CSCI362: Software Engineering

OpenMRS

Team: 404-Error

Ricky Ramos, Matthew Schwarz, Matthew Kay

OpenMRS

MEDICAL RECORD SYSTEM

Table of Contents

Project Summary:

This project was assigned by Dr. Jim Bowring for our CSCI362 Software Engineering class. We were tasked with finding a Humanitarian/Free Open Source Software project and designing an automated testing framework to test some of their methods. The testing framework was designed to work on the Ubuntu operating system. The first five chapters of this document represent a deliverable that was created by our team, and the sixth chapter includes an evaluation of this project.

The H/FOSS project we selected was OpenMRS. OpenMRS is an electronic medical record system supported by a community passionate about healthcare. They are mainly concerned with aiding the developing world where AIDS, tuberculosis, and malaria afflict the lives of millions of people. Their project is written in Java, and is well-documented and easy to read.

Chapter 1: Selecting an H/FOSS Project

1.1 Changing Projects

We were first asked to select three H/FOSS project candidates. Surprisingly, OpenMRS wasn't initially considered as a candidate. Our original three candidates were Teammates, Apereo Sakai, and Amara. Teammates is a project that allows teachers and students to give feedback to each other. Apereo Sakai is a learning management system with an intuitive modern design which include a gradebook, lessons tools, and a testing/quizzing tool. Amara is a subtitle editor that makes it easy to caption and translate videos.

Our group selected Teammates as our project, but we had a lot of difficulty getting it to build on all of our computers. After about two weeks of being unsuccessful, we decided to switch projects. Apereo Sakai's source code was very hard to understand, and the documentation wasn't as easy to follow as we initially thought. Before even looking at Amara again, we stumbled upon OpenMRS, and we were all successful at getting the project to build on all of our computers. OpenMRS is well-documented, and the methods are relatively easy to understand.

Chapter 2: Producing a Detailed Test Plan

2.1 Our Test Plan:

Our task for this deliverable was to produce a detailed test plan for five of the eventual twenty-five tests for our project. We used the following testing schedule to plan out our work for the remainder of the semester.

Testing Schedule:

10/03/2017 - Produce a detailed test plan of five test cases.

10/31/2017 - Design and build an automated testing framework to implement our test plan.

11/14/2017 - Complete the design and implementation of the testing framework. Create 25 test cases that the framework will automatically use to test OpenMRS.

11/21/2017 - Design and inject 5 faults into the code we are testing that will cause at least 5 tests to fail, but not all.

2.2 First Five Test Cases:

Test Case: 001

Requirement Being Tested	This is testing that the compareNatural method will return a 0 if the two strings are equal
Component Being Tested	compareNaturalAscii(String s, String t)
Class	NaturalStrings.java
Driver Being Used	NaturalStringDriver
Test Inputs	test, test
Expected Result	0
Package	org.openmrs.util.

Test Case: 002

Requirement Being Tested	This is testing that the compareNatural method will return a negative number if s lexicographically precedes t
Component Being Tested	compareNaturalAscii(String s, String t)
Class	NaturalStrings.java
Driver Being Used	NaturalStringDriver
Test Inputs	test, white
Expected Result	-3
Package	org.openmrs.util.

Test Case: 003

Requirement Being Tested	This is testing that the compareNatural method will return a positive number if s lexicographically follows t
Component Being Tested	compareNaturalAscii(String s, String t)
Class	NaturalStrings.java
Driver Being Used	NaturalStringDriver
Test Inputs	test, string
Expected Result	1
Package	org.openmrs.util.

Test Case: 004

Requirement Being Tested	This is testing that the compareNatural method will return a positive number if s lexicographically follows t
Component Being Tested	compareNaturalAscii(String s, String t)
Class	NaturalStrings.java
Driver Being Used	NaturalStringDriver
Test Inputs	test, check
Expected Result	17
Package	org.openmrs.util.

Test Case: 005

Requirement Being Tested	This is testing that the compareNatural method will return a positive number if s lexicographically follows t
Component Being Tested	compareNaturalAscii(String s, String t)
Class	NaturalStrings.java
Driver Being Used	NaturalStringDriver
Test Inputs	test, pneumonia
Expected Result	4
Package	org.openmrs.util.

Chapter 3: Automated Testing Framework

3.1 Test and Framework Description

Our task for deliverable 3 was to design and build an automated testing framework that implemented and ran our tests that we designed. The structure of the testing framework was to separate the test case test files into one directory (testCases) and the drivers that run the tests into another directory (testCasesExecutables). The bash script which is used to actually execute the tests and grabs information from both the test case text files and the drivers is located in the "scripts" directory with the file name runAllTests.sh. The script is able to navigate through the different directories and is able to pull out the information needed to run the tests.

3.2 Quick Start Guide

1. Choose a designated directory on your machine and clone our project by opening up the terminal and typing:
git clone https://github.com/CSCI-362-02-2017/404-Error.git

2. Ensure that you have at least Java JDK of minimum version 8.

3. Navigate to the TestAutomation/scripts folder and type:
bash runAllTests.sh

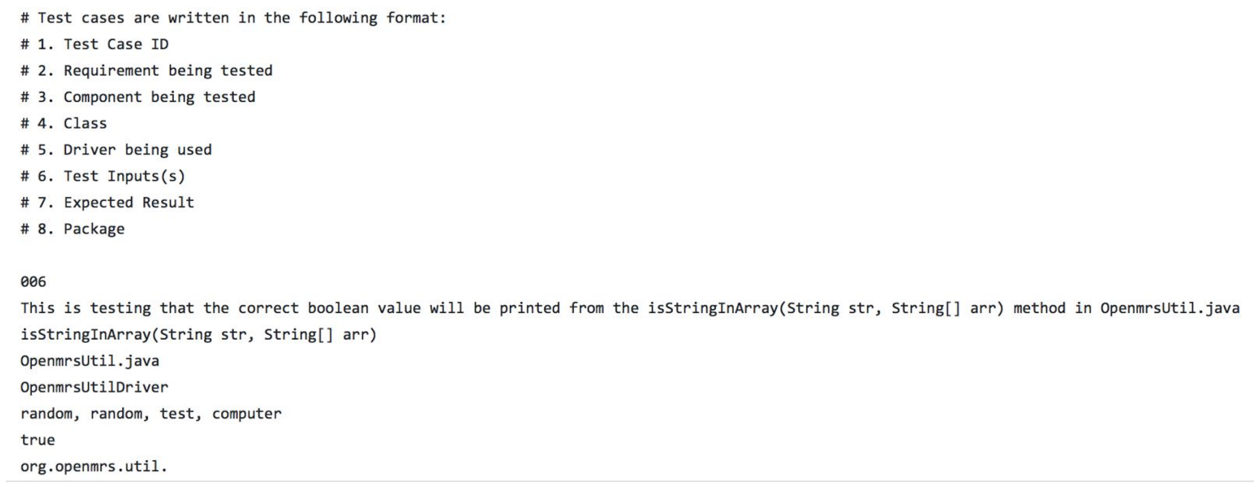
4. The results should be displayed in your terminal as well as in a browser.

