NextDate Class and Function Assignment

Test-Driven Development Example — Test-Case Design Process

SOFE3980U - Software Quality - Dr. El-Darieby

John Howe — 100785128

1- Follow the algorithm to design test cases and group them in a test case table (chapter 6 of the book). Highlight characteristics, domains, blocks, values. Use the pairwise coverage criteria.

1. Identify Testable Functions

The function I will be developing is the NextDate(int Day, int Month, int Year) function.

2. Find All Inputs, Parameters, and Characteristics

Domains

Parameter	Actual Domain	Valid Domain
int Day	{ Day ∈ ℤ }	{ Day ∈ ℤ | 0 < Day ≤ LastDayOfMonth }
int Month	{ Month ∈ ℤ }	{ Month ∈ ℤ | 0 < Month ≤ 12 }
int Year	{ Year ∈ ℤ }	{ Year ∈ ℤ | 1812 ≤ Year ≤ 2212}

Characteristics

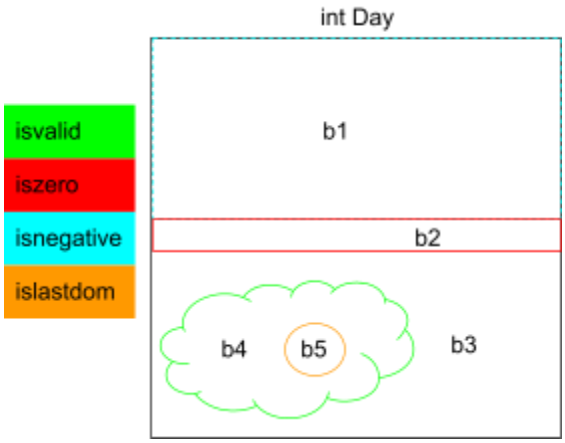
Parameters	Characteristics (Syntax)	Characteristics (Behaviour)
int Day	IsZero, IsNegative	IsValid, IsLastDayofMonth
int Month	IsZero, IsNegative	IsValid, NumberOfDays, IsLastMonthOfYear
int Year	IsZero, IsNegative	IsValid, IsLeapYear

3. Model the Input Domain

int Day

Partitions

Characteristic	b1	b2
isValid	True	False
isZero	True	False
isNegative	True	False
isLastDayOfMonth	True	False



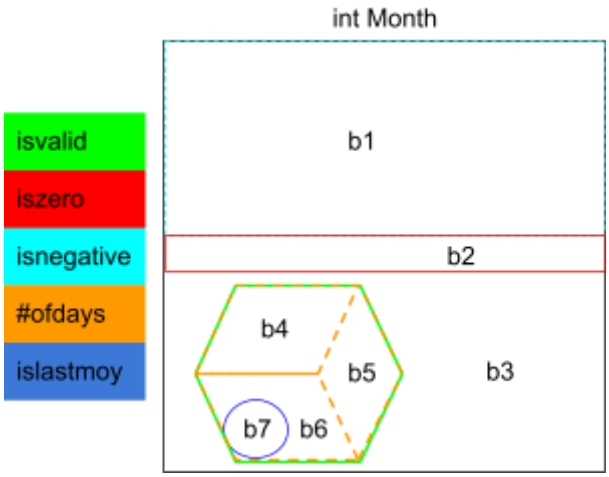
Blocks

Block	isValid	isZero	isNegative	isLastDayOfMonth	Potential Value
b1	F	F	T	F	-3
b2	F	T	F	F	0
b3	F	F	F	F	lengthOfMonth(Month) + 1
b4	T	F	F	F	19
b5	T	F	F	T	lengthOfMonth(Month)

int Month

Partitions

Characteristic	b1	b2	b3	b4
isValid	True	False		
isZero	True	False		
isNegative	True	False		
numberOfDays	0	28/29	30	31
isLastMonthOfYear	True	False		



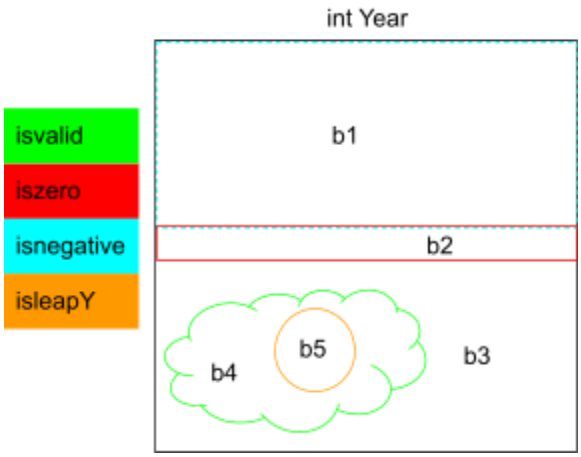
Blocks

Block	isValid	isZero	isNegative	#ofDays	islastMOY	Potential Value
b1	F	F	T	0	F	-4
b2	F	T	F	0	F	0
b3	F	F	F	0	F	15
b4	T	F	F	28/29	F	2
b5	T	F	F	30	F	6
b6	T	F	F	31	F	10
b7	T	F	F	31	T	12

int Year

Partitions

Characteristic	b1	b2
isValid	True	False
isZero	True	False
isNegative	True	False
isLeapYear	True	False



