SOFTWARE ENGINEERING 2 - 2ND THEORETICAL EXERCISE

<u>Exercise 1-Done Raul Moya Campillos and Maja Anna Swierk</u>

1. Write, at least the pseudocode of the identified method.

We didn't develop the pseudocode because to perform the testing it is necessary to implement the class in Java code.

2. Variables that must be considered to test the program:

In this case we only have 3 variables that are the day, month, and year of the date that we are testing.

3. Identify the test values for each one of the variables previously identified, specifying the technique used to obtain each of those values:

For this exercise we decided that we must test 2 methods to develop the problem in a correct way. This is because we check also if the day is valid depending on the month because there are months with different number of days and affects the equivalence classes of the variables.

Moreover, we must consider for the tests that we through an exception when the input value of a variable is a negative value or a value out of the range.

The technique used to establish the testing values is the equivalence classes, with this we can divide the possible inputs in disjoin sets:

- DayWithinMonth():boolean

Day: int   6 equivalence classes

```
(            ] (                           ) [    ) [    ) [    ) (                           )
-∞           0                             28     29     30     31                            ∞
|------------|-----------------------------|------|------|------|-----------------------------|
```

Testing values: {-98, 15, 28, 29, 30, 31, 200}

- Month:int

We can consider these equivalence classes:

1) Equivalence class: 'negative' (-∞, 0)
2) Equivalence class: '30 days' {1,3,5,7,8,10,12}
3) Equivalence class: '31 days' {4,6,9,11}
4) Equivalence class: 'February' {2}

Testing values: {-8, 5, 6, 2}

- IsLeap(): Boolean

We assign these equivalence classes for the variable Year:

1) Equivalence class: ‘negative’ (-∞, 0)
2) Equivalence class: ‘four_only’ {years that are divisible by 4 but not by 100}
3) Equivalence class: ‘four_and_hundred_and_four_hundred ‘{years that are divisible by 4, by 100 and by 400}
4) Equivalence class: ‘not_by_four’ {years that are not divisible by 4}
5) Equivalence class: ‘yes_four_and_hundred_not_four_hundred’ {years that are divisible by 4 and by 100, but not by 400}

Testing values: {-1712, 1992, 1600, 2021, 300}

4. Calculate the maximum possible number of test cases that could be generated from the test values.

As for the ‘Day’ variable we’ve chosen 7 values, for the ‘Month’ 4 values and for the ‘Year’ 5, the maximum possible number of the test cases is 7*4*5, so 140 test cases.

5. Define some test suites using each use

Considering the testing values, we’ve chosen:

- Day = {-98, 15, 28, 29, 30, 31, 200}
- Month = {-8, 5, 6, 2}
- Year = {-1712, 1992, 1600, 2021, 300}

We will create the following test suite:

- Test suite = {(-98, -8, -1712), (15,5,1992), (28,6,1600), (29,2,1600), (30,5,2021), (31,5,1992), (200,6,300)}

6. Define test suits to achieve pairwise coverage by using the proposed algorithm in Lectures. You can check the results by means of the software PICT1

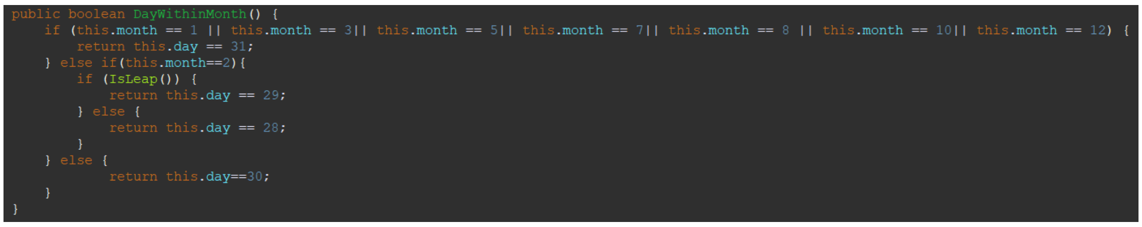
Using the tool Pairwise Pict Online we can generate this test suite:

Day	Month	Year
15	6	1992
28	2	1600
29	-8	-1712
-98	-8	2021
31	2	300
200	2	1992

30	6	2021
30	-8	300
31	-8	1992
30	5	1992
28	6	300
200	5	-1712
200	5	300
29	5	1600
-98	5	300
200	5	2021
28	-8	2021
15	2	-1712
29	2	1992
28	5	-1712
29	6	300
-98	6	1600
28	2	1992
-98	2	1992
15	5	300
30	-8	1600
200	-8	1600
-98	6	-1712
15	2	2021
31	6	1600
31	5	2021
30	2	-1712
15	-8	1600
31	5	-1712
200	6	1600
29	5	2021

