

Amy Wang

MSCI 240 Fall 2018

Instructor: Dr Mark Hancock

September 16th, 2018

Project 1, Part 1

Introduction:

The task at hand is to design a test plan and code for a date class that has various methods. These methods provide the class with many useful functions. However, it is unknown if the code is free of errors which is why, for each method, test cases must be made to check the code's soundness. The test plan should outline all the test cases in plain English and will describe the process of the test and expected result for which the test code will execute with the correct syntax.

Background Info:

Date Class Constructors:

- Date(int year, int month, int day): create a custom date instance
- Date(): creates a date with today's values

Date Class Methods:

- getDay(): returns Date object's day value
- getMonth(): returns Date object's month value (1-12)
- getYear(): returns Date object's year
- getDayOfWeek(): returns Date object's string date value
- isLeapYear(): returns Boolean of if it is a leap year
- nextDay(): increases the day by 1
- toString(): returns string value of Date object

Test Plan:

Part 1 of the test plan will be focused on the constructors to check whether various date objects can be created without errors. First thing is to test if creating today's date instance is correct. Next will be any valid date, a date with an invalid year, a date with an invalid month, a date with an invalid day, a real date with an invalid format, February 29th of a leap year, and February 29th of a non-leap year.

Part 1

- 1.1 Construct a date object for today's date and print to console, the current date is expected to be printed to the console
- 1.2 Construct a custom date with valid year, month, date then print to console, the entered date is expected to be printed to the console
- 1.3 Construct a custom date with an invalid year, an error is expected to be printed
- 1.4 Construct a custom date with an invalid month, an error is expected to be printed
- 1.5 Construct a custom date with an invalid date, an error is expected to be printed
- 1.6 Construct a custom date with a different format but valid date (add zeros in front of date), no error is expected, and the entered date is expected to be printed to the console
- 1.7 Construct a date with a leap year with February 29, entered date is expected to be printed
- 1.8 Construct a date that's not a leap year with February 29, an error is expected to be printed

Part 2

Part 2 of the test plan will test each of "get" methods to check if the outputted results are as expected as well as testing logically incorrect data to observe what occurs. The *toString* methods are also tested in

this section. One valid date, one invalid date, the first date possible, and an extremely high date are created to run the methods with.

Test Date 1: 2018/2/14 | Test Date 2: 2018/02/014 (Invalid format)

Test First Date: 1753/1/1 | Test High Date: 2000000/1/1

- 2.11 Print *getDay* method for test date 1, the expected result is 14
- 2.12 Print *getDay* method for test date 2, the expected result is 14
- 2.13 Print *getDay* method for the first date, the expected result is 1
- 2.21 Print *getMonth* method for test date 1, the expected result is 2
- 2.22 Print *getMonth* method for test date 2, the expected result is 2
- 2.23 Print *getMonth* method for the first date, the expected result is 1
- 2.31 Print *getYear* method for test date 1, the expected result is 2018
- 2.32 Print *getYear* method for the first date, the expected result is 1753
- 2.33 Print *getYear* method for a high date, an error is expected as the year is too high
- 2.41 Print *getDayOfWeek* method for test date 1, the expected result is Wednesday
- 2.42 Print *getDayOfWeek* method for test date 2, the expected result is Wednesday
- 2.43 Print *getDayOfWeek* method for the first date, the expected result is Monday
- 2.44 Print *getDayOfWeek* method for leap year date, the expected result is Monday
- 2.51 Print *toString* method for today, the expected result is 2018/9/16
- 2.52 Print *toString* method for test date 2, the expected result is 2018/2/14
- 2.53 Print *toString* method for first date, the expected result is 1753/1/1

Part 3

Part 3 of the test plan deals with the *nextDay* function and various significant dates. These dates being the last day of the year, the last day of the month, February 28th of a leap year, February 28th of a non-leap year, some valid insignificant date, and some invalid insignificant date. The method *checkNextDay* will create a Date instance, print that, then call the *nextDay* function and print the Date instance again.

- 3.1 Print the next day of some set date (2018/9/16), the expected result is 2018/9/17
- 3.2 Print the next day of February 28th of a leap year (2016/2/28), the expected result is 2016/2/29
- 3.3 Print the next day of February 28th of a non-leap year (2018/9/16), the expected result is 2018/3/1
- 3.4 Print the next day of the day before a new year (2018/12/31), the expected result is 2019/1/1
- 3.5 Print the next day of the day before a new month (2018/9/30), the expected result is 2018/10/1
- 3.6 Print the next day of an invalidly formatted date (2018/9/014), the expected result is 2018/9/16

Part 4

Part 4 tests the *isLeapYear* function to see if it performs as expected. Four dates are created to do just that, a valid date of a leap year, a valid date of a non-leap year, an invalid date of a leap year, and an invalid date of a non-leap year.