Chapter 4: Creating Last 20 Test Cases

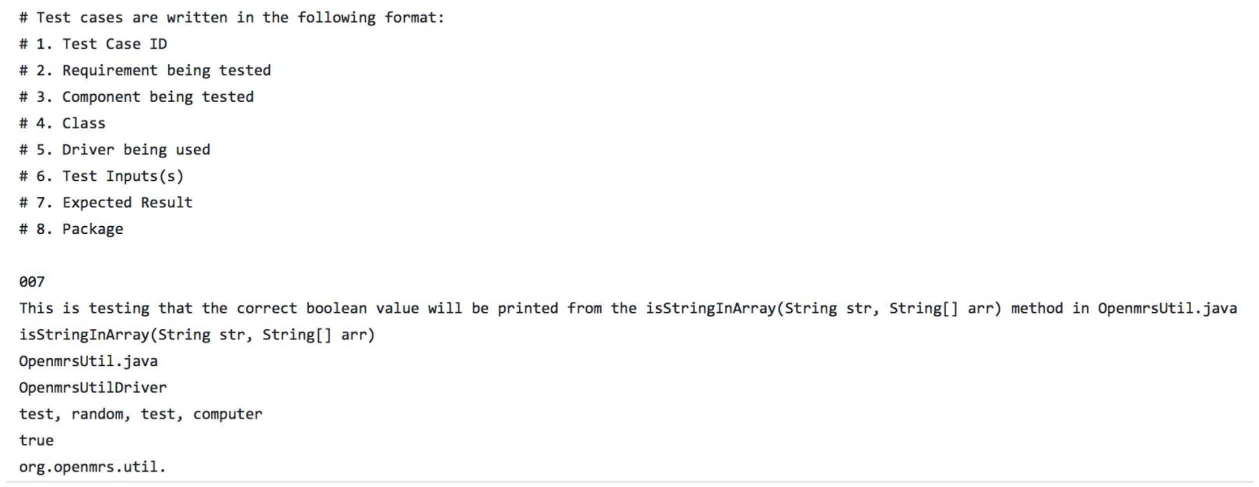
4.1 Finding Test Cases

Our Task was to complete the design and implementation of your testing framework as specified in Deliverable #3. Our team wrote the rest of the twenty test cases. Five test cases for the isStringInArray(String str, String[] arr) method in the OpenmrsUtil class. Five more test cases for the containsDigit(String str) method in the OpenmrsUtil class. Five more test cases for the containsOnlyDigits(String str) method in the OpenmrsUtil class. The last five test cases for the compareNaturalIgnoreCaseAscii(String s, String t) method in the NaturalStrings class.

Below are the rest of our test cases:



```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

006
This is testing that the correct boolean value will be printed from the isStringInArray(String str, String[] arr) method in OpenmrsUtil.java
isStringInArray(String str, String[] arr)
OpenmrsUtil.java
OpenmrsUtilDriver
random, random, test, computer
true
org.openmrs.util.
```



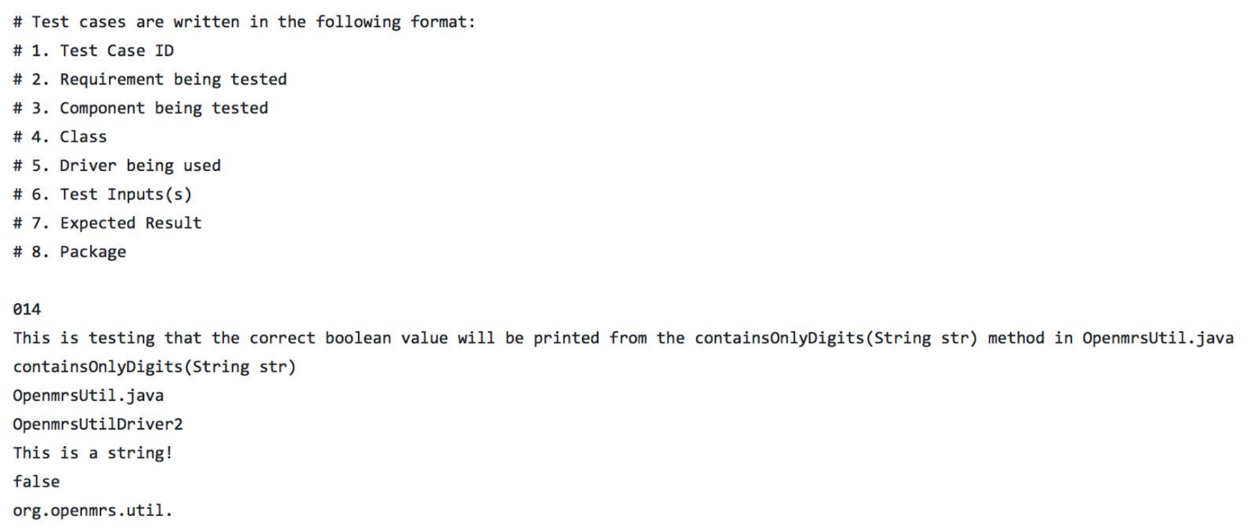
```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

007
This is testing that the correct boolean value will be printed from the isStringInArray(String str, String[] arr) method in OpenmrsUtil.java
isStringInArray(String str, String[] arr)
OpenmrsUtil.java
OpenmrsUtilDriver
test, random, test, computer
true
org.openmrs.util.
```

```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

008
This is testing that the correct boolean value will be printed from the isStringInArray(String str, String[] arr) method in OpenmrsUtil.java
isStringInArray(String str, String[] arr)
OpenmrsUtil.java
OpenmrsUtilDriver
computer, random, test, computer
true
org.openmrs.util.


# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

009
This is testing that the correct boolean value will be printed from the isStringInArray(String str, String[] arr) method in OpenmrsUtil.java
isStringInArray(String str, String[] arr)
OpenmrsUtil.java
OpenmrsUtilDriver
allergy, random, test, computer
false
org.openmrs.util.


# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

010
This is testing that the correct boolean value will be printed from the isStringInArray(String str, String[] arr) method in OpenmrsUtil.java
isStringInArray(String str, String[] arr)
OpenmrsUtil.java
OpenmrsUtilDriver
medicine, random, test, computer
false
org.openmrs.util.
```

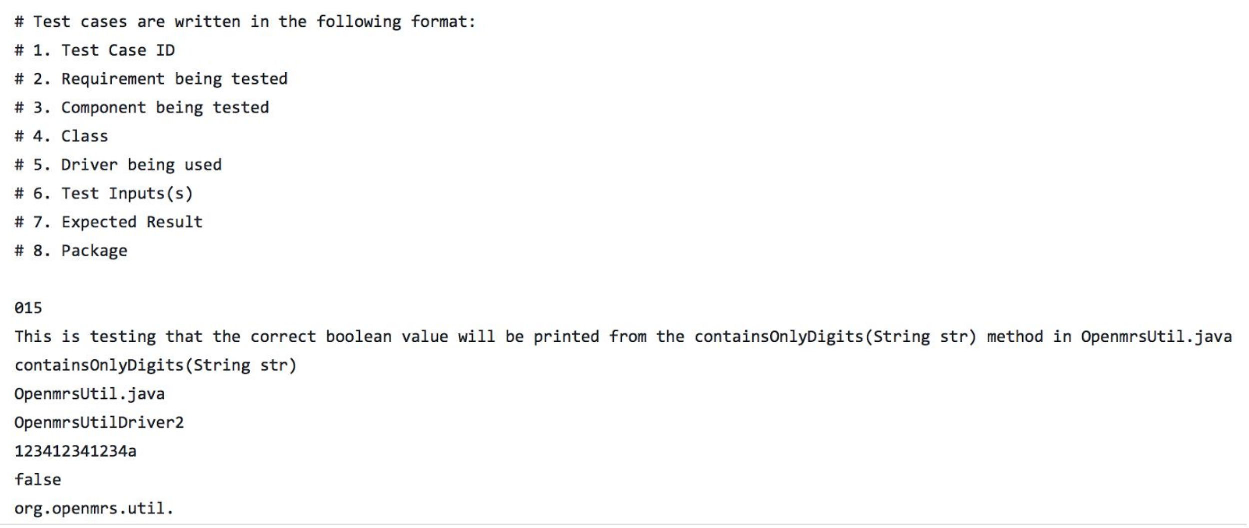
```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

011
This is testing that the correct boolean value will be printed from the containsOnlyDigits(String str) method in OpenmrsUtil.java
containsOnlyDigits(String str)
OpenmrsUtil.java
OpenmrsUtilDriver2
12345
true
org.openmrs.util.
```

```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

012
This is testing that the correct boolean value will be printed from the containsOnlyDigits(String str) method in OpenmrsUtil.java
containsOnlyDigits(String str)
OpenmrsUtil.java
OpenmrsUtilDriver2
67930
true
org.openmrs.util.
```