Blocks

Block	isValid	isZero	isNegative	isLeapYear	Potential Value
b1	F	F	T	F	-5
b2	F	T	F	F	0
b3	F	F	F	F	2213
b4	T	F	F	F	2023
b5	T	F	F	T	2024

4. Apply a Test Criterion to Choose Combinations of Blocks

For this system, I will be using Base Choice Coverage (BCC) with the base choice [19, 2, 2023] to create combinations for all invalid blocks. Using BCC will allow me to test each invalid input type individually to test how they will be handled. For the valid blocks ([Day - b4, b5], [Month - b4, b5, b6, b7], [Year - b4, b5]) I will be using Pairwise Coverage (PWC) to ensure I have included a good coverage for my valid inputs.

5. Refine Combinations of Blocks into Test Inputs

Invalid Test Inputs

Day	Month	Year	Description	Expected Output
-3	2	2023	Negative Day	IllegalArgumentException
19	-4	2023	Negative Month	IllegalArgumentException
19	2	-5	Negative Year	IllegalArgumentException
0	2	2023	Zero Day	IllegalArgumentException
19	0	2023	Zero Month	IllegalArgumentException
19	2	0	Zero Year	IllegalArgumentException
29	2	2023	Positive Invalid Day	IllegalArgumentException
19	15	2023	Positive Invalid Month	IllegalArgumentException
19	2	2213	Positive Invalid Year	IllegalArgumentException

Valid Test Inputs

Day	Month	Year	Expected Output
19	2	2023	20, 2, 2023
19	6	2024	20, 6, 2024
19	10	2023	20, 10, 2023
19	12	2024	20, 12, 2024
30	6	2023	1, 7, 2023
31	10	2024	1, 11, 2024
31	12	2023	1, 1, 2024
29	2	2024	1, 3, 2024

2- Write a JUnit test script

Included below is the JUnit test code for the NextDate function. I have used the @Test and @Tag JUnit annotations to label my test functions as tests and group them as either “invalid inputs” or “valid inputs”. I have used the assertAll() assertion to test the output values from each function call together. assertAll() can take multiple lambda expressions or statements, which allows me to call multiple assertEquals() assertions together. For the invalid inputs tests, I have also used the assertAll() assertion, this time to group together tests using similar invalid inputs. I have used lambda statements in this case, since asserting an exception requires multiple assertions instead of just one.

