Black-box and White-box Testing with JUnit

EECS 4313: Software Engineering Testing

Assignment 2

March 05, 2018

Authors

Daniel McVicar

Glib Sitiugin

Nisha Sharma

Rijul Aggarwal

Varsha Ragavendran

Table of Contents

List of Figures

List of Tables

1.0 Black-box Testing with JUnit

This section describes the specification of the three methods chosen for testing along with a description of black-box testing techniques utilized for testing those methods in JUnit. Evaluation of these testing techniques can also be found in this section.

1.1.0 Specification of the Selected Java Methods

Following are the specifications for three Java methods selected for the purpose of this assignment. Actual source code [1] is available in Appendix.

1.1.1 Selected method one

public static boolean isCompatible(Calendar date, String freq,Collection<Integer> daylist)

This method checks if the given date is valid with a certain repeating event frequency. If the date is valid for the given frequency, the function returns true. Otherwise, it returns false. Frequency strings are generated by providing an integer to the getFreqString(Int i) method. For our test the key frequencies will be:

- Weekday
- Weekend
- Monday, Wednesday, Friday
- Tuesday, Thursday
- Last day of the month.

And less importantly:

- Once, Daily, Weekly, Biweekly, Monthly, or Yearly.

1.1.2 Selected method two

public static boolean isAfter(Date d1, Date d2)

This method checks if one date falls on a later calendar day than another. The method takes two date objects as arguments. If the first date falls on a later day than second date, the method returns true otherwise it returns false. The method utilizes "after" method from "`java.util.Calendar`" class which compares two Calendar time objects; it returns whether this Calendar represents a time after the time represented by the specified Object.

1.1.3 Selected method three

public static String minuteString(int mins)

This method takes in the number of minutes as an integer and returns a human readable String representation of it in hours and minutes. This method expects non-negative integer value of minutes. The returned String takes care of singular or plural representation of hours or minutes in the returned String as well.

1.2.0 Testing Techniques Chosen and Justification

For testing the three methods chosen in the previous section, we decided to use Decision Table Testing, Equivalence Class Testing and Boundary Value Testing respectively. Here we briefly describe why we chose these testing techniques for those methods.

1.2.1 Decision Table Testing

Decision Table Testing (DTT) is particularly well suited for testing the isCompatible method because of the clear relationship between the input conditions - the date and frequency - and the action taken - returning true or false. By building a table of input conditions, it is easy to put together a comprehensive test suite.

1.2.2 Equivalence Class Testing

Equivalence Class Testing (ECT) is appropriate for testing systems that can be represented in the form of a function with one or more variables whose domains have well defined intervals. `“isAfter”` method of `“net.sf.borg.common.DateUtil”` can be represented as a function of two Date variables with domains of well defined intervals. ECT is suited for such complex functions as it assures non-redundancy and completeness of test cases when implemented correctly.

1.2.3 Boundary Value Testing

Boundary Value Testing (BVT) is appropriate for testing `minuteString` method because this it takes in one argument only, for which there is a definite boundary. The argument is non-negative, so it must lie between zero and the maximum value of the integer (inclusive). Hence we can say it is a bounded independent variable. The fault in the code can only have a single source (integer value of minutes) which can only vary within a defined range.

1.3.0 Application of the three Testing Strategies

This section describes how we utilized these testing techniques for our test suite implementation in JUnit. This section also describes the test cases generated.

1.3.1 Decision Table Testing

Decision Table Testing (DDT) requires us to put together a table relating input conditions to actions. There are 14 important input conditions for the method `isCompatible`:

- C1 - Day is a Monday
- C2 - Day is a Tuesday
- C3 - Day is a Wednesday
- C4 - Day is a Thursday

- C5 - Day is a Friday
- C6 - Day is a Saturday
- C7 - Day is a Sunday
- C8 - Day is the last day of the month
- C9 - Frequency is Once, Daily, Weekly, Biweekly, Monthly, or Yearly
- C10 - Frequency is Weekday
- C11 - Frequency is Weekend
- C12 - Frequency is Monday, Wednesday, Friday
- C13 - Frequency is Tuesday, Thursday
- C14 - Frequency is Last Day of Month

There are only two important actions for our program to take:

- A1 - Return True
- A2 - Return False

Following decision tables capture the test cases implemented in JUnit.