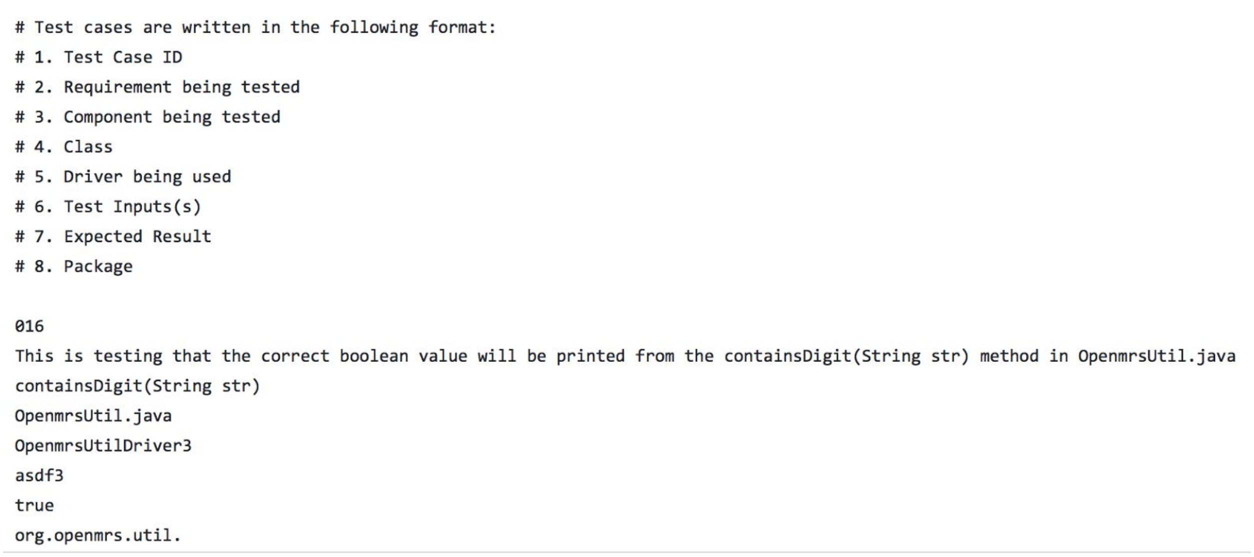
```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

013
This is testing that the correct boolean value will be printed from the containsOnlyDigits(String str) method in OpenmrsUtil.java
containsOnlyDigits(String str)
OpenmrsUtil.java
OpenmrsUtilDriver2
99999999
true
org.openmrs.util.
```



```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

014
This is testing that the correct boolean value will be printed from the containsOnlyDigits(String str) method in OpenmrsUtil.java
containsOnlyDigits(String str)
OpenmrsUtil.java
OpenmrsUtilDriver2
This is a string!
false
org.openmrs.util.
```



```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

015
This is testing that the correct boolean value will be printed from the containsOnlyDigits(String str) method in OpenmrsUtil.java
containsOnlyDigits(String str)
OpenmrsUtil.java
OpenmrsUtilDriver2
123412341234a
false
org.openmrs.util.
```

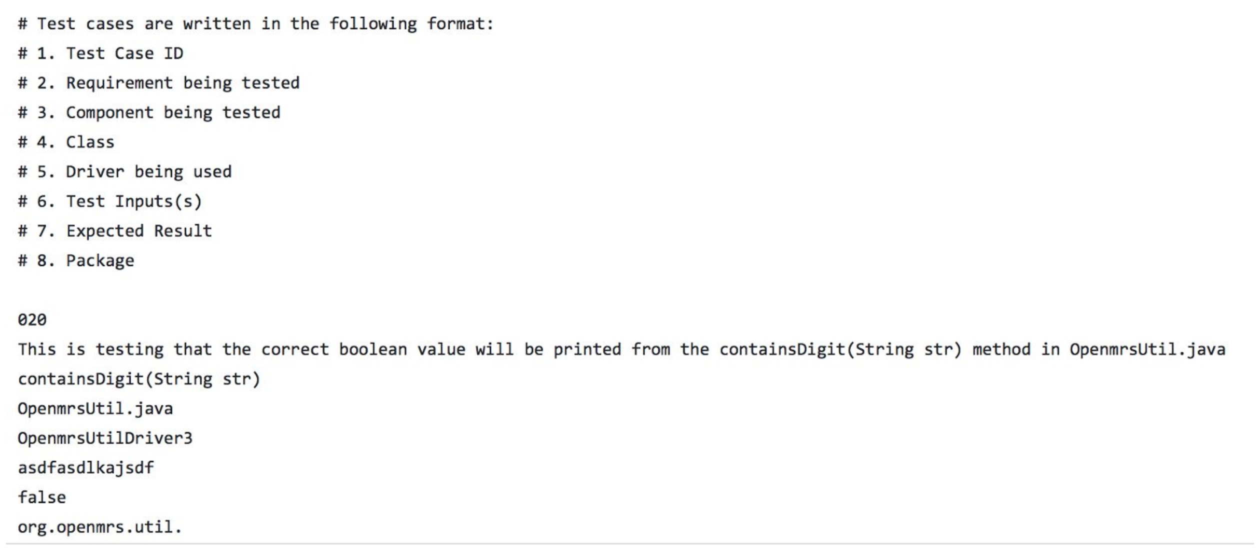


```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

016
This is testing that the correct boolean value will be printed from the containsDigit(String str) method in OpenmrsUtil.java
containsDigit(String str)
OpenmrsUtil.java
OpenmrsUtilDriver3
asdf3
true
org.openmrs.util.
```