7. For code snippets that include decisions, propose a set of test cases to achieve coverage of decisions.

- For the DayWithinMonth() method:



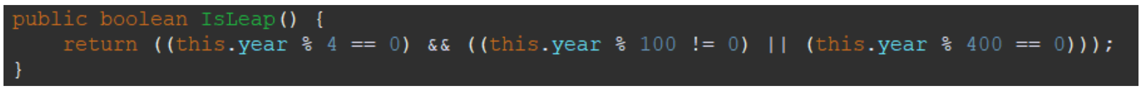
```java
public boolean DayWithinMonth() {
    if (this.month == 1 || this.month == 3|| this.month == 5|| this.month == 7|| this.month == 8 || this.month == 10|| this.month == 12) {
        return this.day == 31;
    } else if(this.month==2){
        if (IsLeap()) {
            return this.day == 29;
        } else {
            return this.day == 28;
        }
    } else {
            return this.day==30;
    }
}
```

The possible test case set is as follows:

Day	Month	Year
12	1	2021

4	2	1900
29	5	1783
21	12	1600
1	4	1410

- For the IsLeap() method:



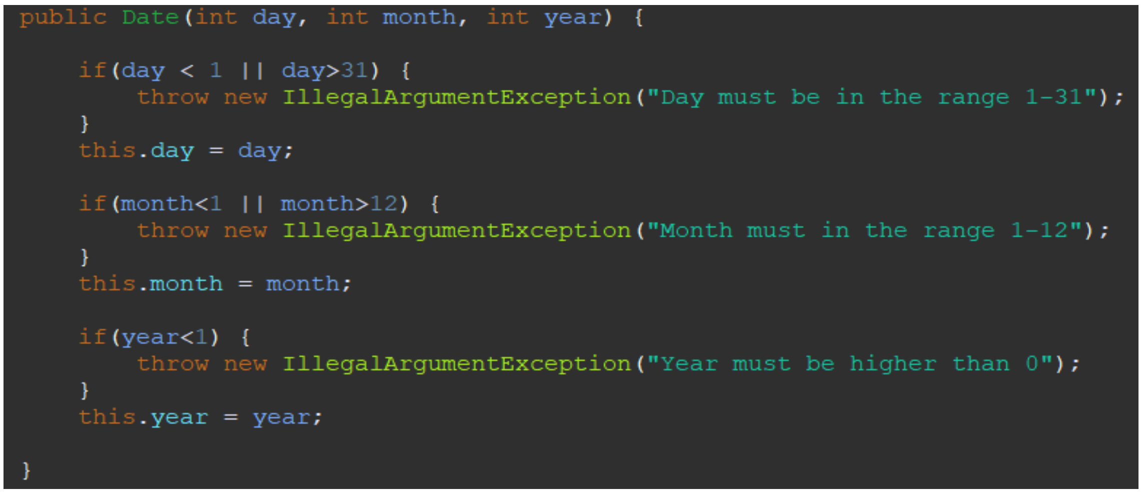
```
public boolean IsLeap() {
    return ((this.year % 4 == 0) && ((this.year % 100 != 0) || (this.year % 400 == 0)));
}
```

we could generate the following test case set:

Day	Month	Year
1	12	1984
23	3	200

8. For code snippets that include decisions, propose test case sets to achieve MC/DC coverage

- For the constructor of class Date():

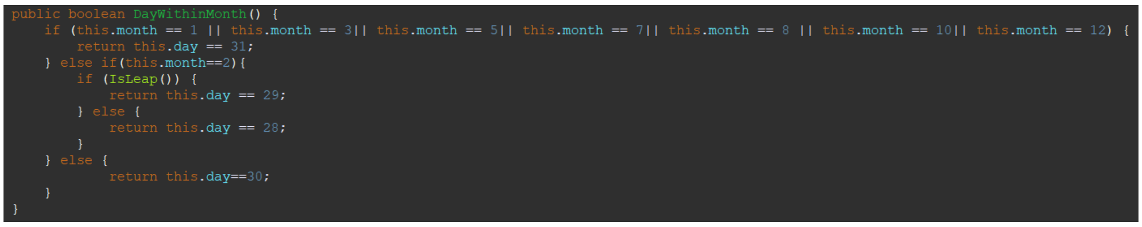


```
public Date(int day, int month, int year) {

    if(day < 1 || day>31) {
        throw new IllegalArgumentException("Day must be in the range 1-31");
    }
    this.day = day;

    if(month<1 || month>12) {
        throw new IllegalArgumentException("Month must in the range 1-12");
    }
    this.month = month;

    if(year<1) {
        throw new IllegalArgumentException("Year must be higher than 0");
    }
    this.year = year;

}
```

