STEVE FINAL REPORT

CSCI 362

STEVE: Michael O’Cain, Chris Tucker, Connor Yates

TABLE OF CONTENTS

CHAPTER 1

PROJECT CHOICE: GLUCOSCIO

Our team chose Glucosio as our project. Glucosio is an Android based app that helps with the management and organization of data involving patients with diabetes.

The app requires both Gradle and Android studio to run. One of our team members was successful in getting the project to load into the Android Studio environment with Gradle. After which, however, the test files could not be run, as the dependencies were not able to be loaded in. Compiling and building the project netted similar results. Whether this was user error or due to the fact that this project was last updated almost 3 years ago is unknown to our team, but we suspect the latter.

CHAPTER 2

TESTING PROCESS

Our script will open up the Glucosio Converter file and run tests to determine the accuracy of the calculations present in the former. The methods in question, round(), kgToLb(), lbToKg, glucoseToA1C(), and a1cToGlucose(), all accept doubles as arguments and output doubles after performing a calculation on the input. Our script will import the test case files from our test case file directory, use the method name in each test case file in a switch statement, and call use the arguments in the test case file for the method calls which will be tied to their respective switch statement path. The method will be returned to a temporary variable which will compare itself to the expected result, which again, will be parsed from the test case file. For each test case the result will be reported and printed to a text file for easy review. Any test failures will be recorded.

REQUIREMENTS

Calculations are accurate
Type casting works as intended (Or fails as expected)
Wrong number of arguments fails

TEST CLASS

GlucosioConverter.java

TEST RESULTS

Test results for each test case will be saved to an easy to read report (reports.html) including any test failures and the reasons for said failures.

CONSTRAINTS

Lack of previous scripting experience for all team members
Innate instability and unreliability of the app we're testing

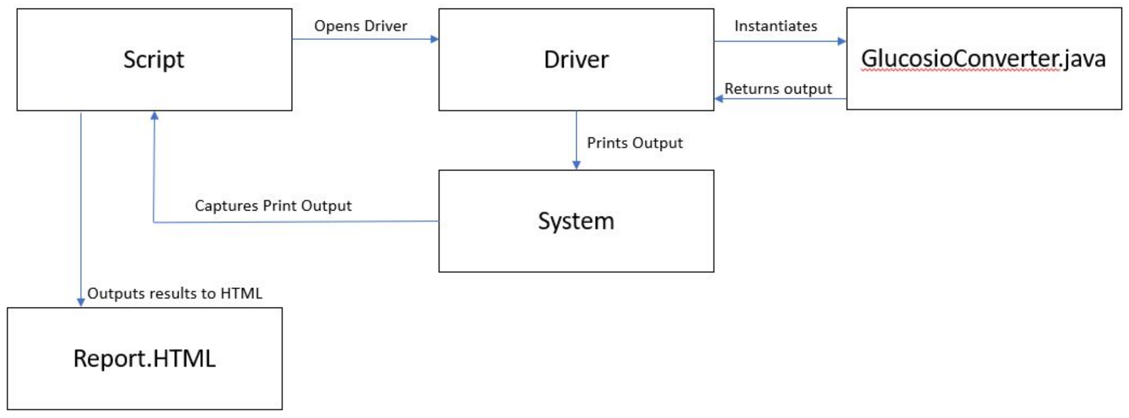
TEST CASE TEMPLATE
Test ID
Method
Argument
Oracle
Description

CHAPTER 3

HOW TO RUN:

1. Download repository from github https://github.com/csci-362-02-2019/Steve.git
2. Run python script from command line using command:
    - (linux/mac) (original) python ~/.../Steve/TestAutomation/scripts/runALLTests.py
    - (Faults) python ~/.../Steve/TestAutomation/scripts/runALLTestsFaults.py
    - (windows) (original) C:...\Steve\TestAutomation\scripts\runALLTests.py
    - (Faults) C:...\Steve\TestAutomation\scripts\runALLTestsFaults.py
3. An HTML report will be saved within Steve/TestAutomation/reports as reports.html and opened

ARCHITECTURE:



SCRIPT STRUCTURE:

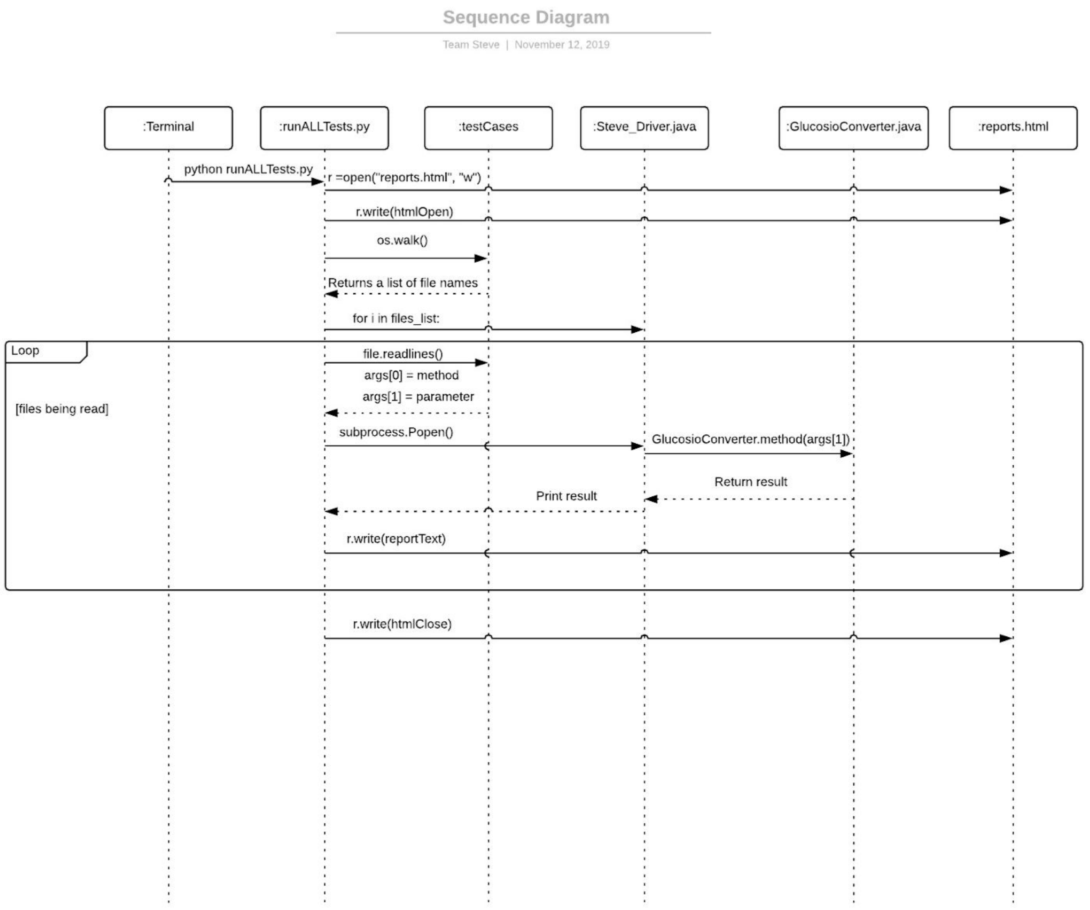
The script opens up the TestCases directory and save each test case file as a string into a list. This list is then iterated through, with each element being parsed and sent into the driver to be tested. The Driver will use System.out.print(return); after completing the specified method. Our script is set up to capture all print statements executed by the driver. After saving the result, our script will compare it to the oracle, which is saved into the test case file and parsed out of the string array with the method and input. It will then write to the HTML whether the test case passed or failed, as well as the result vs the expected result.

CHAPTER 4

DRIVER STRUCTURE:

The driver receives one argument, an array containing the method name and the input, both in string form. The driver then uses the first element of the array to call a switch statement, which is set up for all 5 of our testable methods, as well as a default in the event that the user sends bad data into the driver. It then parses the second index of the array into a double, which is the data type all of our testable methods use. The desired method is run and the return value is saved into a variable, which is then printed out. As stated before our script is set up to capture all print statements made by our driver, meaning it has received the data at this point and our driver closes.

