CS 362 Software Engineering 2
Instructor: Dr. Ali Aburas
Student: Khuong Luu

1. Random Tester

1.1. The development of my random testers and how I improve coverage

I don't jump in to write test at first. Instead, I run the test and generate the report and look at the initial coverage to determine what need to be done. For each method that need to be covered, I look at what has been already covered and what not, then I address the lines and branches that has not been covered.

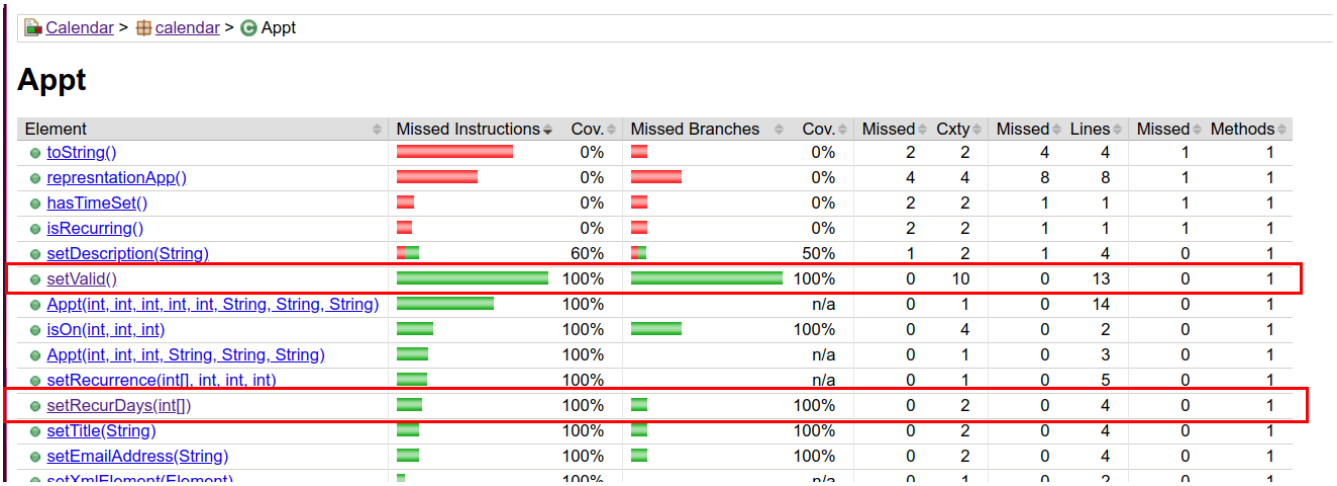
I improve the random testers to get the line coverage before I improve them (the random testers) to get the branch coverage.

When I look at the part where my random testers did not cover in the report, rather than blindly go fix the test, I try to determine the reason why it couldn't by going over the test code and run the code out in my head. There were sometime I had to get the test code print out data to "debug" the test code.

It's also helpful to look at how I test those methods in assignment 2, which is in a non-random way, to recall how this method is call and used. Then, instead of inputing hard-coded values as argument, I find a way to put randomly generated value to it.

Sometime such as in DataHandler.getApptRange method, the value of random argument has some dependencies with each other and so the test get a bit more complicated. There are 2 ways in this case: More random value to reach the valid range in which date1 is before date2. The other way is to make a custom random method to help generate the values that comply the dependencies we want.

1.2. Final coverage for each method


Calendar > calendar > Appt

Appt

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString()		0%		0%	2	2	4	4	1	1
represntationApp()		0%		0%	4	4	8	8	1	1
hasTimeSet()		0%		0%	2	2	1	1	1	1
isRecurring()		0%		0%	2	2	1	1	1	1
setDescription(String)		60%		50%	1	2	1	4	0	1
setValid()		100%		100%	0	10	0	13	0	1
Appt(int, int, int, int, int, String, String, String)		100%		n/a	0	1	0	14	0	1
isOn(int, int, int)		100%		100%	0	4	0	2	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
setEmailAddress(String)		100%		100%	0	2	0	4	0	1
setXmlElement(Element)		100%		n/a	0	1	0	2	0	1