We generate following test case set:

Day	Month	Year
0	12	1000
32	3	879
13	0	456
26	13	2000
1	7	0
31	8	-170
11	1	2023

- For the DayWithinMonth() method:

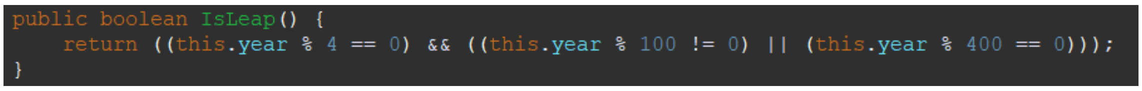


```java
public boolean DayWithinMonth() {
    if (this.month == 1 || this.month == 3|| this.month == 5|| this.month == 7|| this.month == 8 || this.month == 10|| this.month == 12) {
        return this.day == 31;
    } else if(this.month==2){
        if (IsLeap()) {
            return this.day == 29;
        } else {
            return this.day == 28;
        }
    } else {
            return this.day==30;
    }
}
```

The possible test case set is as follows:

Day	Month	Year
31	4	2000
29	2	1005
5	13	1987
16	10	2028

- For the IsLeap() method:



```java
public boolean IsLeap() {
    return ((this.year % 4 == 0) && ((this.year % 100 != 0) || (this.year % 400 == 0)));
}
```

we could generate the following test case set:

Day	Month	Year
1	12	-190
23	3	0
30	8	2022

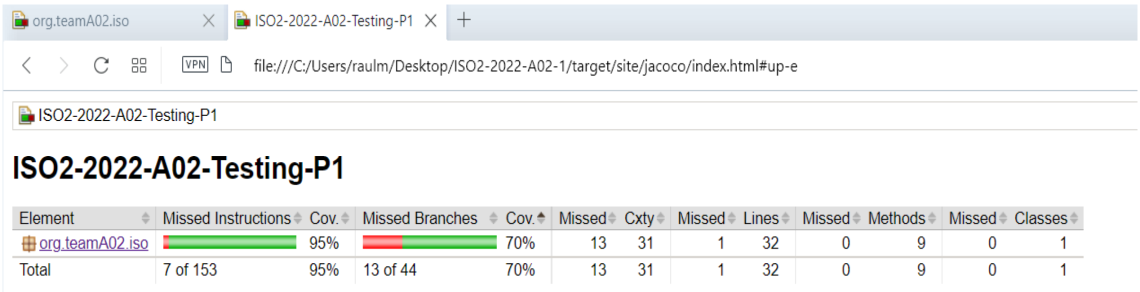
9. Comment on the results of the number of test cases obtained in section 4, 5, and 6, as well as the execution of the oracles: what could be said about the coverage achieved?

Number of test cases obtained in section 4 (140 test cases) is too big to be implemented for this theoretical exercise. We believe that we can perform decent testing, while using smaller test case set.

In section number 6 we got 38 test cases, which is much better result than the previous one. It could be worth implementing all of them, but for the sake of this exercise, we decided to implement smaller number of test cases from the section 5 – a test suite consisting of 7 test cases.

In the second test method we take 13 test cases from the section 6, to do the test with different test cases than in the previous method and using the technique pairwise.