- 4.1 Print *isLeapYear* method of a valid leap year date (2016/2/29), the expected result is true
- 4.2 Print *isLeapYear* method of a valid non-leap year date (2018/2/20), the expected result is false
- 4.3 Print *isLeapYear* method of an invalid leap year date (2016/012/031), the expected result is true
- 4.4 Print *isLeapYear* method of an invalid non-leap year date (2015/012/031), false is expected

Test Code:

```
public static void main(String[] args) {
    // Date instances for Part 2 of Testing
    Date testDate1 = new Date(2018, 2, 14); // Valid
    Date testDate2 = new Date(2018, 02, 014); // Invalid
    Date testFirstDate = new Date(1753, 1, 1); // Earliest Year
    Date testHighDate = new Date(2000000, 1, 1); // Insanely High Year

    // Date instances for Part 4 of Testing
    Date leapDate = new Date(2016, 2, 29); //Valid leap date
    Date nonLeapDate = new Date(2018, 2, 20); //Valid non-leap date
    Date invalidNonLeapDate = new Date(2015, 012, 031); //invalid non-leap date
    Date invalidLeapDate = new Date(2016, 012, 031); //invalid leap date

    // 1.1 Today's Date
    Date today = new Date();
    System.out.println("Today is: " + today);
    // 1.2 Valid Custom Date
    createDate(1912, 1, 14);
    // 1.3 Invalid Year
    createDate(0013, 1, 14);
    // 1.4 Invalid Month
    createDate(2014, 14, 14);
    // 1.5 Invalid Date
    createDate(2015, 1, 35);
    // 1.6 Formatted Incorrectly
    createDate(2016, 01, 016);
    // 1.7 Leap year date 2/29
    createDate(2016, 2, 29);
    // 1.8 Not leap year date 2/29 (DNE)
    createDate(2018, 2, 29);

    // 2.11 Test correctly formatted date
    System.out.println(testDate1 + "'s Day: " + testDate1.getDay());
    // 2.12 Test incorrectly formatted date
    System.out.println(testDate1 + "'s Day: " + testDate2.getDay());
    // 2.13 Test earliest date's day
    System.out.println(testDate1 + "'s Day: " + testFirstDate.getDay());
    // 2.21 Test correctly formatted month
    System.out.println(testDate1 + "'s Month: " + testDate1.getMonth());
    // 2.22 Test incorrectly formatted month
    System.out.println(testDate2 + "'s Month: " + testDate2.getMonth());
    // 2.23 Test earliest date's month
    System.out.println(testFirstDate + "'s Month: " + testFirstDate.getMonth());
    // 2.31 Test correctly formatted year
    System.out.println(testDate1 + "'s Year: " + testDate1.getYear());
    // 2.32 Test earliest date's year
    System.out.println(testFirstDate + "'s Year: " + testFirstDate.getYear());
    // 2.33 Test insanely high year
    System.out.println(testHighDate + "'s Year: " + testHighDate.getYear());
    // 2.41 Test correctly formatted date
    System.out.println(testDate1 + "'s Day of the Week: " + testDate1.getDayOfWeek());
    // 2.42 Test incorrectly formatted date
    System.out.println(testDate2 + "'s Day of the Week: " + testDate2.getDayOfWeek());
    // 2.43 Test earliest date's day of the week (Monday)
    System.out.println(testFirstDate + "'s Day of the Week: " + testFirstDate.getDayOfWeek());
    // 2.44 Test leap year date
    System.out.println(leapDate + "'s Day of the Week: " + leapDate.getDayOfWeek());
    // 2.51 Print toString() for today
    System.out.println("toString(): " + today.toString());
    // 2.52 Print toString() for invalid formatted date
    System.out.println("toString(): " + testDate2.toString());
    // 2.53 Print toString() for invalid formatted date
    System.out.println("toString(): " + testFirstDate.toString());
}
```

```

// 3.1 Test how nextDay() works with some date
checkNextDay(2018, 9, 16);
// 3.2 Test how nextDay() works with leap year date
checkNextDay(2016, 2, 28);
// 3.3 Test how nextDay() works with non-leap year date
checkNextDay(2018, 2, 28);
// 3.4 Test how nextDay() works with Day before new year
checkNextDay(2018, 12, 31);
// 3.5 Test how nextDay() works with Day before new month
checkNextDay(2018, 9, 30);
// 3.6 Test how nextDay() works with an invalidly formatted date
checkNextDay(2018, 9, 014);

// 4.1 Test if the leap year returns as true
System.out.println(leapDate + " Is a Leap Year: " + leapDate.isLeapYear());
// 4.2 Test if the non-leap year returns as false
System.out.println(nonLeapDate + " Is a Leap Year: " + nonLeapDate.isLeapYear());
// 4.2 Test if the invalid formatted leap year returns as true
System.out.println(invalidLeapDate + " Is a Leap Year: " + invalidLeapDate.isLeapYear());
// 4.2 Test if the invalid formatted non-leap year returns as false
System.out.println(invalidNonLeapDate + "Is a Leap Year: " + invalidNonLeapDate.isLeapYear());
}

// creates a custom date object then prints it
public static void createDate(int year, int month, int date) {
    try {
        Date customDate = new Date(year, month, date);
        System.out.println("Created Date: " + customDate);
    } catch (Exception error) {
        error.printStackTrace();
    }
}

// Prints created date, Adds a day to the created Date and prints date again
public static void checkNextDay(int year, int month, int date) {
    try {
        Date customDate = new Date(year, month, date);
        System.out.print("Original Date: " + customDate);
        customDate.nextDay();
        System.out.println("| Next Date: " + customDate);
    } catch (Exception error) {
        error.printStackTrace();
    }
}

