EECS 4313 Assignment 3 Data Flow Testing, Slice-Based Testing and Mutation Testing

Student Name — Student Number — EECS Account

Edward Vaisman — 212849857 — eddyv

Robin Bandzar — 212200531 — cse23028

Kirusanth Thiruchelvam — 212918298 — kirusant

Sadman Sakib Hasan — 212497509 — cse23152

April 3, 2018

Contents

1 BORG Calendar

1.1 Slice Testing

1.1.1 Chosen Method for Testing

- **Class**: *net.sf.borg.common.DateUtil.java*
- **Method**: *minuteString(int mins)*
- **Method Description**: This method generate a human reable string for a particular number of minutes. It returns the string in terms of hours or minutes or both hours and mintues.
  - **mins** - The first argument is of type integer.

Following is the code of the *minuteString* method:

```java
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 = "";
122     } else {
123      minutesString = minsPast + " " +
           Resource.getResourceString("Minutes");
124     }
125
126     // space between hours and minutes
127     if (!hoursString.equals("") && !minutesString.equals(""))
128      minutesString = " " + minutesString;
129
130     return hoursString + minutesString;
131   }
```

1.1.2 Forward Slicing

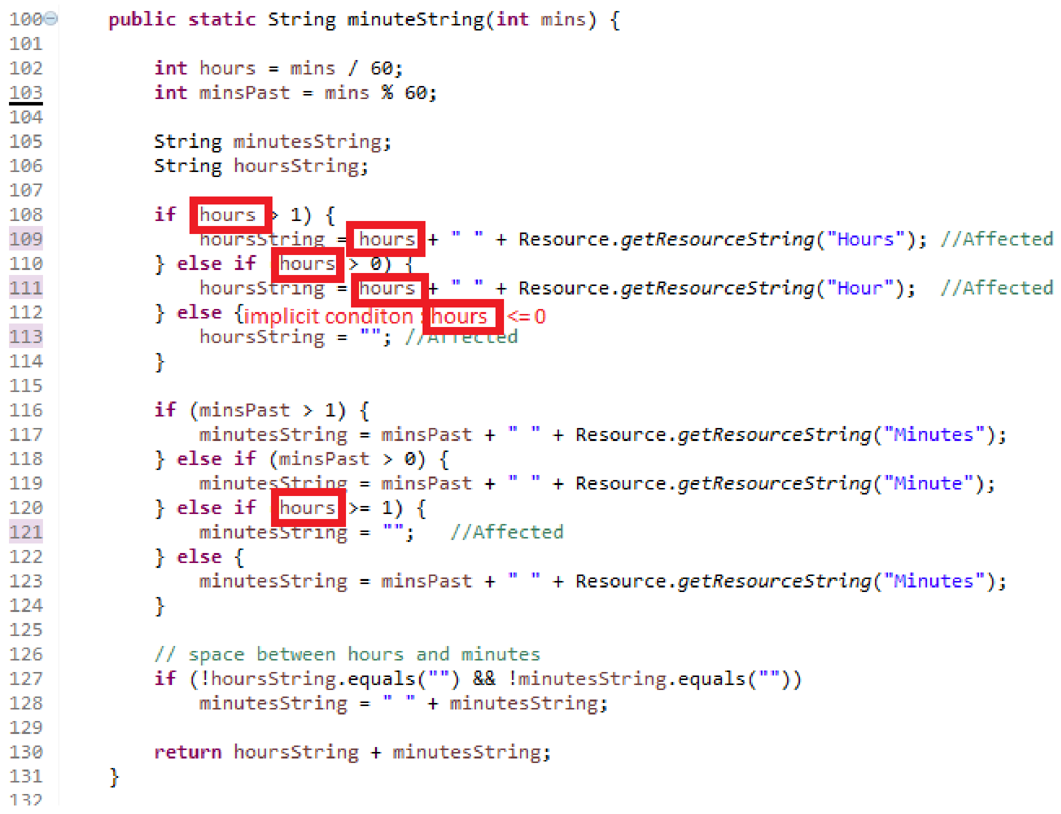
The forward slicing is one of the static program slicing. The forward slicing of a program can be defined in the form of S(v,n) which refers to all the program statments that are affected by the value of v at statement n. In other words, what staments will be affected in the program by modifying the selected varible's value at the selected statment. The Borg Calendar has two cases where we can apply the forward slice-based testing.

