Testing OpenMRS

Meg Krawczyk, Mitch Suzara, Itzayana Carrillo

Department of Computer Science, College of Charleston

CSCI 362: Software Engineering

Dr. Jim Bowring

December 4, 2020

Table of Contents

Introduction

For our Software Engineering course, we have been tasked with creating a Testing framework for an open-source project of our choice. In the following chapters, we will document our experience and the process of creating the assigned testing framework. We will divide the project into five parts each part will be written into a chapter. We will then end with a sixth chapter that will include our final thought and overall experience in the project. Our team is named Team3 and it has three members: Meg Krawczyk, Mitch Suzara, Itzayana Carrillo. We are all very excited to work on this project and to document our experience.

Chapter 1

<u>Our Project</u>

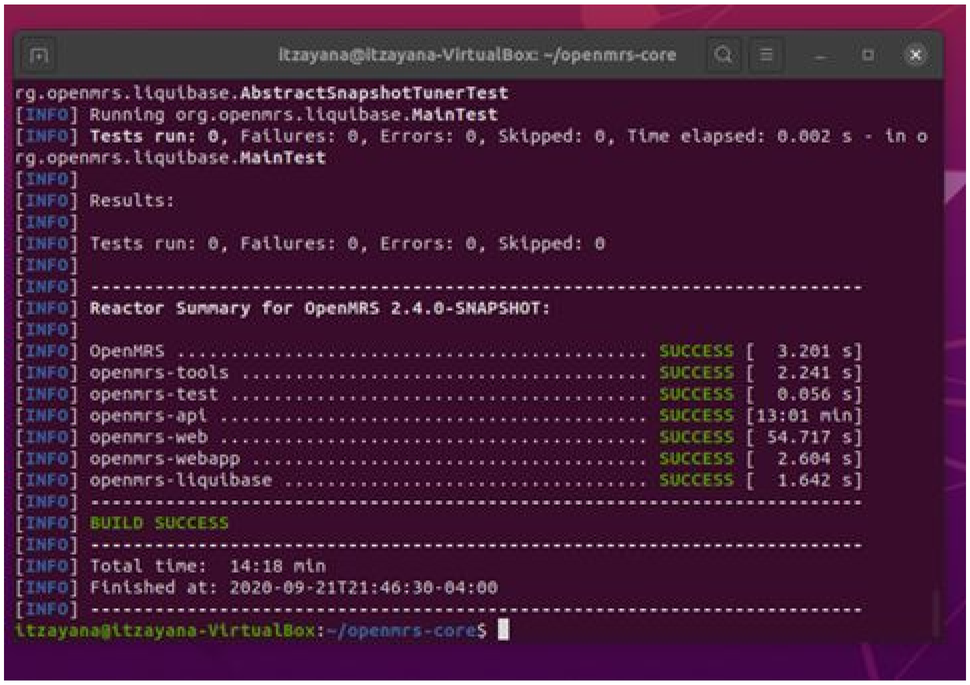
Choosing a project to test was difficult for our team, we looked through, downloaded, and attempted to compile seven to eight projects before finally settling on OpenMRS[1]. To see documentation on our experience with other projects check out our Evaluation report page on our Github Wiki[2]. We decided to go with OpenMRS for a few reasons. First, it is a program that provides an electronic medical record system and its mission is to, “Improve health care delivery in resource-constrained environments by coordinating a global community to create and support this software.”[3] We thought this was a great cause and wanted to contribute to it. Secondly, this was one of the few projects we were able to compile and build making it the best choice for our team.

<u>Building and Compiling OpenMRS</u>

The documentation for building OpenMrs is very clear and easy to execute. Before we did anything, we had to make sure we had the prerequisites to be able to build and compile OpenMRS. In this case, we needed to make sure we had Java JDK, a minimum of Java 8, and

Maven installed on our virtual machines. Once we had the prerequisites installed, we were free to begin the building and compiling process.

The first step was to clone the OpenMrs repository to our local computer by using the command "*git clone https://github.com/openmrs/openmrs-core.git*". Once the repository was on our computer, we could start compiling. To do this we used the command "*cd open-core*" to enter into the open-core directory. From there we built the project by running the command *"mvn clean package."* This command built OpenMrs and ran a few of its tests as seen in Figure 1.



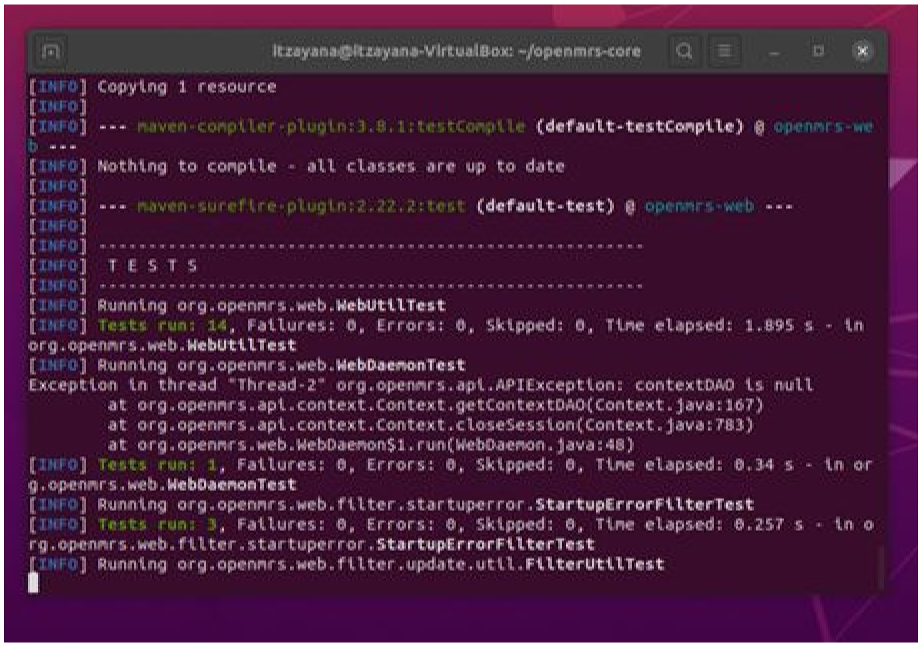
```
rg.openmrs.liquibase.AbstractSnapshotTunerTest
[INFO] Running org.openmrs.liquibase.MainTest
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.002 s - in o
rg.openmrs.liquibase.MainTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 0, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary for OpenMRS 2.4.0-SNAPSHOT:
[INFO]
[INFO] OpenMRS ............................................ SUCCESS [  3.201 s]
[INFO] openmrs-tools ...................................... SUCCESS [  2.241 s]
[INFO] openmrs-test ....................................... SUCCESS [  0.056 s]
[INFO] openmrs-api ........................................ SUCCESS [13:01 min]
[INFO] openmrs-web ........................................ SUCCESS [ 54.717 s]
[INFO] openmrs-webapp ..................................... SUCCESS [  2.604 s]
[INFO] openmrs-liquibase .................................. SUCCESS [  1.642 s]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  14:18 min
[INFO] Finished at: 2020-09-21T21:46:30-04:00
[INFO] ------------------------------------------------------------------------
itzayana@itzayana-VirtualBox:~/openmrs-core$ 
```

Figure 1.1: Successful Build of OpenMRS

In addition to compiling and building OpenMrs, we were also able to get it to deploy to Firefox. We simply ran the commands *"cd openmrs-core/webapp"* and *"mvn jetty:run."* This opened a localhost's page that could be accessed by typing this site handle in the search bar "*localhost:8080/openmrs.*"

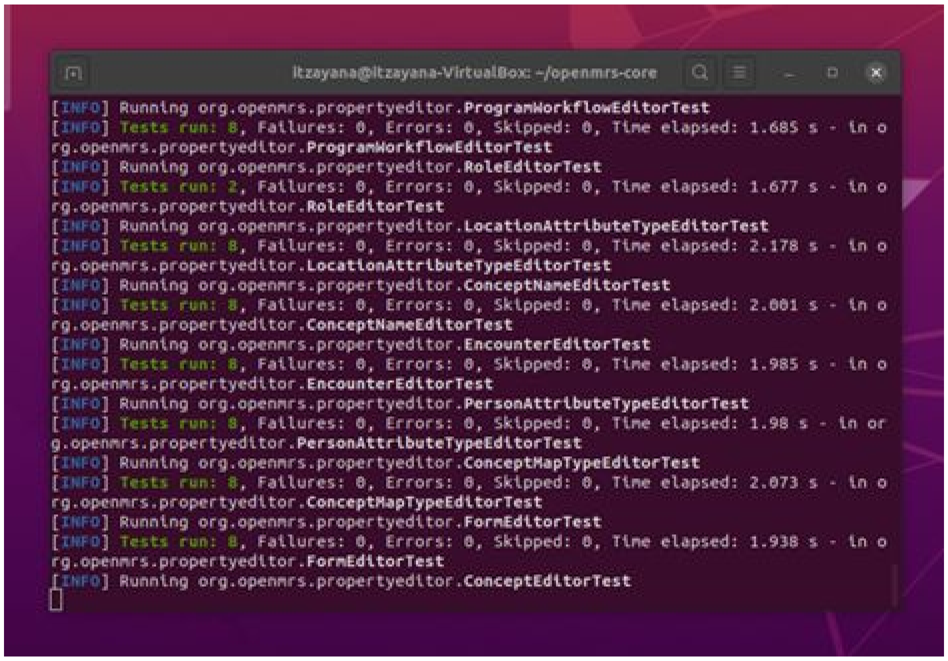
<u>**Running Existing Tests in OpenMRS**</u>

Running the existing tests in OpenMRS turned out to be a lot simpler than we had previously thought. Originally, we were trying to log into OpenMRS on the localhost site, but we kept running into issues with the connection between OpenMRS and MySQL. Luckily, we tried testing from the terminal by changing the directory to the OpenMRS directory and using Maven "*mvn*" test. After a little bit of trial and error, we were able to build and run tests on OpenMRS as seen in Figures 2 and 3. However since we realized we should be testing in the terminal late in our project search, we weren't able to explore the tests thoroughly.



```
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.8.1:testCompile (default-testCompile) @ openmrs-we
b ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.2:test (default-test) @ openmrs-web ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.openmrs.web.WebUtilTest
[INFO] Tests run: 14, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.895 s - in
org.openmrs.web.WebUtilTest
[INFO] Running org.openmrs.web.WebDaemonTest
Exception in thread "Thread-2" org.openmrs.api.APIException: contextDAO is null
        at org.openmrs.api.context.Context.getContextDAO(Context.java:167)
        at org.openmrs.api.context.Context.closeSession(Context.java:783)
        at org.openmrs.web.WebDaemon$1.run(WebDaemon.java:48)
[INFO] Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.34 s - in or
g.openmrs.web.WebDaemonTest
[INFO] Running org.openmrs.web.filter.startuperror.StartupErrorFilterTest
[INFO] Tests run: 3, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.257 s - in o
rg.openmrs.web.filter.startuperror.StartupErrorFilterTest
[INFO] Running org.openmrs.web.filter.update.util.FilterUtilTest
```

Figure 1.2: Testing OpenMrs



```
[INFO] Running org.openmrs.propertyeditor.ProgramWorkflowEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.685 s - in o
rg.openmrs.propertyeditor.ProgramWorkflowEditorTest
[INFO] Running org.openmrs.propertyeditor.RoleEditorTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.677 s - in o
rg.openmrs.propertyeditor.RoleEditorTest
[INFO] Running org.openmrs.propertyeditor.LocationAttributeTypeEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.178 s - in o
rg.openmrs.propertyeditor.LocationAttributeTypeEditorTest
[INFO] Running org.openmrs.propertyeditor.ConceptNameEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.001 s - in o
rg.openmrs.propertyeditor.ConceptNameEditorTest
[INFO] Running org.openmrs.propertyeditor.EncounterEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.985 s - in o
rg.openmrs.propertyeditor.EncounterEditorTest
[INFO] Running org.openmrs.propertyeditor.PersonAttributeTypeEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.98 s - in or
g.openmrs.propertyeditor.PersonAttributeTypeEditorTest
[INFO] Running org.openmrs.propertyeditor.ConceptMapTypeEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 2.073 s - in o
rg.openmrs.propertyeditor.ConceptMapTypeEditorTest
[INFO] Running org.openmrs.propertyeditor.FormEditorTest
[INFO] Tests run: 8, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 1.938 s - in o
rg.openmrs.propertyeditor.FormEditorTest
[INFO] Running org.openmrs.propertyeditor.ConceptEditorTest
```

Figure 1.3: Testing OpenMrs

Final Thoughts and Experience Report

This experience has been difficult so far. Our biggest issue as a team has been knowing what exactly we were to be looking for. Our first major struggle was looking for a project with clear building instructions, and even when we thought we understood the instructions it seemed every time we tried to build and compile a new project we seemed to run into more issues and errors. Our second struggle seemed to be knowing what a compiled and built program looked like. We seemed to think that to successfully build and compile a project we needed to get it up and running in a browser. While this is certainly a step in the right direction, it seems we may have been overthinking it a bit. With the use of Maven mvn test we were able to build and test

OpenMRS. Deliverable One has proven to be a challenge for our team, but we have decided to continue with OpenMRS for Deliverable Two.