Calendar > calendar > CalDay

CalDay

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getFullInfomrationApp(Object)		0%		0%	6	6	23	23	1	1
toString()		0%		0%	3	3	12	12	1	1
iterator()		0%		0%	2	2	3	3	1	1
CalDay()		0%		n/a	1	1	3	3	1	1
getSizeAppts()		0%		n/a	1	1	1	1	1	1
isValid()		0%		n/a	1	1	1	1	1	1
getDay()		0%		n/a	1	1	1	1	1	1
getMonth()		0%		n/a	1	1	1	1	1	1
getYear()		0%		n/a	1	1	1	1	1	1
CalDay(GregorianCalendar)		100%		n/a	0	1	0	10	0	1
addAppt(Appt)		100%		100%	0	4	0	8	0	1
setAppts(LinkedList)		100%		50%	3	4	0	5	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
getAppts()		100%		n/a	0	1	0	1	0	1
Total	254 of 356	28%	19 of 28	32%	20	30	46	76	9	16

Calendar > calendar > DataHandler

DataHandler

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getApptRange(GregorianCalendar, GregorianCalendar)		71%		62%	13	21	22	84	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
DataHandler(String, boolean)		85%		100%	0	2	3	32	0	1
save()		82%		n/a	0	1	4	12	0	1
deleteAppt(Appt)		93%		83%	1	4	1	11	0	1
saveAppt(Appt)		100%		100%	0	4	0	69	0	1
DataHandler()		100%		n/a	0	1	0	3	0	1
DataHandler(String)		100%		n/a	0	1	0	2	0	1
static {...}		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 985	75%	40 of 80	50%	29	57	60	260	1	16

1.3. Part that I failed to cover and why

```
501.
502.     /**
503.      * Deletes the appointment's information from the XML data tree. Does not
504.      * write a new XML file to disk.
505.      * @return True if the appointment is deleted successfully.
506.      **/
507.     public boolean deleteAppt(Appt appt) {
508.         //Do not do anything to invalid appointments
509.         if (!appt.getValid()) {
510.             return false;
511.         }
512.
513.         //Remove the appointment from the XML tree if applicable
514.         Element apptElement = appt.getXmlElement();
515.         if (apptElement == null) {
516.             return false;
517.         }
518.         Node parentNode = apptElement.getParentNode();
519.         parentNode.removeChild(apptElement);
520.         appt.setXmlElement(null);
521.
522.         if (isAutoSave()) {
523.             return save();
524.         }
525.         else {
526.             return true;
527.         }
528.     }
529.
```