The full source code can also be found here on Github. The test file is located at /NextDate/src/test/java/johnh_otu/sofe3980U.

```
@Test @Tag("invalid_inputs")
    public void negativeInputs() {
        assertAll("negative_inputs",
                () -> {
                    Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(-3, 2, 2023));
                    assertEquals(ExceptionMessages.getDay_message(), e.getMessage());
                },
                () -> {
                    Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(19, -4, 2023));
                    assertEquals(ExceptionMessages.getMonth_message(), e.getMessage());
                },
                () -> {
                    Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(19, 2, -5));
                    assertEquals(ExceptionMessages.getYear_message(), e.getMessage());
                }
        );
    }

    @Test @Tag("invalid_inputs")
    public void zeroInputs() {
        assertAll("zero_inputs",
                () -> {
                    Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(0, 2, 2023));
                    assertEquals(ExceptionMessages.getDay_message(), e.getMessage());
                },
                () -> {
```

```
                Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(19, 0, 2023));
                assertEquals(ExceptionMessages.getMonth_message(), e.getMessage());
            },
            () -> {
                Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(19, 2, 0));
                assertEquals(ExceptionMessages.getYear_message(), e.getMessage());
            }
        );
    }

    @Test @Tag("invalid_inputs")
    public void positiveInvalidInputs() {
        assertAll("pos_invalid_inputs",
            () -> {
                Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(29, 2, 2023));
                assertEquals(ExceptionMessages.getDay_message(), e.getMessage());
            },
            () -> {
                Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(19, 15, 2023));
                assertEquals(ExceptionMessages.getMonth_message(), e.getMessage());
            },
            () -> {
                Exception e = assertThrows(IllegalArgumentException.class, () ->
NextDate.nextDate(19, 2, 2213));
                assertEquals(ExceptionMessages.getYear_message(), e.getMessage());
            }
        );
    }

    @Test @Tag("valid_inputs")
    public void test1() {
        int[] out = NextDate.nextDate(19, 2, 2023);
        assertAll("test1",
            () -> assertEquals(20, out[0]),
            () -> assertEquals(2, out[1]),
            () -> assertEquals(2023, out[2])
        );
    }
    @Test @Tag("valid_inputs")
    public void test2() {
```

```
    int[] out = NextDate.nextDate(19, 6, 2024);
    assertAll("test2",
            () -> assertEquals(20, out[0]),
            () -> assertEquals(6, out[1]),
            () -> assertEquals(2024, out[2])
    );
}
@Test @Tag("valid_inputs")
public void test3() {
    int[] out = NextDate.nextDate(19, 10, 2023);
    assertAll("test3",
            () -> assertEquals(20, out[0]),
            () -> assertEquals(10, out[1]),
            () -> assertEquals(2023, out[2])
    );
}
@Test @Tag("valid_inputs")
public void test4() {
    int[] out = NextDate.nextDate(19, 12, 2024);
    assertAll("test4",
            () -> assertEquals(20, out[0]),
            () -> assertEquals(12, out[1]),
            () -> assertEquals(2024, out[2])
    );
}
@Test @Tag("valid_inputs")
public void test5() {
    int[] out = NextDate.nextDate(30, 6, 2023);
    assertAll("test5",
            () -> assertEquals(1, out[0]),
            () -> assertEquals(7, out[1]),
            () -> assertEquals(2023, out[2])
    );
}
@Test @Tag("valid_inputs")
public void test6() {
    int[] out = NextDate.nextDate(31, 10, 2024);
    assertAll("test6",
            () -> assertEquals(1, out[0]),
            () -> assertEquals(11, out[1]),
            () -> assertEquals(2024, out[2])
    );
}
@Test @Tag("valid_inputs")
```

```
public void test7() {
    int[] out = NextDate.nextDate(31, 12, 2023);
    assertAll("test7",
            () -> assertEquals(1, out[0]),
            () -> assertEquals(1, out[1]),
            () -> assertEquals(2024, out[2])
    );
}
@Test @Tag("valid_inputs")
public void test8() {
    int[] out = NextDate.nextDate(29, 2, 2024);
    assertAll("test8",
            () -> assertEquals(1, out[0]),
            () -> assertEquals(3, out[1]),
            () -> assertEquals(2024, out[2])
    );
}
```

3- Write the corresponding Java code

After finishing the test code, and ensuring that the tests failed, I started working on the implementation of the NextDate() function. As I added features I checked to make sure that the corresponding tests passed. The full implementation of my code is on Github. The source code for the NextDate class is found here.

One of the things I found while implementing the function is that I wanted to implement separate functions for certain tasks in the function. Specifically, the isLeapYear() and endOfMonth() functions. Ideally, since these functions work solely within the NextDate class, I think it's suitable to test the class as a single unit rather than multiple. Therefore, the test code I have implemented is suitable for testing the NextDate class as a unit.

I have also implemented an ExceptionMessages class, but this class is only used to centralize the error messages in my code.

4- Run the tests and debug if necessary

Now that the function has been fully implemented, all 11 tests pass.

```
[INFO] --------------------< johnh_otu.sofe3980U:NextDate >--------------------
[INFO] Building NextDate 1.0.0
[INFO]   from pom.xml
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- resources:3.0.2:resources (default-resources) @ NextDate ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\johnh\source\repos\sqTDDhomework\NextDate\src\main\resources
[INFO]
[INFO] --- compiler:3.8.0:compile (default-compile) @ NextDate ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- resources:3.0.2:testResources (default-testResources) @ NextDate ---
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory C:\Users\johnh\source\repos\sqTDDhomework\NextDate\src\test\resources
[INFO]
[INFO] --- compiler:3.8.0:testCompile (default-testCompile) @ NextDate ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- surefire:2.22.1:test (default-test) @ NextDate ---
[INFO]
[INFO] -------------------------------------------------------
[INFO]  T E S T S
[INFO] -------------------------------------------------------
[INFO] Running johnh_otu.sofe3980U.NextDateTest
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 0.077 s - in johnh_otu.sofe3980U.NextDateTest
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 11, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  3.406 s
[INFO] Finished at: 2024-02-25T16:35:18-05:00
[INFO] ------------------------------------------------------------------------
```

Since the majority of the time learning the problem space was spent designing tests, actually implementing and testing the code took very little time, and no bugs were found in my code after implementation. I can see how in a system where this process has been optimized, Test-Driven Development could increase software quality and decrease development time.