"Return Within 30 Days for a Full Refund"

Your final solution will have 3 files (there are 3 classes) so you will need to bundle them together into 1 compressed file. Please submit either a .zip file or a .jar file.

Most return policies give a window of time from the date of purchase, for example 30 days. But how do you find a future date from a given date? Let's write a tool to figure this out.

Objectives

- Defining your own object to a specification
- Write both supplier and client code
- Working with multiple classes and starter code

Specification

This assignment has 2 parts. In part 1, you'll implement a class to a given specification. In part 2, you'll modify starter code to complete a stand-alone program using the class you wrote in part 1.

In both classes, you are limited to the topics covered so far this quarter: chapters 1 - 5, & chapter 8.

Part 1:
You are going to write a class named Date to the following specification:

class Date
<properties -- you need to choose appropriate datatypes> - day - month - year /* Class invariants: The year must be 1600 or later The day must be legal for the month The day and month must be legal for the year For example, if the month is June, the day should never be 31. If the month is February, the day can only be 29 if it is a leap year. All methods that affect the state of a Date object must ensure this. */
<constructor> **+ Date(int month, int day, int year) --** construct a Date object with the given state. If any of the parameter values are invalid, throw an IllegalArgumentException. Make sure that the exception contains a message specific to the problem.

<static methods>
+ boolean isLegal(int month, int day, int year) -- returns true if these 3 values make up a valid date, false otherwise. Leap years are handled correctly.

<accessor methods>
+ **int getDay()** -- returns the day of this Date

+ **int getMonth()** -- returns the numeric value of the month of this Date

+ String getMonthName() - returns the name of the month of this Date. The name starts with a capital letter, such as April or June.

+ **int getYear()** -- returns the year of this Date

+ **String toString()** -- returns this Date as a String of the form: mm/dd/yyyy. An example would be 02/20/1974 or 10/11/2020

+ Date daysFromDate(int days) -- returns a new Date that is "days" in the future from this Date. A valid parameter is in the range [1, 31]. Throw an IllegalArgumentException if the parameter is invalid.

<mutator methods>
+ void setDay(int day) -- update the day of this Date. If the parameter value is invalid, throw an IllegalArgumentException.

+ void setMonth(int mth) -- update the month of this Date. If the parameter value is invalid, throw an IllegalArgumentException.

+ void setYear(int yr) -- update the year of this Date. If the parameter value is invalid, throw an IllegalArgumentException.

Determining a leap year:

A year is a leap year if it is divisible by 4, except if it is divisible by 100. If a year is divisible by 100, then it must be divisible by 400 to be a leap year. For example, the following are leap years: 1976, 1600, and 2000. 1982 and 1900 are not leap years.

Part 2:

To complete the program, you first need to download the starter code. This code creates a graphical user interface (GUI) that allows the user to interact with the program. To start the program, compile the given files, then choose the main() method from the FindADate class. You should see a window open with some text, 3 areas for input, and a button at the bottom (Date in 30 days). Click the button and it should display a message to System.out. If you've gotten this far then you know that the starter code works.

Your task is to modify the FindADate class so that it gets the input and ***displays the correct result to the GUI window***. The class that controls the GUI window is the UserWindow class; FindADate

class will be a client of the UserWindow class. I recommend opening UserWindow to the documentation view in BlueJ. FindADate is also a client of your Date class so make sure you've tested it well.

Though the user interface restricts user input somewhat, there is still the chance that the input is invalid. Make sure your client code calls isLegal() before trying to construct or update a Date object. We don't know yet how to "recover" from a thrown exception. ***If FindADate determines that the input is invalid, have it display an error message to the user, using the error window method in UserWindow.***

Finally, don't be surprised if there are methods in Date that this client program doesn't use. That's ok, but those methods should still work. Your supplier class may be used in a different application that will need those methods, so test your class definition thoroughly.

Documentation & Style

- From this homework forward, you must use Javadoc comments for each class description and before each method definition. Use the @author, @param, @return, and @throws tags as appropriate. Refer to week 4's additional reading as well as the Documentation and Style Guidelines under Course Resources in Canvas.
- Include internal comments of your algorithms in both the Date class and the FindADate class.
- Use class constants as appropriate.
- Since you will be modifying FindADate, make sure to include your name with the @author tag. Something like **@author CSC 142, modified by Jane Doe**.
- There are a number of tasks in the Date class that you'll find yourself repeating in different places. This is a perfect place for procedural decomposition (breaking one method into smaller methods).

Suggestions

- You must build the Date class first in order to complete part 2, so start there. As in the past, implement and test each method one at a time to prevent going too far down a wrong path.
- Make sure to have a good sense of your algorithm design before starting to code. Use either pseudocode or flowcharts or any way you like to get your logic on paper. *You can't build a bridge without plans!*
- As mentioned above, you'll need to test your Date class thoroughly. I encourage you to include a static test method in that class. This way, you write your test code once, and then call it as often as you'd like. This is more efficient than repeatedly using BlueJ's object bench.

Grading

/12 Date class meets the specification correctly and is designed well
/3 Modifications to FindADate class
/5 Documentation and style

Extra Credit (+1) for a thorough static test method in your Date class.

Good Luck!