Data Flow Testing, Slice-Based Testing and Mutation Testing Report

Reuben Ninan - 216315509

Eric Kwok - 216397150

Luke Linigari - 216307639

Ahmed Hagi - 215043896

EECS4313

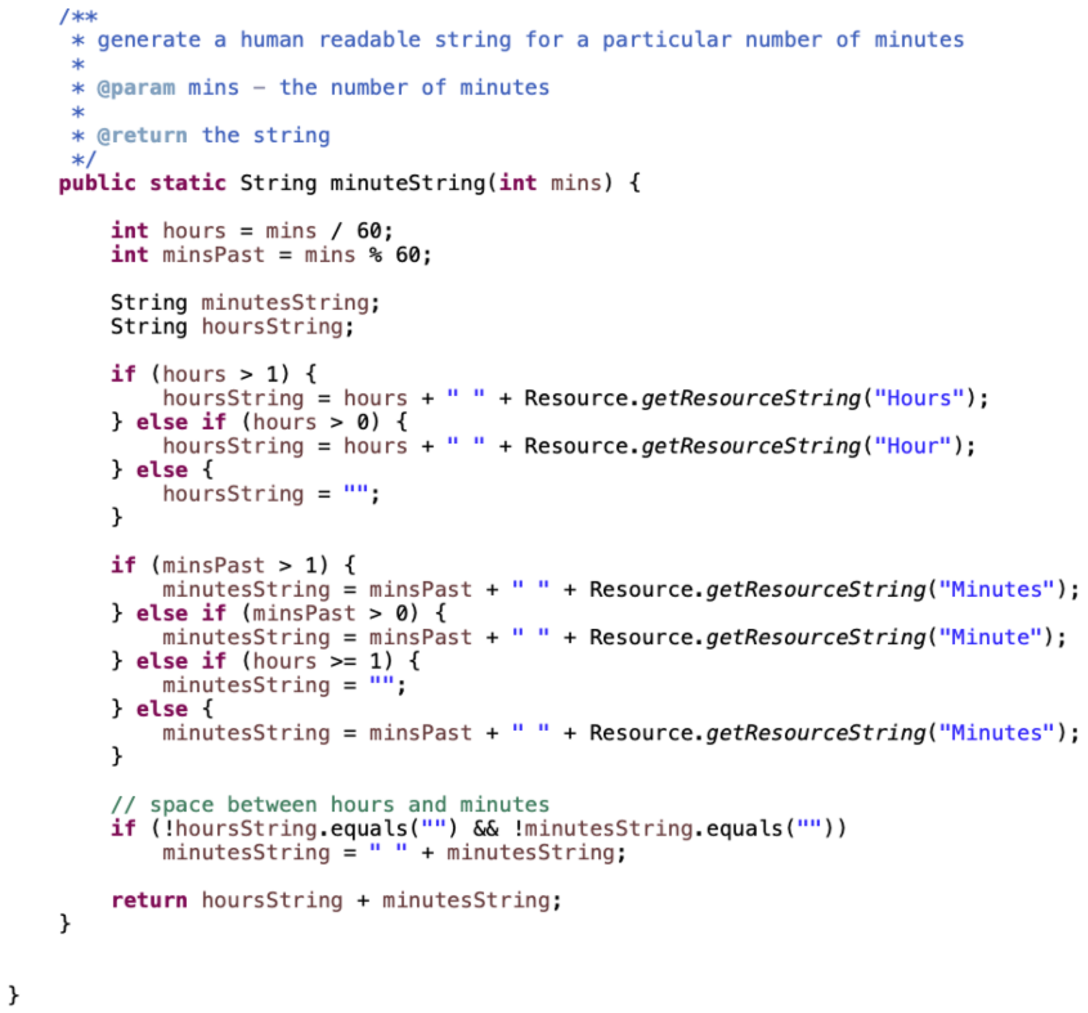
ZhenMing Jiang

December 7th, 2021

Task 1 – Borg Calendar

2.1.1 Data Flow Analysis

We chose to use the munuteString method because it contains primitive variables such as integers. None of our other methods under test contained any primitive data types, thus munuteString was chosen. There was no need for additional tests to get the path coverage to 100%. minuteString is a simple method with simple paths which can be tested vigorously with a range of values. The range of values used in the test were sufficient to get 100% coverage.



```java
    /**
     * generate a human readable string for a particular number of minutes
     *
     * @param mins - the number of minutes
     *
     * @return the string
     */
    public static String minuteString(int mins) {

        int hours = mins / 60;
        int minsPast = mins % 60;

        String minutesString;
        String hoursString;

        if (hours > 1) {
            hoursString = hours + " " + Resource.getResourceString("Hours");
        } else if (hours > 0) {
            hoursString = hours + " " + Resource.getResourceString("Hour");
        } else {
            hoursString = "";
        }

        if (minsPast > 1) {
            minutesString = minsPast + " " + Resource.getResourceString("Minutes");
        } else if (minsPast > 0) {
            minutesString = minsPast + " " + Resource.getResourceString("Minute");
        } else if (hours >= 1) {
            minutesString = "";
        } else {
            minutesString = minsPast + " " + Resource.getResourceString("Minutes");
        }

        // space between hours and minutes
        if (!hoursString.equals("") && !minutesString.equals(""))
            minutesString = " " + minutesString;

        return hoursString + minutesString;
    }

}
```

Figure 1: minuteString screenshot

Only the variables mins, hours and minsPast are primitives, thus the dc paths are generated according to their usage.

Table 1: Definition and use segmentation

Code Segment	ID	Type
public static String minuteString(int mins)	A	Def mins

int hours = mins / 60;
int minsPast = mins % 60
String minutesString;
String hoursString;
	B	Def hours.
		Def minsPast,
		C-Use mins

if (hours > 1) {	C	P-Use hours
hoursString = hours + " " + Resource.getResourceString("Hours");	D	C-Use hours
} else if (hours > 0) {	E	P-Use hours
hoursString = hours + " " + Resource.getResourceString("Hour");	F	C-Use hours

} else {
hoursString = "";
	G	

if (minsPast > 1) {	H	P-Use minsPast
minutesString = minsPast + " " + Resource.getResourceString("Minutes")	I	C-Use minsPast
} else if (minsPast > 0) {	J	P-Use minsPast
minutesString = minsPast + " " + Resource.getResourceString("Minute")	K	C-Use minsPast
} else if (hours >= 1) {	L	P-Use hours
minutesString = "";	M	

} else {
minutesString = minsPast + " " + Resource.getResourceString("Minutes")
	N	C-Use minsPast

This is the flowchart which represents the minuteString method. Each node corresponds to the blocks of code defined in table 1.