The driver was designed from the beginning to be able to accept any of the five methods as an argument (args[0]) to be placed into a switch statement along with its own argument (args[1]) and then be evaluated and return the result. Due to this design, the driver was already complete with Deliverable #3 and finishing Deliverable #4 was a matter of creating twenty additional test case files (five files each for the remaining four methods) and placing into the “testCases” folder. Once the files were placed into the folder, the script was able to pass their arguments along to the driver and capture their results to be placed into the reports.html file in the “reports” folder. Below is our sequence diagram.



CHAPTER 5

FAULT IMPLEMENTATION

The five methods we tested are round(), glucoseToMgDl(), glucoseToMmolL(), glucoseToAlC(), and alcToGlucose(). The latter three all call round(), so when we broke round() by changing the round so that it rounds down on a half value. For glucoseToMgDl(), we changed the variable that the parameter is multiplied by to be a negative number instead of positive, resulting in all of the tests failing. For glucoseToMmolL(), we changed the call to round() to round to 3 decimal places instead of 2. This resulted in test ids 11 and 12 passing but 13, 14, and 15 failing. For glucoseToAlC(), we changed the value of one of the equation variables from 46.7 to 46.6. This caused test ids 18 and 19 to fail but every other test passed. For alcToGlucose(), we increased the value of one of the variables from 26.7 to 26.8 and it caused test ids 21 to 25 to fail, all other test ids passed. Below are the equations of the methods we tested and the changes we made for each one.

Original:

```
public static double round(double value, int places) {

        if (places < 0) throw new IllegalArgumentException();

        BigDecimal bd = BigDecimal.valueOf(value);

        bd = bd.setScale(places, RoundingMode.HALF_UP);

        return bd.doubleValue();

        }
```

Changed: Round rounds down on half now

```
public static double round(double value, int places) {

        if (places < 0) throw new IllegalArgumentException();

        BigDecimal bd = BigDecimal.valueOf(value);

        bd = bd.setScale(places, RoundingMode.HALF_DOWN);

        return bd.doubleValue();

        }
```

Original:

```
public static double glucoseToMgDl(double mmolL) {

        return mmolL * MG_DL_TO_MMOL_CONSTANT;

        }
```

Changed: MG_DL_TO_MMOL_CONSTANT is now negative

```
public static double glucoseToMgDl(double mmolL) {

        return mmolL * -MG_DL_TO_MMOL_CONSTANT;

        }
```

Original:

```
public static double glucoseToMmolL(double mgDl) {

            return round(mgDl / MG_DL_TO_MMOL_CONSTANT, 2);

            }
```

Changed: Second parameter for round is now 3 instead of 2

```
public static double glucoseToMmolL(double mgDl) {

            return round(mgDl / MG_DL_TO_MMOL_CONSTANT, 3);

            }
```

Original:

```
public static double glucoseToA1C(double mgDl) {

            // A1C = (Average glucose + 46.7) / 28.7

            return round((mgDl + 46.7) / 28.7, 2);

            }
```

Changed: 46.7 is now 46.6

```
public static double glucoseToA1C(double mgDl) {

            // A1C = (Average glucose + 46.7) / 28.7

            return round((mgDl + 46.6) / 28.7, 2);

            }
```

Original:

```
public static double a1cToGlucose(double a1c) {

            // Average glucose = (A1C * 28.7) -46.7

            return round((a1c * 28.7) - 46.7, 2);

            }
```

Changed: 28.7 is now 28.8

```
public static double a1cToGlucose(double a1c) {

        // Average glucose = (A1C * 28.7) -46.7

        return round((a1c * 28.8) - 46.7, 2);

        }
```

CHAPTER 6

TEAM EVALUATION

The team overall performed well and worked efficiently. Each member contributed sufficiently to the project. The project was successful. We learned how to use linux, html formatting, gradle building, and learned about scripts and drivers.

ASSIGNMENT EVALUATION

The deliverables were relatively straightforward, requiring teamwork at every step. They seemed daunting at first, but in retrospect were not nearly as insurmountable as they initially seemed. The first was the hardest by far, but we know now that that was by design. Overall the order and timing of the deliverables made sense and don’t warrant any significant criticism beyond our initial frustration.