Even though as you can see at line 494-499 that I can cover the isAutoSave(), I cannot explain why my test cannot cover the same branch in deleteAppt method.

```
127.      **/
128.     public List<CalDay> getApptRange(GregorianCalendar firstDay,
129.             GregorianCalendar lastDay) throws DateOutOfRangeException {
130.
131.             //Internal Diagnositic Messages turned on when true
132.             boolean diagnose = false;
133.
134.             //If the data handler isn't initialized return null
135.             if (isValid() == false) {
136.                 return null;
137.             }
138.
139.             //Make sure that the first day is before the last day
140.             if (!firstDay.before(lastDay)) {
141.                 throw new DateOutOfRangeException("Second date specified is not " +
142.                     "before the first date specified.");
143.             }
144.
145.             //Create a linked list of calendar days to return
146.             LinkedList<CalDay> calDays = new LinkedList<CalDay>();
147.
148.             //Create the first CalDay object with the starting date and add to list
149.             GregorianCalendar nextDay = (GregorianCalendar) firstDay.clone();
150.             while (nextDay.before(lastDay)) {
151.                 calDays.add(new CalDay(nextDay));
152.                 nextDay.add(nextDay.DAY_OF_MONTH, 1);
153.             }
154.
155.             if (diagnose) {
156.                 System.out.println("======================================");
157.                 System.out.println("DEBUGGING GETTING OF APPOINTMENTS      ");
158.             }
159.
160.             //Retrieve the root node - <calendar>
161.             Document doc = getDocument();
162.             Element root = doc.getDocumentElement();
163.
164.             if (diagnose) {
165.                 System.out.println("Root node: " + root.getTagName());
166.                 System.out.println("All following nodes should be appt nodes.");
167.             }
168.
169.             //Retrieve the root's children - <appt> nodes
170.             NodeList appts = root.getChildNodes();
171.             for (int i = 0; i < appts.getLength(); i++) {
172.                 Element currentAppt = (Element) appts.item(i);
173.
174.                 if (diagnose) {
175.                     System.out.println("Nodes under the root: " +
176.                         currentAppt.getTagName());
177.                 }
178.
179.                 //For this appointment, get the values of all fields
180.                 NodeList fieldNodes = currentAppt.getChildNodes();
181.                 Hashtable<String, String> fields = new Hashtable<String, String>();
182.                 if (diagnose) {
183.                     System.out.println("Preparing to read each field for the appt");
184.                 }
185.                 for (int j = 0; j < fieldNodes.getLength(); j++) {
186.                     Element currentField = (Element) fieldNodes.item(j);
187.                     String fieldName = currentField.getTagName();
188.                     if (diagnose) {
189.                         System.out.println("Reading field: " + fieldName);
190.                     }
191.                     String fieldValue = "";
192.                     NodeList fieldValueNodes = currentField.getChildNodes();
193.                     for (int k = 0; k < fieldValueNodes.getLength(); k++) {
194.                         Text text = (Text)fieldValueNodes.item(k);
195.                         fieldValue += text.getData();
196.                     }
197.                     if (diagnose) {
198.                         System.out.println("Reading field's value: " + fieldValue);
199.                     }
200.
201.                     fields.put(fieldName, fieldValue);
```

I failed to cover every “false” branch of if(diagnose) statement because it’s impossible for my test to cover these branch because it cannot modify the value of “diagnose” variable since it is internally set to false, to which the my test don’t have any control. In other words, **it is impossible to cover 100% branch of this method.**

