EECS 4313 Assignment 2
Black-box and White-box Testing with JUnit

Student Name — Student Number — EECS Account
Edward Vaisman — 212849857 — eddyv
Robin Bandzar — 212200531 — cse23028
Kirusanth Thiruchelvam — 212918298 — kirusant
Sadman Sakib Hasan — 212497509 — cse23152

March 5, 2018

Contents

1 Black Box Testing

- Specification of the selected Java methods.
- Justification of the testing technique chosen, i.e., why is it appropriate for this method.
- Description of your application of the three testing strategies. Be clear about which test cases you implemented.
- Evaluation of the test cases derived by the testing technique. Include the screenshots of the test running results. If you had to complement the derived test cases with special value testing, describe that as well. The marker will not read your code in order to see what you tested. You have to describe it.
- 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.

2 White Box Testing

2.1 Equivalence Class Testing

- **Technique**: *Equivalence Class Testing*
- **Class**: *net.sf.borg.common.DateUtil.java*
- **Method**: *minuteString(int mins))*
- **Method Description**: This method generate a human reable string for a particular numbe of minutes.It returns the string interms of hours or minutes or both hours and mintues.
  - **mins** - The argument is an integer
- **Justification**: Equivalence class testing is suitable for this method since the argument of this method is an integer which is an independent variable and input range can be partitioned while assuring disjointness and non-redundancy between each partition set. We have chosen these partition integer range based on when we use 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.

Table 1: Input ECT

M1: {mins | mins % 60 = 0}
M2: {mins | mins % 60 = 1}
M3: {mins | mins % 60 > 1}
H1: {mins | mins / 60 ≥ 1 AND mins / 60 < 2}
H2: {mins | mins / 60 ≥ 2}
H3: {mins | mins / 60 ≥ 0 AND mins / 60}

Table 2: Output ECT

O1: H1, M1 = 1 hour
O2: H1, M2 = 1 hour 1 minute
O3: H1, M3 = 1 hour y minutes
O4: H2, M1 = x hours
O5: H2, M2 = x hours 1 minute
O6: H2, M3 = x hours y minutes
O7: H3, M1 = 0 minutes
O8: H3, M2 = 1 minute
O9: H3, M3 = y minutes

– x and y are integer variables

Table 3: ECT-Matrix

M1	O1	O4	O7
M2	O2	O5	O8
M3	O3	O6	O9
	H1	H2	H3

The method did not specifity how negative minutes should be treated, so we omit the negative integers as an argument for this method. For example, -75 can be converted as -1 hour and 15 minutes or 45 minutes or any other way. Therefore, this case is tested in the whitebox testing thorugh additional test cases after analyzing structure of the method.

- **Statement Coverage Using Black-Box Methods :** 100% [refer to Figure 1 & 2]
- **Justification for getting 100% in blackbox tests :** Since the method converts the interger in terms of human readable minutes and hours, the range of valid

outputs can be partition as described in the method description. Since we know how the conversion of integer values into hours and minutes are implemented, there is a need to check negative integers.