Coverage obtained by testing:

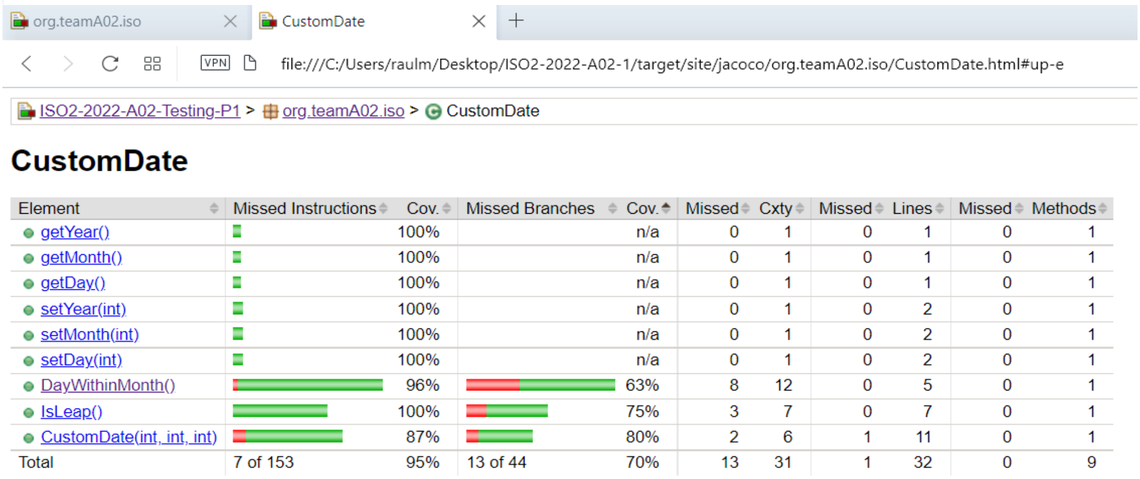

org.teamA02.iso | ISO2-2022-A02-Testing-P1

file:///C:/Users/raulm/Desktop/ISO2-2022-A02-1/target/site/jacoco/index.html#up-e

ISO2-2022-A02-Testing-P1

ISO2-2022-A02-Testing-P1

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
org.teamA02.iso		95%		70%	13	31	1	32	0	9	0	1
Total	7 of 153	95%	13 of 44	70%	13	31	1	32	0	9	0	1


org.teamA02.iso | CustomDate

file:///C:/Users/raulm/Desktop/ISO2-2022-A02-1/target/site/jacoco/org.teamA02.iso/CustomDate.html#up-e

ISO2-2022-A02-Testing-P1 > org.teamA02.iso > CustomDate

CustomDate

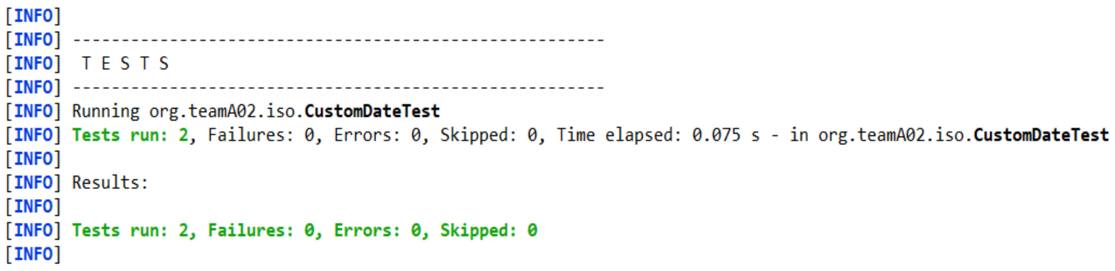
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
getYear()		100%		n/a	0	1	0	1	0	1
getMonth()		100%		n/a	0	1	0	1	0	1
getDay()		100%		n/a	0	1	0	1	0	1
setYear(int)		100%		n/a	0	1	0	2	0	1
setMonth(int)		100%		n/a	0	1	0	2	0	1
setDay(int)		100%		n/a	0	1	0	2	0	1
DayWithinMonth()		96%		63%	8	12	0	5	0	1
IsLeap()		100%		75%	3	7	0	7	0	1
CustomDate(int, int, int)		87%		80%	2	6	1	11	0	1
Total	7 of 153	95%	13 of 44	70%	13	31	1	32	0	9

The coverage is the degree in which certain parts of the system have been subject under test. We didn't test the getters and setters because in the constructor of the Object Date we check that the values introduced for the variable's day, month and year are in the range expected.

With the two important methods for the we achieve a good coverage considering the missed instructions.

In the other hand, considering the missed branches we would get a better coverage result, but the number of test cases implemented was small and it is possible that we didn't reach a specific condition in the method. Also, we can assume that we won't find more errors using other testing values.

We think that with the number of test cases that we have we achieve to maximize the coverage. If we add all the test cases generated by the pairwise strategy the coverage will be better.


```
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running org.teamA02.iso.CustomDateTest
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.075 s - in org.teamA02.iso.CustomDateTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
```