```

Test Console Output:

```
|Today is: 2018/9/16
Created Date: 1912/1/14
java.lang.IllegalArgumentException: 11/1/14 year too small: 11
    at teacher.Date.<init>(Date.java:39)
    at part1.DateClient.createDate(DateClient.java:109)
    at part1.DateClient.main(DateClient.java:37)
java.lang.IllegalArgumentException: 2014/14/14 month out of range: 14
    at teacher.Date.<init>(Date.java:42)
    at part1.DateClient.createDate(DateClient.java:109)
    at part1.DateClient.main(DateClient.java:39)
java.lang.IllegalArgumentException: 2015/1/35 day out of range: 35
    at teacher.Date.<init>(Date.java:45)
    at part1.DateClient.createDate(DateClient.java:109)
    at part1.DateClient.main(DateClient.java:41)
Created Date: 2016/1/14
Created Date: 2016/2/29
java.lang.IllegalArgumentException: 2018/2/29 day out of range: 29
    at teacher.Date.<init>(Date.java:45)
    at part1.DateClient.createDate(DateClient.java:109)
    at part1.DateClient.main(DateClient.java:47)
2018/2/14's Day: 14
2018/2/14's Day: 12
2018/2/14's Day: 1
2018/2/14's Month: 2
2018/2/12's Month: 2
1753/1/1's Month: 1
2018/2/14's Year: 2018
1753/1/1's Year: 1753
2000000/1/1's Year: 2000000
2018/2/14's Day of the Week: Wednesday
2018/2/12's Day of the Week: Monday
1753/1/1's Day of the Week: Monday
2016/2/29's Day of the Week: Monday
toString(): 2018/9/16
toString(): 2018/2/12
toString(): 1753/1/1
Original Date: 2018/9/16| Next Date: 2018/9/17
Original Date: 2016/2/28| Next Date: 2016/2/29
Original Date: 2018/2/28| Next Date: 2018/3/1
Original Date: 2018/12/31| Next Date: 2019/1/1
Original Date: 2018/9/30| Next Date: 2018/10/1
Original Date: 2018/9/12| Next Date: 2018/9/13
2016/2/29 Is a Leap Year: true
2018/2/20 Is a Leap Year: false
2016/10/25 Is a Leap Year: true
2015/10/25 Is a Leap Year: false
```

Non-Acknowledgment of Receiving Assistance or Use of Others' Ideas

I received the following help, assistance, or any ideas from classmates, other knowledgeable people, books or non-course websites (please include a description of discussions with the TA or the instructor):

None

Record of Giving Assistance to Others

I gave the following help, assistance, or ideas to the following classmates (please describe what assistance to whom was given by you):

None

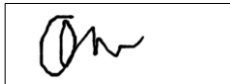
Declaration

I declare that except for the assistance noted above, assistance provided on the course website, and material provided by the instructor and/or TAs that this is my original work.

I have neither given nor received an electronic or printed version of any part of this code to/from anyone.

I declare that any program output submitted as part of the assignment was generated by the program code submitted and not altered in any way.

Signature: _____

A handwritten signature in black ink, appearing to be "Om", is enclosed within a rectangular box.