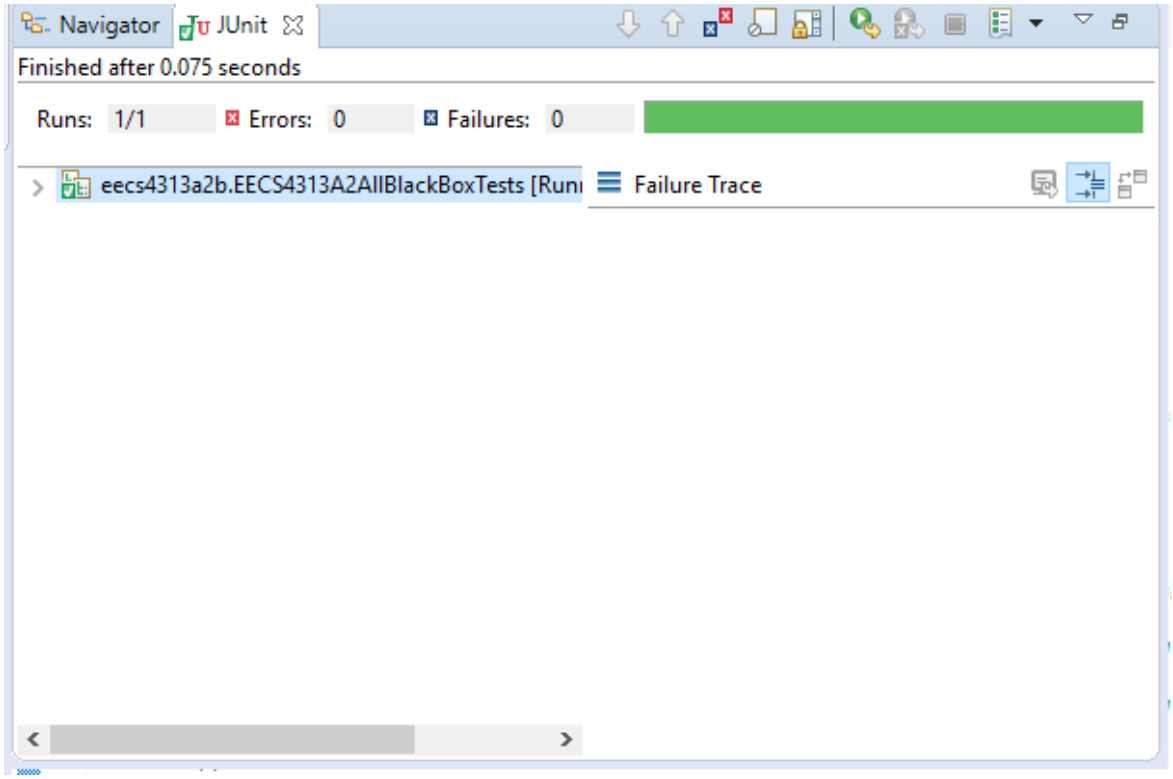

Figure 1: Test run result for the minuteString functionn

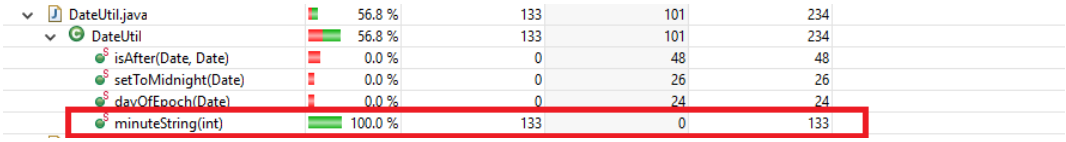

Figure 2: Statement Coverage View for the minuteString function

- **Additional cases:** Since hours are implemented by dividing integer by 60, the negative integers can produce negative hours. Also, negative hours produced emptystring according to the implementation.However, minutes are computed by taking the remainder of the division (mins%60) this will produce a positive

number since reminder cannot be negative. Therefore, only two cases we can check in this whitebox testing procedure. They are :

– Mins/60 < 0 and Mins%60 > 1 - To test negative hours with more than 1 minute [Class 10]
– Mins/60 < 0 and Mins%60 = 1 - To test negative with 1 minute [Class 11]
This covers the two invalid cases possible for the test based on the implementation of the method. since we can only convert negative integers in terms of minutes according to the implementation of the method. However these two cases produced a bug because the method is producing negative results instead of positive results. The test cases fail for the two cases are shown in figure 4.

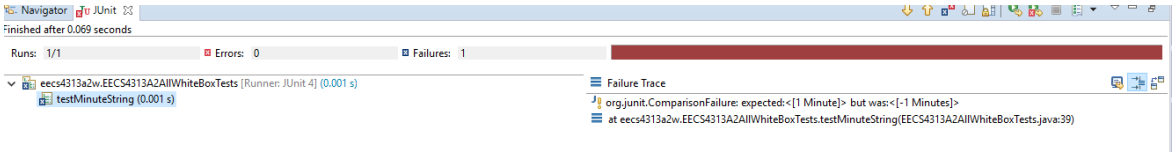

Figure 3: Additional test cases fail due to a bug in the minuteString function

2.1.1 Bug Report

– **Bug Report Title**: DateUtil.minuteString Method does not output the correct result for negative integers
– **Reported by**: Kirusanth Thiruchelvam
– **Date reported**: March 3, 2018
– **Program (or component) name**: net.sf.borg.common.DateUtil.minuteString()
– **Configuration(s)**:
  * Operating System: Windows 10 Pro
  * Version: 10.0.1.16299 Build 16299
  * System Manufacturer: SAMSUNG ELECTRONICS CO., LTD.
  * System Model: QX310/QX410/QX510/SF310/SF410/SF510
  * BIOS: AMIBIOS Version 03MX.M005.20101011.SCY
  * Processor: Intel(R) Core(TM) i5 CPU M 460 @ 2.53 GHZ (4CPUs), 2.5GHz
  * Memory: 8192 MB RAM
  * Display Device: Intel(R) HD Graphics (Core i5)