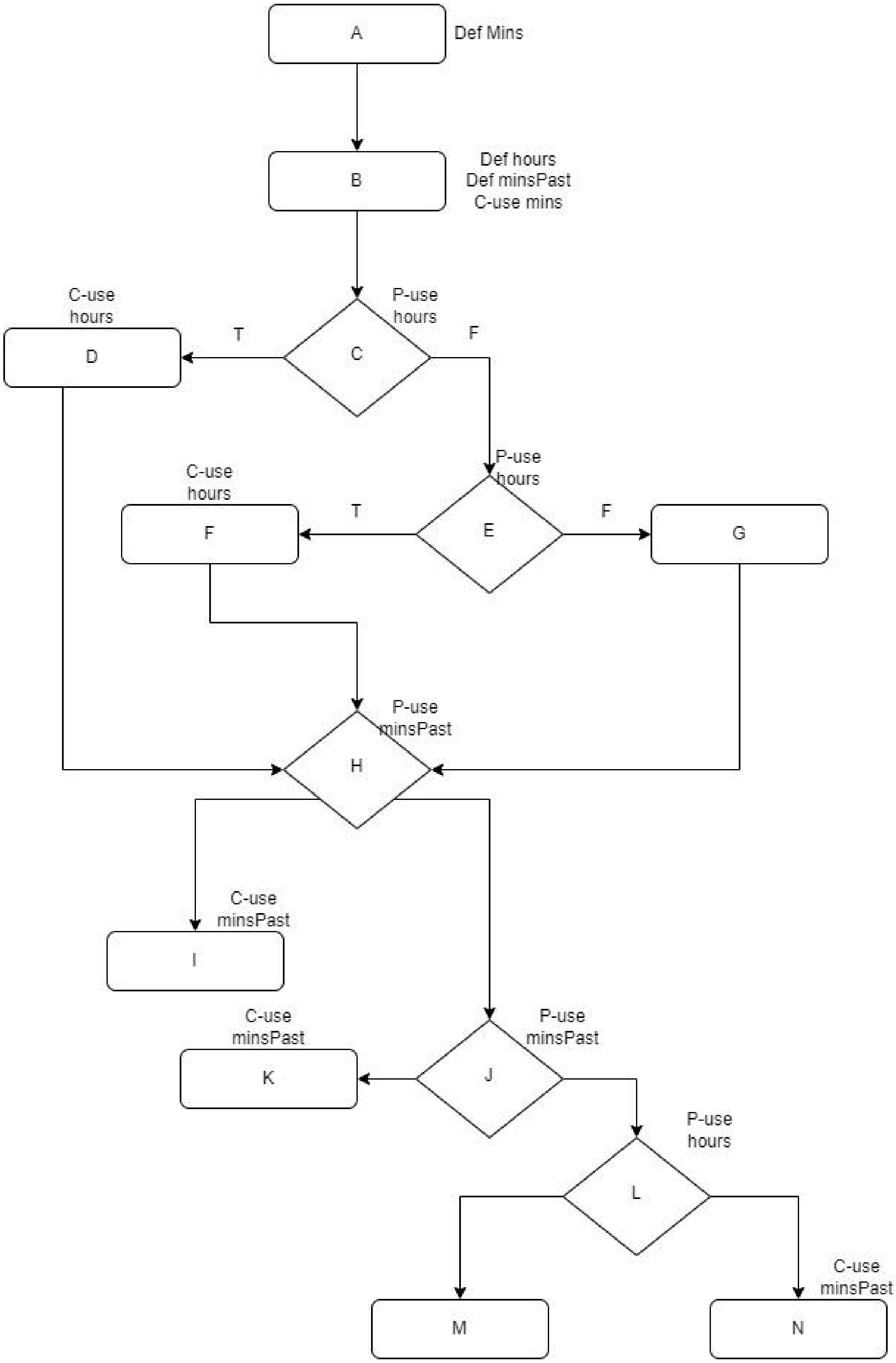


Figure 2: flow chart using the nodes defined in table 1

Tests

There were a total of 9 tests performed in the black box testing for the minuteString method. All 9 tests have a cumulative coverage of 100%. These are the tests, their names and the paths they take when going through minuteString.

Test Name	**Test Code**	**Path**

H1_M1	String s = d.minuteString(0);
	assertTrue(s.equals("0 Minutes"));
		A B C E G H J L N

H1_M2	String s = d.minuteString(1);
	assertTrue(s.equals("1 Minute"));
		A B C E G H J K

H1_M3	String s = d.minuteString(37);
	assertTrue(s.equals("37 Minutes"));
		A B C E G H I

H2_M1	String s = d.minuteString(60);
	assertTrue(s.equals("1 Hour"));
		A B C E F H J L M

H2_M2	String s = d.minuteString(61);
	assertTrue(s.equals("1 Hour 1 Minute"));
		A B C E F H J K

H2_M3	String s = d.minuteString(86);
	assertTrue(s.equals("1 Hour 26 Minutes"));
		A B C E F H I

H3_M1	String s = d.minuteString(660);
	assertTrue(s.equals("11 Hours"));
		A B C D H J L M

H3_M2	String s = d.minuteString(361);
	assertTrue(s.equals("6 Hours 1 Minute"));
		A B C D H J K

H3_M3	String s = d.minuteString(988);
	assertTrue(s.equals("16 Hours 28 Minutes"));
		A B C D H I

Coverage

Paths with (F or G) signify that a test that has a path containing either F or G in that spot can be used as a valid test for that variable coverage. It does not matter what branch is chosen as it is just used as a convenient way to get to the target variable further down.

All Uses

At least one path of each variable definition to each p-use and each c-use of the definition

Variable	Path	Test
mins	A B	Any test will suffice
hours	B C D B C E B C E F B C E (F or G) H J L	H3_M3 H1_M3 H2_M1 H2_M1
minsPast	B C E (F or G) H B C E (F or G) H I B C E (F or G) H J K B C E (F or G) H J L N	H1_M3 H1_M3 H2_M2 H1_M1

All Defs

Each definition of each variable for at least one use of the definition

Variable	Path	Test
mins	A B	All test will suffice
hours	B C	All tests will suffice
minsPast	B C E (F or G) H or(B C D H)	H1_M3 H3_M3

All-C-uses/Some-P-uses

At least one path of each variable definition to each c-use of the variable. If any variable definitions are not covered, use p-use

Variable	Path	Test
mins	A B	All tests will suffice
hours	B C D B C E F	H3_M3 H2_M1

minsPast	B C E (F or G) H I B C E (F or G) H J K B C E (F or G) H J L N	H2_M3 M2_M2 H1_M1

All-P-uses/Some-C-uses

At least one path of each variable definition to each p-use of the variable. If any variable definitions are not covered by p-use, then use c-use

Variable	Path	Test
mins	A B	All tests will suffice
hours	B C B C E B C E (F or G) H J L or(B C D H J L)	All tests will suffice H1_M1 H2_M1 H3_M1

minsPast	B C E (F or G) H
	B C E (F or G) H J
	or(B C D H)
	or(B C D H J)
		H1_M2
		H2_M2
		H3_M2
		H3_M2

