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 5, 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 generates a human readable string for a particular number of minutes. It returns the string in terms of hours or minutes or both hours and minutes.
  - **mins** - The first argument is of type integer.

Following is the code of the *minuteString* method:

```
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");
110    } else if (hours > 0) {
111      hoursString = hours + " " + Resource.getResourceString("Hour");
112    } else {
113      hoursString = "";
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 = "";
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 Backward Slicing

Backward slicing is in the form of S(v,n) where the slices are code fragments that contribute to variable v at statement n. Slices are only done for primitive values and their All-defs and P-use paths defined in the data flow analysis part.

S(hours, 102)

```
100   public  static  String  minuteString(int mins) {
101
102     int  hours = mins / 60;
```

S(minsPast, 103)

```
100   public  static  String  minuteString(int mins) {
101
102
103     int  minsPast = mins % 60;
```

The following test case covers the two slices listed above and covers the All-def, P-use for *mins*.

```
assertEquals("1 Hour",DateUtil.minuteString(60));
```

S(hours, 108)

```
100 public static String minuteString(int mins) {
101
102   int hours = mins / 60;
103
104
105
106
107
108   if (hours > 1) {
109   }
```

S(hours, 120)

```
100 public static String minuteString(int mins) {
101
102   int hours = mins / 60;
103   int minsPast = mins % 60;
104
105
106
107
108
109
110
111
112
113
114
115
116   if (minsPast > 1) {
117
118   } else if (minsPast > 0) {
119
120   } else if (hours >= 1) {
121   }
```

The following test case covers the previous two slices for *hours*.

```
assertEquals("3 Hours",DateUtil.minuteString(180));
```

S(minsPast, 116)

```
100  public static String minuteString(int mins) {
101
102
103      int minsPast = mins % 60;
104
105
106
107
108
109
110
111
112
113
114
115
116      if (minsPast > 1) {
117
118      }
```

The following test case covers the previous slice for *minsPast*.

```
assertEquals("2 Hours 1 Minute",DateUtil.minuteString(121));
```

S(hours, 110)

```
100  public static String minuteString(int mins) {
101
102      int hours = mins / 60;
103
104
105
106
107
108      if (hours > 1) {
109
110      } else if (hours > 0) {
111
112      }
```

S(minsPast, 118)

```
100 public static String minuteString(int mins) {
101
102
103   int minsPast = mins % 60;
104
105
106
107
108
109
110
111
112
113
114
115
116   if (minsPast > 1) {
117
118   } else if (minsPast > 0) {
119
120
121   }
```

The following test case covers the previous slice for *minsPast*.

```
assertEquals("1 Hour 1 Minute",DateUtil.minuteString(61));
```

S(hours, 112)

```
100 public static String minuteString(int mins) {
101
102   int hours = mins / 60;
103
104
105
106
107
108   if (hours > 1) {
109
110   } else if (hours > 0) {
```

```
111
112     } else {
113
114     }
```

The following test case covers the previous slice for *hours*.

```
assertEquals("0 Minutes",DateUtil.minuteString(0));
```

S(hours, 122)

```
100   public static String minuteString(int mins) {
101
102     int hours = mins / 60;
103     int minsPast = mins % 60;
104
105
106
107
108
109
110
111
112
113
114
115
116     if (minsPast > 1) {
117
118     } else if (minsPast > 0) {
119
120     } else if (hours >= 1) {
121
122     } else {
123
124     }
```

S(minsPast, 122)

```
100   public static String minuteString(int mins) {
101
102     int hours = mins / 60;
```

```
103     int minsPast = mins % 60;
104
105
106
107
108
109
110
111
112
113
114
115
116     if (minsPast > 1) {
117
118     } else if (minsPast > 0) {
119
120     } else if (hours >= 1) {
121
122     } else {
123
124     }
```

The following test case covers the previous two slices for *hours*.

```
assertEquals("50 Minutes",DateUtil.minuteString(50));
```

This concludes all the backward slices related to the All-defs and P-uses of the primitive types in the *minuteString* function.

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).