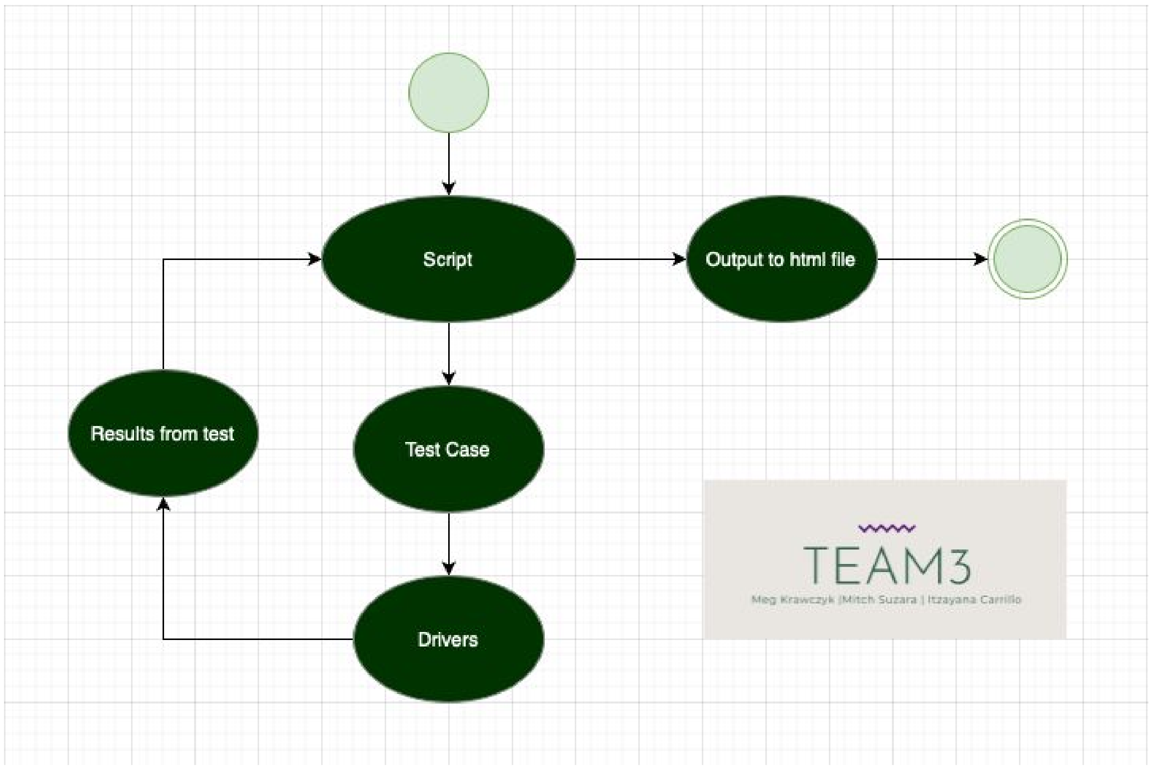
Chapter 3

Team 3: Meg, Mitch, Itzayana

CSCI 362 Project: OpenMrs

Architectural Framework

Our architectural framework consists of a control script runAllTests.sh. This script controls the flow of the testing being done and 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.

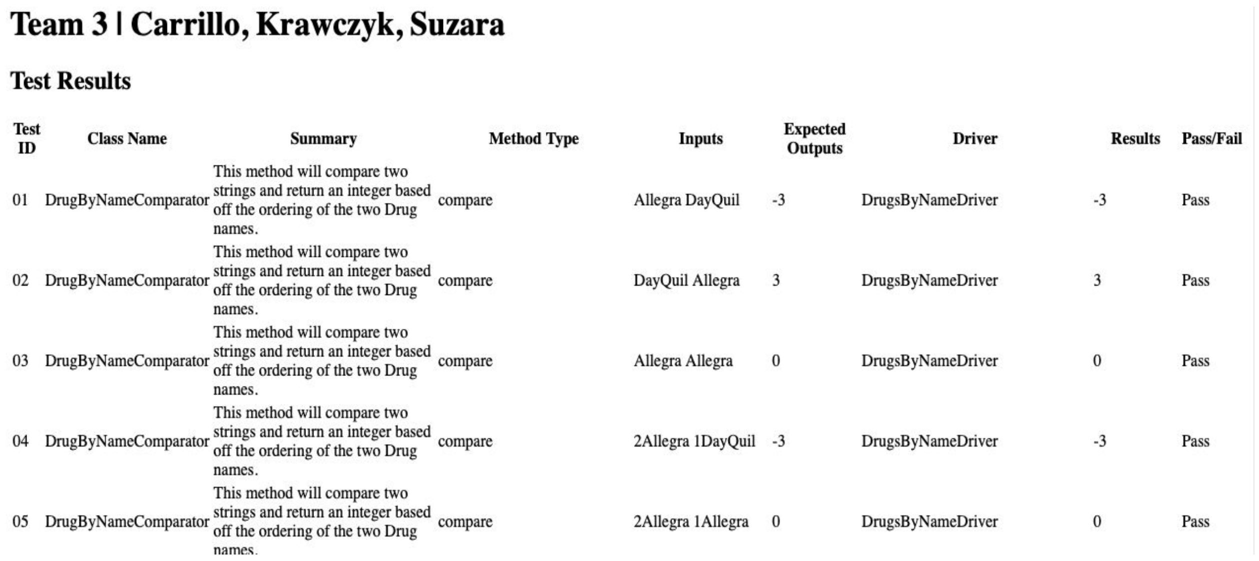


How to Run the Automated Test Framework

- Download Team 3 Repository
- To run the script make file executable
- Run script on Ubuntu terminal by entering the following:
    - `cd to Team3/TestAutomation/scripts`
    - `chmod u+x runAllTests.sh`
    - `./runAllTests.sh`
- HTML report will open on browser

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 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 which is being tested. Each individual 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

<u>Experience In Building Framework</u>

1. <u>Picking Methods</u>

On our first round looking for methods to test we struggled. OpenMRS is a 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 this, these classes and methods had a lot of dependencies making them hard to extract while keeping the logic of the method in tact

Upon further inspection, we were able to find a few static methods that had had very few dependencies, with the exception of a 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()

2. Creating Test Cases

Once we had our methods picked, we could start to make out test cases for each method. Luckily, methods we chose to test all had decent comments specifying what the results should be based on various types of inputs. This documentation 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 off the compare method, it should return -3. Which it did. We have 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	02 DrugByNameComparator This method will compare	03 DrugByNameComparator This method will compare	04 DrugByNameComparator This method will	05 DrugByNameComparator This method will

two strings and return an integer based on the ordering of the two Drug names. compare Allegra DayQuil -3 DrugsByNameDriver	two strings and return an integer based on the ordering of the two Drug names. compare DayQuil Allegra 3 DrugsByNameDriver	two strings and return an integer based on the ordering of the two Drug names. compare Allegra Allegra 0 DrugsByNameDriver	compare two strings and return an integer based on the ordering of the two Drug names. compare 2Allegra 1DayQuil -3 DrugsByNameDriver	compare two strings and return an integer based off the ordering of the two Drug names. compare 2Allegra 1Allegra 0 DrugsByNameDriver

3. Creating the Drivers

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 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); } }
```

4. Building the Automated Script

Overall the development of the script wasn’t difficult, rather 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 testCase text files. After some research, he discovered a debugging tool used from 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
}
```