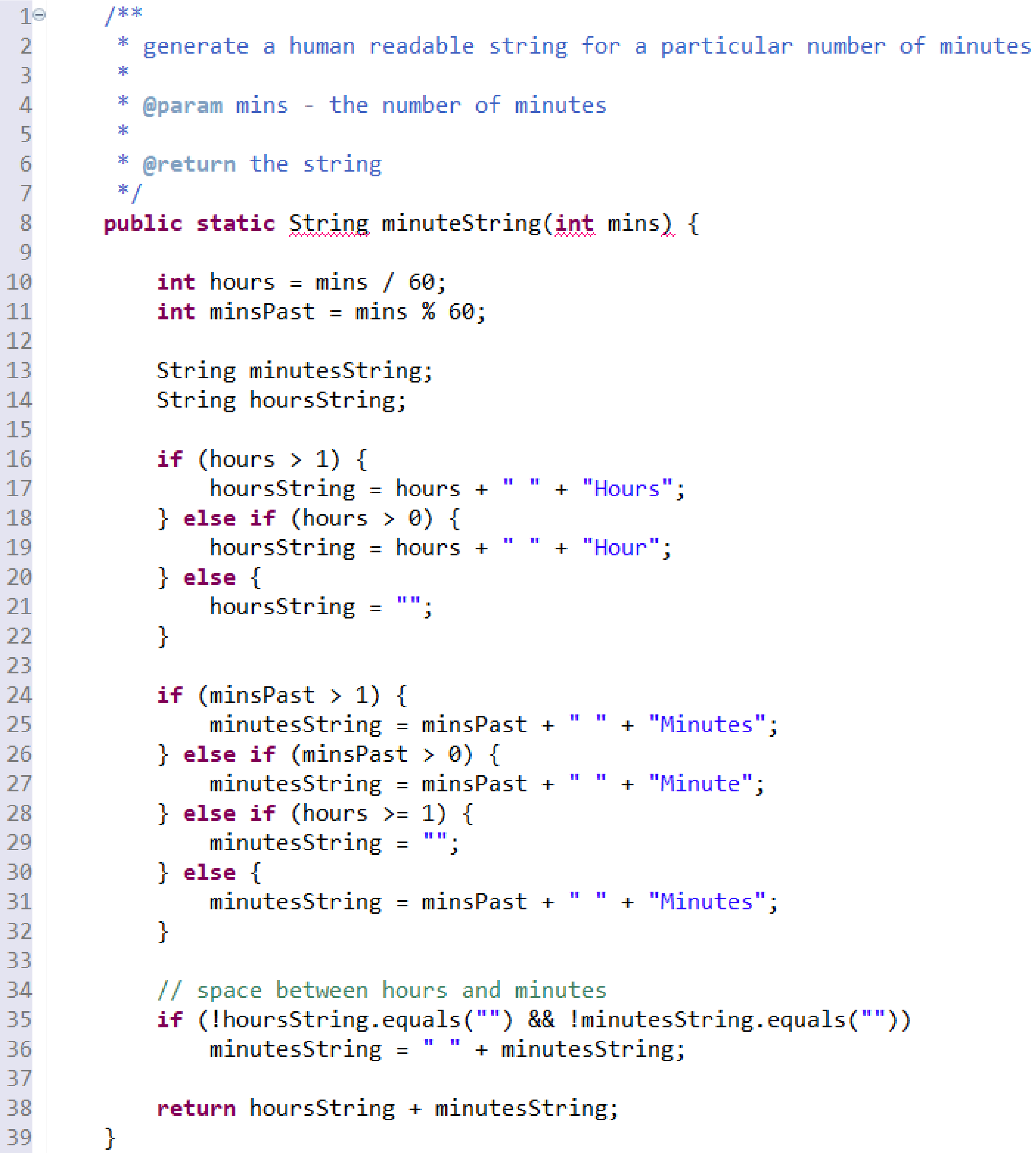
2.1.2 Slice Testing

Our goal is to derive the backward slices of the program with respect to a program point p and a set of program variables V. This consists of all statements and predicates in the program that could affect the value of values in **V at p.**

The criterion for our slices will be **Slice(Variable, At Line #)**

Our Testing covers 100% of the slices that we derived based on our data flow analysis

We will be basing our slices on this method

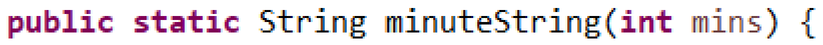


```
 1      /**
 2       * generate a human readable string for a particular number of minutes
 3       *
 4       * @param mins - the number of minutes
 5       *
 6       * @return the string
 7       */
 8      public static String minuteString(int mins) {
 9  
10          int hours = mins / 60;
11          int minsPast = mins % 60;
12  
13          String minutesString;
14          String hoursString;
15  
16          if (hours > 1) {
17              hoursString = hours + " " + "Hours";
18          } else if (hours > 0) {
19              hoursString = hours + " " + "Hour";
20          } else {
21              hoursString = "";
22          }
23  
24          if (minsPast > 1) {
25              minutesString = minsPast + " " + "Minutes";
26          } else if (minsPast > 0) {
27              minutesString = minsPast + " " + "Minute";
28          } else if (hours >= 1) {
29              minutesString = "";
30          } else {
31              minutesString = minsPast + " " + "Minutes";
32          }
33  
34          // space between hours and minutes
35          if (!hoursString.equals("") && !minutesString.equals(""))
36              minutesString = " " + minutesString;
37  
38          return hoursString + minutesString;
39      }
```

minuteString Function

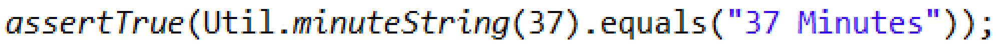
A-Def Slices

Slice(mins, 8)



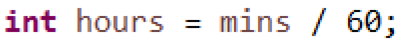
```
public static String minuteString(int mins) {
```

One of the tests that covers this is



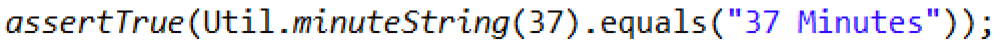
```
assertTrue(Util.minuteString(37).equals("37 Minutes"));
```

Slice(hours, 10)



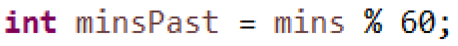
```
int hours = mins / 60;
```

One of the tests that covers this is



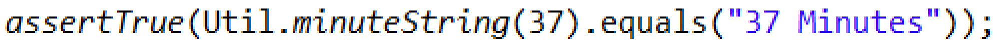
```
assertTrue(Util.minuteString(37).equals("37 Minutes"));
```

Slice(minsPast, 11)



```
int minsPast = mins % 60;
```

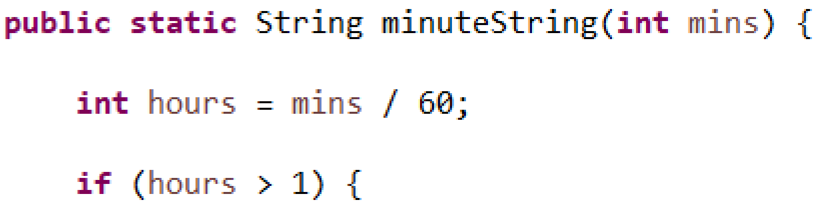
One of the tests that covers this is



```
assertTrue(Util.minuteString(37).equals("37 Minutes"));
```

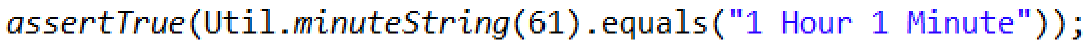
P-Use Slices

Slice(hours, 16)



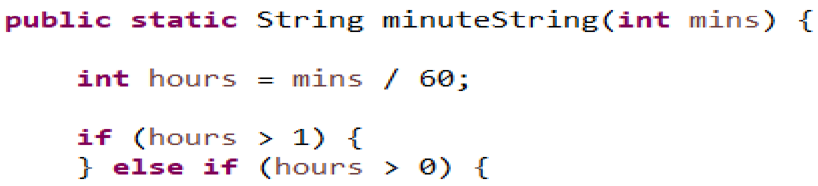
```
public static String minuteString(int mins) {
    int hours = mins / 60;
    if (hours > 1) {
```

One of the tests that covers this is



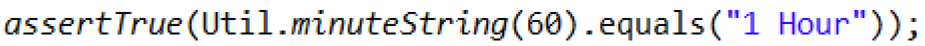
```
assertTrue(Util.minuteString(61).equals("1 Hour 1 Minute"));
```

Slice(hours, 18)