---

[1] https://github.com/openmrs/openmrs-core

[2] https://github.com/csci-362-02-2020/Team3/wiki/Evaluation-Report

[3] https://openmrs.org/

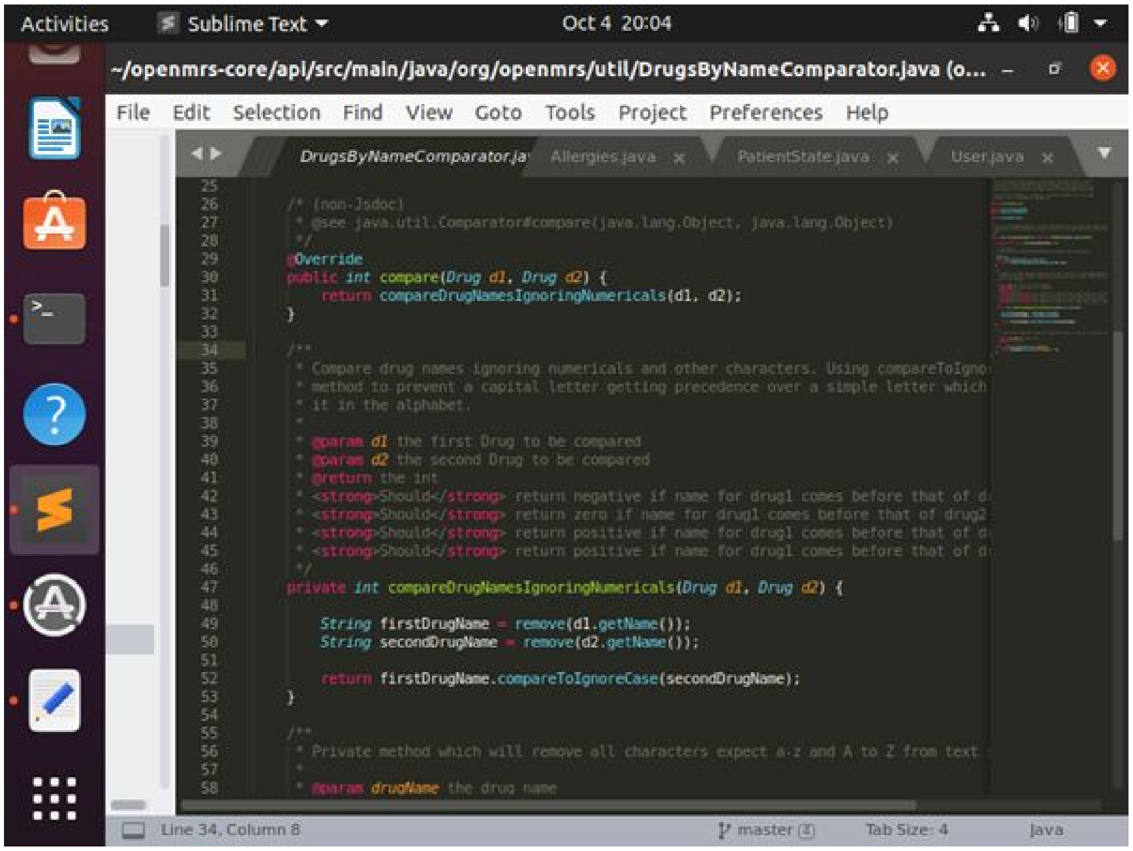
Chapter 2

Method Selection

OpenMRS had plenty of files for our team to search through to find 5 methods to test. The sheer number of Java files was very overwhelming at first and finding methods fit for testing proved to be difficult. Since OpenMrs is more or less a Database that stores medical information it is mostly made up of getter and setter methods, but as we continued to parse through the different classes, we eventually found several methods that we considered fit for testing. As of now we are looking at more than the required amount of methods since a lot of these classes have several dependencies, and we are not sure what will be manageable and what will not be. For this Deliverable we will be looking at the compare() method from the DrugsByNameComparator class.

Method

The compare() method is DrugsByNameComparitor takes two objects of type Drug and compares their names by using the compare interface. An image of the code is below. This method should take two objects of type Drug, get each one's name, remove any numbers from each name, and then compare them alphabetically. If the first drug should come before the second, then the method should return a positive integer. If the second drug should come first, then it should return a negative integer. If both drugs have the same name, then it should return a zero. Since this method is more or less just comparing two strings, our team may tweak the code so that compare() takes two strings instead of two Drug objects for ease of testing and to avoid any unnecessary dependencies.



```java

    /* (non-Jsdoc)
     * @see java.util.Comparator#compare(java.lang.Object, java.lang.Object)
     */
    @Override
    public int compare(Drug d1, Drug d2) {
        return compareDrugNamesIgnoringNumericals(d1, d2);
    }

    /**
     * Compare drug names ignoring numericals and other characters. Using compareToIgno
     * method to prevent a capital letter getting precedence over a simple letter which
     * it in the alphabet.
     *
     * @param d1 the first Drug to be compared
     * @param d2 the second Drug to be compared
     * @return the int
     * <strong>Should</strong> return negative if name for drug1 comes before that of d
     * <strong>Should</strong> return zero if name for drug1 comes before that of drug2
     * <strong>Should</strong> return positive if name for drug1 comes before that of d
     * <strong>Should</strong> return positive if name for drug1 comes before that of d
     */
    private int compareDrugNamesIgnoringNumericals(Drug d1, Drug d2) {

        String firstDrugName = remove(d1.getName());
        String secondDrugName = remove(d2.getName());

        return firstDrugName.compareToIgnoreCase(secondDrugName);
    }

    /**
     * Private method which will remove all characters expect a-z and A to Z from text
     *
     * @param drugName the drug name
```

Figure 2.1: Method 1

Test Plan

- **Testing Process**
  - We will create an automated testing framework that will run through the 25 test cases our group has created for OpenMRS. We will be testing five different methods from OpenMRS, each method will have 5 different test cases. After the testing is complete the framework will parse the results, and save them in an HTML file.
- **Requirements Traceability**

  - Each test case will have a unique test ID and unique requirements, which will allow for easy traceability of all the test cases. The requirements for each method are presented in the table below.
- **Items Tested**
  - As stated before, we have selected more methods than needed to ensure we can find a method that doesn't have too many dependencies or that we can tweak enough to keep the logic the same but make it easier to test. Below are the classes and methods we plan to test.

Class	**Method**	**Requirements**
Allergies	Add(Allergy a1)	adds an object of Type Allergy to a list
DateUtil	truncateToSeconds(Date date)	truncates the date to the second
DrugByNameComparator	compare(Drug d1, Drug d2)	Compares the names of two drugs
DrugOrder	hasSameOrderableAs(DrugOrder d1)	Returns true if the drugs have they say orderable
Graph	topographicalSort()	Sorts the graph
OrderUtil	isType(OrderType o1, OrderType o2)	Checks to see if the two orders have the same type
PateintState	/compareTo(PatientState p)	Compares the states of two patients
PersonMergeLogData	computeHashTable()	Creates a hash table of people
UserByNameComparator	compare(User u1, User u1)	Compare the name of two users

Figure 2.2: Method 1

- **Testing Schedule**
  - Deliverable 3: Automated Testing Framework. (11/05/20)
    - Extract methods and classes to be tested
    - Create a script that can automatically run several test cases
    - Create 5-10 test cases a week
  - Deliverable 4: Run 25 Test Cases and Pipe them to HTML File (11/17/20)
  - Deliverable 5: Test deliberate errors in the Code (11/24/20)

- Final Report and Presentation (12/03/20)

- **Test recording procedures**
  - After our driver runs all tests, the output will be sent to an HTML file and displayed in a web browser.
- **Hardware and software requirements**
  - Hardware: Computer
  - Software: VirtualBox, Linux, Terminal, Java, Github, and Git
- **Constraints**
  - Meeting Schedule: as a team, we have scheduling constraints. Each team member has a variable work and school schedule, which limits our availability to meet.
- **Unit tests**
  - For testing our methods, we have not yet created an automated testing script, but we have a few preliminary test cases in our Github Repository[1]. The basic information in them can be found in the graph below.
  - TestCase Key
    - TestID
    - Class name
    - Requirements
    - Method name
    - Summary
    - Inputs
    - Expected outputs
    - Driver
  - TestCase 1
    - 01
    - DrugsByNameComparator
    - This class will compare two strings and put them in alphabetical order

1

    - compare(String d1, String d2)
    - This test will test the compare() when d1 comes before d2 alphabetically
    - Allegra Dayquil
    - 3
    - Driver

- <u>TestCase 2</u>
    - 02
    - DrugsByNameComparator
    - This class will compare two strings and put them in alphabetical order
    - compare(String d1, String d2)
    - This test will test the compare() when d2 comes before d1 alphabetically
    - Dayquil Allegra
    - -3
    - Driver

- <u>TestCase 3</u>
    - 03
    - DrugsByNameComparator
    - This class will compare two strings and put them in alphabetical order
    - compare(String d1, String d2)
    - This test will test the compare() when d1 and d2 are the same
    - Allegra Allegra
    - 0
    - Driver

- <u>TestCase 4</u>
    - 04
    - DrugsByNameComparator
    - This class will compare two strings and put them in alphabetical order
    - compare(String d1, String d2)
    - This test will test the compare() when d1 comes before d2 alphabetically, but d2 has a number in it that would make it come first
    - 2Allegra 1Dayquil
    - -3
    - Driver

- <u>TestCase 5</u>
    - 05
    - DrugsByNameComparator
    - This class will compare two strings and put them in alphabetical order

- compare(String d1, String d2)
- This test will test the compare() when d1 is a string and d2 is null
- Allegra null
- Null pointer exception
- Driver

https://github.com/csci-362-02-2020/Team3/tree/master/testCases

Final Thoughts and Evaluation Report

We have learned a lot as a team, finding good testable methods has proven difficult since most of the methods in OpenMRS are setter and getter methods. We also did a team exercise to help learn how to compile a java program from the command line. We are still running into some errors when we try to run OpenMRS, namely with packages and other strange dependencies, hopefully, we can overcome that and continue with our testing.