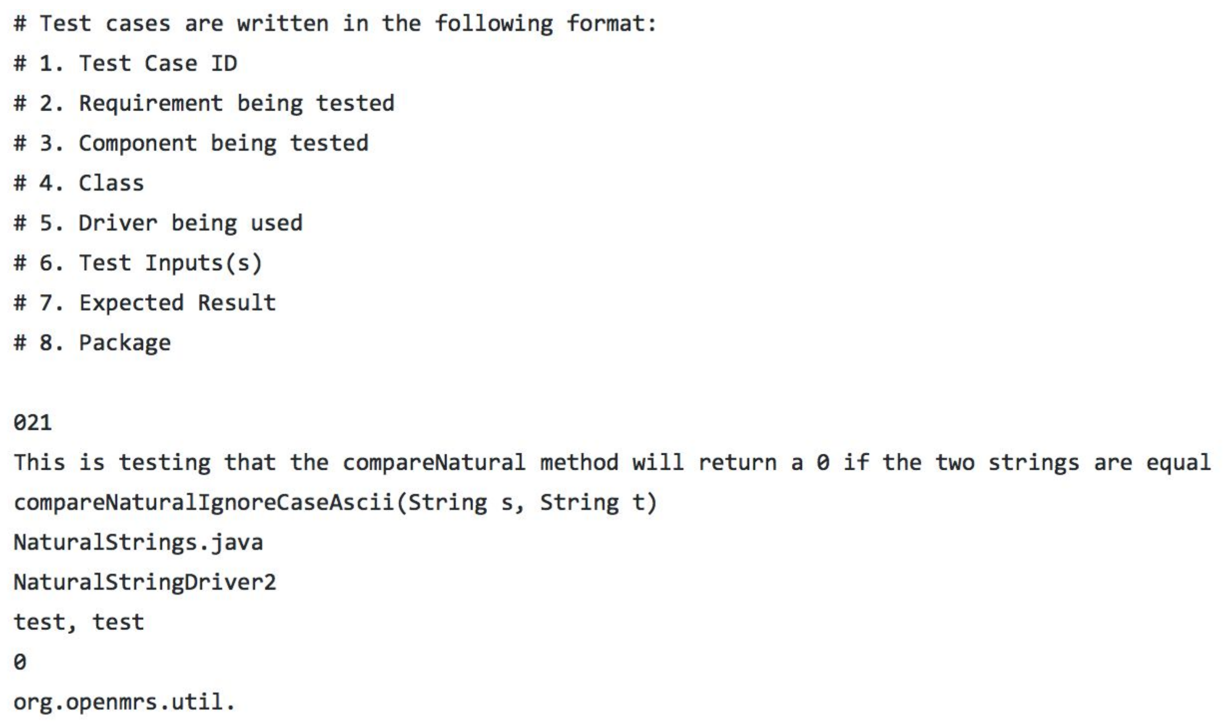
```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

017
This is testing that the correct boolean value will be printed from the containsDigit(String str) method in OpenmrsUtil.java
containsDigit(String str)
OpenmrsUtil.java
OpenmrsUtilDriver3
ddddddd9
true
org.openmrs.util.
```

```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

018
This is testing that the correct boolean value will be printed from the containsDigit(String str) method in OpenmrsUtil.java
containsDigit(String str)
OpenmrsUtil.java
OpenmrsUtilDriver3
11234
true
org.openmrs.util.
```

```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

019
This is testing that the correct boolean value will be printed from the containsDigit(String str) method in OpenmrsUtil.java
containsDigit(String str)
OpenmrsUtil.java
OpenmrsUtilDriver3
string
false
org.openmrs.util.
```



```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

020
This is testing that the correct boolean value will be printed from the containsDigit(String str) method in OpenmrsUtil.java
containsDigit(String str)
OpenmrsUtil.java
OpenmrsUtilDriver3
asdfasdlkajsdf
false
org.openmrs.util.
```



```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

021
This is testing that the compareNatural method will return a 0 if the two strings are equal
compareNaturalIgnoreCaseAscii(String s, String t)
NaturalStrings.java
NaturalStringDriver2
test, test
0
org.openmrs.util.
```

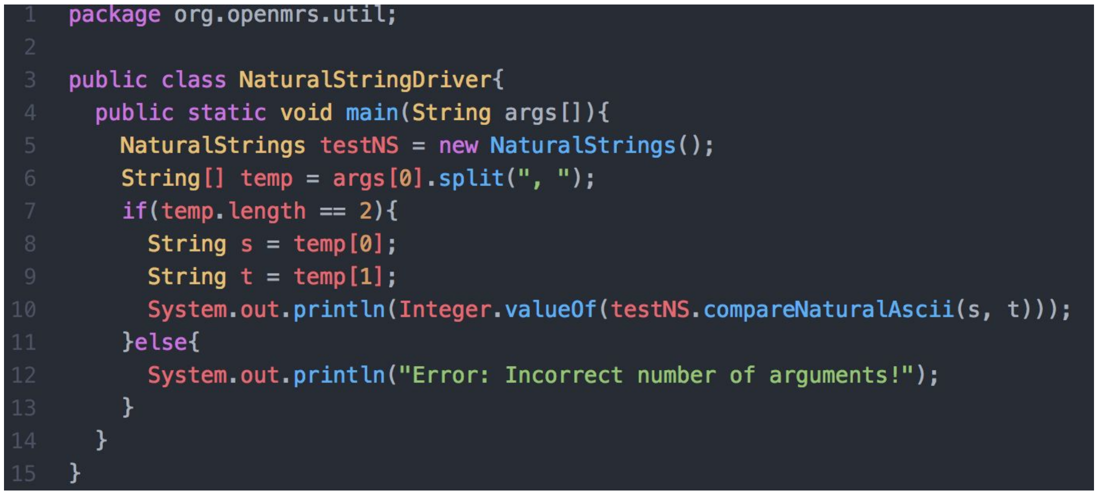
```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

022
Testing that the compareNatural method will return a negative number if s lexicographically precedes t
compareNaturalIgnoreCaseAscii(String s, String t)
NaturalStrings.java
NaturalStringDriver2
Test, whITe
-3
org.openmrs.util.
```

```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

023
Testing that the compareNatural method will return a positive number if s lexicographically followes t
compareNaturalIgnoreCaseAscii(String s, String t)
NaturalStrings.java
NaturalStringDriver2
test, strING
1
org.openmrs.util.
```

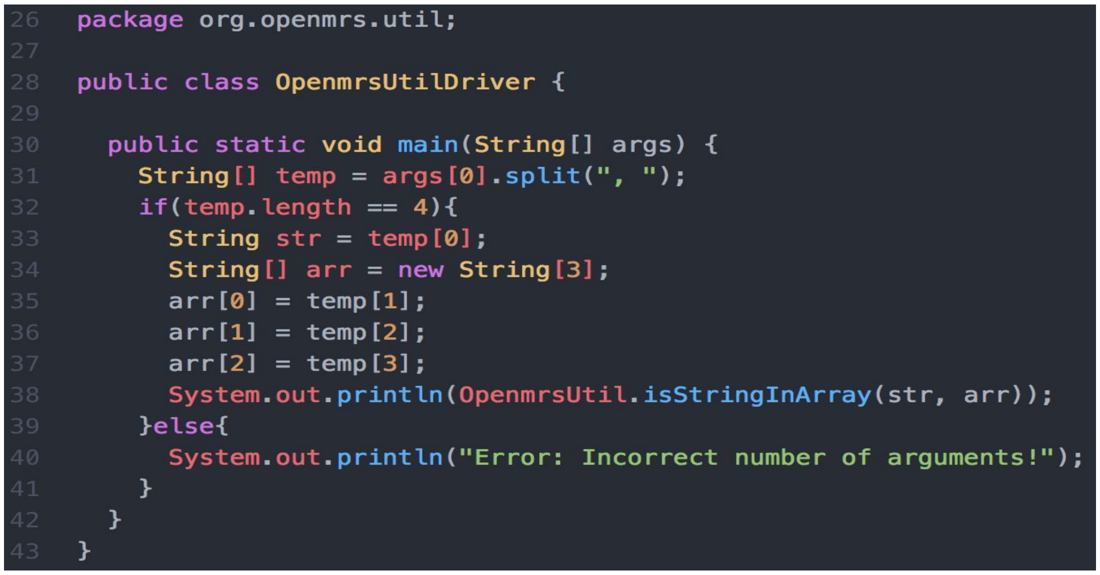
```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

024
Testing that the compareNatural method will return a positive number if s lexicographically follows t
compareNaturalIgnoreCaseAscii(String s, String t)
NaturalStrings.java
NaturalStringDriver2
TEST, Check
17
org.openmrs.util.
```

```
# Test cases are written in the following format:
# 1. Test Case ID
# 2. Requirement being tested
# 3. Component being tested
# 4. Class
# 5. Driver being used
# 6. Test Inputs(s)
# 7. Expected Result
# 8. Package

025
Testing that the compareNatural method will return a positive number if s lexicographically follows t
compareNaturalIgnoreCaseAscii(String s, String t)
NaturalStrings.java
NaturalStringDriver2
tESt, pneUMOnia
4
org.openmrs.util.
```