Table 1: Weekday Frequency Check

Condition							
C1	T	F	F	F	F	F	F
C2	F	T	F	F	F	F	F
C3	F	F	T	F	F	F	F
C4	F	F	F	T	F	F	F
C5	F	F	F	F	T	F	F
C6	F	F	F	F	F	T	F
C7	F	F	F	F	F	F	T
C8	-	-	-	-	-	-	-
C9	F	F	F	F	F	F	F
C10	T	T	T	T	T	T	T
C11	F	F	F	F	F	F	F
C12	F	F	F	F	F	F	F
C13	F	F	F	F	F	F	F
C14	F	F	F	F	F	F	F
Action							
A1	T	T	T	T	T	F	F
A2	F	F	F	F	F	T	T

<u>Table 2:</u> Weekend Frequency Check

Condition							
C1	T	F	F	F	F	F	F
C2	F	T	F	F	F	F	F
C3	F	F	T	F	F	F	F
C4	F	F	F	T	F	F	F
C5	F	F	F	F	T	F	F
C6	F	F	F	F	F	T	F
C7	F	F	F	F	F	F	T
C8	-	-	-	-	-	-	-
C9	F	F	F	F	F	F	F
C10	F	F	F	F	F	F	F
C11	T	T	T	T	T	T	T
C12	F	F	F	F	F	F	F
C13	F	F	F	F	F	F	F
C14	F	F	F	F	F	F	F
Action							
A1	F	F	F	F	F	T	T
A2	T	T	T	T	T	F	F

<u>Table 3:</u> Monday, Wednesday, Friday Frequency Check

Condition							
C1	T	F	F	F	F	F	F
C2	F	T	F	F	F	F	F
C3	F	F	T	F	F	F	F
C4	F	F	F	T	F	F	F
C5	F	F	F	F	T	F	F
C6	F	F	F	F	F	T	F
C7	F	F	F	F	F	F	T
C8	-	-	-	-	-	-	-
C9	F	F	F	F	F	F	F
C10	F	F	F	F	F	F	F
C11	F	F	F	F	F	F	F

C12	T	T	T	T	T	T	T
C13	F	F	F	F	F	F	F
C14	F	F	F	F	F	F	F
Action							
A1	T	F	T	F	T	F	F
A2	F	T	F	T	F	T	T

Table 4: Tuesday, Thursday Frequency Check

Condition							
C1	T	F	F	F	F	F	F
C2	F	T	F	F	F	F	F
C3	F	F	T	F	F	F	F
C4	F	F	F	T	F	F	F
C5	F	F	F	F	T	F	F
C6	F	F	F	F	F	T	F
C7	F	F	F	F	F	F	T
C8	-	-	-	-	-	-	-
C9	F	F	F	F	F	F	F
C10	F	F	F	F	F	F	F
C11	F	F	F	F	F	F	F
C12	F	F	F	F	F	F	F
C13	T	T	T	T	T	T	T
C14	F	F	F	F	F	F	F
Action							
A1	F	T	F	T	F	F	F
A2	T	F	T	F	T	T	T

Table 5: Last Month of the Day Frequency

Condition							
C1	-	-	-	-	-	-	-
C2	-	-	-	-	-	-	-
C3	-	-	-	-	-	-	-
C4	-	-	-	-	-	-	-

C5	-	-	-	-	-	-	-
C6	-	-	-	-	-	-	-
C7	-	-	-	-	-	-	-
C8	T	T	T	T	T	T	T
C9	F	F	F	F	F	F	F
C10	F	F	F	F	F	F	F
C11	F	F	F	F	F	F	F
C12	F	F	F	F	F	F	F
C13	F	F	F	F	F	F	F
C14	T	T	T	T	T	T	T
Action							
A1	T	T	T	T	T	T	T
A2	F	F	F	F	F	F	F

Table 6: Once, Daily, Weekly, Biweekly, Monthly, or Yearly Frequency

Condition							
C1	-	-	-	-	-	-	-
C2	-	-	-	-	-	-	-
C3	-	-	-	-	-	-	-
C4	-	-	-	-	-	-	-
C5	-	-	-	-	-	-	-
C6	-	-	-	-	-	-	-
C7	-	-	-	-	-	-	-
C8	-	-	-	-	-	-	-
C9	T	T	T	T	T	T	T
C10	F	F	F	F	F	F	F
C11	F	F	F	F	F	F	F
C12	F	F	F	F	F	F	F
C13	F	F	F	F	F	F	F
C14	F	F	F	F	F	F	F
Action							
A1	T	T	T	T	T	T	T
A2	F	F	F	F	F	F	F

