# 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**

March 13, 2018

# Contents

# 1 BORG Calendar

## 1.1 Data Flow Testing

### 1.1.1 Chosen Method

- **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 snippet of the *minuteString* method:

```
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;
  }
```

### 1.1.2 Variable set

Following are the list of variables used in the *minuteString* method along with their types:

| Variable Name | Data Type | Is Primitive? |
|---|---|---|
| mins | int | Yes |
| hours | int | Yes |
| minsPast | int | Yes |
| minutesString | String | No |
| hoursString | String | No |

Table 1: List of variables for the minuteString method

**Note**: The last two variables, *minutesString* and *hoursString*, are **NOT** primitive types, and will **NOT** be used to perform the data flow analysis as specified by the instructor.

### 1.1.3 Program Segmentation

The following table 2 contains the *minuteString* method broken down into segments:

| | |
|---|---|
| public static string minuteString(int mins) | A |
| int hours = mins / 60;<br>int minsPast = mins % 60; | B |
| String minutesString;<br>String hoursString; | C |
| if (hours > 1) { | D |
| hoursString = hours + " " + Resource.getResourceString("Hours"); | E |
| } else if (hours > 0) { | F |
| hoursString = hours + " " + Resource.getResourceString("Hour"); | G |
| } else { hoursString = ""; } | H |
| if (minsPast > 1) { | I |
| minutesString = minsPast + " " + Resource.getResourceString("Minutes"); | J |
| } else if (minsPast > 0) { | K |
| minutesString = minsPast + " " + Resource.getResourceString("Minute"); | L |
| } else if (hours >= 1) { | M |
| minutesString = ""; | N |
| } else {<br>minutesString = minsPast + " " + Resource.getResourceString("Minutes");<br>} | O |
| if (!hoursString.equals("") && !minutesString.equals("")) | P |
| minutesString = " " + minutesString; | Q |
| return hoursString + minutesString; | R |

Table 2: Program Segmentation for the minuteString method

### 1.1.4 Program graph

The following diagram 1 represents the control flow graph for the *minuteString* method:

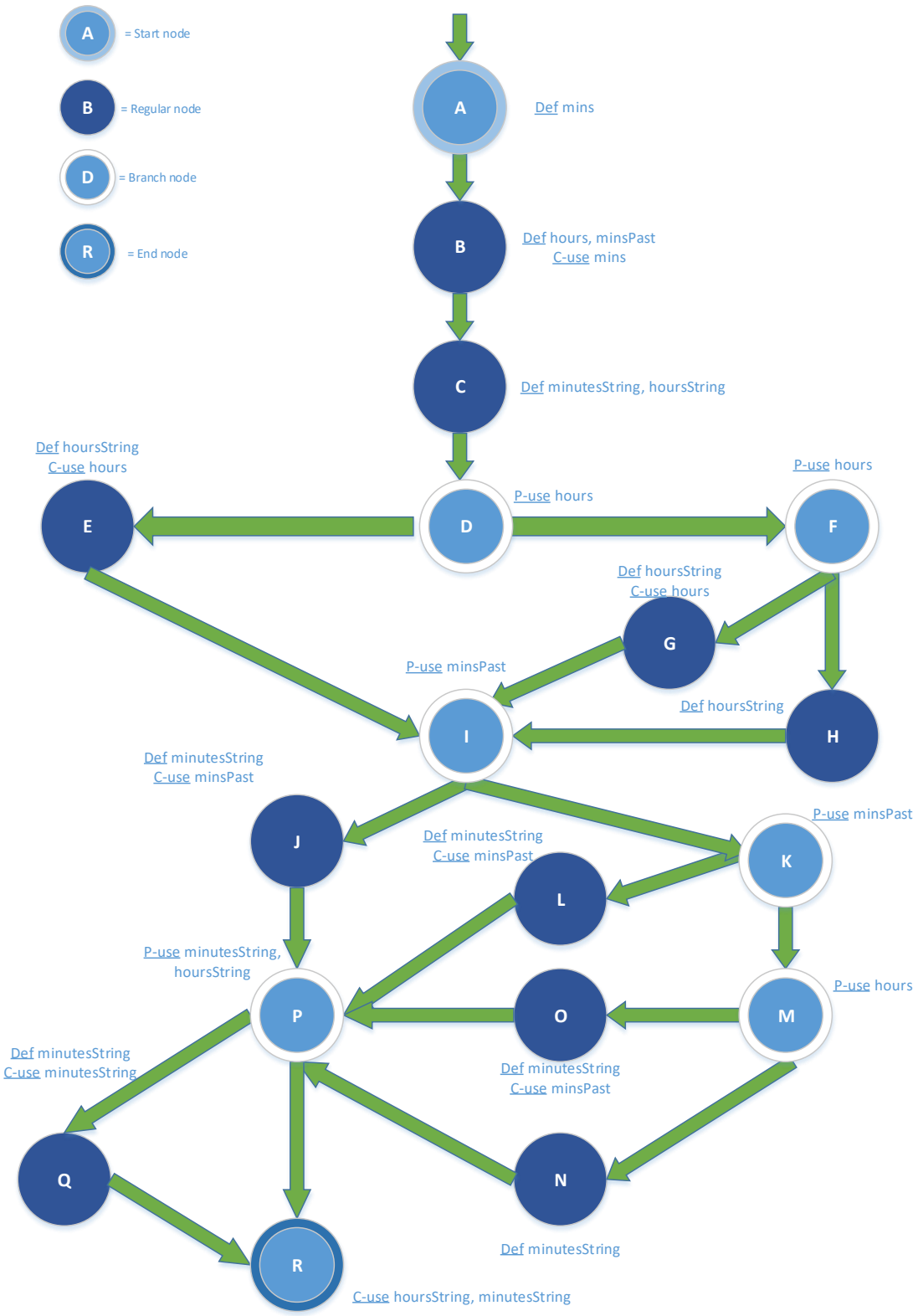


Figure 1: Control Flow Graph for the minuteString method

### 1.1.5 Data Flow Analysis

1. **du-paths for *mins***
   - **All-Defs**: *AB*
   - **All-C-Uses/Some-P-Uses**: *AB*
   - **All-P-Uses/Some-C-Uses**: *AB*
   - **All-Uses**: *AB*
2. **du-paths for *hours***
   - **All-Defs**: *BCD*
   - **All-C-Uses/Some-P-Uses**: *BCDE, BCDFG*
   - **All-P-Uses/Some-C-Uses**: *BCD, BCDF, BCDEIKM, BCDFGIKM, BCDFHIKM*
   - **All-Uses**: *BCD, BCDE, BCDF, BCDEIKM, BCDFG, BCDFGIKM, BCDFHIKM*
3. **du-paths for *minsPast***
   - **All-Defs**: *BCDEI*
   - **All-C-Uses/Some-P-Uses**: *BCDEIJ, BCDFGIJ, BCDFHIJ, BCDEIKL, BCDFGIKL, BCDFHIKL, BCDEIKMO, BCDFGIKMO, BCDFHIKMO*
   - **All-P-Uses/Some-C-Uses**: *BCDEI, BCDFGI, BCDFHI, BCDEIK, BCDFGIK, BCDFHIK*
   - **All-Uses**: *BCDEI, BCDEIJ, BCDFGIJ, BCDFHIJ, BCDEIKL, BCDFGIKL, BCDFHIKL, BCDEIKMO, BCDFGIKMO, BCDFHIKMO, BCDFGI, BCDFHI, BCDEIK, BCDFGIK, BCDFHIK*

### 1.1.6 Coverage Metrics

Following are the existing test cases from Assignment-2 and their coverage metrics using the data flow analysis for the *minuteString* method:

1. **Test Case 1**:
   - **Input**: mins = 60
   - **Path**: *ABCDEGIKMNPR*
   - **Coverage**: 100*(12/18) = 66.67%
2. **Test Case 2**:
   - **Input**: mins = 61

- **Path**: *ABCDFGIKLPQR*
- **Coverage**: 100*(14/18) = 77.78%

3. **Test Case 3**:
   - **Input**: mins = 75
   - **Path**: *ABCDFGIJPQR*
   - **Coverage**: 100*(15/18) = 83.33%
4. **Test Case 4**:
   - **Input**: mins = 180
   - **Path**: *ABCDEIKMNPR*
   - **Coverage**: 100*(16/18) = 88.88%
5. **Test Case 5**:
   - **Input**: mins = 121
   - **Path**: *ABCDEIKLPQR*
   - **Coverage**: 100*(16/18) = 88.88%
6. **Test Case 6**:
   - **Input**: mins = 145
   - **Path**: *ABCDEIJPQR*
   - **Coverage**: 100*(16/18) = 88.88%
7. **Test Case 7**:
   - **Input**: mins = 0
   - **Path**: *ABCDFHIOPR*
   - **Coverage**: 100*(18/18) = 100.0%

**Rationale**: The testing technique used for testing this method is *Equivalence Class Testing*. It is a suitable technique for this method since the argument is an integer which is an independent variable and the input range can be partitioned assuring disjointness and non-redundancy between each partition set. We have chosen the partition integer range based on usages of minute, minutes, hour, and hours. In order to partition the integer argument into hours and minutes, we divide the minutes by 60 to get the range of hours and the remainder (minutes % 60) to get the range of the minutes. By covering all the cases, where the given input returns a concatenation of hours and minutes string, we were able to reach a 100% coverage for this method.

- 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).