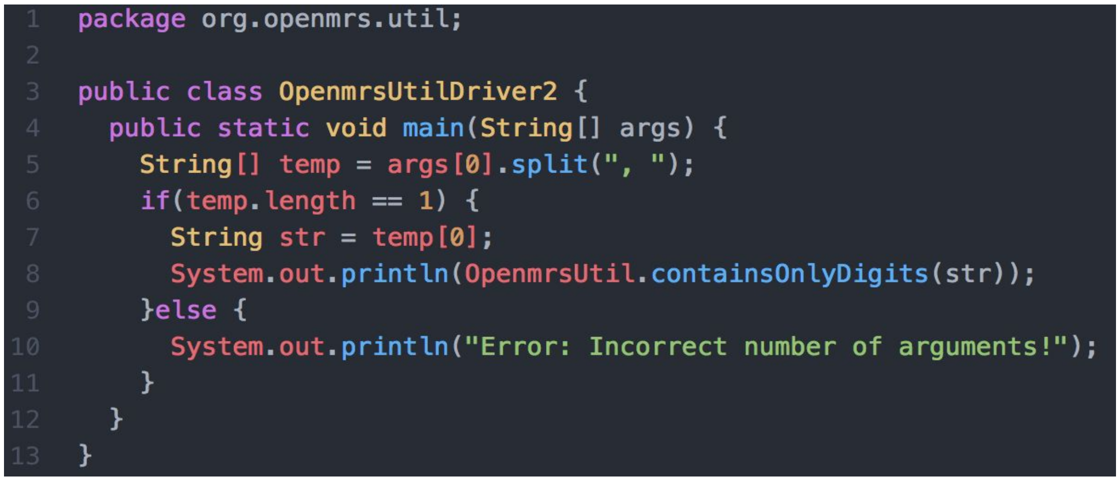
4.2 Drivers:



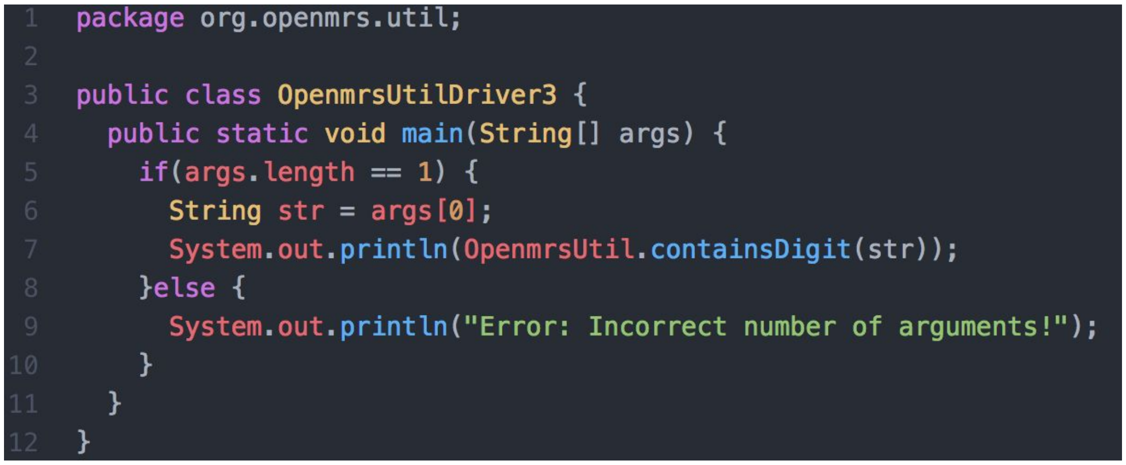
```java
package org.openmrs.util;

public class NaturalStringDriver{
  public static void main(String args[]){
    NaturalStrings testNS = new NaturalStrings();
    String[] temp = args[0].split(", ");
    if(temp.length == 2){
      String s = temp[0];
      String t = temp[1];
      System.out.println(Integer.valueOf(testNS.compareNaturalAscii(s, t)));
    }else{
      System.out.println("Error: Incorrect number of arguments!");
    }
  }
}
```



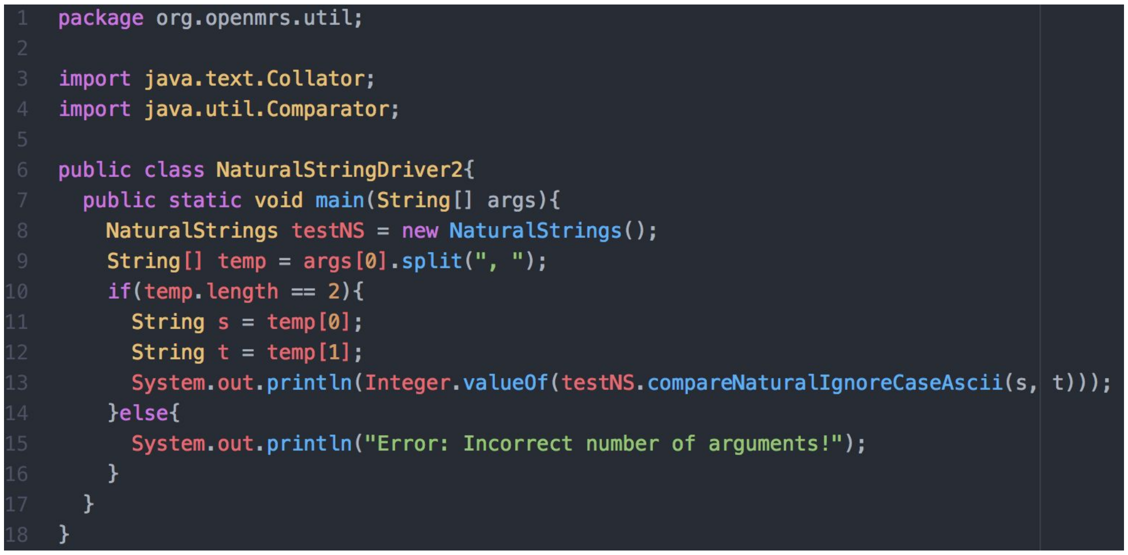
```java
package org.openmrs.util;

public class OpenmrsUtilDriver {

  public static void main(String[] args) {
    String[] temp = args[0].split(", ");
    if(temp.length == 4){
      String str = temp[0];
      String[] arr = new String[3];
      arr[0] = temp[1];
      arr[1] = temp[2];
      arr[2] = temp[3];
      System.out.println(OpenmrsUtil.isStringInArray(str, arr));
    }else{
      System.out.println("Error: Incorrect number of arguments!");
    }
  }
}
```



```java
package org.openmrs.util;

public class OpenmrsUtilDriver2 {
  public static void main(String[] args) {
    String[] temp = args[0].split(", ");
    if(temp.length == 1) {
      String str = temp[0];
      System.out.println(OpenmrsUtil.containsOnlyDigits(str));
    }else {
      System.out.println("Error: Incorrect number of arguments!");
    }
  }
}
```



```java
package org.openmrs.util;

public class OpenmrsUtilDriver3 {
  public static void main(String[] args) {
    if(args.length == 1) {
      String str = args[0];
      System.out.println(OpenmrsUtil.containsDigit(str));
    }else {
      System.out.println("Error: Incorrect number of arguments!");
    }
  }
}
```



```java
package org.openmrs.util;

import java.text.Collator;
import java.util.Comparator;

public class NaturalStringDriver2{
  public static void main(String[] args){
    NaturalStrings testNS = new NaturalStrings();
    String[] temp = args[0].split(", ");
    if(temp.length == 2){
      String s = temp[0];
      String t = temp[1];
      System.out.println(Integer.valueOf(testNS.compareNaturalIgnoreCaseAscii(s, t)));
    }else{
      System.out.println("Error: Incorrect number of arguments!");
    }
  }
}
```