Case 1: S (hours, 102) - modifying value of the variable *hours* at statment 102

In this case, we cannot slice any statment of the program since all of them are depend on variable *hours*. We have two nested if-else statments and both of them uses variable *hours* in their esle if condition. This makes other conditons on the nested if-else struture to be depend upon the variable of *hour* to execute the program but the value of it would not affect them. Since there is a need for return statment to exeute the code, we cannot slice that statment as well. In order to produce the correct spaces in the outputs we need the if condion made for hourstring and minutes string at statment 127.

However, we only need four testcases here to identify how the chage in value of hour affectes the program since other statments in the code do not depend on the vlaue at all and they merely depend on the varible to execute the program successfully due the structure of nested if-else.

The following figure show which statments in the method affected by changing value of hours.



```
100      public static String minuteString(int mins) {
101
102          int hours = mins / 60;
103          int minsPast = mins % 60;
104
105          String minutesString;
106          String hoursString;
107
108          if (hours > 1) {
109              hoursString = hours + " " + Resource.getResourceString("Hours"); //Affected
110          } else if (hours > 0) {
111              hoursString = hours + " " + Resource.getResourceString("Hour");  //Affected
112          } else {implicit conditon : hours <=0
113              hoursString = ""; //Affected
114          }
115
116          if (minsPast > 1) {
117              minutesString = minsPast + " " + Resource.getResourceString("Minutes");
118          } else if (minsPast > 0) {
119              minutesString = minsPast + " " + Resource.getResourceString("Minute");
120          } else if (hours >= 1) {
121              minutesString = "";   //Affected
122          } else {
123              minutesString = minsPast + " " + Resource.getResourceString("Minutes");
124          }
125
126          // space between hours and minutes
127          if (!hoursString.equals("") && !minutesString.equals(""))
128              minutesString = " " + minutesString;
129
130          return hoursString + minutesString;
131      }
132
```

Figure 1: Affected lines due to change in the value of variable *hours*

In order to check the affected four lines we have implemented threer test cases :

1. assertEquals(”3 Hours”,DateUtil.minuteString(180)) - checks staments 108- 109 and statment 120 -121
2. assertEquals(”1 Hour”,DateUtil.minuteString(60)) - checks statments 110 - 111 and statment 120 -121
3. assertEquals(”0 Minutes”,DateUtil.minuteString(0)) - checks statments 110 - 111

Since the cases provide full coverage of the affected lines, we only need to consider changing those lines to produce the correct result when we modify the vairable *hours*

Case 2: S (minsPast, 103) - modifying value of the variable *minsPast* at statment 103

- The data flow analysis you performed and the calculation of the coverage metrics. You must show which test cases are responsible for which dc-paths.
- A description of the test cases you added to improve coverage. If your coverage was already high, discuss how your testing was able to achieve this.
- The slices that you identified and the percentage of slices that your testing covers. You must show which test cases are responsible for which slices.
- A description of the test cases you added to improve slice coverage. If your coverage was already high, discuss how your testing was able to achieve this.
- Evaluate the effectiveness of your test cases using mutation testing. Discuss and address any issues if you have found in your written report.
- Attaching bug reports if bugs are discovered using your testing methods. You should use the same bug report format as in Assignment 1. Do not file these bug reports to the projects bug report system.
- An appendix with the specification of the methods you are testing

2 JPetStore

- The test scenarios that you have created;
- The request rates and the duration of the load tests;
- The analysis of your load tests and the description of any problems that you have found (if there are any).