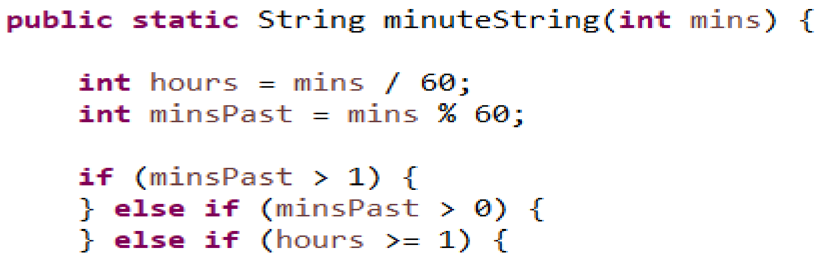
```
public static String minuteString(int mins) {

    int hours = mins / 60;

    if (hours > 1) {
    } else if (hours > 0) {
```

One of the tests that covers this is



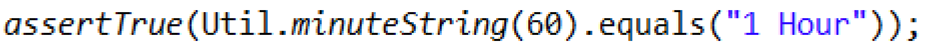
```
assertTrue(Util.minuteString(60).equals("1 Hour"));
```

Slice(hours, 28)



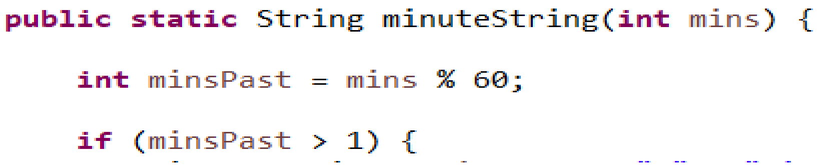
```
public static String minuteString(int mins) {

    int hours = mins / 60;
    int minsPast = mins % 60;

    if (minsPast > 1) {
    } else if (minsPast > 0) {
    } else if (hours >= 1) {
```

One of the tests that covers this is



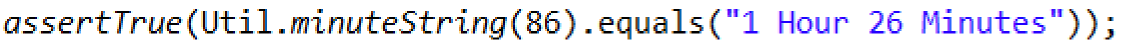
```
assertTrue(Util.minuteString(60).equals("1 Hour"));
```

Slice(minsPast, 24)



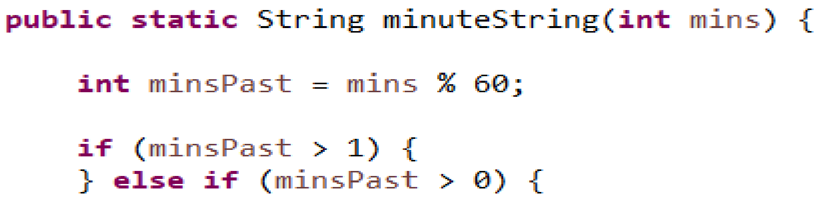
```
public static String minuteString(int mins) {

    int minsPast = mins % 60;

    if (minsPast > 1) {
```

One of the tests that covers this is



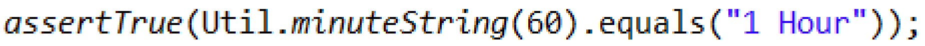
```
assertTrue(Util.minuteString(86).equals("1 Hour 26 Minutes"));
```

Slice(minsPast, 26)



```
public static String minuteString(int mins) {

    int minsPast = mins % 60;

    if (minsPast > 1) {
    } else if (minsPast > 0) {
```

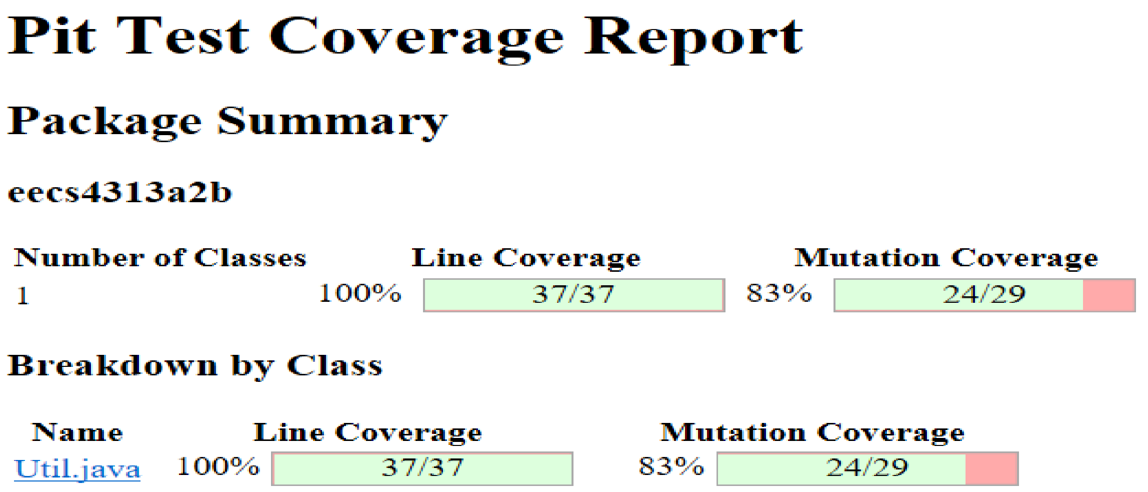
One of the tests that covers this is



```
assertTrue(Util.minuteString(60).equals("1 Hour"));
```

2.1.3 Mutation Testing

To easily analyze the methods chosen, we moved them into a file called Util.java where we will be performing the PIT mutation testing on. Thus, the results will all be displayed to us on one file. Here are the results for our current tests so far in assignment 2 & 3.



Pit Test Coverage Report

Package Summary

eecs4313a2b

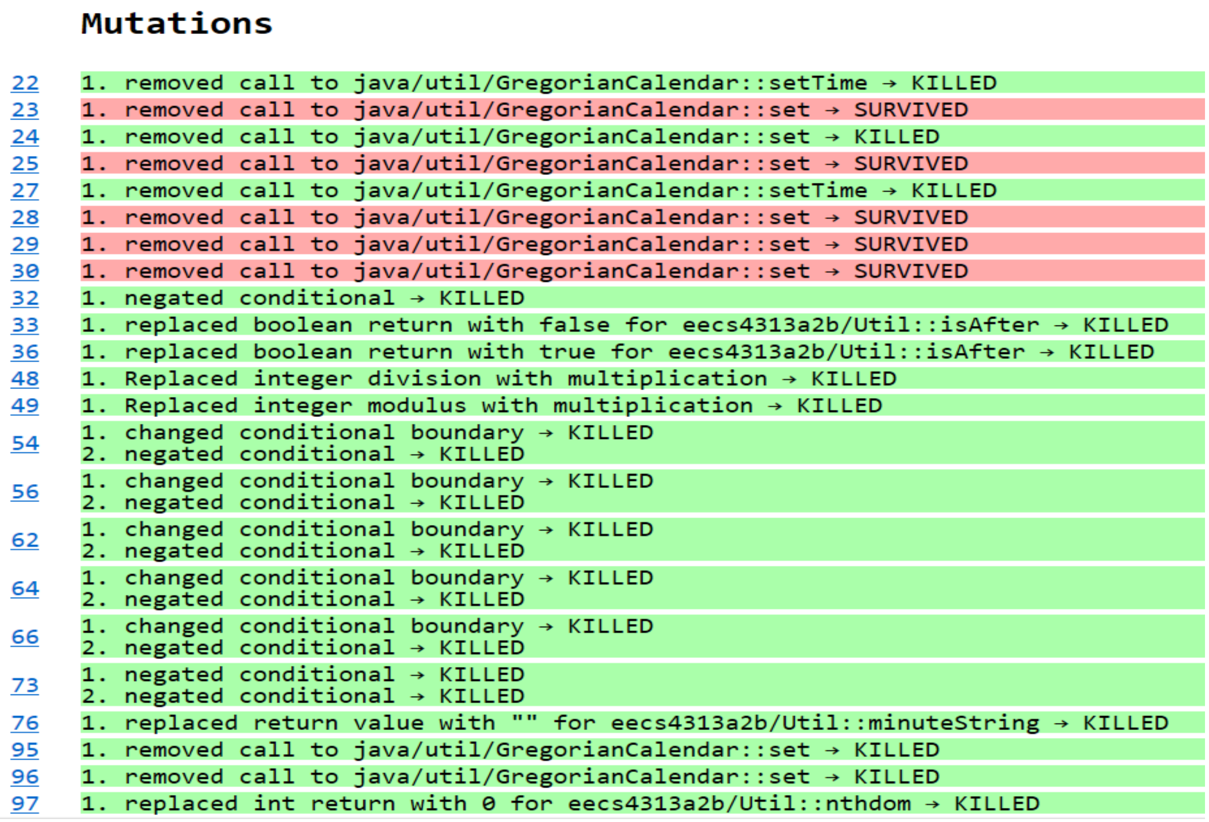
Number of Classes	Line Coverage		Mutation Coverage	
1	100%	37/37	83%	24/29

