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 4, 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

- The statement coverage measurements for your Assignment 2 test suite.
- A description of the test cases that you added in this assignment to improve statement coverage. The marker will not read your code in order to see what you tested. You have to describe it.
- The statement coverage measurements for your final submission. Include the screenshots of the test running results and the screenshots of the coverage measurement. If your coverage is not 100
- The Control Flow Graph you created. Indicate the segments clearly (you will probably need to include the code for this).
- The path coverage discussion described in section 2 above.
- 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 (if there are new ones).

2.1 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.

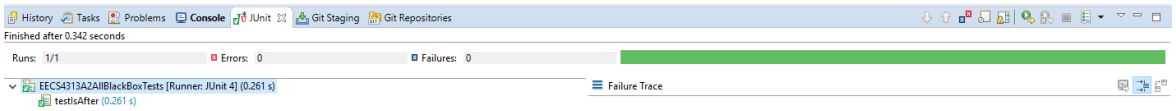

Figure 1: Test run result for the isAfter function

```java
/**
 * 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
 */
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;
}
```

Figure 2: 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 3: Statement Coverage Metrics for the isAfter function

2.1.1 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 1: CFG Segment Table for the isAfter method

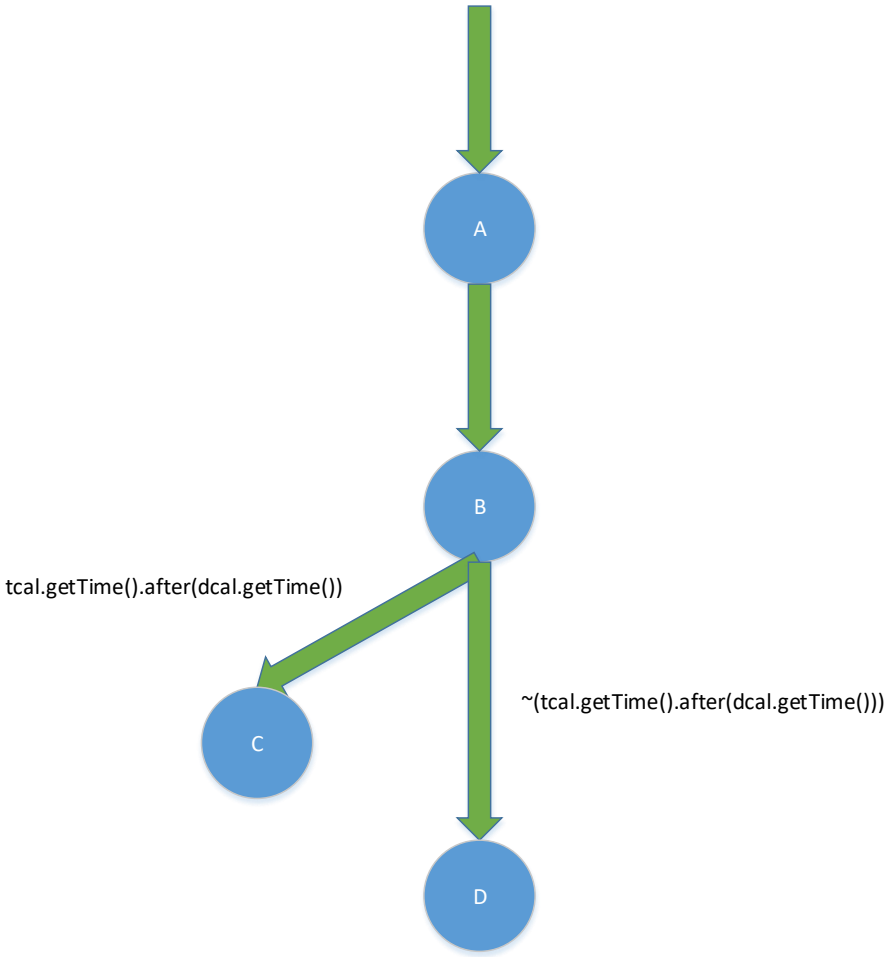

Figure 4: Control Flow Graph for the isAfter function

- **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 **??**, 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 **??**, 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);
```