1.3.2 Equivalence Class Testing

Equivalence Class Testing (ECT) removes redundancy from Boundary Value Testing (BVT). For testing "`isAfter`" with Equivalence Class Testing, we used following strategy:
We chose to do Weak Normal ECT and Weak Robust ECT testing for the purpose of this assignment. We divided the test case into mutually disjoint subsets and then chose one test case from each of these subsets. The subsets for Weak Normal consider valid inputs for date objects. However, the subsets for Weak Robust considers combinations of both valid and invalid inputs for date objects.
<u>Valid Date:</u> Date within the valid date range and is a real date (days range from 1-31, months range from 0-11, years range from 1900 onwards)
<u>Invalid Date:</u> Date not within the valid date range or not a real date

Weak Normal test cases check if following cases are met:
WN1: Valid date change: Input variables with varying valid date; day of first date after second one (date1: 12/1/2018, date2: 1/1/2018)
WN2: Valid month change: Input variables with varying valid month; month of first date after second one (date1: 1/6/2018, date2: 1/1/2018)
WN3: Valid year change: Input variable with varying valid year; year of first date after second one (date1: 15/7/2018, date2: 15/7/2010)

Weak Robust test cases add to Weak Normal test cases by checking if following cases are met as well:
WR1: Both dates are Invalid second one is before the first one (date1: 29/1/2018, date2: 31/5/2010)
WR2: First date is invalid, but the second one is valid and after the first one (date1: 31/3/2018, date2: 20/5/2020)
WR3: First date is valid, but the second one is invalid and after the first one (date1: 3/3/2018, date2: 29/1/2020)
WR4: Both dates are valid and the second date is before the first one (date1: 3/5/2018, date2: 20/5/2050)

It should be noted that varying the equivalence relation would result in a different number of test cases. Strong Robust and Strong Normal may have same or more number of test cases compared to Weak Normal and/or Weak Robust.

1.3.3 Boundary Value Testing

Boundary Value Testing (BVT) helps make simple and effective testing for method by trying to test the corner/edge cases rapidly for independent variables.

The boundary for the input variable is non-negative integers, i.e. 0 - Integer.MAX_VALUE.

Hence The test was done with the following cases:

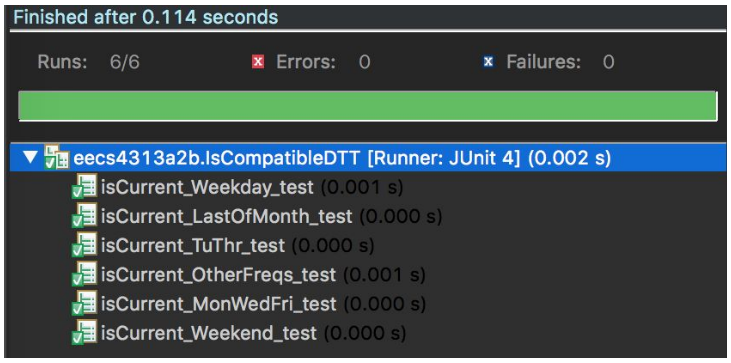
Case	Argument
Minimum	0
Just above minimum	1
Nominal	320
Just below maximum	Integer.MAX_VALUE - 1
Maximum	Integer.MAX_VALUE

1.4.0 Evaluation of the Test Cases

In this section, we evaluate the test cases developed in section 1.3.0 by presenting the results of running the test suite in JUnit.

1.4.1 Decision Table Testing

Figure 1 demonstrates the results of the test suite developed. The only important special value tested was for the "Last day of month" frequency, which was tested with and without an end of the month day value. All other test cases correspond to a frequency setting being tested with every day of the week.

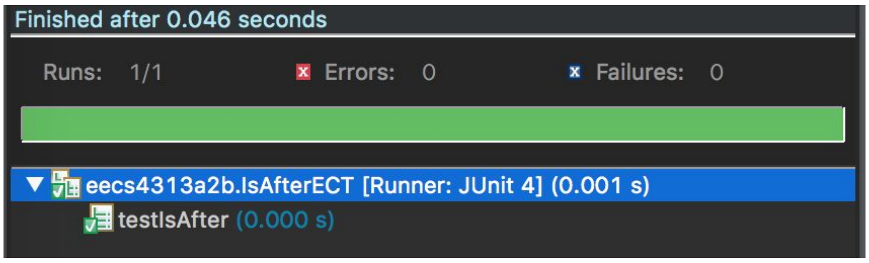


```
Finished after 0.114 seconds

Runs: 6/6        Errors: 0        Failures: 0

eecs4313a2b.IsCompatibleDTT [Runner: JUnit 4] (0.002 s)
    isCurrent_Weekday_test (0.001 s)
    isCurrent_LastOfMonth_test (0.000 s)
    isCurrent_TuThr_test (0.000 s)
    isCurrent_OtherFreqs_test (0.001 s)
    isCurrent_MonWedFri_test (0.000 s)
    isCurrent_Weekend_test (0.000 s)
```

Figure 1: JUnit test results for Decision Table Testing on method "*isCompatible*"

1.4.2 Equivalence Class Testing

Figure 2, below is a screenshot of test running results. There was no special value testing involved. According to the testing results, the source code demonstrates expected behavior. The method returns true whenever the first date argument is after the second date arguments.

