding
Contributors:

Denny,	Matt	(16mmmd1)
Laxdal,	Kai	(16kibl)
Littlefield,	Brent	(16bml1)
Liu,	Tong	(9tl14)

For our testing procedure we mainly followed the general outline for testing which is laid out at the following link, https://github.com/CISC-CMPE-327/CI-Java-Maven. Since we are developing our interpretation of Quinterac using JAva we will be implementing our testing using JUnit and specifically the Maven framework.

The folder structure we used to conduct the testing of our program can be seen below.



```
C:.
│   .gitignore
│   LICENSE
│   pom.xml   ============> define dependencies and your main class
│   README.md
│
├───.github
│   └───workflows
│           maven.yml   =========> defined CI process for GitHub Actions
│
├───src
    ├───main
    │   └───java
    │       └───ca
    │           └───queensu
    │               └───cisc327
    │                       App.java
    │
    └───test
        ├───java
           └───ca
               └───queensu
                   └───cisc327
                          AppTest.java
```

Inside of our AppTest.java file we have followed the format of the file to spec. Our testing file contains the AppTest which will help us conduct all of our test cases we need to take account of. Inside the AppTest class we have all of our tests to make sure that the Front-end of Quinterac is behaving properly and all of the required features that the front end must have are working the way they are intended to do. The general outline of our test cases are as follows:

```
@Test
 public void R1T1() throws Exception {
        runTest(Arrays.asList("logout", "--quit"),
                        Arrays.asList("Please enter a transaction: “
                        , "Cannot logout before login.", "Please enter a transaction: "));
}
```

In the AppTest file we also needed a way to run all of these tests, so we needed to create a “RunTest” method that would act upon each of the test cases we have created. We needed to create two “RunTest” methods as there was a need to check of the expected terminal output was as expected or if the output file created contained the expected information. The two method headers that were created are as follows:

```
runTest(List<String> terminal_input, List<String> expected_terminal_tails)
runTest(List<String> terminal_input, List<String> expected_terminal_tails, String
expected_output_file)
```

For some of the requirements we had come up with in Assignment 1 required that we make sure that the inputted files into the frontEnd of Quinterac are valid themselves just so we do not have any issues later on when a user might want to perform a transaction of some sort. The fileTest method is very similar to the RunTest methods but checks if the files are formatted valid and only contain correct entries. The following method header is for the method we would call when we want to conduct a test to check if the input files valid.

```
fileTest(String inputFile, List<String> expected_terminal_tails)
```

Overall, we stuck closely to the format outlined on the GitHub repository linked above in order to conduct our testing, but adapted the format to form to our architecture and how we designed the overall structure of the front-end of Quinterac.