| Marks distribution for testing exercise 1 | | | TOTAL |
|---|---|---|---|
| Hiba | Alicia & Álvaro | Lucía & Irem | |
| 3.33 | 3.33 | 3.33 | 10 |

**1) Write at least the pseudocode corresponding to the identified method or method**

The code its already create in its repository, in this case it was implemented by the pair of Alicia and Alvaro

**2) Identify the test units.**

Each method will be considered a test unit. So the test units for this problem would be :

For the class MyDate:

- MyDate(),
- daysInMonth()
- isLeapYear ()

**3) Identify the variables that must be considered to test the relevant methods included in the test units.**

For MyDate(): Three integers (day, month, year).

For daysInMonth(): Two integers (month, year).

For isLeapYear(): One integer (year).

**4) Identify the test values for each of the above variables, which will constitute the corresponding test cases using the three techniques discussed in theory (equivalence classes, boundary value analysis, and error guessing), specifying for each value obtained which technique has been used.**

**MyDate(int day, int month, int year):**

| Variable | Partitions | Limit | Guess | Final | explanation |
|---|---|---|---|---|---|
| year | -5 | -1,0,1 | MAXINT | 0,1,2024 | Some values are not included in the final column because they are being controlled in the command line interface, so they will never appear in our code. |
| | 2024 | | MININT | | |
| | | | ¨hello¨ | | |
| day | -2 | 0,1,2 | “hello” | 0,1,2,15,27,28,29,30,31,33 | |
| | 15 | 27,28,29 | | | |
| | 29 | 28,29,30 | | | |
| | 30 | 29,30,31 | | | |

| | | | | | |
|---|---|---|---|---|---|
| | 31 | 30,31,32 | | | |
| | 33 | | | | |
| month | -3 | 0,1,2 | “hello” | 0,1,2,5,11,12,13,14 | |
| | 5 | 11,12,13 | | | |
| | 14 | | | | |
| | | | | | |

**daysInMonth(int month, int year)**

| Variable | Partitions | Limit | Guess | Final | explanation |
|---|---|---|---|---|---|
| month | -3 | 0,1,2 | “hello” | 0,1,2,5,11,12,13,14 | Some values are not included in the final column because they are being controlled in the command line interface, so they will never appear in our code. |
| | 5 | 11,12,13 | | | |
| | 14 | | | | |
| year | -5 | -1,0,1 | MAXINT | 0,1,2024 | |
| | 2024 | | MININT | | |

**isLeapYear(int y)**

| Variable | Partitions | Limit | Guess | Final | Explanation |
|---|---|---|---|---|---|
| y | -5 | -1,0,1 | MAXINT | 0,1,2024 | Some values are not included in the final column because they are being controlled in the command line interface, so they will never appear in our code. |
| | 2024 | | MININT | | |
| | | | ¨hello¨ | | |

**5) Calculate the maximum possible number of test cases that could be generated from the test values (combinatorial).**

MyDate(int day, int month, int year): 10 * 8 * 3 = 240

daysInMonth(int month, int year): 8 * 3 = 24

isLeapYear(int y): 3

**6) Define a set of test cases to fulfill each use (each value at least once).**

To guarantee that every test value is used at least once, the minimum number of test cases for each method is the maximum number of values among its variables:

**MyDate()**

| case | year | month | day | IsLeapYear |
|---|---|---|---|---|
| 1 | 2024 | 2 | 29 | True |

| | | | | |
|---|---|---|---|---|
| 2 | 1 | 5 | 15 | False |
| 3 | 0 | 0 | 0 | invalidDateException |
| 4 | 1 | 1 | 1 | False |
| 5 | 1 | 11 | 27 | False |
| 6 | 2024 | 2 | 28 | True |
| 7 | 1 | 13 | 30 | invalidDateException |
| 8 | 2024 | 14 | 31 | invalidDateException |
| 9 | 1 | 5 | 33 | invalidDateException |
| 10 | 2024 | 5 | 2 | true |