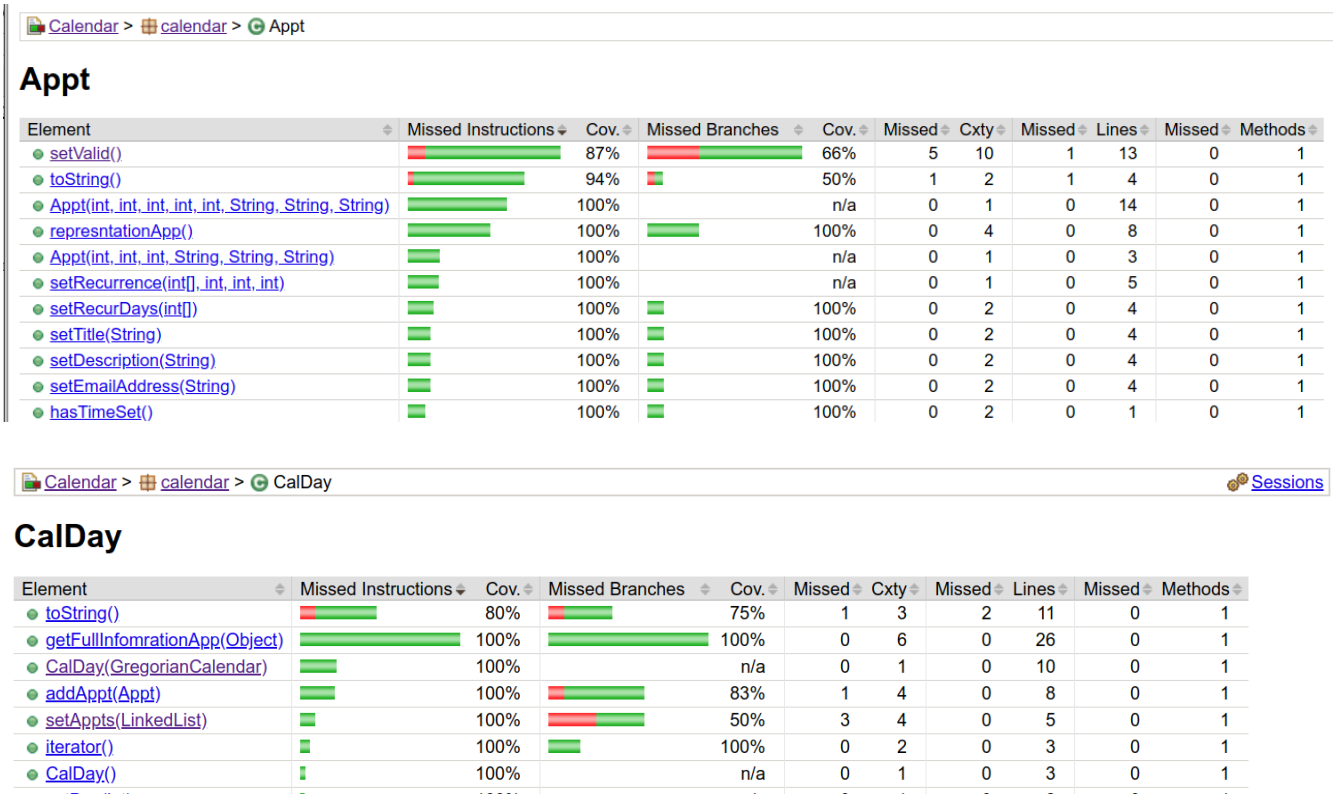
2. Unit vs. Random

2.1. **Between your random tests and your unit tests, how the tests differ in ability to detect faults**

Random test is a short and easy way to reach coverage but slower and take more time and computing resources to run. However, if there are not sufficient test trial (NUM_TEST and eslapsed time in this program), random test is worse than unit test. So, in short, random test is limited by time and computing resource.

In the other hands, the ability to detect fault of unit test is limited by the ability of the tester, or in other words, limited by the people who write test (the one who program the unit tests). So if the tester is not good, so is the test. If the tester is good, she can cover all edge case and special cases, then the unit test is very good. Another weakness of unit test is that if the number of combination is too large, then unit test cannot cover all of them.

2.2. **My test coverage from assignment 2:**



Calendar > calendar > Appt

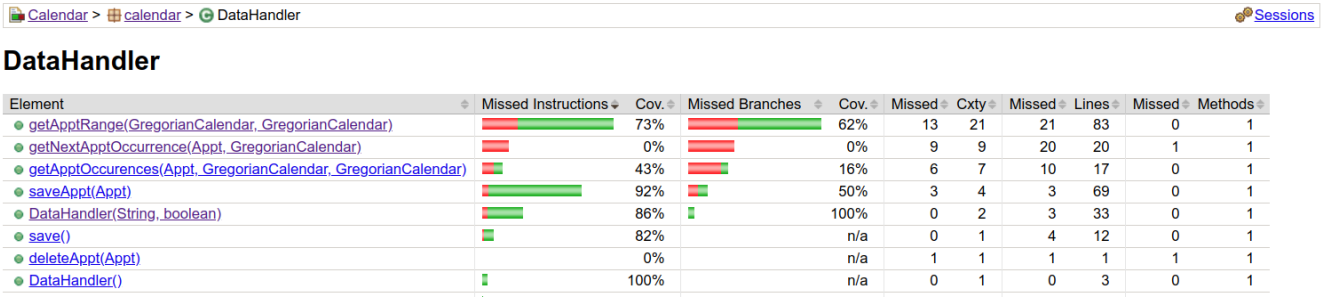
Appt

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

Calendar > calendar > CalDay Sessions

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



2.3. **Which tests had higher coverage – unit or random?**

From the screenshot in section 1.2. and section 2.2 above, it can be concluded that in this program. random tests had higher coverage.

2.4. **Which tests had better fault detection capability?**

Random test had better fault detection capability because with random tests can cover branches more efficient and more thorough than unit test which we generate value manually and may miss the edge case or there may be rare combination that I have overlooked – Evidently, for this program specifically, it's in Appt.setValid() method and CalDay.addAppt() method in which my unit tests in assignment 2 overlooked some special combination and edge cases while my random tests in this assignment were able to cover them.