Chapter 3

Architectural Framework

Our architectural framework consists of a control script runAllTests.sh. This script controls the flow of the testing being done, takes the results, and outputs them into an HTML file. Once the script runs, it will access all test cases and then match the test case to its driver. Once the drive completes the test the results will be directed back to the script which outputs the results to an HTML file that opens in a browser.

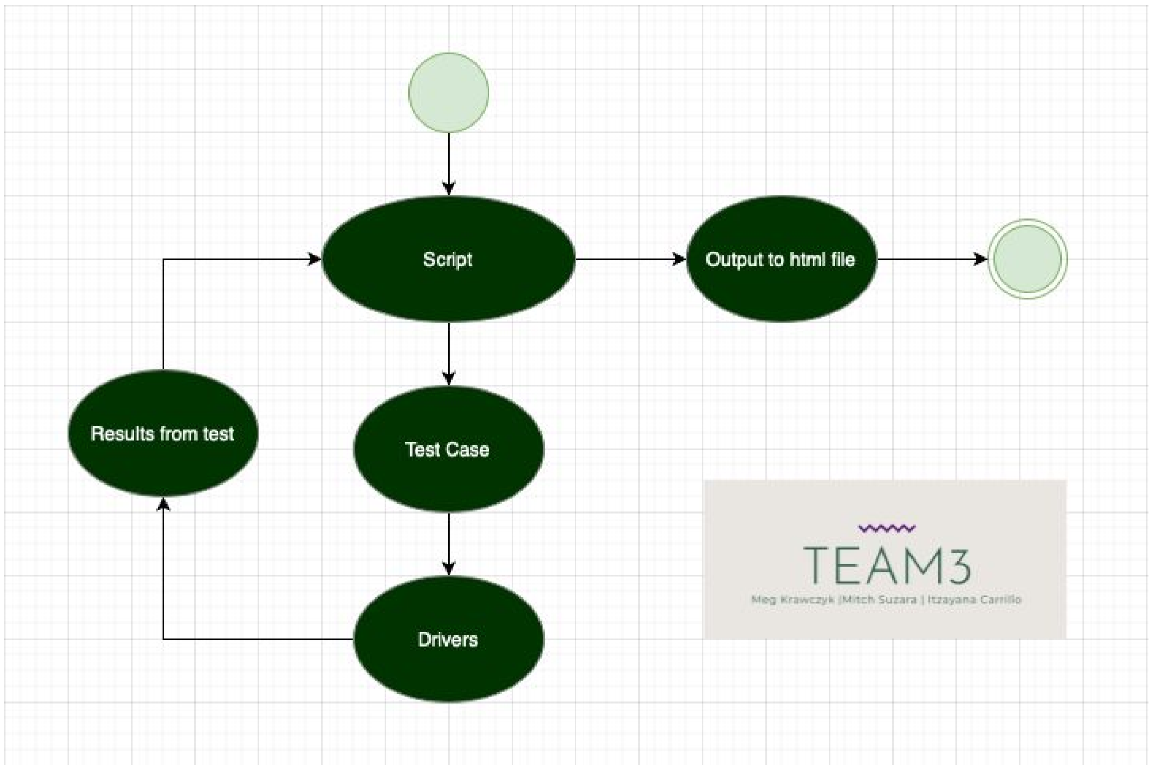


Figure 3.1: Sequence Diagram

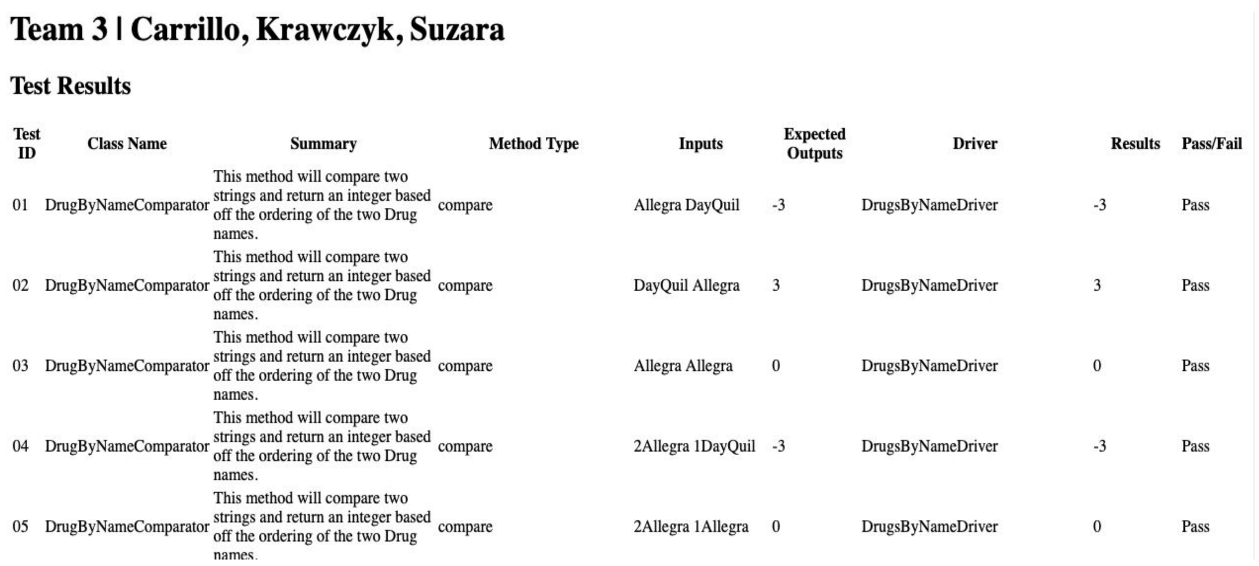
How to Run the Automated Test Framework

- Before staring be sure you have java 8 and git installed on your machine

- Go to our Github repository: https://github.com/csci-362-02-2020/Team3
- Click the download code button and copy the URL link of our repository.
- Open your terminal and cd into the folder you would copy our repository to
- Next, run the command *git clone "insert URL*" This will create a copy of our repository on your computer.
- Run the command *cd Team3/TestAutomation/*
- To run the script make the file executable
- Run script on Ubuntu terminal by entering the following:
    - `bash ./scripts/runAllTests.sh`
- HTML report will open on browser, displaying the test results

Test Output Example

Below is the expected output that will appear in the browser after executing our script. Each test is given a unique test ID. The only class being tested below is DrugsByNameComparator. This class is expected to determine if the inputs are in lexicographical order. If they are, the expected output is "3". If the inputs are the same then a "0" is expected to be returned. If they are not in order then the output should be "-3." There is a summary for each test which explains what the class is expected to do. Each test has a method type that is being tested. Each test is given unique inputs and expected outputs. The name of the driver is also included for each test case. Finally, the results of running the test are provided followed by a pass or fail. The pass or fail output is determined by comparing each expected result with the actual result. The image below shows that our five tests of the class DrugByNameComparator passed as well as the driver.



Team 3 | Carrillo, Krawczyk, Suzara

Test Results

Test ID	Class Name	Summary	Method Type	Inputs	Expected Outputs	Driver	Results	Pass/Fail
01	DrugByNameComparator	This method will compare two strings and return an integer based off the ordering of the two Drug names.	compare	Allegra DayQuil	-3	DrugsByNameDriver	-3	Pass
02	DrugByNameComparator	This method will compare two strings and return an integer based off the ordering of the two Drug names.	compare	DayQuil Allegra	3	DrugsByNameDriver	3	Pass
03	DrugByNameComparator	This method will compare two strings and return an integer based off the ordering of the two Drug names.	compare	Allegra Allegra	0	DrugsByNameDriver	0	Pass
04	DrugByNameComparator	This method will compare two strings and return an integer based off the ordering of the two Drug names.	compare	2Allegra 1DayQuil	-3	DrugsByNameDriver	-3	Pass
05	DrugByNameComparator	This method will compare two strings and return an integer based off the ordering of the two Drug names.	compare	2Allegra 1Allegra	0	DrugsByNameDriver	0	Pass

Figure 3.2: Outputs for Test Cases 1-5

Experience In Building Framework

1. Picking Methods

On our first round looking for methods to test we struggled. OpenMRS is more or less a database for medical records, which meant that there would be a lot of getter and setter methods, which for our purposes were not the best options for testing. In addition to this, these classes and methods had a lot of dependencies making them hard to extract while keeping the logic of the method intact. Upon further inspection, we were able to find a few static methods that had very few dependencies, except for few standard Java libraries. These methods were easily extracted and we decided to use them for our project. The final methods and classes we chose can be found in the chart below.

Class	Method
DateUtil	dateUtil()

OpenmrsUtil	convertToInteger()
OpenmrsUtil	containsUpperAndLowerCase()
OpenmrsUtil	containsOnlyDigits()
DrugByNameComparitor	DrugsByNameComparitor()

Figure 3.3: Chosen Methods

2. <u>Creating Test Cases</u>

Once we had our methods picked, we could start to make test cases for each method. Luckily, the methods we chose to test all had decent comments specifying what the results should be based on various types of inputs. The documentation in the methods made creating the test cases much easier.

One thing we learned about the drugsByNameComparitor() method is because it uses the compareTo method in the Java comparator class. Which means it will return a positive or negative result repenting upon the order in which the inputs are called. Meg was under the impression that it would return -1, 1, or 0. Those being the only numbers that it could produce. However, after researching into the compareTo() method we soon discovered that the method returned a positive or negative value, but not necessarily -1 or 1. Upon this discovery, we realized that our expected output was incorrect, and based on the compare method, it should return -3. There are examples of the test cases for the DrugsByNameComparitor() method below.

TestCase 1	**TestCase 2**	**TestCase 3**	**TestCase 4**	**TestCase 5**

01 DrugByNameComparator This method will compare two strings and return an integer based on the ordering of the two Drug names. compare Allegra DayQuil -3 DrugsByNameDriver	02 DrugByNameComparator This method will compare two strings and return an integer based on the ordering of the two Drug names. compare DayQuil Allegra 3 DrugsByNameDriver	03 DrugByNameComparator This method will compare two strings and return an integer based on the ordering of the two Drug names. compare Allegra Allegra 0 DrugsByNameDriver	04 DrugByNameComparator This method will compare two strings and return an integer based on the ordering of the two Drug names. compare 2Allegra 1DayQuil -3 DrugsByNameDriver	05 DrugByNameComparator This method will compare two strings and return an integer based on the ordering of the two Drug names. compare 2Allegra 1Allegra 0 DrugsByNameDriver

Figure 3.4:Test Cases 1-5

3. <u>Creating the Drivers</u>

Creating the drivers was not too difficult. The more challenging part was figuring out how to make sure the driver would be able to access the class that was being tested. The solution was found in using a package. Meg had never used packages before, but after some research, she quickly learned how to use them and more importantly how they affect the compiling and running of code. So instead of *javac FileName.java* to compile the code the command was *javac -d . FileName.java*. Then to run the program, instead of *java Fileman "Inputs"* for the testCaseExecutables package the command was *java testCaseExecutables.FileName "Inputs".* The Driver for the DrugsByNameComparitor method is below.

```
package testCaseExecutables;
public class DrugsByNameDriver{
```

```
            public static void main(String args[]){

            DrugByNameComparator drugs = new DrugByNameComparator();

            String inputString[] = args[0].split(" ");
            int output =
drugs.compareDrugNamesInoringNumericals(inputString[0],inputString[1]);
            System.out.println(output); } }
```