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Util.java	100%	37/37	83%	24/29

Pit Test Coverage Report

As we can see, the mutation coverage isn't 100% and there are mutants that havent been killed through the testing process. So, we will look at each of them and see what tests we can add to be able to possibly achieve this. After that the results will be visited again to see how much the coverage improved by.

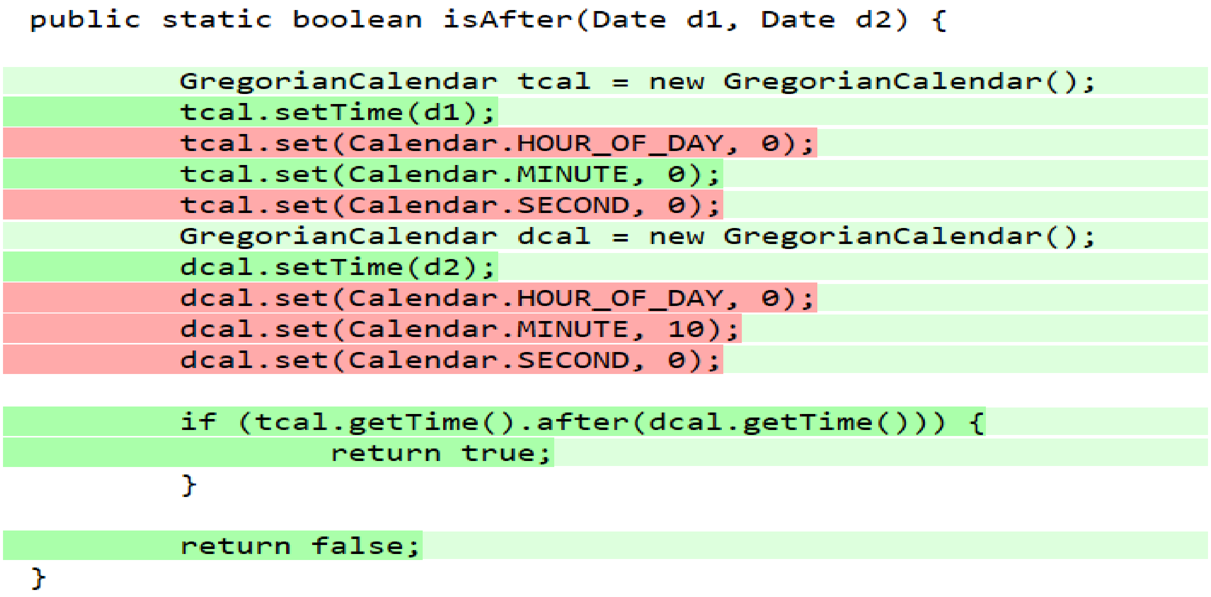


Mutations

```
22   1. removed call to java/util/GregorianCalendar::setTime → KILLED
23   1. removed call to java/util/GregorianCalendar::set → SURVIVED
24   1. removed call to java/util/GregorianCalendar::set → KILLED
25   1. removed call to java/util/GregorianCalendar::set → SURVIVED
27   1. removed call to java/util/GregorianCalendar::setTime → KILLED
28   1. removed call to java/util/GregorianCalendar::set → SURVIVED
29   1. removed call to java/util/GregorianCalendar::set → SURVIVED
30   1. removed call to java/util/GregorianCalendar::set → SURVIVED
32   1. negated conditional → KILLED
33   1. replaced boolean return with false for eecs4313a2b/Util::isAfter → KILLED
36   1. replaced boolean return with true for eecs4313a2b/Util::isAfter → KILLED
48   1. Replaced integer division with multiplication → KILLED
49   1. Replaced integer modulus with multiplication → KILLED
54   1. changed conditional boundary → KILLED
     2. negated conditional → KILLED
56   1. changed conditional boundary → KILLED
     2. negated conditional → KILLED
62   1. changed conditional boundary → KILLED
     2. negated conditional → KILLED
64   1. changed conditional boundary → KILLED
     2. negated conditional → KILLED
66   1. changed conditional boundary → KILLED
     2. negated conditional → KILLED
73   1. negated conditional → KILLED
     2. negated conditional → KILLED
76   1. replaced return value with "" for eecs4313a2b/Util::minuteString → KILLED
95   1. removed call to java/util/GregorianCalendar::set → KILLED
96   1. removed call to java/util/GregorianCalendar::set → KILLED
97   1. replaced int return with 0 for eecs4313a2b/Util::nthdom → KILLED
```

Mutations

isAfter



```
public static boolean isAfter(Date d1, Date d2) {

        GregorianCalendar tcal = new GregorianCalendar();
        tcal.setTime(d1);
        tcal.set(Calendar.HOUR_OF_DAY, 0);
        tcal.set(Calendar.MINUTE, 0);
        tcal.set(Calendar.SECOND, 0);
        GregorianCalendar dcal = new GregorianCalendar();
        dcal.setTime(d2);
        dcal.set(Calendar.HOUR_OF_DAY, 0);
        dcal.set(Calendar.MINUTE, 10);
        dcal.set(Calendar.SECOND, 0);

        if (tcal.getTime().after(dcal.getTime())) {
                return true;
        }

        return false;
}
```

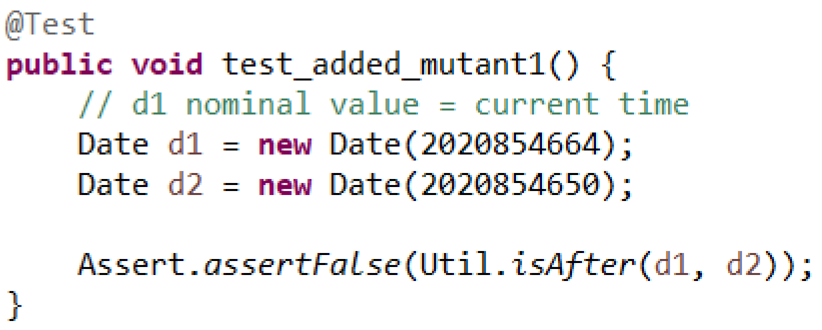
isAfter Coverage

Added Tests

removed call to java/util/GregorianCalendar::set → SURVIVED

Here we can see that the mutations on the lines that use the set function all survive even when they are removed from the function. We can observe from this that these lines set the Hour and Second Values to 0 for the GregorianCalender objects created to convert the date object from. The minute value is an exception where the one that survived sets its date value to 10. This setting is because the function only cares about the date itself and the time would be irrelevant to the function.