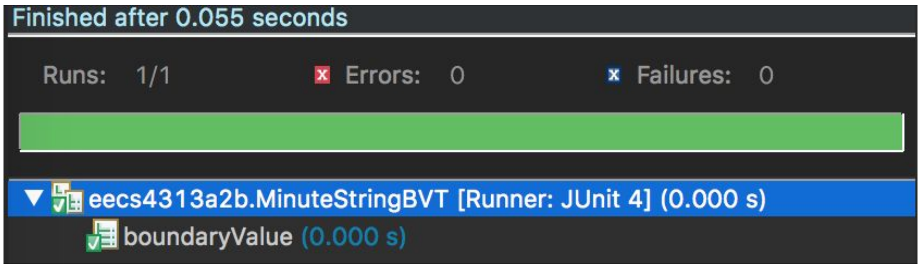


```
Finished after 0.046 seconds
Runs: 1/1        Errors: 0        Failures: 0
▼ eecs4313a2b.IsAfterECT [Runner: JUnit 4] (0.001 s)
    testIsAfter (0.000 s)
```

Figure 2: JUnit test results for Equivalence Class Testing on method "*isAfter*"

1.4.3 Boundary Value Testing

Figure 3, below is a screenshot of test running results. There was no special value testing involved. According to the testing results, the source code demonstrates expected behavior. The method returns the expected human readable String representation of the integer value of minutes passed as argument. The method correctly works near the boundaries, and takes care of the singular/plural tense.



```
Finished after 0.055 seconds
Runs: 1/1        Errors: 0        Failures: 0
▼ eecs4313a2b.MinuteStringBVT [Runner: JUnit 4] (0.000 s)
    boundaryValue (0.000 s)
```

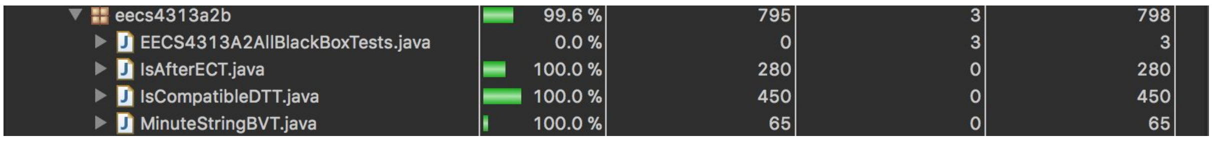
Figure 3: JUnit test result for Boundary Value Testing on method *"minuteString"*

1.5 Bug Reports

No bugs were found during testing of "`isCompatable`" method.
No bugs were found during testing of "`isAfter`" method.
No bugs were found during testing of "`minuteString`" method.

2.0 White-box Testing with JUnit

2.1.0 Statement Coverage Measurements before White-Box Testing



eecs4313a2b	99.6 %	795	3	798
EECS4313A2AllBlackBoxTests.java	0.0 %	0	3	3
IsAfterECT.java	100.0 %	280	0	280
IsCompatibleDTT.java	100.0 %	450	0	450
MinuteStringBVT.java	100.0 %	65	0	65

Figure 4: Statement Coverage Performed on black-box test cases

2.2.0 Description of the Test Cases

The statement coverage as shown above in Figure 4, resulted in 100% coverage for the black-box testing approach. Although the coverage shows as 100%, we took a look into the code to see if further test cases can be added to strengthen the coverage and ensure correctness of the code implementation.

2.2.1 Additional Test Cases for Boundary Value Testing:

Looking at the implementation for public static String `minuteString(int mins)` located in `net.sf.borg.common.DateUtil`, we noticed negative inputs were also considered for minutes. Therefore the minimum value and the just above minimum value considered for boundary value testing now incorporates the use of Integer.MIN_VALUE *(-2^{31})* and Integer.MIN_VALUE+1 *($-2^{31}+1$)*, respectively.