Figure 3.5: DrugsByNameComparitor Driver

4. Building the Automated Script

Overall the development of the script wasn't difficult, it was a fun challenge. Mitch used the List.sh file we created earlier this year and used it as a starting point for the script. He ran into a few issues with bugs and being able to read from the test case text files. After some research, he discovered a debugging tool used for scripting. He said that the discovery of this tool helped him find and fix bugs in the script. The script, excluding the HTML portion, is found below.

```
#!/bin/bash
clear

# for debugging purposes
# exec 5> command.txt
# BASH_XTRACEFD="5"

echo "----------Running test script----------"

# Constants
DIRECTORY=${PWD##*/}
TITLE="Team 3 | Carrillo, Krawczyk, Suzara"
RIGHT_NOW="$(date +"%x %r %Z")"
TIME_STAMP="Updated on $RIGHT_NOW by $USER"
FILENAME="testResults.html"
PACKAGE="testCasesExecutables"
```

```
# Create the HTML file
touch ../reports/$FILENAME
> ../reports/$FILENAME

# list contents of current directory
list_directory(){ printf '%s\n' *;}

# cd to testCaseExecutables
cd ../testCasesExecutables

# clean any previous files and directories
rm -f ../testCaseExecutables/*.class


# compile all test case executables
javac  -d . *.java
echo "All test executables have been compiled"

# create array to read text case files
declare -a array

# function to run tests and add results to HTML table
function run_tests() {
   for file in ../testCases/*.txt
   do
      i=0;
      echo \<tr\>
      while read line || [ -n "$line" ];
      do
         echo \<td\>$line\<\/td\>
         array[$i]="$line"
         # echo $array[$i]
         i=$((i+1))
      done < $file

         # testID=${array[0]}
         # class=${array[1]}
         # requirements=${array[2]}
         # method=${array[3]}
         input=${array[4]}
         expected_output=${array[5]}
         driver_name=${array[6]}

      if [[ $driver_name == "containsOnlyDigitsDriver" ]]; then
         result=$(java testCaseExecutables.containsOnlyDigitsDriver "$input")
      fi
```

```
        if [[ "$driver_name" == "containsUpperAndLowerCaseDriver" ]]; then
            result="$(java testCaseExecutables.containsUpperAndLowerCaseDriver
"$input")"
        fi

        if [[ "$driver_name" == "convertToIntegerDriver" ]]; then
            result="$(java testCaseExecutables.convertToIntegerDriver "$input")"
        fi

        if [[ "$driver_name" == "DateUtilDriver" ]]; then
            result="$(java testCaseExecutables.DateUtilDriver "$input")"
        fi

        if [[ $driver_name == "DrugsByNameDriver" ]]; then
            result=$(java testCaseExecutables.DrugsByNameDriver "$input")
        fi

            # set -x
            echo \<td\>$result\<\/td\>
            if [[ $result==$expected_output ]]; then

                echo \<td\>"Pass"\<\/td\>
            else
                echo \<td\>"Fail"\<\/td\>
            fi
            # set +x
        echo \</tr\>
    done
}
```

Figure 3.6: Script

Chapter 4

Project Updates

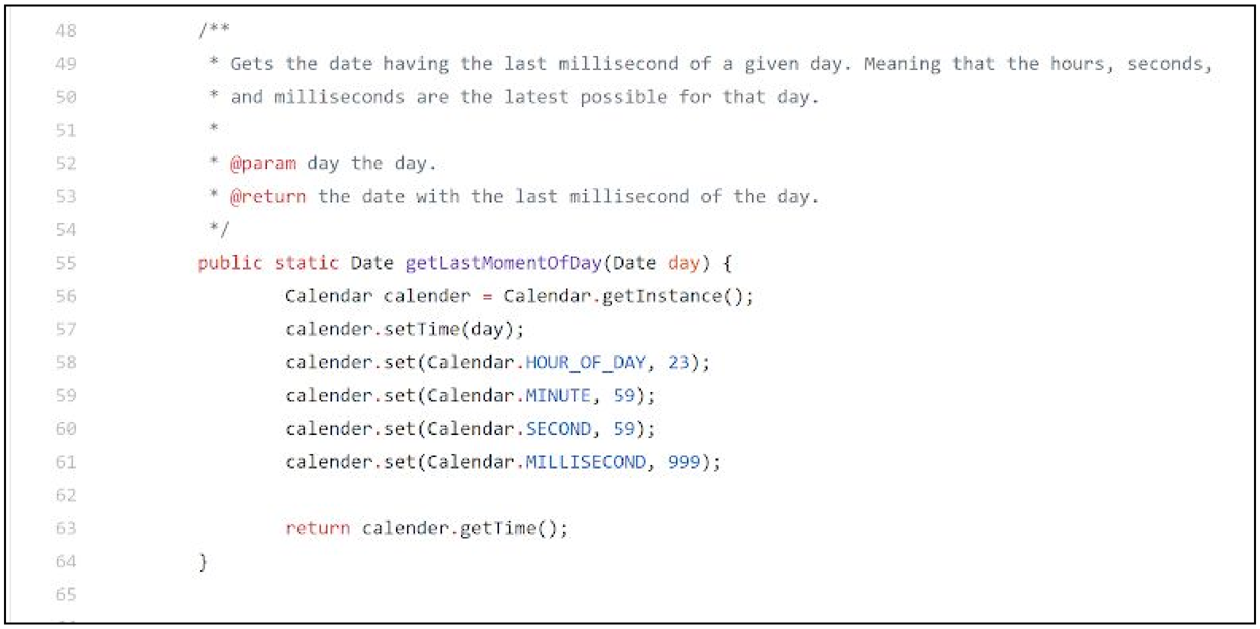
After our presentation on Thursday, we had a couple of changes to make to our project. First, we decided to replace the compareDrugsByName method in DrugsByNameComparitor class with the LastMomentOfDay method found in the OpenmrsUtils class. The main reason we decided to swap the methods was due to the unpredictability of the output of the compareDrugsByName method. According to its requirements, it would return a positive number both when the second drug came before and after the first drug alphabetically. Since drastically different inputs could result in the same output it became very hard to test. In addition, there were no instructions on how this function came up with the number it output, which again made it very hard to test. As a result, we decided to toss it and replace it with the LastMomentOfDay method, which takes a date and shifts the time of day to be the last second of the day. This proved to be much easier to calculate the actual output.

The second thing we made changes to was the script. The first issue with our script was that it was not actually comparing the input and output, but rather just defaulting to pass. This was not what we wanted, but now our script has been updated to actually compare the two values and correctly determine if the test passed or failed. In this change, our team has discovered an odd quirk in the DateUtil method. When the script is run on Mitch's computer the test fails, claiming that the expected output is one second ahead of the actual output. However when the script runs on Meg's or Itzayana's computer the test passes. We're not sure why this happens, but we think it has something to do with the Date class which has been deprecated.

Finally, the last thing we did was to create drivers and test cases for the LastMomentOfDay method. This didn't take too long, since we just modeled them after the first example we had already completed. Luckily, because of the way we changed our script all the test cases and drivers are working as planned.

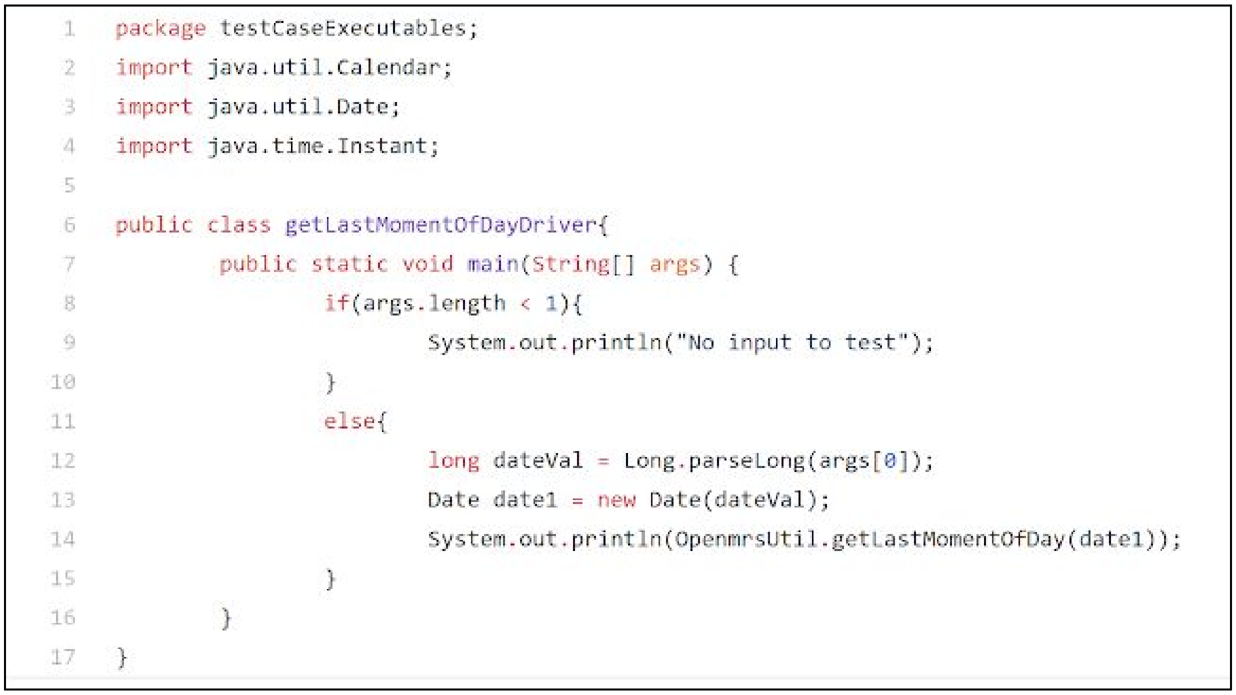
Methods, Drivers and Test Cases

getLastMomentOfDay



```java
	/**
	 * Gets the date having the last millisecond of a given day. Meaning that the hours, seconds,
	 * and milliseconds are the latest possible for that day.
	 *
	 * @param day the day.
	 * @return the date with the last millisecond of the day.
	 */
	public static Date getLastMomentOfDay(Date day) {
		Calendar calender = Calendar.getInstance();
		calender.setTime(day);
		calender.set(Calendar.HOUR_OF_DAY, 23);
		calender.set(Calendar.MINUTE, 59);
		calender.set(Calendar.SECOND, 59);
		calender.set(Calendar.MILLISECOND, 999);

		return calender.getTime();
	}

```

Figure 4.1: getLlastMomentOfDay Method found in the OpenmrsUtil Class



```java
package testCaseExecutables;
import java.util.Calendar;
import java.util.Date;
import java.time.Instant;

public class getLastMomentOfDayDriver{
	public static void main(String[] args) {
		if(args.length < 1){
			System.out.println("No input to test");
		}
		else{
			long dateVal = Long.parseLong(args[0]);
			Date date1 = new Date(dateVal);
			System.out.println(OpenmrsUtil.getLastMomentOfDay(date1));
		}
	}
}
```

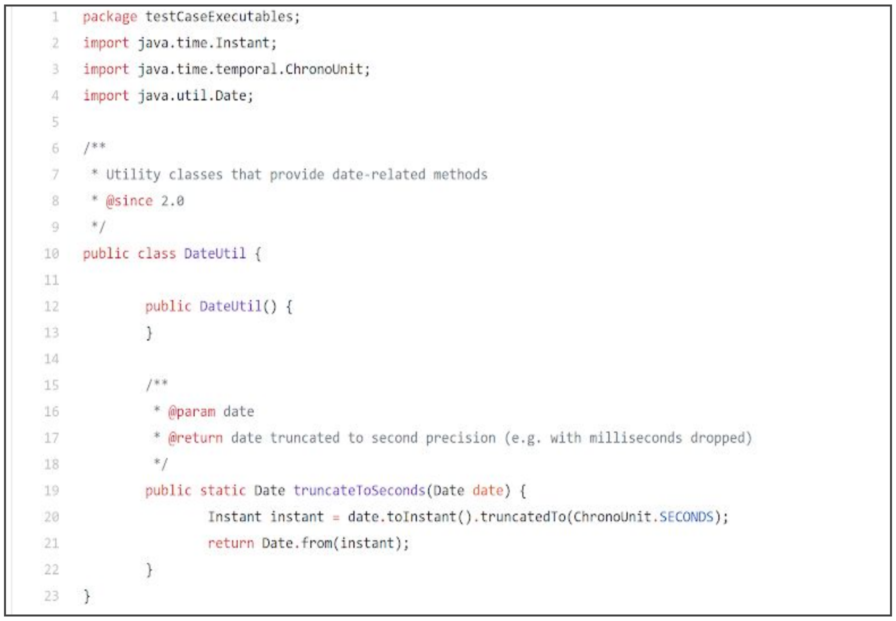
Figure 4.2: getLastMomentOfDayDriver Class