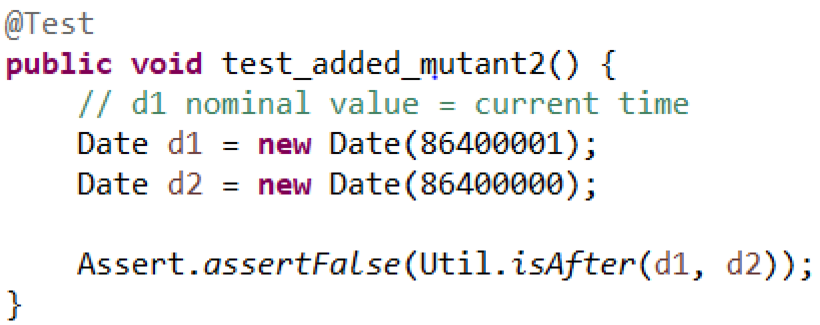
To improve the mutation test results 2 different tests were added to cover some of the surviving mutations.



```
@Test
public void test_added_mutant1() {
    // d1 nominal value = current time
    Date d1 = new Date(2020854664);
    Date d2 = new Date(2020854650);

    Assert.assertFalse(Util.isAfter(d1, d2));
}
```

First Added Test

This test was added to cover the mutation for **tcal.set(Calendar.HOUR_OF_DAY, 0)** where two dates with an hour value greater than 0 and the first one being after the second one. The return value would now change if we didn't set the hour of the first of the two dates to 0.



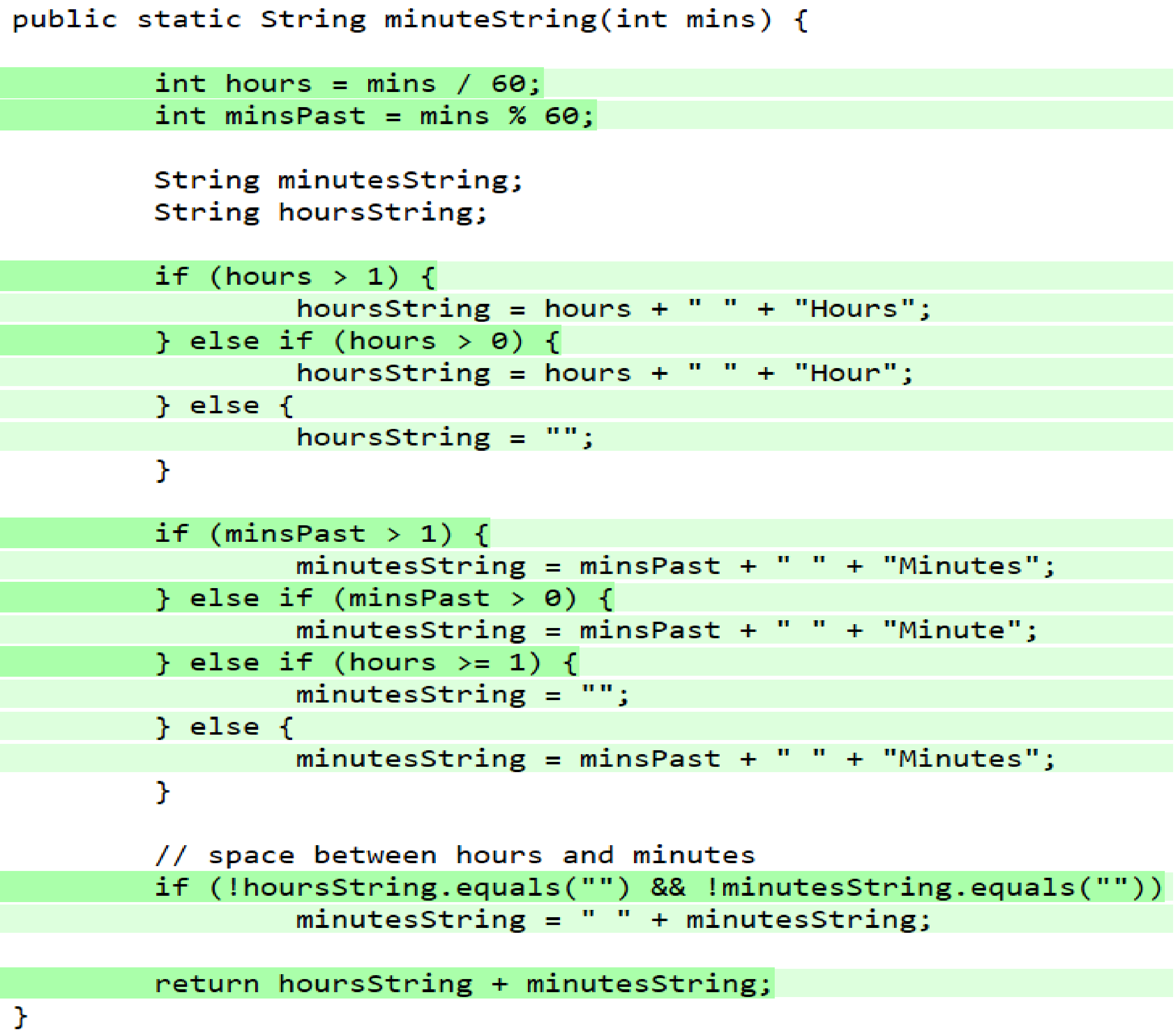
```java
@Test
public void test_added_mutant2() {
    // d1 nominal value = current time
    Date d1 = new Date(86400001);
    Date d2 = new Date(86400000);

    Assert.assertFalse(Util.isAfter(d1, d2));
}
```

Second Added Test

This test was added to cover the mutation for **dcal.set(Calendar.MINUTE, 10)** where two dates correlated to exactly a day with the first one being one millisecond over it. This change meant that when the dcal minute set to 10 was removed then the first date would now be after the second date.

minuteString

This function had no survived mutations

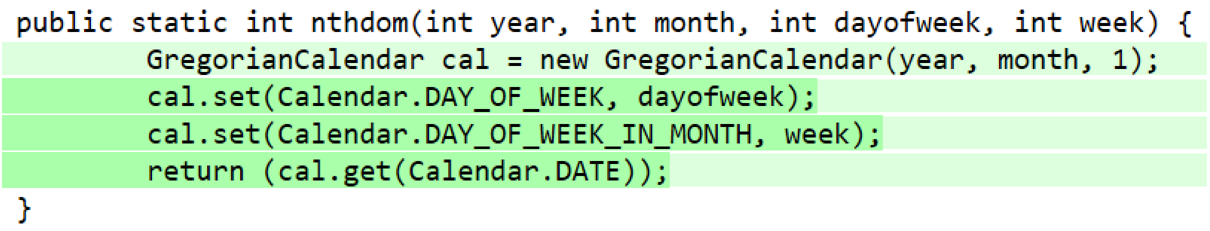


```java
public static String minuteString(int mins) {

        int hours = mins / 60;
        int minsPast = mins % 60;

        String minutesString;
        String hoursString;

        if (hours > 1) {
                hoursString = hours + " " + "Hours";
        } else if (hours > 0) {
                hoursString = hours + " " + "Hour";
        } else {
                hoursString = "";
        }

        if (minsPast > 1) {
                minutesString = minsPast + " " + "Minutes";
        } else if (minsPast > 0) {
                minutesString = minsPast + " " + "Minute";
        } else if (hours >= 1) {
                minutesString = "";
        } else {
                minutesString = minsPast + " " + "Minutes";
        }

        // space between hours and minutes
        if (!hoursString.equals("") && !minutesString.equals(""))
                minutesString = " " + minutesString;

        return hoursString + minutesString;
}
```