The following two test cases were added to test the cases mentioned above,

```java
private String MINIMUM;
private String ABOVE_MINIMUM;

@Before
public void setup() {
 MINIMUM = "-8 Minutes";
 ABOVE_MINIMUM = "-7 Minutes";
}

@Test
public void boundaryValue() {
 // minuteString also accepts negative values
 // base minimum
 minutes = Integer.MIN_VALUE;
 Assert.assertEquals(MINIMUM, DateUtil.minuteString(minutes));

 // just a bit above minimum
 minutes = Integer.MIN_VALUE + 1;
 Assert.assertEquals(ABOVE_MINIMUM, DateUtil.minuteString(minutes))
}
```

2.2.2 Additional Test Cases for Equivalence Class Testing

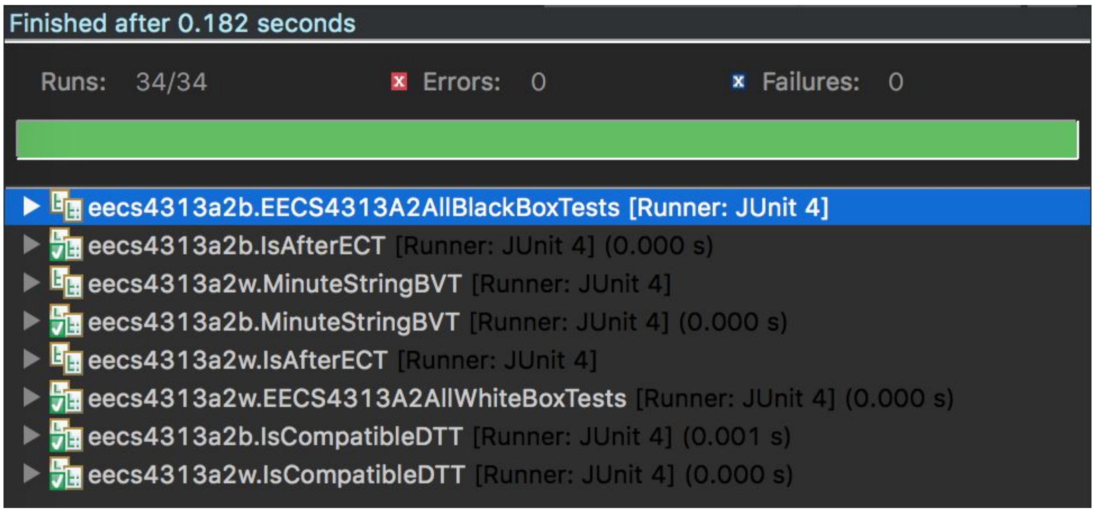
Viewing the code of the method isAfter() in the DateUtil class (available in Appendix) revealed strange implementation details. In particular, the minute count of the second date parameter was set to 10, while hours and seconds were set to 0. Seconds, minutes and hours of the first date parameter were set to 0. We applied Weak Normal ECT while taking advantage of implementation details mentioned above. In other words, we asserted the equality of output of results of isAfter() method in DateUtil class and after() method of GregorianCalendar classes The testing did not reveal any bugs resulting from the way the method was implemented by developers.

```
private Date date1;
private Date date2;
private Date date3;
private Date date4;
private Date date5;
private Date date6;

@Before
public void setup() {
    date1 = new Date(2012, 1, 1);
    date2 = new Date(2012, 0, 31);
    date3 = new Date(2000, 1, 29);
    date4 = new Date(2000, 1, 28);
    date5 = new Date(2018, 3, 1);
    date6 = new Date(2018, 3, 30);
}

@Test
public void testIsAfter() {

    GregorianCalendar cal1 = new GregorianCalendar();
    GregorianCalendar cal2 = new GregorianCalendar();

    cal1.setTime(date1);
    cal2.setTime(date2);
    assertEquals(cal1.after(cal2), DateUtil.isAfter(date1, date2));
    assertEquals(cal2.after(cal1), DateUtil.isAfter(date2, date1));

    cal1.setTime(date3);
    cal2.setTime(date4);
    assertEquals(cal1.after(cal2), DateUtil.isAfter(date3, date4));
    assertEquals(cal2.after(cal1), DateUtil.isAfter(date4, date3));

    cal1.setTime(date5);
    cal2.setTime(date6);
    assertEquals(cal1.after(cal2), DateUtil.isAfter(date5, date6));
    assertEquals(cal2.after(cal1), DateUtil.isAfter(date6, date5));
}
```

2.2.3 Additional Test Cases for Decision Table Testing:

The test cases created for the `static public boolean isCompatible(Calendar date, String freq,Collection<Integer> daylist)` method found in `net.sf.borg.model.Repeat` class during black-box testing covered all paths the method considered. Therefore, no further test cases were added.