Test Cases
01 OpenmrsUtil This method will take a date, convert it to a calendar, then shift the time to be the last second of the day on that date. getLastMomentOfDay 10000 Wed Dec 31 23:59:59 EST 1969 getLastMomentOfDayDriver
02 OpenmrsUtil This method will take a date, convert it to a calendar, then shift the time to be the last second of the day on that date. getLastMomentOfDay -10000000000 Sun Sep 07 23:59:59 EDT 1969 getLastMomentOfDayDriver
03 OpenmrsUtil This method will take a date, convert it to a calendar, then shift the time to be the last second of the day on that date. getLastMomentOfDay 0 Wed Dec 31 23:59:59 EST 1969 getLastMomentOfDayDriver
04 OpenmrsUtil This method will take a date, convert it to a calendar, then shift the time to be the last second of the day on that date. getLastMomentOfDay 1000000000000 Sat Sep 08 23:59:59 EDT 2001 getLastMomentOfDayDriver
05 OpenmrsUtil This method will take a date, convert it to a calendar, then shift the time to be the last second of the day on that date. getLastMomentOfDay 914209500000 Sun Dec 20 23:59:59 EST 1998

getLastMomentOfDayDriver

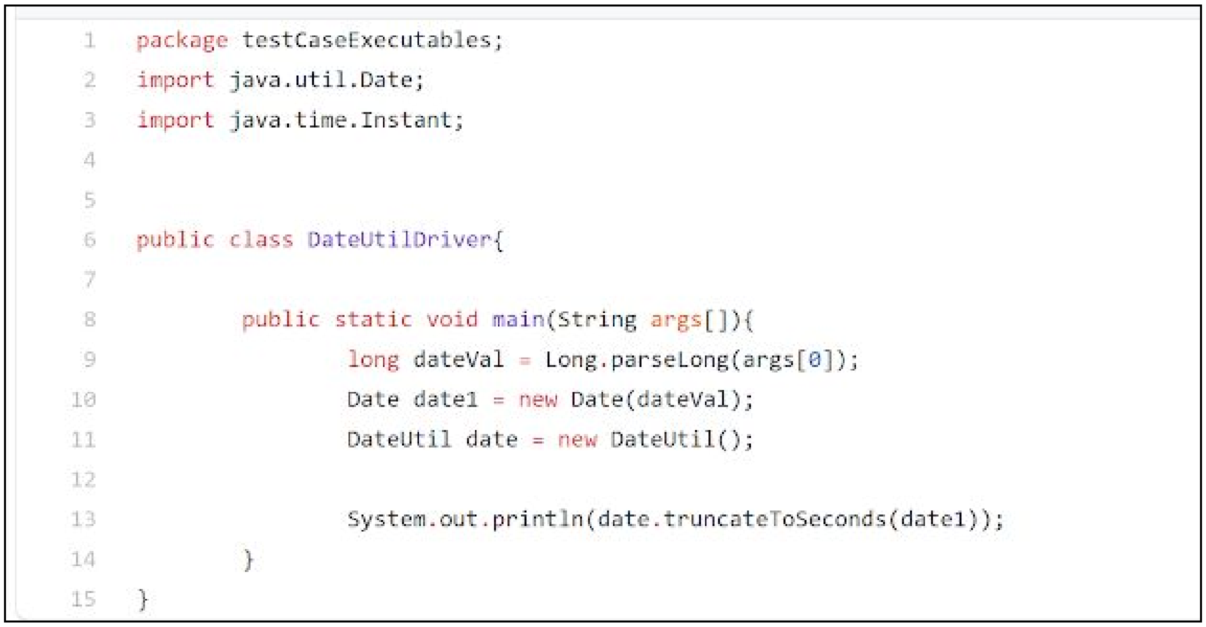
Figure 4.3: Test Cases 01- 05 Update

truncateToSeconds



```
package testCaseExecutables;
import java.time.Instant;
import java.time.temporal.ChronoUnit;
import java.util.Date;

/**
 * Utility classes that provide date-related methods
 * @since 2.0
 */
public class DateUtil {

        public DateUtil() {
        }

        /**
         * @param date
         * @return date truncated to second precision (e.g. with milliseconds dropped)
         */
        public static Date truncateToSeconds(Date date) {
                Instant instant = date.toInstant().truncatedTo(ChronoUnit.SECONDS);
                return Date.from(instant);
        }
}
```

Figure 4.4: truncateToSeconds Method found in the DateUtil Class



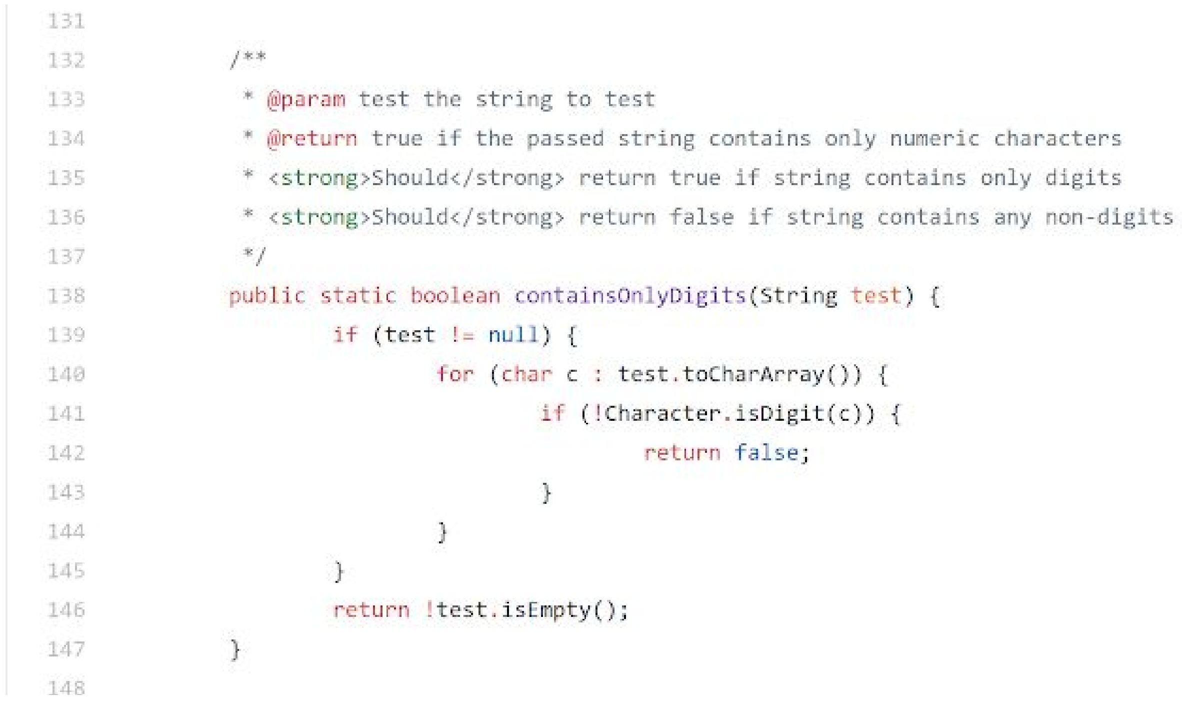
```
package testCaseExecutables;
import java.util.Date;
import java.time.Instant;


public class DateUtilDriver{

        public static void main(String args[]){
                long dateVal = Long.parseLong(args[0]);
                Date date1 = new Date(dateVal);
                DateUtil date = new DateUtil();

                System.out.println(date.truncateToSeconds(date1));
        }
}
```

Figure 4.5: DateUtilDriver Class

Test Cases
06 DateUtil This method will return a Date with the milliseconds truncated truncateToSeconds 1234094 Wed Dec 31 19:20:34 EST 1969 DateUtilDriver
07 DateUtil This method will return a Date with the milliseconds truncated truncateToSeconds 12340940087655343 Sun Oct 28 13:14:15 EDT 393038 DateUtilDriver
08 DateUtil This method will return a Date with the milliseconds truncated truncateToSeconds 0 Wed Dec 31 19:00:00 EST 1969 DateUtilDriver
09 DateUtil This method will return a Date with the milliseconds truncated truncateToSeconds -847350384 Sun Dec 21 23:37:30 EST 1969 DateUtilDriver
10 DateUtil This method will return a Date with the milliseconds truncated truncateToSeconds -949474704827 Thu Nov 30 12:01:36 EST 1939 DateUtilDriver

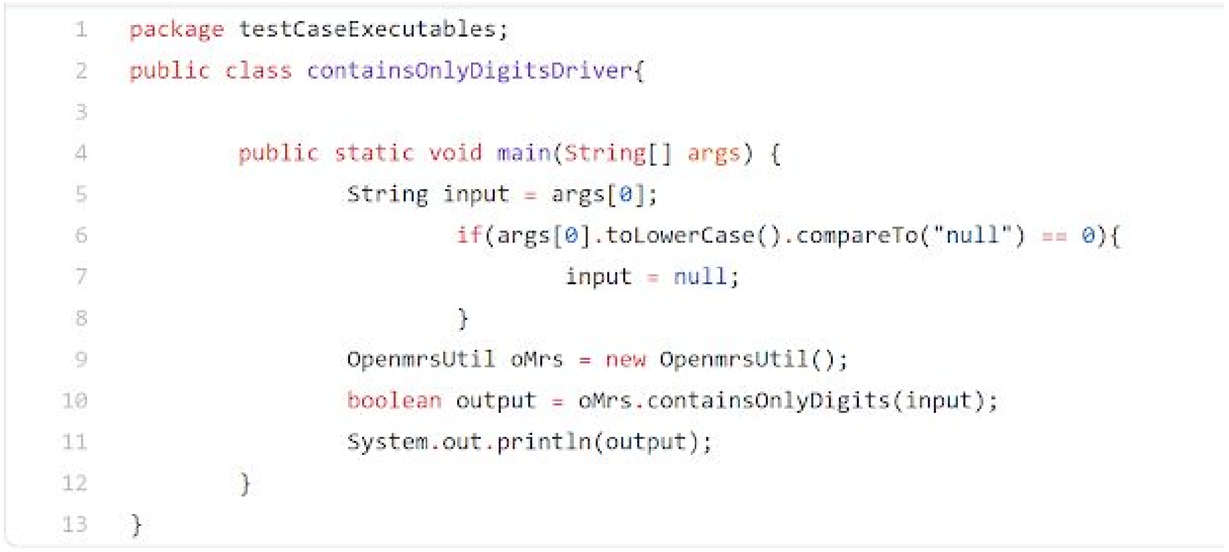
Figure 4.6: Test Cases 06 -10

containsOnlyDigits



```java

	/**
	 * @param test the string to test
	 * @return true if the passed string contains only numeric characters
	 * <strong>Should</strong> return true if string contains only digits
	 * <strong>Should</strong> return false if string contains any non-digits
	 */
	public static boolean containsOnlyDigits(String test) {
		if (test != null) {
			for (char c : test.toCharArray()) {
				if (!Character.isDigit(c)) {
					return false;
				}
			}
		}
		return !test.isEmpty();
	}

