TEAM #18

Testing Document

TEAM MEMBERS

- Naveen Muralidhar Prakash Person# 50208032
- Barath Eswer Nagasubramaniyan Person# 50207356
- Khushbu Mittal Person# 50169099
- Jiawei Zhao Person# 50207069
- Vanshika Nigam Person# 50208031

Unit Testing

Simple loop test

```
//Loop not executed

@Test
public void shouldUploadNoTests() {

    // List containing zero test case file
    ArrayList<File> testList = new ArrayList<>();

    // Create a professor class instance
    Professor professor = new Professor();

    boolean uploadStatus = professor.addTestcase(testList);

    int testCount = professor.getTestCount();

    assertEquals(testCount, 0);
}


// Loop running once

@Test
public void shouldUploadOneTests() {

    // List containing ONE test case file
    ArrayList<File> testList = new ArrayList<>();

    testList.add(testFile);

    // Create a professor class instance
    Professor professor = new Professor();

    boolean uploadStatus = professor.addTestcase(testList);

    int testCount = professor.getTestCount();

    assertEquals(testCount, 1);
}



// Loop executing multiple times

@Test
public void shouldUploadAllTests() {

    // List containing ONE test case file
    ArrayList<File> testList = new ArrayList<>();

    testList.add(testFile);

    // Create a professor class instance
    Professor professor = new Professor();

    boolean uploadStatus = professor.addTestcase(testList);

    int testCount = professor.getTestCount();

    assertEquals(testCount, 28);
}
```

Boundary analysis

```java
// Boundary analysis
@Test

public void shouldInBoundary() {

    // Test divide by 0
    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    // Returns method percentage
    float percentage = ccCalculator.calculate_method_coverage(this.zero_line_method());

    assertEquals(Math.round(percentage), 0);
}
```

Bad-data

```java
// Bad Data

@Test(expected = CodeCoverageException.class)

public void hasBadData() {

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_method_coverage(this.not_valid_method());
}
```

Good-data

```java
//Good Data

@Test
public void shouldContainGoodData() {

    // Create a class with known coverage %
    String className = "/path/ValidClass.class";

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_class_coverage(className);

    assertEquals(94.5, percentage, 0.0);
}
```

All-Branch coverage

```
// All branch coverage

@Test(expected = CodeCoverageException.class)
public void shouldReturnNotFoundInClassCoverage() {

    String className = "/path/Invalid_Class.class";

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_class_coverage(className);
}
```

```
@Test(expected = CodeCoverageException.class)

public void shouldReturnNoMainInClassCoverage() {

    String className = "path/No_Main_Class.class";

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_class_coverage(className);
}
```

```
@Test
public void shouldReturnClassCoveragePercent() {

    String className = "path/Valid_Class.class";

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_class_coverage(className);

    assertEquals(93.5, percentage, 0.0);
}
```

Algorithm:

// Calculates class coverage percentage

calculate_class_coverage (String className)

1. If className is not valid
    a. throw CodeCoverageException
2. Else if className is not exposed
    a. throw CodeCoverageException
3. Else
    a. calculate class coverage percentage
    b. return percentage

Explanation:

The all branch coverage algorithms implements the calculate_class_coverage() which has the following conditions:

- If the class name parameter is invalid, it throws a custom exception
- If the parameter class doesn't contain a main method or if the class is not exposed, it throws an exception
- When a valid class name is passed it calculates and returns the code coverage in percentage
- All the above branches are tested with individual JUnit methods

Integration testing

Integration test 1:

```
// Integration testing

// Integration - 1

@Test
public void checkClassCodeCoverage() {

    // Combining Professor and CodeCoverage classes
    String className = "/path/Valid_Class.class";

    Professor professor = new Professor();

    boolean uploadStatus = professor.addTestcase(testList);

    assertEquals(uploadStatus, true);

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_class_coverage(className);

    float reportValue = ccCalculator.generateReport(percentage);

    assertEquals(93.0,reportValue, 0.0);
}
```

Purpose: To check if the calulate_method_coverage() method returns the method coverage percentage

Methods used in this test: add_testcases(), calulate_method_coverage(), generate_report()

Step-by-step directions to perform this test:

- The professor updates the test cases for the current project in add_testcases()
- The professor calls the calulate_method_coverage() for every submission
- The calulate_method_coverage() returns the coverage percentage to generate_report()
- The generate_report() method returns the method coverage in percentage

Step-by-step expected results:

- If the test cases file is submitted successfully by the professor, the output is “File Uploaded!”
- Method coverage - 93.4%

Integration test 2:

```
// Integration - 2
@Test
public void checkMethodCodeCoverage() {

    // Combining Student and CodeCoverage classes

    String location = "path/ClassName.java";

    Student student = new Student();

    boolean uploadStatus = student.submit_file(location);

    assertEquals(uploadStatus, true);

    CodeCoverageCalculator ccCalculator = new CodeCoverageCalculator();

    float percentage = ccCalculator.calculate_class_coverage(this.valid_method());

    float reportValue = ccCalculator.generateReport(percentage);

    assertEquals(94.5, reportValue, 0.0);

}
```

Purpose: To check if the calulate_class_coverage() method returns the class coverage

Methods used in this test: submit_file(), calulate_class_coverage(), generate_report()

Step-by-step directions to perform this test:

- The student submits the .java file
- The professor calls the calulate_class_coverage() for the submission
- The calulate_class_coverage() methods returns the coverage percentage to generate_report()
- The generate_report() method returns the class coverage in percentage

Step-by-step expected results:

- If the file is submitted successfully by the student, the output is “File Uploaded!”
- Class coverage - 94.5%

Validation Tests

Validation Test 1

Purpose
To make sure the TA can update tests and manage the student enrollment in the course grading system

Methods used in this test

- login()
- update_tests()
- add_user()
- remove_user()

Step-by-step directions to perform this test:

- The TA logs in with the valid credentials
- The TA updates the test cases from professor to code coverage system
- The TA adds a new student to the grading system using add_user(Student)
- The TA removes a student from the grading system using remove_user(Student)

Step-by-step expected results:

- If the tests are uploaded successfully, the output is true
- The newly added student can successfully login
- The student removed from the database cannot login to the system

Validation Test 2

Purpose
To make sure the student can submit the file, run the test cases and view the submissions

Methods used in the test

- login()
- submit_file()
- calculate_method_coverage()
- calculate_class_coverage()
- view_submissions()
- generate_report()

Step-by-step directions to perform this test:

- The student logs in with valid credentials
- The student submits the java file using the submit_file() method
- The CodeCoverageCalculator runs the test cases against the student's submission
- The CodeCoverageCalculator system generates a report of coverage percentage
- The student can view file submissions using view_submissions()

Step-by-step expected results:

- If the file is submitted successfully by the student, the output is "File Uploaded!"
- Method coverage - 93.5%
- Class coverage - 94.5%