  - BORG Calendar Version: 1.8.3
  - Java Version: 1.8.0_161

- **Report type**: Coding Error
- **Reproducibility**: 100%
- **Severity**: Low
- **Problem summary**: When inputing a negative integer as an argument for the minuteString method in DataUtil class. It produces the number in negative which is not correct since the reminder of the negative integer cannot be ne negaitive.
- **Problem description**:
  Steps to Reproduce
  1. Load the source code of Borg Calender in Java
  2. Create a new junit test case
  3. call the net.sf.borg.common.DateUtil.minuteString() in the class with -70 as an argument

  Results
- Expected Results: After inputing the -70, it should gives 10 as the output since hour produce empty string and only minutes produce the results.The expected result is to see positive 10 but it gives -10 since (-70) mod 60 is 10 minutes.
- Real Results: The actual result produce the bug since it shows -10 as the outputs.
- **New or old bug**: New

2.2 Decision Table Testing

- **Technique**: *Decision Table Testing*
- **Class**: *net.sf.borg.common.DateUtil.java*
- **Method**: *isAfter(Date d1, Date d2)*
- **Method description**: The method checks if a given date *d1* falls on a later calendar day than date *d2*. It returns **true** if *d1* does fall on a later calendar day than *d2* and **false** otherwise.
  - **d1** - The first argument is of type Java Date Object.
  - **d2** - The second argument is of type Java Date Object.
- **Justification**: Decision table testing technique is an appropriate testing technique for this method because there are decision making to be done among the input variables. It consists of logical relationships among the input variables, i.e date *d1* appearing before, after or at the same time as date *d2*, which directly affects the output.

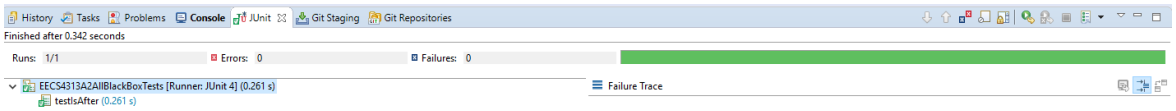

Figure 4: Test run result for the isAfter function

```
30    /**
31     * Checks if one date falls on a later calendar day than another.
32     *
33     * @param d1
34     *            the first date
35     * @param d2
36     *            the second date
37     *
38     * @return true, if is after
39     */
40    public static boolean isAfter(Date d1, Date d2) {
41
42        GregorianCalendar tcal = new GregorianCalendar();
43        tcal.setTime(d1);
44        tcal.set(Calendar.HOUR_OF_DAY, 0);
45        tcal.set(Calendar.MINUTE, 0);
46        tcal.set(Calendar.SECOND, 0);
47        GregorianCalendar dcal = new GregorianCalendar();
48        dcal.setTime(d2);
49        dcal.set(Calendar.HOUR_OF_DAY, 0);
50        dcal.set(Calendar.MINUTE, 10);
51        dcal.set(Calendar.SECOND, 0);
52
53        if (tcal.getTime().after(dcal.getTime())) {
54            return true;
55        }
56
57        return false;
58    }
```

Figure 5: Statement Coverage View for the isAfter function

DateUtil	22.2 %	48	168	216
minuteString(int)	0.0 %	0	115	115
setToMidnight(Date)	0.0 %	0	26	26
dayOfEpoch(Date)	0.0 %	0	24	24
isAfter(Date, Date)	100.0 %	48	0	48

Figure 6: Statement Coverage Metrics for the isAfter function

2.2.1 Control Flow Graph

The following is the code snippet for the *isAfter* function:

```
/**
 * Checks if one date falls on a later calendar day than
 * another.
 *
 * @param d1
 *            the first date
 * @param d2
 *            the second date
 *
 * @return true, if is after
 */
1. public static boolean isAfter(Date d1, Date d2) {

2.      GregorianCalendar tcal = new GregorianCalendar();
3.      tcal.setTime(d1);
4.      tcal.set(Calendar.HOUR_OF_DAY, 0);
5.      tcal.set(Calendar.MINUTE, 0);
6.      tcal.set(Calendar.SECOND, 0);
7.      GregorianCalendar dcal = new GregorianCalendar();
8.      dcal.setTime(d2);
9.      dcal.set(Calendar.HOUR_OF_DAY, 0);
10.     dcal.set(Calendar.MINUTE, 10);
11.     dcal.set(Calendar.SECOND, 0);

12.     if (tcal.getTime().after(dcal.getTime())) {
13.             return true;
14.     }

15.     return false;
16. }
```

Name	Covered Statements
A	from line 1 to 11
B	if (tcal.getTime().after(dcal.getTime()))
C	return true;
D	return false;