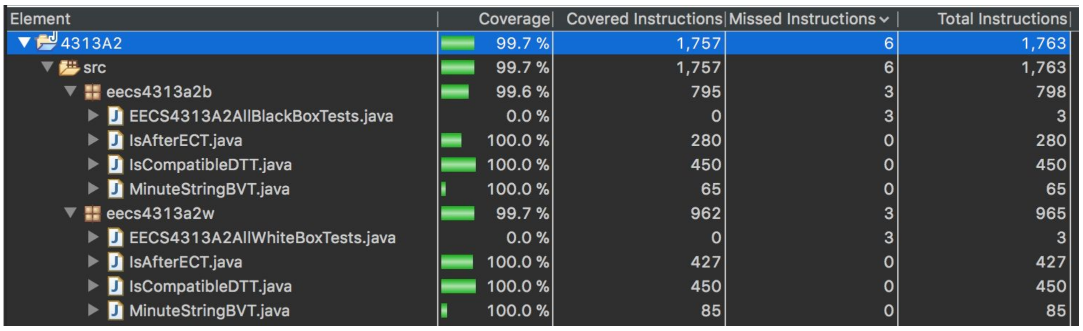
2.3.0 Statement Coverage Measurements after White-Box Testing



```
Finished after 0.182 seconds

Runs:   34/34          Errors:   0          Failures:   0

▶ eecs4313a2b.EECS4313A2AllBlackBoxTests [Runner: JUnit 4]
▶ eecs4313a2b.IsAfterECT [Runner: JUnit 4] (0.000 s)
▶ eecs4313a2w.MinuteStringBVT [Runner: JUnit 4]
▶ eecs4313a2b.MinuteStringBVT [Runner: JUnit 4] (0.000 s)
▶ eecs4313a2w.IsAfterECT [Runner: JUnit 4]
▶ eecs4313a2w.EECS4313A2AllWhiteBoxTests [Runner: JUnit 4] (0.000 s)
▶ eecs4313a2b.IsCompatibleDTT [Runner: JUnit 4] (0.001 s)
▶ eecs4313a2w.IsCompatibleDTT [Runner: JUnit 4] (0.000 s)
```

Figure 5 : Test Results for White-box and Black-box Test Cases

The statement coverage measurements of the white-box testing approach is displayed below in [Figure 6](#). All Test classes have 100% coverage, except for the test suites created for all the black box tests and all the white box tests as there is no code in these classes. These two test suites will be used for automated testing via command line.



Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ 4313A2	99.7 %	1,757	6	1,763
▼ src	99.7 %	1,757	6	1,763
▼ eecs4313a2b	99.6 %	795	3	798
▶ EECS4313A2AllBlackBoxTests.java	0.0 %	0	3	3
▶ IsAfterECT.java	100.0 %	280	0	280
▶ IsCompatibleDTT.java	100.0 %	450	0	450
▶ MinuteStringBVT.java	100.0 %	65	0	65
▼ eecs4313a2w	99.7 %	962	3	965
▶ EECS4313A2AllWhiteBoxTests.java	0.0 %	0	3	3
▶ IsAfterECT.java	100.0 %	427	0	427
▶ IsCompatibleDTT.java	100.0 %	450	0	450
▶ MinuteStringBVT.java	100.0 %	85	0	85

Figure 6 : Statement Coverage Performed on White-box Test Cases

2.4.0 Control Flow Graph

The control flow graph below is for the `static public boolean isCompatible(Calendar date, String freq,Collection<Integer> daylist)` method found in `net.sf.borg.model.Repeat` class.

The steps executed at each node of the control flow graph are labelled based on the implementation of the `static public boolean isCompatible(Calendar date, String freq,Collection<Integer> daylist)` method found in the Appendix.