This part was pretty straight forward in what was required. We needed to find 4 new methods to have 20 new test cases to test. Our group found methods to use fairly quickly and easily without problems. We ran into some issues when trying to get some of our drivers to work for a couple of the new methods, but after looking at them for a while, we were able to alleviate our issues and make them work. Our HTML output was still very basic in its implementation and presentation, but that is something that we improved upon for the next deliverable to make it more aesthetically pleasing. We still had our driver and classes compiling directly by reference in our script, but that is something that we were aware of how to fix and did soon after. Overall, the team experience for this deliverable was good, and it went smoother than previous deliverables that we had issues on.

Chapter 5: Fault Injections

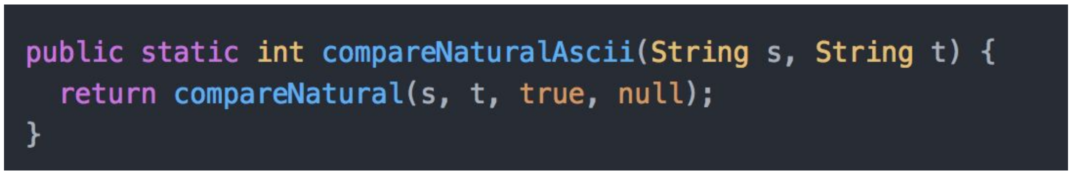
Our task was to design and inject 5 faults into the code we were testing that will cause at least 5 tests to fail, but hopefully not all tests to fail. Then, exercise our framework and analyze the results. A single fault was inserted into each of our five methods. All faults essentially flip a boolean value (true to false or false to true). The majority of our test cases still passed. Our first method, compareNaturalAscii(String s, String t), was the only method in which our test case results remained the same even after the fault was injected.

5.1 First Fault Injection:

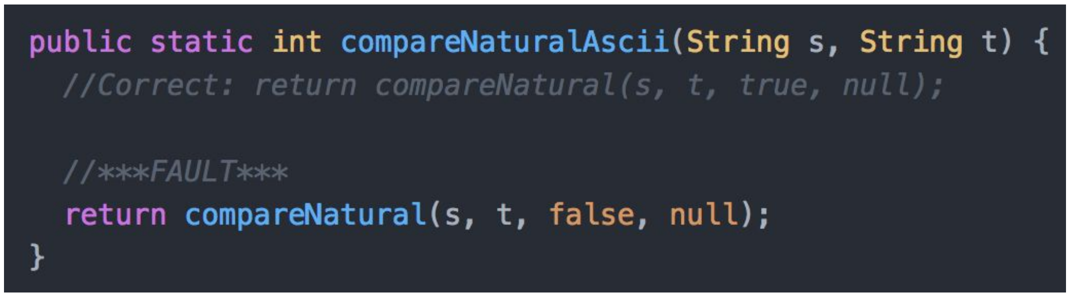
Method 1: compareNaturalAscii(String s, String t)
Test Cases: 001 - 005
Description of Fault: The third (boolean) parameter in the method called within compareNaturalAscii(String s, String t) was switched from true to false. That parameter deals with case sensitivity (true if you want it, false if you don't).
Test Case Results:

Original: 5 Passed / 0 Failed
After Fault: 5 Passed / 0 Failed

Analysis: All of our tests were already using strings that were all lowercase, so the change in case sensitivity had no effect on these specific test cases.



```
public static int compareNaturalAscii(String s, String t) {
  return compareNatural(s, t, true, null);
}
```



```
public static int compareNaturalAscii(String s, String t) {
  //Correct: return compareNatural(s, t, true, null);

  //***FAULT***
  return compareNatural(s, t, false, null);
}
```

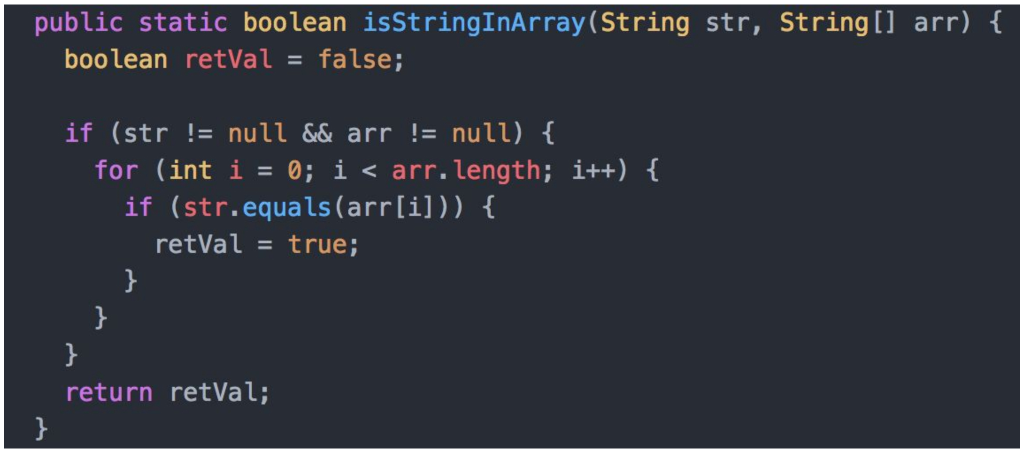
5.2 Second Fault Injection:

Method 2: isStringInArray(String str, String[] arr)
Test Cases: 006 - 010
Description of Fault: The variable retVal has been initialized to true instead of false. This method will always return true.
Test Case Results:

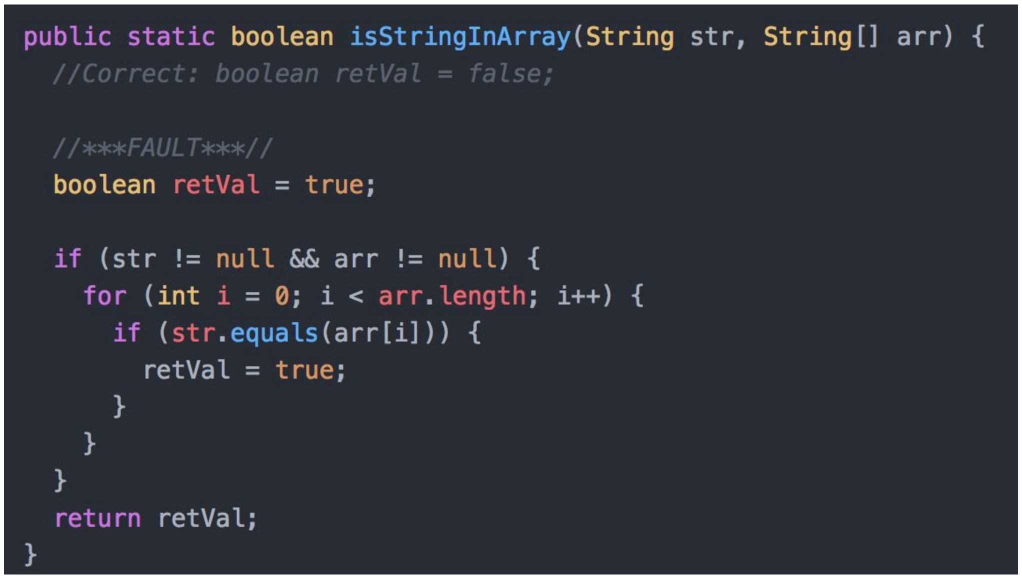
  Original: 5 Passed / 0 Failed

  After Fault: 3 Passed (006,007,008) / 2 Failed (009,010)

Analysis: Because of the fault, the method will always return true meaning that the input string will always be considered inside the input string array, even if it is not. In our first three test cases, the input string is actually in the input string array, so those three still passed. In the last two tests, the string was not in the array, so those should have failed.



```java
public static boolean isStringInArray(String str, String[] arr) {
  boolean retVal = false;

  if (str != null && arr != null) {
    for (int i = 0; i < arr.length; i++) {
      if (str.equals(arr[i])) {
        retVal = true;
      }
    }
  }
  return retVal;
}
```



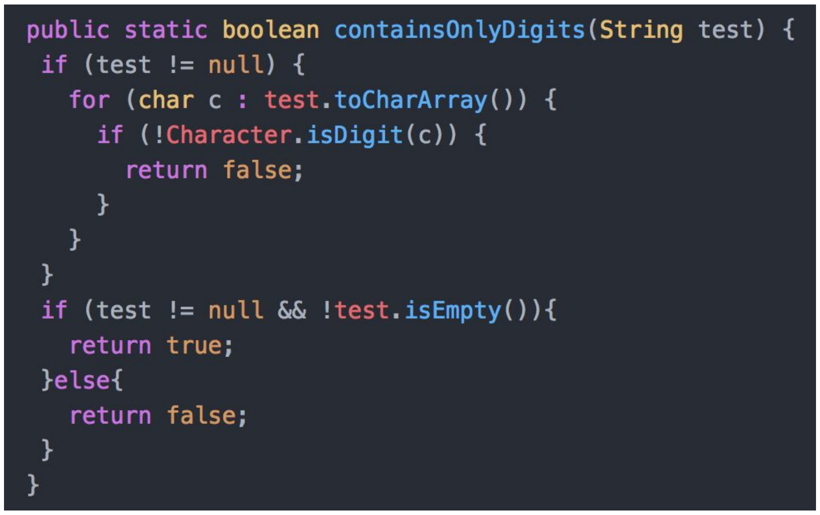
```java
public static boolean isStringInArray(String str, String[] arr) {
  //Correct: boolean retVal = false;

  //***FAULT***//
  boolean retVal = true;

  if (str != null && arr != null) {
    for (int i = 0; i < arr.length; i++) {
      if (str.equals(arr[i])) {
        retVal = true;
      }
    }
  }
  return retVal;
}
```

5.3 Third Fault Injection:

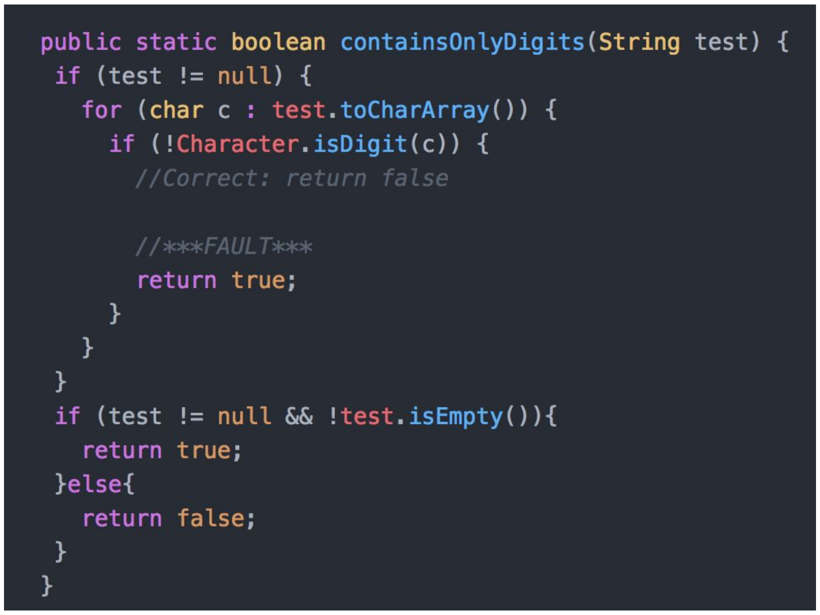
Method 3: containsOnlyDigits(String test)
Test Cases: 011 - 015
Description of Fault: The method now returns true instead of false for a non-empty string if the string contains a letter.
Test Case Results:

Original: 5 Passed / 0 Failed
After Fault: 3 Passed (011,012,013) / 2 Failed (014,015)

Analysis: Our first 3 test cases for this method still passed because they use input strings that aren't null or empty. The last two test cases failed because the first letter the method would see would make the method return true, which contradicted the expected output.



```java
public static boolean containsOnlyDigits(String test) {
 if (test != null) {
   for (char c : test.toCharArray()) {
     if (!Character.isDigit(c)) {
       return false;
     }
   }
 }
 if (test != null && !test.isEmpty()){
   return true;
 }else{
   return false;
 }
}
```



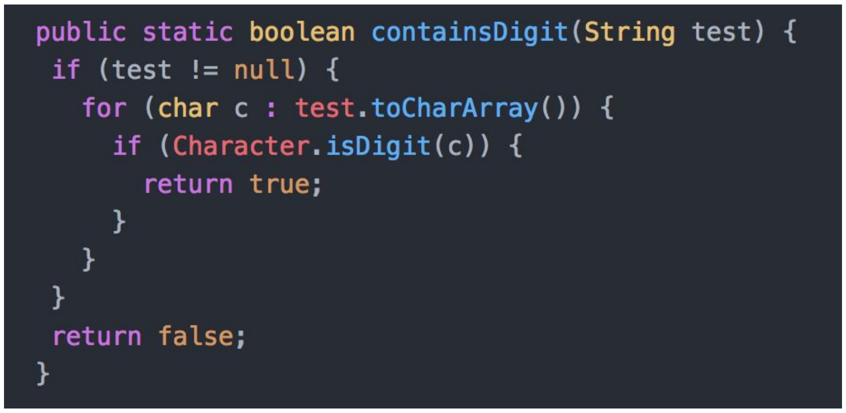
```java
 public static boolean containsOnlyDigits(String test) {
  if (test != null) {
    for (char c : test.toCharArray()) {
      if (!Character.isDigit(c)) {
        //Correct: return false

        //***FAULT***
        return true;
      }
    }
  }
  if (test != null && !test.isEmpty()){
    return true;
  }else{
    return false;
  }
 }
```

