CS 362 – Software Engineering 2
Professor Ali Aburas – Spring 2018 – On-Campus
Assignment 2 – Code Coverage
Khuong Luu

1. The Utility of the code coverage tool

1.1. Screenshot of my code coverage

Appt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
setValid()		87%		66%	5	10	1	13	0	1
toString()		94%		50%	1	2	1	4	0	1
Appt(int, int, int, int, int, String, String, String)		100%		n/a	0	1	0	14	0	1
represntationApp()		100%		100%	0	4	0	8	0	1
Appt(int, int, int, String, String, String)		100%		n/a	0	1	0	3	0	1
setRecurrence(int[], int, int, int)		100%		n/a	0	1	0	5	0	1
setRecurDays(int[])		100%		100%	0	2	0	4	0	1
setTitle(String)		100%		100%	0	2	0	4	0	1
setDescription(String)		100%		100%	0	2	0	4	0	1
setEmailAddress(String)		100%		100%	0	2	0	4	0	1
hasTimeSet()		100%		100%	0	2	0	1	0	1
setXmlElement(Element)		100%		n/a	0	1	0	2	0	1
setStartHour(int)		100%		n/a	0	1	0	2	0	1
setStartMinute(int)		100%		n/a	0	1	0	2	0	1
setStartDay(int)		100%		n/a	0	1	0	2	0	1
setStartMonth(int)		100%		n/a	0	1	0	2	0	1
setStartYear(int)		100%		n/a	0	1	0	2	0	1
setRecurBy(int)		100%		n/a	0	1	0	2	0	1
setRecurIncrement(int)		100%		n/a	0	1	0	2	0	1
setRecurNumber(int)		100%		n/a	0	1	0	2	0	1
getXmlElement()		100%		n/a	0	1	0	1	0	1
getStartHour()		100%		n/a	0	1	0	1	0	1
getStartMinute()		100%		n/a	0	1	0	1	0	1
getStartDay()		100%		n/a	0	1	0	1	0	1
getStartMonth()		100%		n/a	0	1	0	1	0	1
getStartYear()		100%		n/a	0	1	0	1	0	1
getTitle()		100%		n/a	0	1	0	1	0	1
getDescription()		100%		n/a	0	1	0	1	0	1
getEmailAddress()		100%		n/a	0	1	0	1	0	1
getRecurNumber()		100%		n/a	0	1	0	1	0	1
getRecurBy()		100%		n/a	0	1	0	1	0	1
getRecurDays()		100%		n/a	0	1	0	1	0	1
getRecurIncrement()		100%		n/a	0	1	0	1	0	1
getValid()		100%		n/a	0	1	0	1	0	1
isOn(int, int, int)		100%		n/a	0	1	0	1	0	1
isRecurring()		100%		n/a	0	1	0	1	0	1
Total	11 of 349	96%	7 of 36	80%	6	54	2	98	0	36

CalDay

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString()		80%		75%	1	3	2	11	0	1
getFullInfomrationApp(Object)		100%		100%	0	6	0	26	0	1
CalDay(GregorianCalendar)		100%		n/a	0	1	0	10	0	1
addAppt(Appt)		100%		83%	1	4	0	8	0	1
setAppts(LinkedList)		100%		50%	3	4	0	5	0	1
iterator()		100%		100%	0	2	0	3	0	1
CalDay()		100%		n/a	0	1	0	3	0	1
setDay(int)		100%		n/a	0	1	0	2	0	1
setMonth(int)		100%		n/a	0	1	0	2	0	1
setYear(int)		100%		n/a	0	1	0	2	0	1
getSizeAppts()		100%		n/a	0	1	0	1	0	1
isValid()		100%		n/a	0	1	0	1	0	1
getAppts()		100%		n/a	0	1	0	1	0	1
getDay()		100%		n/a	0	1	0	1	0	1
getMonth()		100%		n/a	0	1	0	1	0	1
getYear()		100%		n/a	0	1	0	1	0	1
Total	14 of 348	95%	5 of 28	82%	5	30	2	78	0	16