minuteString Coverage

nthdom

This function had no survived mutations

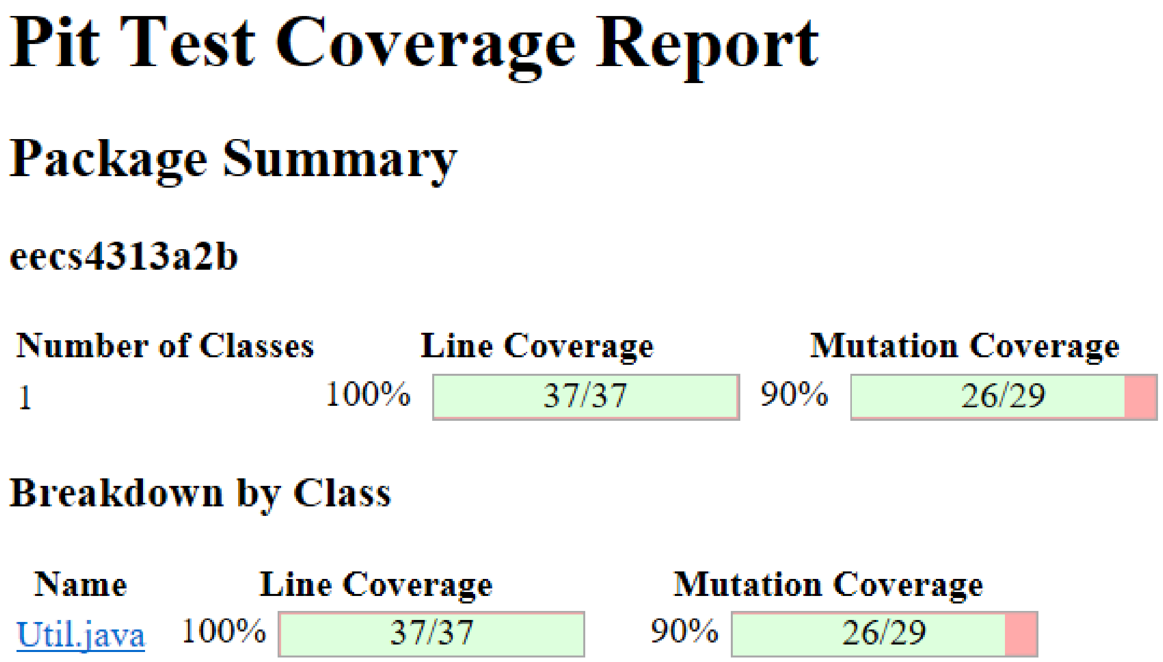


```
public static int nthdom(int year, int month, int dayofweek, int week) {
        GregorianCalendar cal = new GregorianCalendar(year, month, 1);
        cal.set(Calendar.DAY_OF_WEEK, dayofweek);
        cal.set(Calendar.DAY_OF_WEEK_IN_MONTH, week);
        return (cal.get(Calendar.DATE));
}
```

nthdom Coverage

Final Results

Upon the test cases added to cover more of the mutation cases we managed to have an increase in the testing coverage from 83% to 90%. The rest of the mutation cases could not have been adjusted due to the fact that the setting to 10 minutes for the dcal variable makes it so that the second value will never affect the output.



Pit Test Coverage Report

Package Summary

eecs4313a2b

Number of Classes	Line Coverage		Mutation Coverage	
1	100%	37/37	90%	26/29

Breakdown by Class

Name	Line Coverage		Mutation Coverage	
Util.java	100%	37/37	90%	26/29

Pit Test Coverage Report With Added Tests

Task 2 – JPetStore

3.1 Getting started

Test scenarios

Two load tests were performed on the JPetStore website. The first test scenario describes the flow of a purchase and checkout and the second scenario describes the flow of a current user modifying their user account. The details of the test scenario steps, response rates and duration are shown below as well as their respective response time graphs.

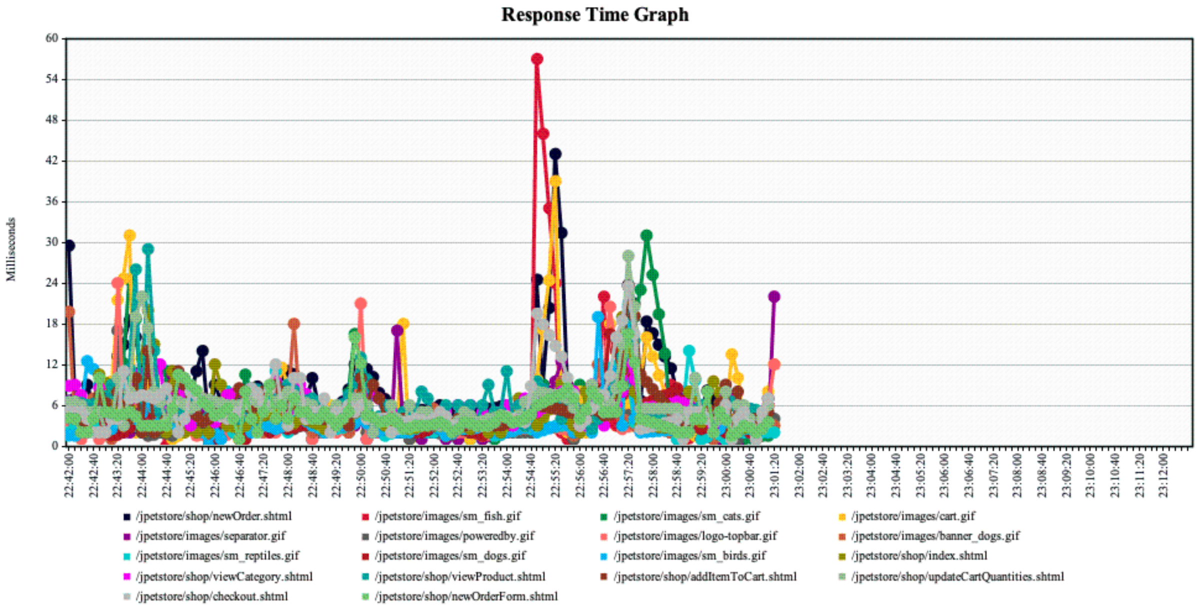
Scenario 1

Scenario Description

1. View Fish Category
2. View Product ID FI-FW-01
3. Add Item EST-4 to cart
4. Update EST-4 quantity to 2
5. Checkout
6. New order with personal information
7. Confirm order