5.4 Fourth Fault Injection:

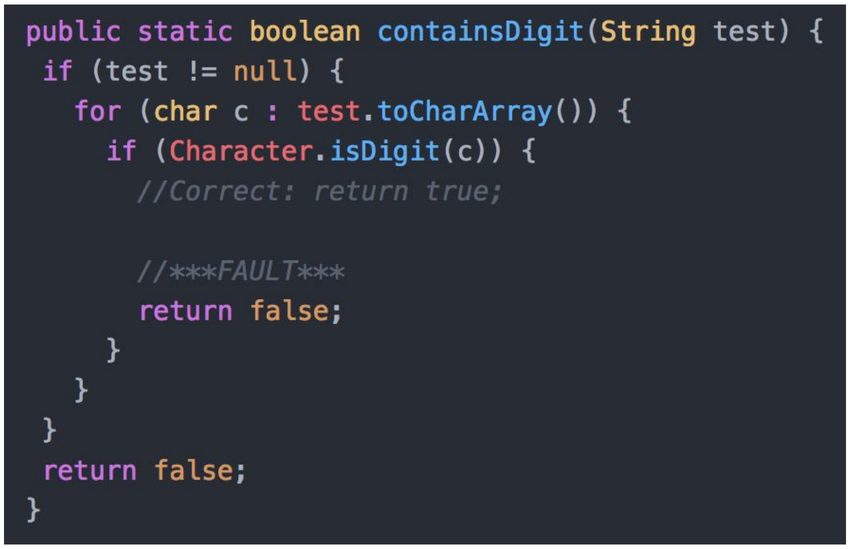
Method 4: containsDigit(String test)
Test Cases: 016 - 020
Description of Fault: The method now returns false if a digit appears in a string.
Test Case Results:

> **Original:** 5 Passed / 0 Failed
> **After Fault:** 2 Passed (019,020) / 3 Failed (016,017,018)

Analysis: Our first three test cases that failed after the fault injection analyze input strings with at least one digit in them which results in the method returning false. The method will also return false regardless.



```java
public static boolean containsDigit(String test) {
 if (test != null) {
   for (char c : test.toCharArray()) {
     if (Character.isDigit(c)) {
       return true;
     }
   }
 }
 return false;
}
```



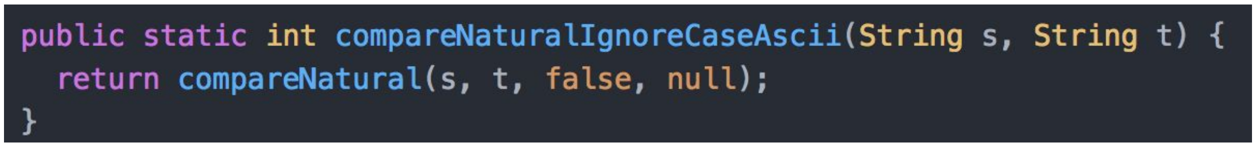
```java
public static boolean containsDigit(String test) {
 if (test != null) {
   for (char c : test.toCharArray()) {
     if (Character.isDigit(c)) {
       //Correct: return true;

       //***FAULT***
       return false;
     }
   }
 }
 return false;
}
```

5.5 Fifth Fault Injection:

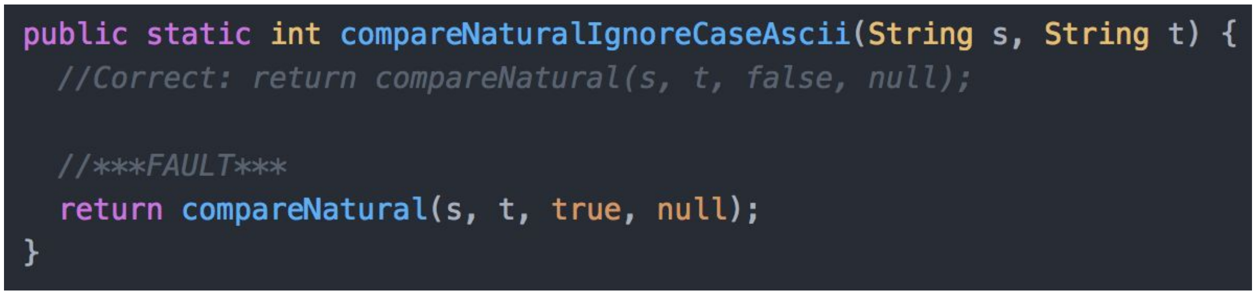
Method 5: compareNaturalIgnoreCaseAscii(String s, String t)
Test Cases: 021 - 025
Description of Fault: The third (boolean) parameter in the method called within compareNaturalIgnoreCaseAscii(String s, String t) was switched from false to true. That parameter deals with case sensitivity (true if you want it, false if you don't).
Test Case Results:

Original: 5 Passed / 0 Failed
After Fault: 4 Passed (021,023,024,025) / 1 Failed (022)

Analysis: Because of the fault, case sensitivity is now being taken into account. The only test case to fail because of the fault is 22 because capital 'T' is now being compared to lower case 'w'.



```java
public static int compareNaturalIgnoreCaseAscii(String s, String t) {
  return compareNatural(s, t, false, null);
}
```



```java
public static int compareNaturalIgnoreCaseAscii(String s, String t) {
  //Correct: return compareNatural(s, t, false, null);

  //***FAULT***
  return compareNatural(s, t, true, null);
}
```

Chapter 6: Evaluations

6.1 Overall Experience and Lessons Learned:

Overall, our team was pleased with this project. It was definitely the most tedious project that any of our members have done in the computer science program. OpenMRS was a great project to work on because it was easy to build on all of our computers, and the code was very intuitive. All of our members have a strong foundation in Java, so using OpenMRS gave us very little problems. The members of our group did not have much previous experience in writing scripts, if any, so writing the bash script was tough at first, but the more we worked on it the more we understood what we were doing and needed to do. We gained extensive knowledge in writing tests for methods and getting java to compile outside of an IDE using just the terminal and the .java and .class files. We struggled initially getting the java to compile correctly without directly stating the directory path in the script, but after conversing with Professor Bowring on the subject and learning more about using the classpath, we were able to figure it out and get our java to compile based on inputs from the test case text files.

6.2 Team Self-Evaluation:

Our team worked incredibly well together. Each member was able to contribute a good amount of work to each deliverable. Though there were times where we struggled to get something working on time, we ultimately fixed everything by the next deadline. We all took Dr.Bowring’s constructive criticism well and used it improve our work. When we first started, we did have some procrastination issues, but as the semester progressed and the project evolved, we became much better at planning out our time and giving roles for each member of the team so that everyone would be able to contribute equally and have something to do.

6.3 Project Evaluation and Suggestions:

Overall, we believe that the project was a great experience. Software engineering is inherently difficult, so we already expected the class project to be difficult and time-consuming. We believe that ample time was given inside and outside of class for team members to meet and work on the project. Github tutorials were introduced at the beginning of the semester, which were nice resources. Other than the large quantity of blogs that we had this semester, everything else should be kept the same.