Created with JaCoCo 0.8.0.201801022044

DataHandler

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getApptRange(GregorianCalendar, GregorianCalendar)		73%		62%	13	21	21	83	0	1
getNextApptOccurrence(Appt, GregorianCalendar)		0%		0%	9	9	20	20	1	1
getApptOccurences(Appt, GregorianCalendar, GregorianCalendar)		43%		16%	6	7	10	17	0	1
saveAppt(Appt)		92%		50%	3	4	3	69	0	1
DataHandler(String, boolean)		86%		100%	0	2	3	33	0	1
save()		82%		n/a	0	1	4	12	0	1
deleteAppt(Appt)		0%		n/a	1	1	1	1	1	1
DataHandler()		100%		n/a	0	1	0	3	0	1
static {...}		100%		n/a	0	1	0	2	0	1
DataHandler(String)		100%		n/a	0	1	0	2	0	1
setDocument(Document)		100%		n/a	0	1	0	2	0	1
setFileName(String)		100%		n/a	0	1	0	2	0	1
isAutoSave()		100%		n/a	0	1	0	1	0	1
isValid()		100%		n/a	0	1	0	1	0	1
getDocument()		100%		n/a	0	1	0	1	0	1
getFileName()		100%		n/a	0	1	0	1	0	1
Total	240 of 936	74%	42 of 74	43%	32	54	62	250	2	16

Created with JaCoCo 0.8.0.201801022044

1.2. Explanation

The only class that I have less than 80% coverage (74% of Instructions and 43% of branches) is the DataHandler class in which the class is too “close out” for access from other classes (including my test class) to set up the appropriate context for testing. Thus my test class was only able to set up some limited testing context. This is the reason that causes my test class to miss branches as in some context the execution flow read the return statement very soon and goes out of the class, which make the rest code of the class untested.

Further more, there was a bug in my test code that I couldn’t resolve. Specifically, the DataHandler object always loads 1194 instances of Appointment while the real execution (run without test with java -c command) does not behave this way. This make the condition checking if-statement in some class return early and make the rest of the code untested.

There are also exception catching that will never happen because some configuration was preset by code, not by user’s input parameters.

1.3. Commentary

That being said, besides my DataHandler class, I'm very proud of the fact that all of my other class has very high code coverage: Appt class has 96% instruction coverage and 80% branches coverage. CalDay class has 95% instruction coverage and 82% branches coverage.

Below are some code that my tests don't cover

```
        if (!occurrenceDay.before(lastDay)) {
            return result;
        }

        //Make sure that there is a limited number of recurrences
        for (int i = 0; i < appt.getRecurNumber()+1; i++) {

            //Add the day of occurrence to the list if it is after the first day
            if (!occurrenceDay.before(firstDay)) {
                result.add(occurrenceDay);
            }

            //Calculate the next recurrence day
            occurrenceDay = getNextApptOccurrence(appt, occurrenceDay);
            if (occurrenceDay == null) {
                break;
            }

            //Keep cycling while the occurence day is in range
            if (!occurrenceDay.before(lastDay)) {
                break;
            }
        }

        return result;
    }

    /**
     * Calculates the next recurring day in the given appointment. If the
     * appointment does not recur it returns null. If the date cannot be
     * calculated for some reason it returns null.
     **/
    private static GregorianCalendar getNextApptOccurrence(Appt appt,
            GregorianCalendar day) {
        //If the appointment does not recur then return null
        if (!appt.isRecurring()) {
            return null;
        }
        // Always indicate "does not recur" or "cannot be calculated"
        // if (true) {
        //     return null;
        // }

        //Leave the original day untouched.
        GregorianCalendar nextDay = (GregorianCalendar)day.clone();

        //This depends on the recurrence settings
        switch (appt.getRecurBy()) {
            case Appt.RECUR_BY_WEEKLY:
                int[] recurDays = appt.getRecurDays();

                //If the user specified weekly recurrence and didn't specify
                //which week days, then assume it is the same week day of the
                //first occurrence
                if (recurDays.length == 0) {
                    //Add 7 days and return that by default
                    nextDay.add(nextDay.DAY_OF_MONTH, 7);
                    return nextDay;
                }

                //The user did specify weekly recurrence, so increment the
                //day until it falls on a weekday the user specified
                for (int k = 0; k < 7; k++) {
                    nextDay.add(nextDay.DAY_OF_MONTH, 1);
                    int newDayOfWeek = nextDay.get(nextDay.DAY_OF_WEEK);

                    for (int i = 0; i < recurDays.length; i++) {
                        //If the calendar is set to a day of the week that the
                        //appt recurs on then return that day.
```

My test was not able to cover the method getNextApptOccurence because there is an if-statement that check a certain condition and if true, return out of the method which would call the getNextApptOccurence method otherwise. I couldn't figure out the bug in which the said condition is always true while it doesn't in normal run (run without test).

1.4. Advantage of using JaCoCo

Section 1.3 also shows a clear "thump up" of using JaCoCo – I'm able to see the very details statistics of code coverage, and the best thing is I'm able to see which part of the code my test suits weren't able to cover. It is also able to exclude private methods which makes its statistics more accurate and better reflect the code coverage.