```

Figure 4.7: containsOnlyDigits Method found in the OpenmrsUtil Class



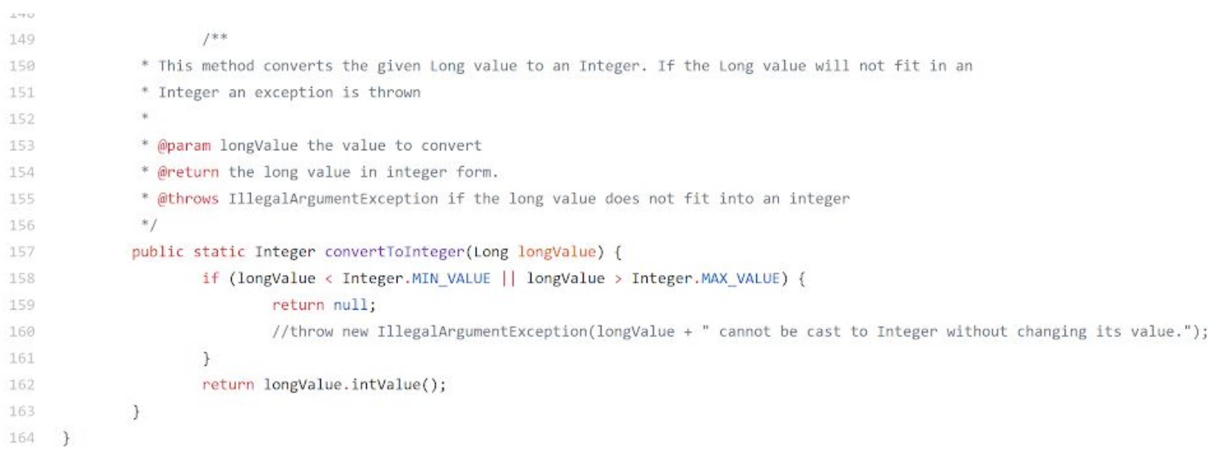
```java
package testCaseExecutables;
public class containsOnlyDigitsDriver{

	public static void main(String[] args) {
		String input = args[0];
			if(args[0].toLowerCase().compareTo("null") == 0){
				input = null;
			}
		OpenmrsUtil oMrs = new OpenmrsUtil();
		boolean output = oMrs.containsOnlyDigits(input);
		System.out.println(output);
	}
}
```

Figure 4.8: contiansOnlyDigits Driver Class

Test Cases
11 OpenmrsUtil This method returns true if a string only contains digits containsOnlyDigits 808760 true containsOnlyDigitsDriver
12 OpenmrsUtil This method returns true if a string only contains digits containsOnlyDigits -842 false containsOnlyDigitsDriver
13 OpenmrsUtil This method returns true if a string only contains digits containsOnlyDigits 8940 true containsOnlyDigitsDriver
14 OpenmrsUtil This method returns true if a string only contains digits containsOnlyDigits 34hi80 false containsOnlyDigitsDriver
15 OpenmrsUtil This method returns true if a string only contains digits containsOnlyDigits 60 78 false containsOnlyDigitsDriver

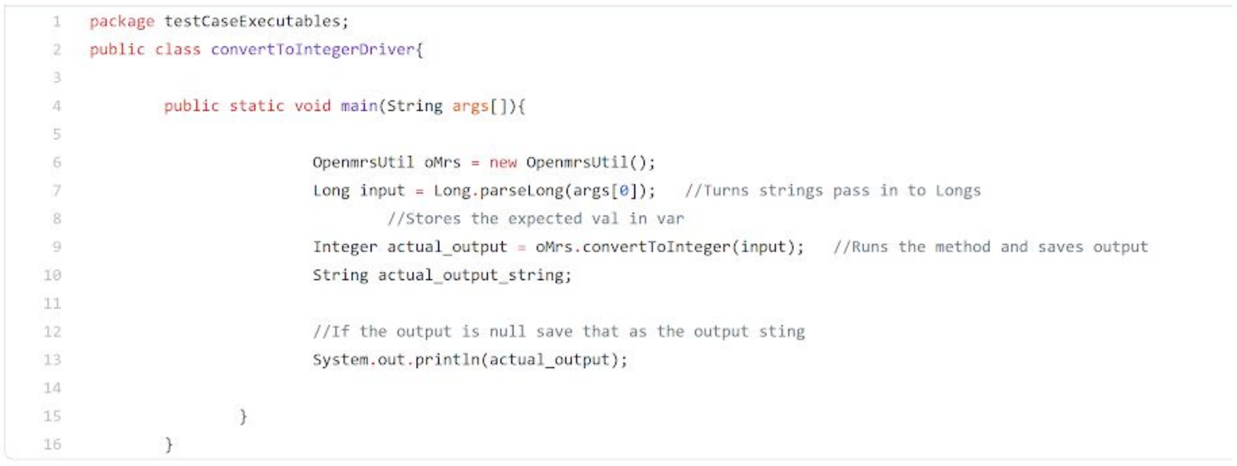
Figure 4.9: Test Cases 11- 15

convertToInteger



```
		/**
	 * This method converts the given Long value to an Integer. If the Long value will not fit in an
	 * Integer an exception is thrown
	 *
	 * @param longValue the value to convert
	 * @return the long value in integer form.
	 * @throws IllegalArgumentException if the long value does not fit into an integer
	 */
	public static Integer convertToInteger(Long longValue) {
		if (longValue < Integer.MIN_VALUE || longValue > Integer.MAX_VALUE) {
			return null;
			//throw new IllegalArgumentException(longValue + " cannot be cast to Integer without changing its value.");
		}
		return longValue.intValue();
	}
}
```

Figure 4.10: convertToInteger Method found in the OpenmrsUtil Class



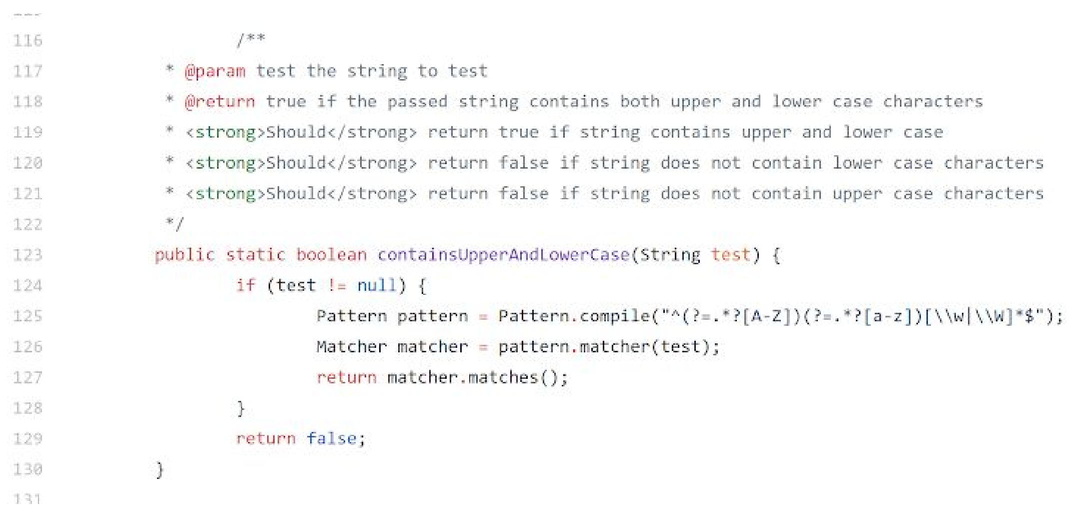
```
package testCaseExecutables;
public class convertToIntegerDriver{

	public static void main(String args[]){

			OpenmrsUtil oMrs = new OpenmrsUtil();
			Long input = Long.parseLong(args[0]);   //Turns strings pass in to Longs
				//Stores the expected val in var
			Integer actual_output = oMrs.convertToInteger(input);   //Runs the method and saves output
			String actual_output_string;

			//If the output is null save that as the output sting
			System.out.println(actual_output);

		}
	}
```

Figure 4.11: convertToIntegerDriver Class

Test Cases
16 OpenmrsUtil This method turns a long into an integer, as long as it does not exceed the int memory limit convertToInteger 600 600 convertToIntegerDriver
17 OpenmrsUtil This method turns a long into an integer, as long as it does not exceed the int memory limit convertToInteger 6000000000 null convertToIntegerDriver
18 OpenmrsUtil This method turns a long into an integer, as long as it does not exceed the int memory limit convertToInteger -83878527402 null convertToIntegerDriver
19 OpenmrsUtil This method turns a long into an integer, as long as it does not exceed the int memory limit convertToInteger 2147483647 2147483647 convertToIntegerDriver
20 OpenmrsUtil This method turns a long into an integer, as long as it does not exceed the int memory limit convertToInteger -2147483648 -2147483648 convertToIntegerDriver

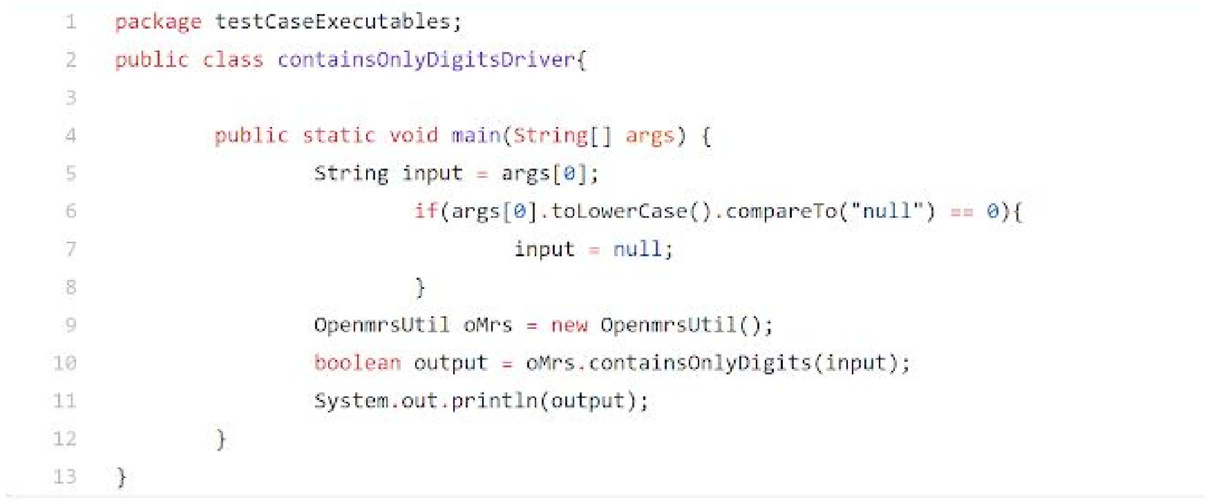
Figure 4.12: Test Cases 16 - 20

containsUpperAndLower



```
        /**
         * @param test the string to test
         * @return true if the passed string contains both upper and lower case characters
         * <strong>Should</strong> return true if string contains upper and lower case
         * <strong>Should</strong> return false if string does not contain lower case characters
         * <strong>Should</strong> return false if string does not contain upper case characters
         */
        public static boolean containsUpperAndLowerCase(String test) {
                if (test != null) {
                        Pattern pattern = Pattern.compile("^(?=.*?[A-Z])(?=.*?[a-z])[\\w|\\W]*$");
                        Matcher matcher = pattern.matcher(test);
                        return matcher.matches();
                }
                return false;
        }
```