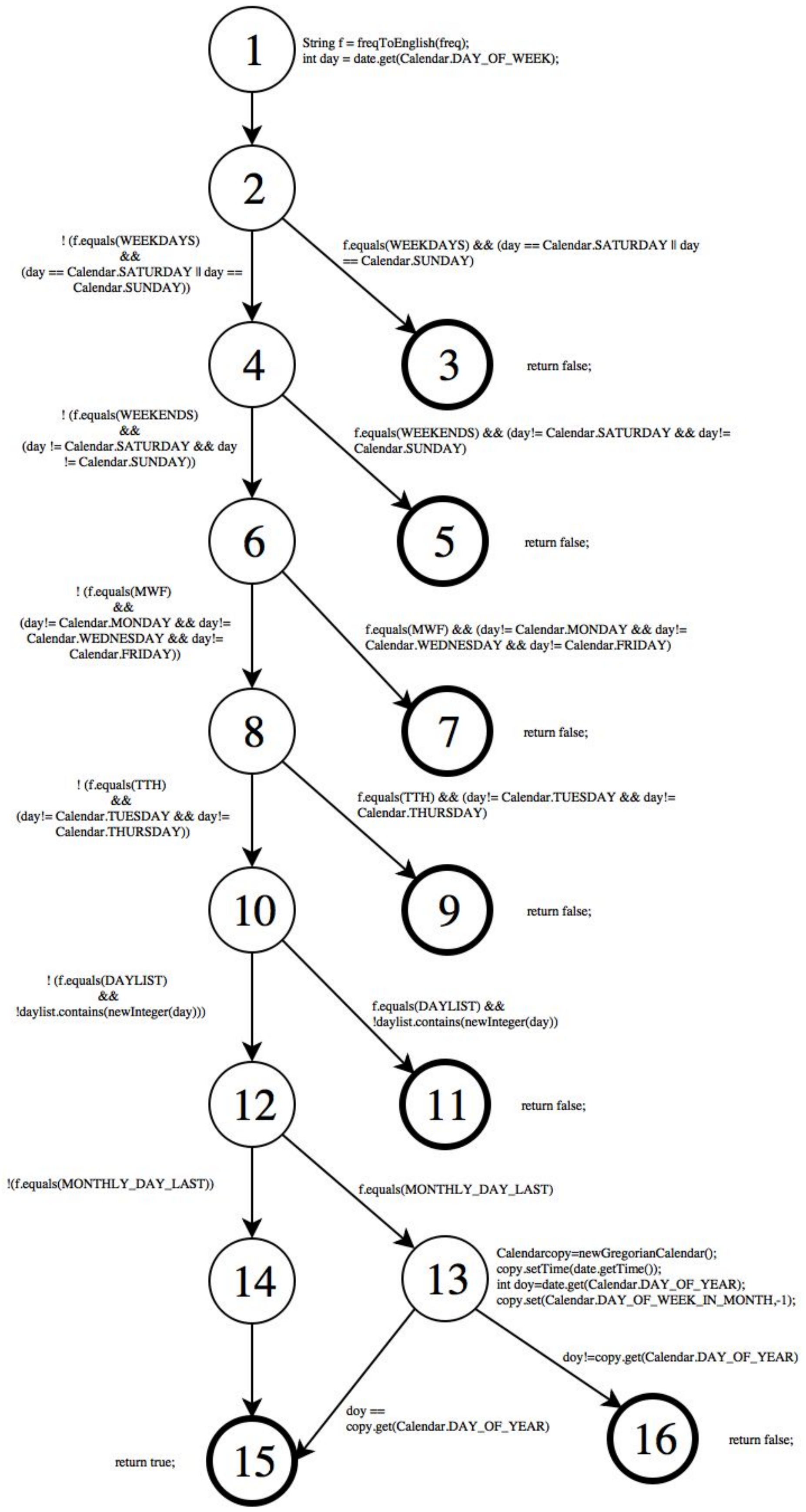

1
String f = freqToEnglish(freq);
int day = date.get(Calendar.DAY_OF_WEEK);
2
! (f.equals(WEEKDAYS) && (day == Calendar.SATURDAY || day == Calendar.SUNDAY))
f.equals(WEEKDAYS) && (day == Calendar.SATURDAY || day == Calendar.SUNDAY)
4
3
return false;
! (f.equals(WEEKENDS) && (day != Calendar.SATURDAY && day != Calendar.SUNDAY))
f.equals(WEEKENDS) && (day!= Calendar.SATURDAY && day!= Calendar.SUNDAY)
6
5
return false;
! (f.equals(MWF) && (day!= Calendar.MONDAY && day!= Calendar.WEDNESDAY && day!= Calendar.FRIDAY))
f.equals(MWF) && (day!= Calendar.MONDAY && day!= Calendar.WEDNESDAY && day!= Calendar.FRIDAY)
8
7
return false;
! (f.equals(TTH) && (day!= Calendar.TUESDAY && day!= Calendar.THURSDAY))
f.equals(TTH) && (day!= Calendar.TUESDAY && day!= Calendar.THURSDAY)
10
9
return false;
! (f.equals(DAYLIST) && !daylist.contains(newInteger(day)))
f.equals(DAYLIST) && !daylist.contains(newInteger(day))
12
11
return false;
!(f.equals(MONTHLY_DAY_LAST))
f.equals(MONTHLY_DAY_LAST)
14
13
Calendarcopy=newGregorianCalendar();
copy.setTime(date.getTime());
int doy=date.get(Calendar.DAY_OF_YEAR);
copy.set(Calendar.DAY_OF_WEEK_IN_MONTH,-1);
doy!=copy.get(Calendar.DAY_OF_YEAR)
doy == copy.get(Calendar.DAY_OF_YEAR)
16
return false;
return true;
15