2. Unit Testing Effort

This is the first time I use code coverage in my code and it was very interesting honestly. To be able to instantly see visual feedback of my work and see myself making progress is encouraging and rewarding. With code coverage, I was able to test faster than I think I could be because I know where to add test, where to fix my test case, etc.

I was able to find most of the bugs that I made in assignment 1. The details of how I found the bugs are described in the table below:

Bug #	Method Name	Line #	What I've changed	Incorrect result
1	Appt.setValid	180	change || to &&	The app accepts invalid appointment start day because the check condition will always be evaluated to false since no day is both < 1 and > NumDaysInMonth
	Test case: ApptTest.testConstructor1() How: the apptInvalidMonth should make the toString method return an error text rather than an actual toString() data from the object. Because the Appt.setValid always return true for failed startDay variable, this test failed			

	@Test(timeout = 4000)
	public void testConstructor1() throws Throwable {
	Appt apptValidData = new Appt(15, 30, 9, 12, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	String apptString1 = apptValidData.toString();
	assertEquals("\t12/9/2018 at 3:30pm ,Birthday Party, This is my birthday party\n", apptString1);
	
	Appt apptHourZero = new Appt(0, 30, 9, 12, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	String apptString2 = apptHourZero.toString();
	assertEquals("\t12/9/2018 at 12:30am ,Birthday Party, This is my birthday party\n", apptString2);
	
	Appt apptInvalidMonth = new Appt(25, 30, 9, 14, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	String apptString3 = apptInvalidMonth.toString();
	assertEquals("\tThis appointment is not valid", apptString3);
	}			

2	DataHandler.getNextApptOccurence	334-336	add an "if trap" that make the function always return **null**	I cannot make the app display the next appointment of a recurrent appointment. It return error.
	Unable to find since my tests were able to cover all this method			
3	Appt.isOn	288	always return true	the Calendar app always output no appointment at every day on the calendar
	Test case: ApptTest.testIsOn How: These test failed because the test create a new appoinment which a certain date and then assert for a different date but the isOn method still return true.			

	@Test(timeout = 4000)
	public void testIsOn2() throws Throwable {
	Appt appt = new Appt(15, 30, 9, 12, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	assertFalse(appt.isOn(2, 2, 2020));			

	}			
4	Appt.getStartDay	249	change return value to -1	the calendar app always return -1 for day values.
	Test case: ApptTest.Constructor1 How: The toString method return different string than the expected string in which the startDay is always -1.			

	@Test(timeout = 4000)
	public void testConstructor1() throws Throwable {
	Appt apptValidData = new Appt(15, 30, 9, 12, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	String apptString1 = apptValidData.toString();
	assertEquals("\t12/9/2018 at 3:30pm ,Birthday Party, This is my birthday party\n", apptString1);
	
	Appt apptHourZero = new Appt(0, 30, 9, 12, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	String apptString2 = apptHourZero.toString();
	assertEquals("\t12/9/2018 at 12:30am ,Birthday Party, This is my birthday party\n", apptString2);
	
	Appt apptInvalidMonth = new Appt(25, 30, 9, 14, 2018, "Birthday Party", "This is my birthday party", "xyz@gmail.com");
	String apptString3 = apptInvalidMonth.toString();
	assertEquals("\tThis appointment is not valid", apptString3);
	}			

5	CalDay.CallDay()	67	change valid = true to valid = false	The new calendar day is always “invalid”. So the Calendar App throw exception at Runtime: The number of appointments between 04/13/2018 (inclusive) and 04/14/2018 (exclusive) Exception in thread "main" java.lang.NullPointerException at calendar.CalDay.getFullInfomrationApp(CalDay.java:214) at calendar.CalendarMain.main(CalendarMain.java:171)
	Test case: CalDayTest.testContructor2() How: Despite being constructed with valid data, the isValid() method always return false, which make the assert statement fail.			
	@Test(timeout = 4000) public void testConstructor2() throws Throwable {			

	GregorianCalendar gCal = new GregorianCalendar(2018, 4, 23);
	CalDay cal = new CalDay(gCal);
	assertEquals(2018, cal.getYear());
	assertEquals(4+1, cal.getMonth());
	assertEquals(23, cal.getDay());
	assertNotEquals(null, cal.getAppts());
	assertEquals(0, cal.getSizeAppts());
	assertEquals(true, cal.isValid());
	}

Note: I fixed bug #4 and bug #5 while testing to get the more code coverage.