Table 4: CFG Segment Table for the isAfter method

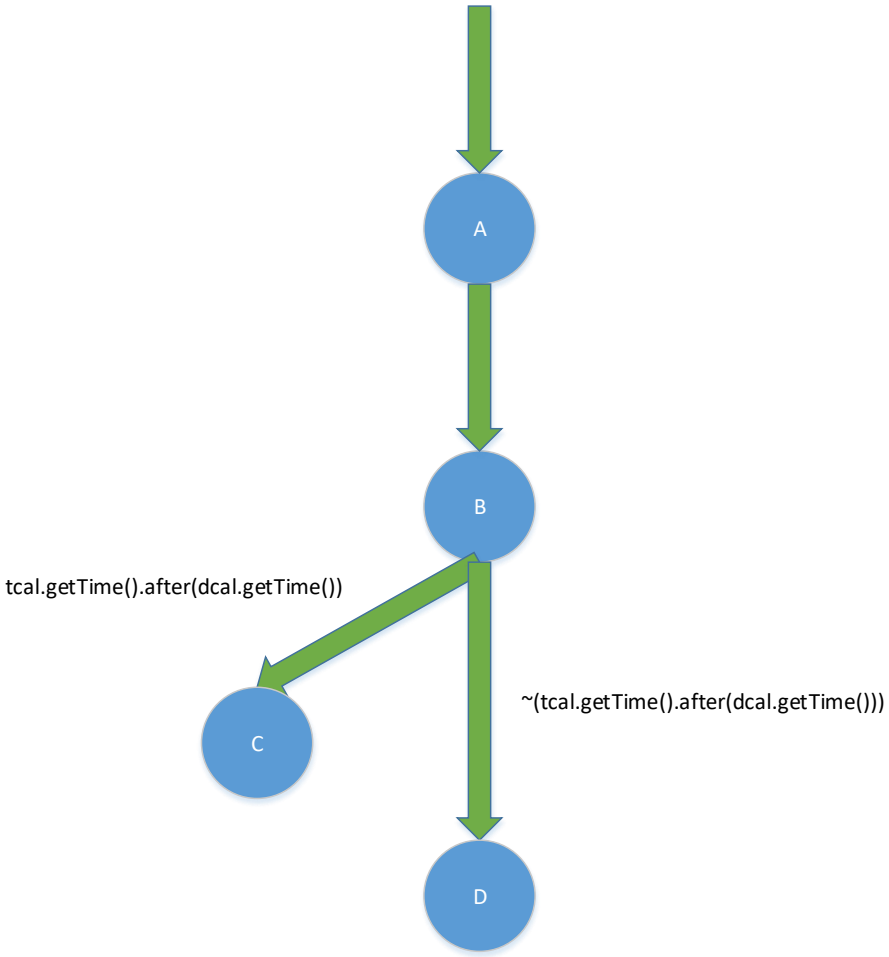

Figure 7: Control Flow Graph for the isAfter function

- **Number of paths in the method**: 2
  - **Path P1** - The first path is when the method returns *true* if date d1 is after date d2. Following the CFG diagram above, refer to diagram 8, this is path is: **ABC**.
  - **Path P2** - The second path is when the method returns *false* if date d1 is not after date d2 (i.e d1 is either equal to or before d2). Following the CFG diagram above, refer to diagram 8, this is path is: **ABD**.
- **Estimated % of path covered in the test**: 100%

The following code snippet depicts the test case and path coverages for *isAfter* function:

```
@Test
public void testIsAfter() {
        /** Method used: Decision Table Testing **/

        Date d1 = new Date(117, 11, 3);
        Date d2 = new Date(117, 11, 3);
        boolean result;

        // date d1 is equal to d2
        result = DateUtil.isAfter(d1, d2);
        assertFalse("Date d1 is equal to d2", result);

        // date d1 is before d2
        d1.setDate(2);
        result = DateUtil.isAfter(d1, d2);
        assertFalse("Date d1 is before d2", result);

        // date d1 is after d2
        d1.setDate(4);
        result = DateUtil.isAfter(d1, d2);
        assertTrue("Date d1 is after d2", result);
}
```

Path coverage **P1**:

```
// date d1 is after d2
d1.setDate(4);
result = DateUtil.isAfter(d1, d2);
assertTrue("Date d1 is after d2", result);
```

Path coverage **P2**:

```
// date d1 is equal to d2
result = DateUtil.isAfter(d1, d2);
assertFalse("Date d1 is equal to d2", result);

// date d1 is before d2
d1.setDate(2);
result = DateUtil.isAfter(d1, d2);
assertFalse("Date d1 is before d2", result);
```