Problem: Leap Year Determination & Date Validation
MEMBERS: AMIN AND FRAN

Goal:

The task consists of writing and testing a method that, given an object of type MyDate, returns whether its year is a leap year.

Invalid inputs (letters, negative numbers, invalid months or days) must throw specific exceptions.

The program must follow good encapsulation and separation of concerns.

Additionally:

A command-line user interface must be implemented to read values from the keyboard (integers, strings, doubles, dates).

Exceptions must be used to control incorrect input and invalid dates.

1) Pseudocode corresponding to the identified method or methods:

Pseudocode isLeapYear(int year):

```
isLeapYear(year : int) : boolean
  if year < 0:
    throw InvalidDateException("Year cannot be negative")

  if year mod 400 == 0:
    return true
  else if year mod 100 == 0:
    return false
  else if year mod 4 == 0:
    return true
  else:
    return false
```

Pseudocode — Full Date Constructor Validation:

```
MyDate(year : int, month : int, day : int)
  validateYear(year)
  validateMonth(month)
  validateDay(year, month, day)
  this.year = year
  this.month = month
  this.day = day
```

validateYear(year):

```
if year < 0:
  throw InvalidDateException("Year cannot be negative")
```

validateMonth(month):

```
if month < 1 or month > 12:
  throw InvalidDateException("Invalid month")
```

validateDay(year, month, day)

```
maxDay = daysInMonth(year, month)

if day < 1 or day > maxDay:
  throw InvalidDateException("Invalid day for this month/year")
```

daysInMonth(year, month)

```
if month == 2:
   if isLeapYear(year):
      return 29
   else:
      return 28
else if month in {4, 6, 9, 11}:
   return 30
else:
   return 31
```

Pseudocode isLeapYear(MyDate date)

```
isLeapYear(date : MyDate) : boolean
   year = date.getYear()
   return isLeapYear(year)
```

2) Identified Test Units

TU1 — MyDate constructor + validation
TU2 — LeapYearCalculator.isLeapYear(int)
TU3 — LeapYearCalculator.isLeapYear(MyDate)

3) Identification of Relevant Variables

1. Year (integer)

Used by both MyDate and LeapYearCalculator.
Relevant because it influences date validity and leap-year logic.
Ranges:

- Negative → invalid (exception in MyDate)
- Divisible by 400 → leap
- Divisible by 100 (not 400) → not leap
- Divisible by 4 (not 100) → leap
- Others → not leap

2. Month (integer)

Used only in MyDate validation.
Ranges:

- <1 → invalid
- 12 → invalid
- 1–12 → valid

3. Day (integer)

Depends on:

- month
- leap year

Ranges:

- <1 → invalid

- daysInMonth(year,month) → invalid
- 1–28/29/30/31 depending on the month

Special cases:

- February 29 depends on leap-year logic
- 30-day months: (4, 6, 9, 11)
- 31-day months: rest

4. MyDate object (composed variable)

Relevant for TU3.
Affects:

- valid date → isLeapYear(MyDate) runs normally
- invalid date → constructor throws exception before calling isLeapYear(int)

4)Identified Test Values

Year

Equivalence Class	Values
Negative (invalid)	-1
Zero boundary	0
Non-leap	1, 2023
Leap (divisible by 4)	2024
Not leap (divisible 100)	1900
Leap (divisible 400)	2000
TOTAL:6	

Month

Equivalence Class	Values
Invalid (<1)	0
Valid	1, 2, 4, 6, 12
Invalid (>12)	13
Total:3	

Day

Equivalence Class	Values
Invalid (<1)	0
Valid range	1, 28
Edge invalid values	29,30, 31
invalid (>31)	40
Total: 4	

5. Maximum Number of Possible Test Case Combinations

Approximation:

Each-Use

Max number of values of any variable:

➞ 6 cases

Pairwise

Max(values(year) × values(month)):

➞ 6 × 4= 24

Full Combinatorial (N-wise)

All combinations:

➞ 6× 3 × 4 = **72cases**

6. Each-Use Test Case Set

Case	Year	Month	Day	Expected Result
1	-1	1	1	Invalid year
2	0	2	1	Valid date (leap)
3	1	12	31	Valid date (non-leap)
4	2023	2	29	Invalid day
5	2024	2	29	Valid leap date (divisible 4)
6	1900	2	29	Invalid, not leap
7	1200	5	10	Leap (divisible 400)

7) Pairwise Test Case Set

Case	Year	Month	Day	Expected Result
TC1	-1	2	0	Invalid Date
TC2	-1	2	1	Invalid Date
TC3	-1	2	28	Invalid Date
TC4	-1	2	40	Invalid Date
TC5	0	2	0	Invalid Date
TC6	0	2	1	Valid Date
TC7	0	2	28	Valid Date
TC8	0	2	40	Invalid Date
TC9	1	2	0	Invalid Date
TC10	1	2	1	Invalid Date
TC11	1	2	28	Invalid Date
TC12	1	2	40	Invalid Date
TC13	2024	2	0	Invalid Date
TC14	2024	2	1	Valid Date
TC15	2024	2	28	Valid Date
TC16	2024	2	40	Invalid Date
TC17	1900	2	0	Invalid Date
TC18	1900	2	1	Invalid Date
TC19	1900	2	28	Invalid Date
TC20	1900	2	40	Invalid Date
TC21	2000	2	0	Invalid Date
TC22	2000	2	1	Valid Date
TC23	2000	2	28	Valid Date
TC24	2000	2	40	Invalid Date

8) Test Case Set for Decision Coverage

1. First decision: A = (year % 400 == 0)

A	Value A
0	1900
1	2000

2. Second decision: B = (year % 100 == 0)

A	B	A or B	Value A (for A=1)	Value B (for B=1)
0	0	0		2023
0	1	1		1900
1	X	1	2000	

3. Third decision: C = (year % 4 == 0)

A	B	C	A or B or C	Value C
0	0	0	0	2023
0	0	1	1	2024
0	1	X	1	1900
1	X	X	1	2000

Final result — Minimum test cases for Decision Coverage

Case	Year	A (400?)	B (100?)	C (4?)	Expected
TC1	2000	1	X	X	true
TC2	1900	0	1	X	false
TC3	2024	0	0	1	true
TC4	2023	0	0	0	false

9) Set of Test Cases for MC/DC Coverage

First decision: if (year % 400 == 0)

A	DOMINANT	Decision
0	A	year = **2023** → FALSE
1	A	year = **2000** → TRUE

Second decision: else if (year % 100 == 0)

A	B	C	A or B	DOMINANT	Decision
0	0	1	1	C	year = **2024** → TRUE
0	1	X	1	B	year = **1900** → FALSE

Third decision: else if (year % 4 == 0)

A	B	C	A or B or C	DOMINANT	Decision
0	0	0	0	C	year = **2023** → FALSE
0	0	1	1	C	year = **2024** → TRUE

10. Coverage Analysis

Full Combinatorial (N-wise):

Full combinatorial coverage requires testing **all possible combinations** of the identified test values:

- Year: 6 values
- Month: 3 values
- Day: 4 values

Total combinations:

$$6 \times 3 \times 4 = 72 \text{ test cases}$$

Advantage: Detects any defect caused by multi-variable interactions.

Disadvantage: High execution and design cost; unnecessary for simple decision structures.

Each-use:

This level of coverage ensures that **each individual test value** identified in section 4 is used **at least once**.

- Year: 6 values
- Month: 3 values
- Day: 4 values

Total:

$$6 + 3 + 4 = 13 \text{ test cases}$$

Advantage: Low cost and fast to execute; ensures that all equivalence classes are represented.

Disadvantage: Does not guarantee interaction coverage between variables.

Pairwise:

Pairwise coverage ensures that **each pair of variable values** appears together in at least one test case.

Using the selected values for Year, Month, and Day, the minimal number of test cases needed to achieve pairwise coverage is:

18 test cases

Advantage: Provides a good balance between test cost and defect detection; catches most interaction faults while remaining compact.

Disadvantage: May miss defects that require combinations involving three or more variables simultaneously.