Figure 7: Control Flow Graph for Method "*static public boolean isCompatible*"

2.5.0 Path Coverage

<u>Possible Execution Paths</u>

The possible execution paths based on the Control Flow Graph in **Figure 7**, of the `public boolean isCompatible(Calendar date, String freq,Collection<Integer> daylist)` method found in package `net.sf.borg.model.Repeat class` are :

- 1→2→3
- 1→2→4→5
- 1→2→4→6→7
- 1→2→4→6→8→9
- 1→2→4→6→8→10→11
- 1→2→4→6→7→10→12→13→16
- 1→2→4→6→8→10→12→13→15
- 1→2→4→6→8→10→12→15

As shown in the coverage metrics in **Figure 6** , the decision table testing class has 100% coverage and therefore all possible execution paths of the `public boolean isCompatible(Calendar date, String freq,Collection<Integer> daylist)` method has been covered by our test suite.

For example, the following test case below was implemented to test the path 1→2→3:

```
private Calendar saturday_calendar;
private Calendar sunday_calendar;
private String weekday_freq;
private Collection < Integer > daylist;

@Before
public void setup() {
 saturday_calendar = new GregorianCalendar(2018, 2, 10);
 sunday_calendar = new GregorianCalendar(2018, 2, 11);
 weekday_freq = Repeat.getFreqString(8);
 daylist = new ArrayList < Integer > ();
}

@Test
public void isCurrent_Weekday_test() {
 assertFalse(Repeat.isCompatible(saturday_calendar, weekday_freq, daylist));
 assertFalse(Repeat.isCompatible(sunday_calendar, weekday_freq, daylist));
}
```

2.6.0 Bug Reports

No bugs were discovered while performing white box testing.

References

[1] "mikeberger/borg_calendar", *GitHub*, 2018. [Online]. Available: https://github.com/mikeberger/borg_calendar/tree/master/BORGCalendar. [Accessed: 01- Mar- 2018].

Appendix

Method 1

The `minuteString(int mins)` method can be found in `net.sf.borg.common.DateUtil` package.

```java
/**
 * generate a human readable string for a particular number of minutes
 *
 * @param mins - the number of minutes
 *
 * @return the string
 */
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;
}
```

Method 2

The method `isAfter(Date d1, Date d2)` can be found in `net.sf.borg.common.DateUtil` package.

```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;
}
```

Method 3

The method `isCompatible(Calendar date, String freq, Collection<Integer> daylist)` can be found in `net.sf.borg.model.Repeat` package.

```java
/**
     * Checks to see if a particular day is valid for certain strict repeat
     * types
     *
     * @param date
     *            the date
     * @param freq
     *            the frequency
     * @param daylist
     *            the daylist (for repeat with a list of days)
     *
     * @return true, if the daye is compatible with the repeat frequency
     */
    static public boolean isCompatible(Calendar date, String freq,
            Collection<Integer> daylist) {
        String f = freqToEnglish(freq);
        int day = date.get(Calendar.DAY_OF_WEEK);
        if (f.equals(WEEKDAYS)
                && (day == Calendar.SATURDAY || day == Calendar.SUNDAY))
            return false;
        else if (f.equals(WEEKENDS)
                && (day != Calendar.SATURDAY && day != Calendar.SUNDAY))
            return false;
        else if (f.equals(MWF)
                && (day != Calendar.MONDAY && day != Calendar.WEDNESDAY && day !=
Calendar.FRIDAY))
            return false;
        else if (f.equals(TTH)
                && (day != Calendar.TUESDAY && day != Calendar.THURSDAY))
            return false;
        else if (f.equals(DAYLIST) && !daylist.contains(new Integer(day)))
            return false;
        else if( f.equals(MONTHLY_DAY_LAST))
        {
            Calendar copy = new GregorianCalendar();
            copy.setTime(date.getTime());
            int doy = date.get(Calendar.DAY_OF_YEAR);
            copy.set(Calendar.DAY_OF_WEEK_IN_MONTH,-1);
            if( doy != copy.get(Calendar.DAY_OF_YEAR))
                return false;
        }
        return true;
    }
```