**Amy Wang**

**MSCI 240 Fall 2018**

**Instructor: Dr Mark Hancock**

**September 22th, 2018**

**Project 1, Part 2**

**Introduction:**

The goal is to implement a class for a Date object in Java. The Date object has a year, month, and a date therefore, we need constructors to assign values to the class variables. One default constructor which creates a date object with the current date and one constructor to create a custom date. The created dates must also be within reasonable merit. For example, 1753 is the earliest year recorded thus, the creation of a year less than 1753 should be disallowed. Various functions related to date should also be created. The class needs accessor methods to get the year, month, date, day of the week, or the string value of the object, and to check whether the Date is a leap year. Additionally, the class needs a mutator method which increases the date objects date by 1. In this report, all the planning and logic will be explained in pseudocode or plain English.

**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

**Description of Implementation:**

**Import:** Import java’s temporal ChronoUnit and LocalDate

Constants: One array for months that have only 30 days (4,6,9,11)

**Private Class Modules (Helper methods):**

**isLeapYear(int year):**  Takes the input of a year and returns a true if it is a leap year and false if it’s not

create a Boolean that starts off as false then check these conditions

if the year is not divisible by 4 then off the bat it is not a leap year

otherwise if year modulus by 4 and 400 both equal 0 then it is a leap year, change the Boolean to true

if year modulus by 4 and 400 and 100 are all 0 then it is not a leap year, change the Boolean to false

If the year is divisible by only 4 and not 100 or 400 then it is a leap year change the Boolean to true

Return the boolean

**daysSince(int baseYear):** Meant to calculate how many days have passed since Base Year/1/1

for each year from base year to object’s year add 365 if its not a leap year and add 366 if it is

the loop goes up to the object’s year but does not include the days elapsed in that year

return all the accumulated days from the years passed then add the year’s day of year (/365 or /366)

**calcDayOfYear():** track the total amount of elapsed days for the object’s year

for each month up to the date object’s month -1, whether the month has 30 days, 31 days, 29 days (with leap year), or 28 days (non leap)

the elapsed days will accumulate up until Month-1

then it will return the elapsed days accumulated plus the object’s day value

**date(valid):** calls methods which check if the year, month, and date are valid by passing the necessary parameters

call checkYear and pass in year, call checkMonth and pass in month

call CheckDay and pass in year, month, and day

**checkYear(int year):** If the year is less than 1753 throw error

**checkMonth (int month):** if the month is not within (1,12) then throw error

**checkDay(int year, int month, int day):** checks if the day is valid given the values of month and year

first check if the day is between 1 and 31

next check if the date is in February, it is a leap year, then the day must not be greater than 29

if the date is in February, it not is a leap year, then the day must not be greater than 28

otherwise loop through the month constant which holds months with 30 days and check if the date are any of those months and greater than 30 then throw an error

**Public Class Methods**

**getDayOfWeek():** Will return the day of the week

Knowing that 1753 starts on a Monday we use the constant baseYear2 to calculate daysSince baseYear2/1/1 and the object’s date

Create an integer day and set it to 0, day will keep track of the day of the week

Next loop from 0 to the daysSince calculated and loop from 0-6 and reset to 0 after it reaches 6

After the Then we can return the dayOfWeek corresponding to the index of the day.

**nextday():** Will add a date to the object and change either month or year based on certain criteria

first checks end of year case if its 12 and 31 then it will reset month and day to 1

next check if it is February, a leap year, and 29th then it will increase month and reset day

next check if it is February, a non-leap year, and 28th then it will increase month and reset day

next check if it is April, June, September or November then the month will increase, and days reset at 30

otherwise if the day is 31 and none of the other criteria are met then it’ll increase month and reset day

this ends the conditions and a day is added.

**setDate(int year, int month, int day):** setter method, sets object attributes to values passed in

**toString():** getter method which formats the attributes into a string format for date (YYYY,MM,DD)

**getYear():** getter method just returns year of instance

**getMonth():** getter method returns month of instance

**getDay():** getter method gets returns of instance

**Date():** The default constructor should create a Date instance with the current date’s values.

Create a LocalDate instance for today’s value

Create a starting LocalDate object

Set daysSince to get days between start LocalDate and today LocalDate

Set the date to the January 1st of the base year

Run the next day method daysSince amount of times to get the current date

**Date(int year, int month, int day):** Constructor used to make custom dates

First call dateValid method to check if any errors are thrown

If no errors occur run the setDate method which will assign the passed in values to the object

**Test Console Output:**



**Test Plan:**

|  |  |  |  |
| --- | --- | --- | --- |
| Test # | Brief Summary | Expected result | Actual Result |
| 1.1 | Today’s date | 2018/9/26 | 2018/9/26 |
| 1.2 | Create any date | 1912/1/14 | 1912/1/14 |
| 1.3 | Date with year too small | Error | Error |
| 1.4 | Date with month too high | Error | Error |
| 1.5 | Date with invalid day | Error | Error |
| 1.6 | Very High Date | 3000/12/31 | 3000/12/31 |
| 1.7 | Leap Year Feb. 29 | 2016/2/29 | 2016/2/29 |
| 1.8 | Non-leap year Feb. 29 | Error | Error |
| 2.11 | Get valid date day | 14 | 14 |
| 2.12 | Get high date day | 31 | 31 |
| 2.13 | Get first date day | 1 | 1 |
| 2.21 | Get valid date month | 2 | 2 |
| 2.22 | Get high date month | 12 | 12 |
| 2.23 | Get first date month | 1 | 1 |
| 2.31 | Get valid date year | 2018 | 2018 |
| 2.32 | Get high date year | 3000 | 3000 |
| 2.33 | Get first date year | 1753 | 1753 |
| 2.41 | Get valid date day of week | Wednesday | Wednesday |
| 2.42 | Get high date day of week | Wednesday | Wednesday |
| 2.43 | Get first date day of year | Monday | Monday |
| 2.44 | Get leap year date day of year | Monday | Monday |
| 2.51 | Today to string | 2018/9/26 | 2018/9/26 |
| 2.52 | High date to string | 3000/12/31 | 3000/12/31 |
| 2.53 | Earliest date to string | 1753/1/1 | 1753/1/1 |
| 3.1 | Next day of some date | 2018/9/17 | 2018/9/17 |
| 3.2 | Next day of Feb 28 of leap year | 2016/2/29 | 2016/2/29 |
| 3.3 | Next day of Feb 28 of non-leap year | 2018/3/1 | 2018/3/1 |
| 3.4 | Next day of New Year’s Eve | 2019/1/1 | 2019/1/1 |
| 3.5 | Next day of end of month | 2018/10/1 | 2018/10/1 |
| 3.6 | Next day of Earliest date | 1753/1/2 | 1753/1/2 |
| 4.1 | Check if leap year is a leap year | True | True |
| 4.2 | Check if non-leap year is a leap year | False | False |
| 4.3 | Check if earliest leap year is leap year | True | true |

For part 1 of the project we created a Date Client to test the premade Date class. In part 2 I used the same Date client to test the Date class created. The expected results should have been the same as the output of the last part. The Date class created, passed all the tests in the test plan for part 1, part 2, part 3, and part 4. It is evident in the console output attached above and explained in the table above.

# 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: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_