**7) Define test sets to achieve pairwise coverage using the algorithm explained in class. The results can be verified using the PICT program5 .**

**MyDate():**

| case | year | month | day | IsLeapYear |
|---|---|---|---|---|
| 1 | 2024 | 2 | 29 | True |
| 2 | 1 | 2 | 29 | invalidDateException |
| 3 | 1 | 4 | 31 | invalidDateException |
| 4 | 2024 | 4 | 30 | True |
| 5 | 1 | 1 | 31 | False |
| 6 | 0 | 5 | 15 | invalidDateException |
| 7 | 1 | 13 | 15 | invalidDateException |
| 8 | 1 | 5 | 33 | invalidDateException |
| 9 | 1 | 2 | 28 | false |
| 10 | 2024 | 5 | 2 | true |

**8) For code sections involving decisions, propose a set of test cases to achieve decision coverage.**

| case | year | month | day | IsLeapYear |
|---|---|---|---|---|
| 1 | 2004 | 2 | 29 | True |
| 2 | 1 | 2 | 29 | true |
| 3 | 2023 | 3 | 29 | false |
| 4 | 2010 | 3 | 28 | False |
| 5 | 1996 | 5 | 31 | True |
| 6 | 2023 | 4 | 30 | false |
| 7 | 2000 | 1 | 31 | true |
| 8 | 1700 | 6 | 26 | false |
| | | | | |

**9) For code sections involving decisions, propose a set of test cases to achieve MC/DC coverage.**

| case | year | A (%400) | B(%4) | C(!=%100) | Expression | Result |
|---|---|---|---|---|---|---|

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2000 | T | T | F | T | true |
| 2 | 1900 | F | T | F | F | False |
| 3 | 2016 | F | T | T | T | True |
| 4 | 2017 | F | F | T | F | False |

**10) Comment on the results regarding the number of test cases obtained in sections 4, 5, and 6: what can be said about the coverage achieved? To what extent does the implementation of the program influence the design and implementation of the test cases??**

In this problem its very easy to see the test cases and coverage. In the section 4 we acknowledge the main elements to test per method, which will make easier to test our program, for starters as we are checking dates we would have 366 dates that can be valid, and you can put any invalid date. But in the section 5 we calculate the maximum to check our program completely. Which even if its not an ostentatious number its still big.

So that's why the part 6 its so important, because we use a criterion to check everything with a much lesser number of tests.

The coverage its great, and the implementation of the code its very important because if you put everything in a method you will have a very big number of possible combinations. So the design and design will change completely and also the time invested in doing the testing part.

**The final test cases:**

the test cases to be implemented are the following:

| **Input Values** | **Expected result** |
|---|---|
| (1, 5, 15) | True |
| (0, 0, 0) | InvalidDateException is thrown |
| (1, 1, 1) | True |
| (1, 11, 27) | True |
| (2024, 2, 28) | True |
| (1, 13, 30) | InvalidDateException is thrown |
| (2024, 14, 31) | InvalidDateException is thrown |
| (2024, 2, 29) | True |

| | |
|---|---|
| (1, 2, 29) | InvalidDateException is thrown |
| (1, 4, 31) | InvalidDateException is thrown |
| (2024, 4, 30) | True |
| (1, 1, 31) | True |
| (0, 5, 15) | InvalidDateException is thrown |
| (1, 13, 15) | InvalidDateException is thrown |
| (1, 5, 33) | InvalidDateException is thrown |
| (1, 2, 28) | True |
| (2000, 2, 29) | True |
| (2023, 3, 29) | True |
| (2010, 3, 28) | True |
| (1996, 5, 31) | True |
| (2023, 4, 30) | True |
| (2000, 1, 31) | True |
| (2000, 1, 1) | True |
| (1900, 1, 1) | True |
| (2016, 1, 1) | True |
| (2017, 1, 1) | True |