Figure 4.13: containsUpperAndLowerCase Method



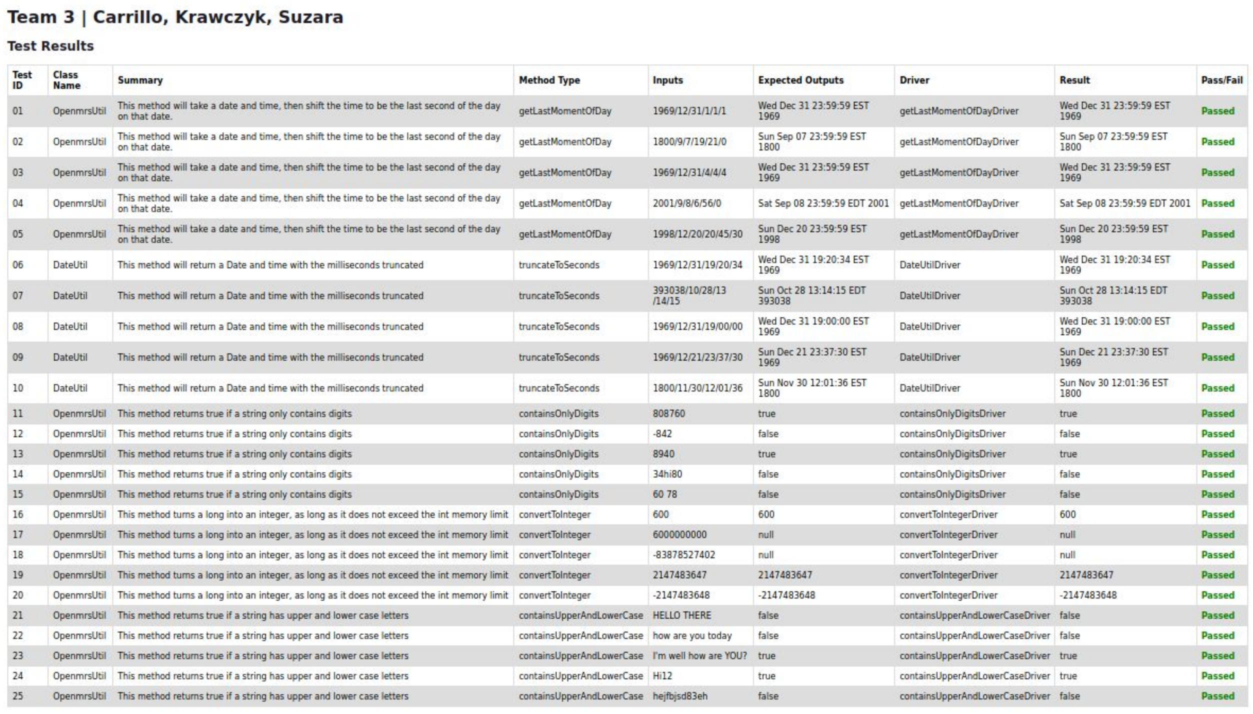
```
package testCaseExecutables;
public class containsOnlyDigitsDriver{

        public static void main(String[] args) {
                String input = args[0];
                        if(args[0].toLowerCase().compareTo("null") == 0){
                                input = null;
                        }
                OpenmrsUtil oMrs = new OpenmrsUtil();
                boolean output = oMrs.containsOnlyDigits(input);
                System.out.println(output);
        }
}
```

Figure 4.14: containsUpperAndLowerCase Class

Test Cases
21 OpenmrsUtil This method returns true if a string has upper and lower case letters containsUpperAndLowerCase HELLO THERE false containsUpperAndLowerCaseDriver
22 OpenmrsUtil This method returns true if a string has upper and lower case letters containsUpperAndLowerCase how are you today false containsUpperAndLowerCaseDriver
23 OpenmrsUtil This method returns true if a string has upper and lower case letters containsUpperAndLowerCase I'm well how are YOU? true containsUpperAndLowerCaseDriver
24 OpenmrsUtil This method returns true if a string has upper and lower case letters containsUpperAndLowerCase Hi12 true containsUpperAndLowerCaseDriver
25 OpenmrsUtil This method returns true if a string has upper and lower case letters containsUpperAndLowerCase hejfbjsd83eh false containsUpperAndLowerCaseDriver

Figure 4.15: Test Cases 11- 15

Output Table



Team 3 | Carrillo, Krawczyk, Suzara

Test Results

Test ID	Class Name	Summary	Method Type	Inputs	Expected Outputs	Driver	Result	Pass/Fail
01	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1969/12/31/1/1/1	Wed Dec 31 23:59:59 EST 1969	getLastMomentOfDayDriver	Wed Dec 31 23:59:59 EST 1969	Passed
02	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1800/9/7/19/21/0	Sun Sep 07 23:59:59 EST 1800	getLastMomentOfDayDriver	Sun Sep 07 23:59:59 EST 1800	Passed
03	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1969/12/31/4/4/4	Wed Dec 31 23:59:59 EST 1969	getLastMomentOfDayDriver	Wed Dec 31 23:59:59 EST 1969	Passed
04	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	2001/9/8/6/56/0	Sat Sep 08 23:59:59 EDT 2001	getLastMomentOfDayDriver	Sat Sep 08 23:59:59 EDT 2001	Passed
05	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1998/12/20/20/45/30	Sun Dec 20 23:59:59 EST 1998	getLastMomentOfDayDriver	Sun Dec 20 23:59:59 EST 1998	Passed
06	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1969/12/31/19/20/34	Wed Dec 31 19:20:34 EST 1969	DateUtilDriver	Wed Dec 31 19:20:34 EST 1969	Passed
07	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	393038/10/28/13 /14/15	Sun Oct 28 13:14:15 EDT 393038	DateUtilDriver	Sun Oct 28 13:14:15 EDT 393038	Passed
08	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1969/12/31/19/00/00	Wed Dec 31 19:00:00 EST 1969	DateUtilDriver	Wed Dec 31 19:00:00 EST 1969	Passed
09	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1969/12/21/23/37/30	Sun Dec 21 23:37:30 EST 1969	DateUtilDriver	Sun Dec 21 23:37:30 EST 1969	Passed
10	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1800/11/30/12/01/36	Sun Nov 30 12:01:36 EST 1800	DateUtilDriver	Sun Nov 30 12:01:36 EST 1800	Passed
11	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	808760	true	containsOnlyDigitsDriver	true	Passed
12	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	-842	false	containsOnlyDigitsDriver	false	Passed
13	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	8940	true	containsOnlyDigitsDriver	true	Passed
14	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	34hi80	false	containsOnlyDigitsDriver	false	Passed
15	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	60 78	false	containsOnlyDigitsDriver	false	Passed
16	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	600	600	convertToIntegerDriver	600	Passed
17	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	6000000000	null	convertToIntegerDriver	null	Passed
18	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	-83878527402	null	convertToIntegerDriver	null	Passed
19	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	2147483647	2147483647	convertToIntegerDriver	2147483647	Passed
20	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	-2147483648	-2147483648	convertToIntegerDriver	-2147483648	Passed
21	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	HELLO THERE	false	containsUpperAndLowerCaseDriver	false	Passed
22	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	how are you today	false	containsUpperAndLowerCaseDriver	false	Passed
23	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	I'm well how are YOU?	true	containsUpperAndLowerCaseDriver	true	Passed
24	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	Hi12	true	containsUpperAndLowerCaseDriver	true	Passed
25	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	hejfbjsd83eh	false	containsUpperAndLowerCaseDriver	false	Passed

Figure 4.16: Output Table

Experience

Our experience working on OpenMrs continues to grow our knowledge of both open-source projects and automated testing. Through this small sprint of work, we learned a few valuable lessons about consistently using the same operating systems, understanding method requirements, and scripting. We learned that even writing what seems to be an ordinary text file in windows can have an effect on how that text file is read later.

Chapter 5

Error Insertion

Down below are step by step instructions on how to insert these same errors into the code.

1. Open the OpenmrsUtil.java file
    a. Comment out line 27, this will remove the correct code segment
    b. Uncomment line 28, this will activate the error in the code
    c. Comment out line 46, this will remove the correct code segment
    d. Uncomment line 47, this will activate the error in the code
    e. Comment out line 68, this will remove the correct code segment
    f. Uncomment line 69, this will activate the error in the code
    g. Save the OpenmrsUtil.java file
2. Open the DateUtil.java file
    a. Comment out line 20, this will remove the correct code segment
    b. Uncomment line 21, this will activate the error in the code

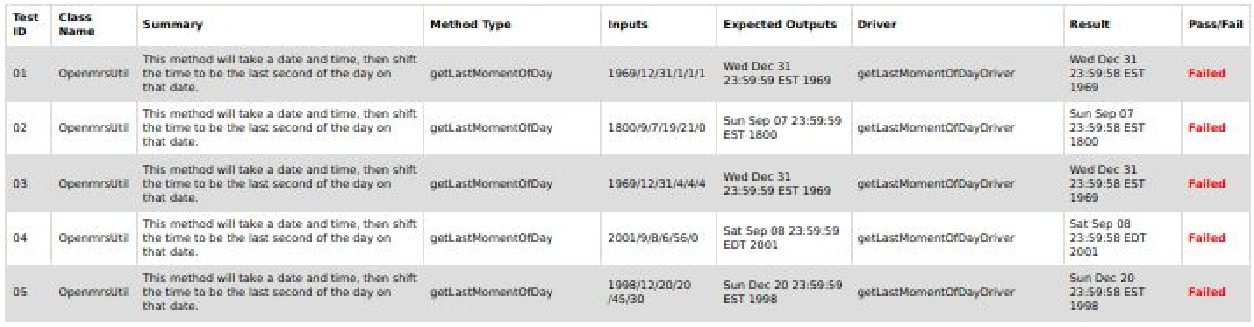
Error Results

getLastMomentOfDay

For the getLastMomentOfDay method, we decided that the error we would implement would be to set the seconds to be 58 instead of 59. This seemed like an error that could easily occur if someone was typing fast or didn't read the requirements. When we ran our script with the error in place, the error was caught in all cases. This result makes sense because the 58 was a hardcoded mistake and as a result, it affected all the test cases.

```
calender.set(Calendar.SECOND, 59);
calender.set(Calendar.SECOND, 58); //THIS IS AN ERROR
```

Figure 5.1: getLastMomentOfDay Error



Test ID	Class Name	Summary	Method Type	Inputs	Expected Outputs	Driver	Result	Pass/Fail
01	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1969/12/31/1/1/1	Wed Dec 31 23:59:59 EST 1969	getLastMomentOfDayDriver	Wed Dec 31 23:59:58 EST 1969	Failed
02	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1800/9/7/19/21/0	Sun Sep 07 23:59:59 EST 1800	getLastMomentOfDayDriver	Sun Sep 07 23:59:58 EST 1800	Failed
03	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1969/12/31/4/4/4	Wed Dec 31 23:59:59 EST 1969	getLastMomentOfDayDriver	Wed Dec 31 23:59:58 EST 1969	Failed
04	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	2001/9/8/6/56/0	Sat Sep 08 23:59:59 EDT 2001	getLastMomentOfDayDriver	Sat Sep 08 23:59:58 EDT 2001	Failed
05	OpenmrsUtil	This method will take a date and time, then shift the time to be the last second of the day on that date.	getLastMomentOfDay	1998/12/20/20/45/30	Sun Dec 20 23:59:59 EST 1998	getLastMomentOfDayDriver	Sun Dec 20 23:59:58 EST 1998	Failed

Figure 5.2: getLastMomentOfDay Output Errors

How to implement this error in getLastMomentOfDay:

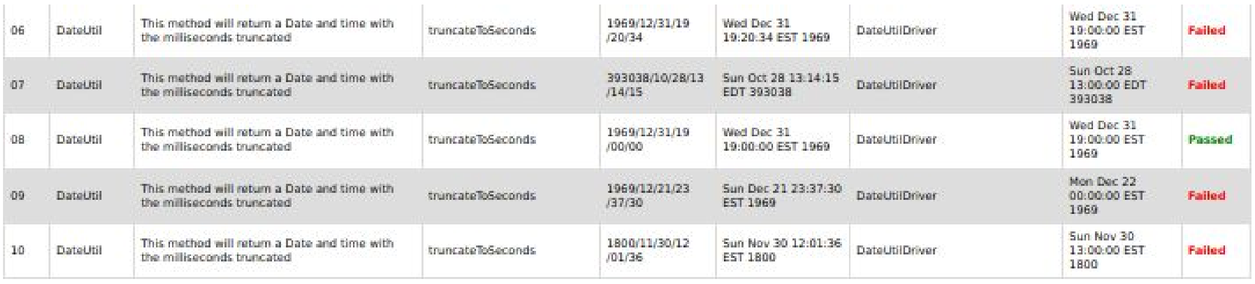
1. Open the OpenmrsUtil.java file
2. Comment out line 27, this will remove the correct code segment
3. Uncomment line 28, this will activate the error in the code

DateUtil

For the DataUtil method, we decided that the error we would implement would be to truncate hours instead of seconds. By truncating by the hour, the second's section of time will always be 00 rather than the actual number of seconds. When we ran our script with the error in place, it caught the error in all cases. In that case, the seconds and hours time segments had been set to zero anyway so truncating the seconds didn't make a difference.

```
Instant instant = date.toInstant().truncatedTo(ChronoUnit.SECONDS);
Instant instant = date.toInstant().truncatedTo(ChronoUnit.HOURS); //THIS IS THE ERROR
```

Figure 5.3: truncateToSeconds Error



06	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1969/12/31/19/20/34	Wed Dec 31 19:20:34 EST 1969	DateUtilDriver	Wed Dec 31 19:00:00 EST 1969	Failed
07	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	393038/10/28/13/14/15	Sun Oct 28 13:14:15 EDT 393038	DateUtilDriver	Sun Oct 28 13:00:00 EDT 393038	Failed
08	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1969/12/31/19/00/00	Wed Dec 31 19:00:00 EST 1969	DateUtilDriver	Wed Dec 31 19:00:00 EST 1969	Passed
09	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1969/12/21/23/37/30	Sun Dec 21 23:37:30 EST 1969	DateUtilDriver	Mon Dec 22 00:00:00 EST 1969	Failed
10	DateUtil	This method will return a Date and time with the milliseconds truncated	truncateToSeconds	1800/11/30/12/01/36	Sun Nov 30 12:01:36 EST 1800	DateUtilDriver	Sun Nov 30 13:00:00 EST 1800	Failed

Figure 5.4: truncateToSeconds Output Error

How to implement this error in DateUtil:

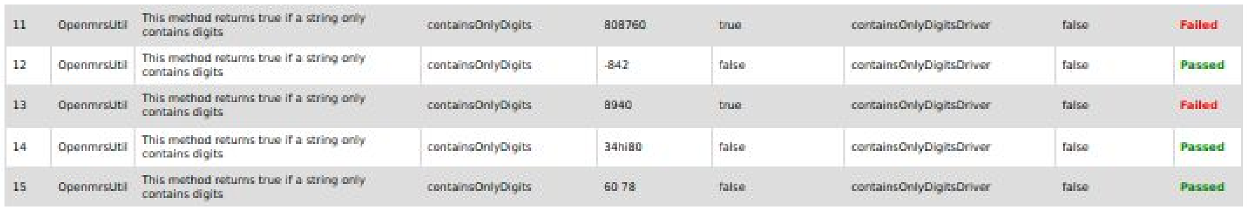
1. Open the DateUtil.java file
2. Comment out line 20, this will remove the correct code segment
3. Uncomment line 21, this will activate the error in the code

containsOnlyDigits

For the containsOnlyDigits method, we decided that the error we would implement would be to return if a string is empty instead of if it's not. If a string makes it to the end of the program it will return the opposite of what we want. When we ran our script with the error in place, the error was not caught in the test cases that were expected to be false. This was confusing at first, but after going back and looking at the code we realized that this method had a separate return statement that is triggered when a string contains something other than digits. Which left those cases untouched by our error.

```
return !test.isEmpty();
return test.isEmpty(); //THIS IS THE ERROR
```

Figure 5.5: containsOnlyDigits Error



11	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	808760	true	containsOnlyDigitsDriver	false	Failed
12	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	-842	false	containsOnlyDigitsDriver	false	Passed
13	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	8940	true	containsOnlyDigitsDriver	false	Failed
14	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	34hi80	false	containsOnlyDigitsDriver	false	Passed
15	OpenmrsUtil	This method returns true if a string only contains digits	containsOnlyDigits	60 78	false	containsOnlyDigitsDriver	false	Passed

Figure 5.6: containsOnlyDigits Output Error

How to implement this error in containsOnlyDigits:

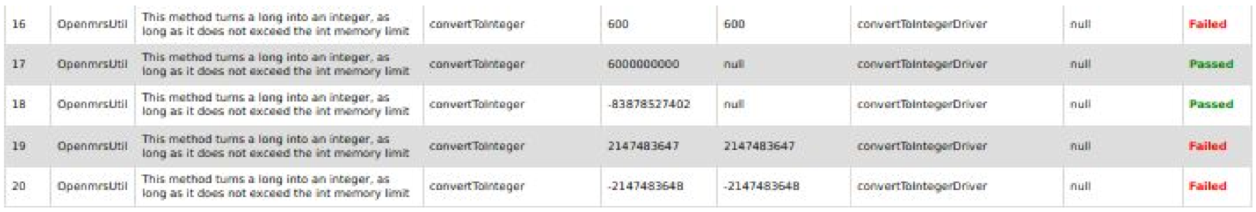
1. Open the OpenmrsUtil.java file
2. Comment out line 68, this will remove the correct code segment
3. Uncomment line 69, this will activate the error in the code

convertToInteger

For the convertToInteger method, we decided that the error we would implement would be to flip the greater than and less than symbols in an if-statement. This error will make the program think that convertible numbers are invalid. When we ran our script with the error in place, the error was not caught in the test cases that were expected to return a null. This makes sense because now the range of numbers that will trigger that if-statement is all possible numbers. So any number I put in as input will return a null.

```
if (longValue < Integer.MIN_VALUE || longValue > Integer.MAX_VALUE) {
if (longValue > Integer.MIN_VALUE || longValue < Integer.MAX_VALUE) { //THIS IS THE ERROR
```

Figure 5.7: convertToInteger Error



16	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	600	600	convertToIntegerDriver	null	Failed
17	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	6000000000	null	convertToIntegerDriver	null	Passed
18	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	-83878527402	null	convertToIntegerDriver	null	Passed
19	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	2147483647	2147483647	convertToIntegerDriver	null	Failed
20	OpenmrsUtil	This method turns a long into an integer, as long as it does not exceed the int memory limit	convertToInteger	-2147483648	-2147483648	convertToIntegerDriver	null	Failed

Figure 5.8: convertToInteger Output Error

How to implement this error in convertToInteger:

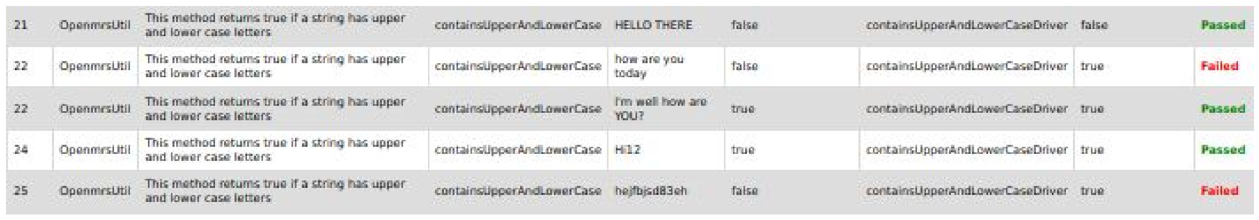
1. Open the OpenmrsUtil.java file
2. Comment out line 82, this will remove the correct code segment
3. Uncomment line 83, this will activate the error in the code

containsUpperAndLower

For the containsUpperAndLower method, we decided that the error we would implement would take out the part of the pattern that recognized Uppercase letters. This error will make the program return true as long as the input contains lower case letters. When we ran our script with the error in place, the error was not caught in the test cases that contained only lowercase letters. This makes sense because now the inputs with both upper and lowercase letters would still pass the test because this error only checks for lowercase and returns true if the input contains lowercase letters.

```
Pattern pattern = Pattern.compile("^(?=.*?[A-Z])(?=.*?[a-z])[\\w|\\W]*$");
Pattern pattern = Pattern.compile("(?=.*?[a-z])[\\w|\\W]*$"); //THIS IS THE ERROR
```

Figure 5.9: containsUpperAndLowerCase Error



21	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	HELLO THERE	false	containsUpperAndLowerCaseDriver	false	Passed
22	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	how are you today	false	containsUpperAndLowerCaseDriver	true	Failed
22	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	I'm well how are YOU?	true	containsUpperAndLowerCaseDriver	true	Passed
24	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	Hi12	true	containsUpperAndLowerCaseDriver	true	Passed
25	OpenmrsUtil	This method returns true if a string has upper and lower case letters	containsUpperAndLowerCase	hejfbjsd83eh	false	containsUpperAndLowerCaseDriver	true	Failed

Figure 5.10: containsUpperAndLowerCase Output Error

How to implement this error in containsUpperAndLower:

1. Open the OpenmrsUtil.java file
2. Comment out line 46, this will remove the correct code segment
3. Uncomment line 47, this will activate the error in the code

<u>Experience</u>

Our experience for this portion of the project was not as substantial as other portions due to it being such a short time between chapter 5 and chapter 4. With that being said, we did learn a few lessons about the importance of readability in our test cases. Dr. Bowring mentioned to us that passing in the date in the form of milliseconds made two of our test cases very hard to read.

To remedy this issue, we decided to pass in the date via a string, the format goes: year, month, day, hour, minute, second, each separated by a "/". For example, "1990/12/25/10/20/20" would be the date Dec 25th, 1990 at 10:20:20 am. This makes reading the resulting output much easier to read.

The second thing we had to do this sprint is put in a few errors in our program and see whether our automated testing framework would catch these errors. The way we decided to implement these errors was through comments as described above. In each method, there is a comment that contains a snippet of code with an error in it. To run the code with the error in it simply uncomment that code segment and comment out the segment above it.

Chapter 6

Final Thoughts

We have overall had a very rewarding experience. We hit many challenges especially in the beginning which we overcame. Just opening and running the open-source project was a challenge. Not only was working with an open-source project new to all of us, but also working on a Linux Virtual Machine and using our command-line/terminal. It was overall a great learning experience.

Team Self-evaluation

Working in a team was a little challenging. The hardest part was scheduling meetings when all three members had different schedules. Most of our meetings outside of class ended up being on Sundays at 2 pm. In all other aspects working in Team3 was fun. It was nice to have someone else to review and test your work. It was great to have the emotional support of a team when errors appeared or things just didn't go as expected. Three heads are definitely better than one.

Assignment Evaluation

We can agree that CSCI 362 has not been a conventional course. The assignment itself has given us a little hint of what is actually expected of you as a software developer. We gain a lot of knowledge from experience rather than from books. When we encountered an error we had to find a way to solve it ourselves through research. This assignment also gave us a good feel of what it is like to work in a team. In previous couseres, much of the work was aimed to be completed individually and we were expected to not look for answers online which was completely opposite in this course. The assigned readings were also very helpful and insightful. They help us in planning and organizing our project. We even included a sequence diagram similar to one we had to create for one of our blog assignments. The only suggestion we can make that could improve the course and the overall experience of the course would be to include a crash course on using bash and command line. At the beginning that took much of our time. Other than that we all enjoyed Software Engineering 362.