Request rate and the duration of the load test

- 1 Request / 1000ms
- 19 minutes of testing



Response Time Graph for Scenario 1

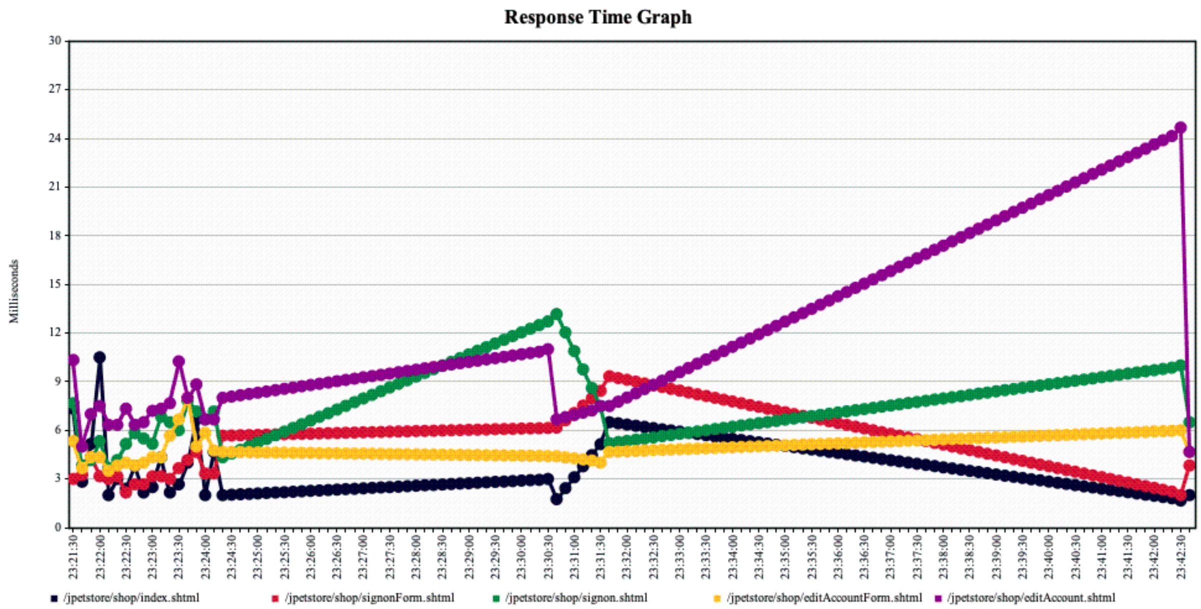
Scenario 2

Scenario Description

1. Go to index page
2. Sign in with username and password
3. Open account editor
4. Modify account with new first name and last name

Request rate and the duration of the load test

- 1 request / 1000ms
- 21 minutes of load testing



Response time graph for Scenario 2

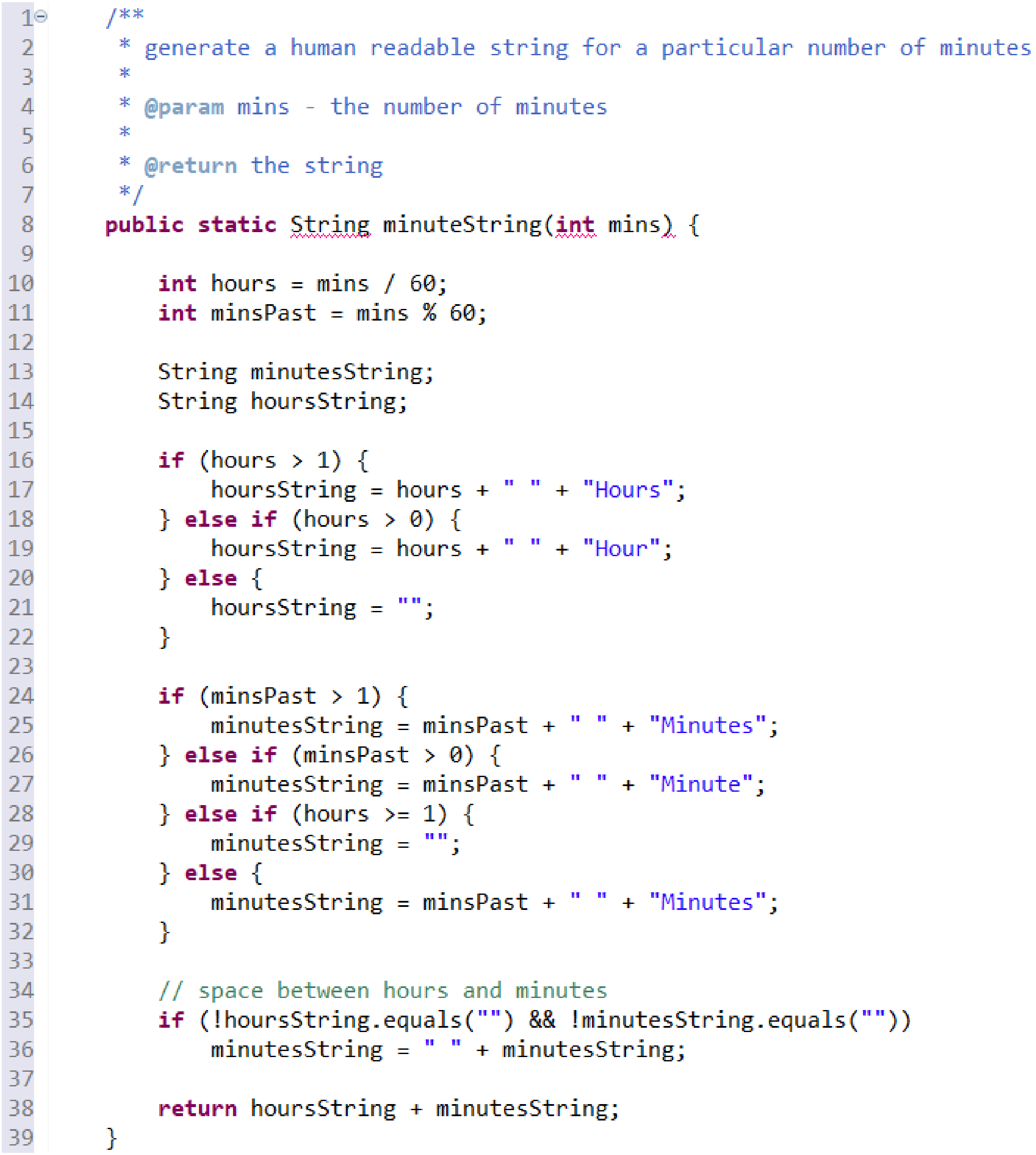
Analysis of Load Tests

During the load tests the system did not crash, restart or have long latency times and ended smoothly. All of the requests given to the system during the load tests were completed as well and produced no errors. This was probably due to the fact that we do not overly stress the system with extremely high load. The load tests were designed using lower request rates in order to expose us to how the Jmeter software obtains and displays load tests data and to ensure that everything in the system was working as intended. These conclusions can also be seen in the response time graphs above. In Scenario 1 it is shown that most of the gif files have recorded higher response times when compared to the shtml files. In Scenario 2 account.shtml had a very high response time towards the end of the load test when compared to the rest of the samples.

Since the load tests only accounted for the response time metrics, other metrics such as cpu utilization and system memory were not accounted for. With those metrics we would be able to determine known problems such as memory leaks and deadlocks. Also, we did not have any previous load tests to use for further analysis.

Appendix

Method Under test



```java
/**
 * generate a human readable string for a particular number of minutes
 *
 * @param mins - the number of minutes
 *
 * @return the string
 */
public static String minuteString(int mins) {

    int hours = mins / 60;
    int minsPast = mins % 60;

    String minutesString;
    String hoursString;

    if (hours > 1) {
        hoursString = hours + " " + "Hours";
    } else if (hours > 0) {
        hoursString = hours + " " + "Hour";
    } else {
        hoursString = "";
    }

    if (minsPast > 1) {
        minutesString = minsPast + " " + "Minutes";
    } else if (minsPast > 0) {
        minutesString = minsPast + " " + "Minute";
    } else if (hours >= 1) {
        minutesString = "";
    } else {
        minutesString = minsPast + " " + "Minutes";
    }

    // space between hours and minutes
    if (!hoursString.equals("") && !minutesString.equals(""))
        minutesString = " " + minutesString;

    return hoursString + minutesString;
}
```