CSC 142
Assignment #4: Birthday/Date[1]
(25 points)
See our Canvas online site for due date.

Submissions made beyond the last day of winter'14 classes are NOT accepted.

Design and implement and test a Java class: Date

This program focuses on classes and objects. Turn in two files named **DateClient.java** and **Date.java**.
Begin by prompting the user for today's date and for his/her birthday in month day year format (see examples below).
Use this information to print the user's birthday in year/month/day format, the day of the week the user was born, and how old the user is in days. If the user was born in a leap year, you print an additional message letting them know their birth year was a leap year.
Below are several example logs of execution from the program; user input is in **bold**. Output is in *italics*. Your program's output should match these examples exactly given the same input.

What is today's date (month day year)? **8 8 2010**
What is your birthday (month day year)? **1 26 1971**
You were born on 1971/1/26, which was a Tuesday.
You are 14439 days old.

What is today's date (month day year)? **8 8 2010**
What is your birthday (month day year)? **2 2 1996**
You were born on 1996/2/2, which was a Friday.
1996 was a leap year.
You are 5301 days old.

What is today's date (month day year)? **8 8 2010**
What is your birthday (month day year)? **11 8 1900**
You were born on 1900/11/8, which was a Thursday.
You are 40085 days old.

What is today's date (month day year)? **7 11 1856**
What is your birthday (month day year)? **10 2 1800**
You were born on 1800/10/2, which was a Thursday.
You are 20371 days old.

[1] Adopted from UW CSE 142 Homework and our textbook.

Solve this problem using Date objects. The methods and behavior of each Date object are described on the next page.

To figure out the number of days old the user is, represent today and the birthday as Date objects and use the methods found in the Date objects to figure out how many days are between them. Note: Some of the methods provided modify the state of the object on which they are called. See the next page for method descriptions.

You can construct a Date object as follows:
Date name = new Date (year, month, day);

You do need to take leap years into account for this assignment. A leap year occurs roughly every 4 years and adds a 29th day to February, making the year 366 days long. (Leap day babies usually celebrate their birthdays on February 28 or March 1 on non-leap years.) Date objects have various methods that can help you to detect and handle the leap year case.
You may assume that neither today nor the user's birthday is the rare "leap day" of February 29.

Assume valid input (that the user will always type a year between 1753 – 9999, a month between 1-12, and a day between 1 and the end of that month).

Date.java, class of objects:
Implement a class named Date, stored in a file named Date.java.

In the descriptions below the phrase "this Date object" refers to the object on which the method was called. Assume that all parameters passed to all methods are valid. Your Date class should implement the following behavior:

- **public Date(int year, int month, int day)**

Constructs a new Date representing the given year, month, and day.

- **public int getYear()**

This method should return this Date object's year, between 1753 and 9999. For example, if this Date object represents August 17, 2010, this method should return 2010.

- **public int getMonth()**

This method should return this Date object's month of the year, between 1 and 12. For example, if this Date object represents August 17, 2010, this method should return 8.

- **public int getDay()**

This method should return this Date object's day of the month, between 1 and the number of days in that month (which will be between 28 and 31). For example, if this Date object represents August 17, 2010, this method should return 17.

- **public String toString()**

This method should return a String representation of this Date in a year/month/day format. For example, if this Date object represents May 24, 2010, return "2010/5/24". If this Date object represents December 3, 1973, return "1973/12/3". Note that this method returns the string; it does not print any output.

- **public boolean equals(Date d)**

Returns true when this Date object represents the same date as the given Date parameter. Returns false otherwise.

- **public boolean isLeapYear()**

Returns whether this Date's year is a leap year. Leap years are all years that are divisible by 4, except for years that are divisible by 100 but not by 400. For example, 1756, 1952, 2004, 1600, and 2000 are all leap years, but 1753, 2005, 1700, and 1900 are not.

- **public void nextDay()**

Modifies the state of this Date object by advancing it 1 day in time. For example, if this Date object represents September 19, a call to this method should modify this Date object's state so that it represents September 20. Note that depending on the date, a call to this method might advance this Date object into the next month or year. For example, the next day after August17, 2010 is August 18, 2010; the next day after February 28, 1997 is March 1,1997; and the next day after December 31, 1978 is January 1, 1979.

- **public int advanceTo(Date endDay)**

Modifies the state of this Date object by advancing it to the given end Date. This method returns the number of days it took to advance this Date object to the given end Date. For example, if this Date object represents September 19, 2010 and the endDay represents September 29, 2010, a call to this method modifies this Date object's state so it also represents September 29, 2010 and returns 10. Depending on the date, a call to this method might advance this Date object into a different month or year. You may assume the given Date object always comes after this Date object.

- **public String getDayOfWeek()**

Returns a String representing what day of the week this Date falls on. The String will be "Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", or

"Saturday". For example, August 8, 2010 fell on a Sunday, so the return value for a new Date(2010, 8, 8) object would be "Sunday". At the end of a call to this method, the state of this Date object should be the same as it was at the start of the call. You should base your calculations on the fact that January 1, 1753 was a Monday.

None of the methods listed above should print any output to the console. You may not use any of Java's date-related classes such as java.util.Date or java.util.GregorianCalendar. You are limited to features in Chapters 1 through 8.

DateClient.java is a testing program; it should call all of your Date methods to test them in a very exhaustive way to try all cases and combinations.

Development Strategy and Hints:

- Write the constructor and getYear, getMonth, and getDay and methods first.
- Then implement toString, equals, and isLeapYear.
- Next, try to write nextDay. (You may wish to write a helping method that returns the number of days in this Date's month, to help you implement nextDay. If you decide to write this method, please remember that if it is a leap year, February will have 29 days.)
- Next, write advanceTo. You should be able to use methods you have already written to write this.
- Lastly, write getDayOfWeek. You might want to construct a Date object for January 1, 1753 to help you. Again, you should use the other methods you have already written to help you write this method. Remember that the Date represented by the object should be the same as it was after a call to getDayOfWeek (this is different from nextDay and advanceTo which modify the state of the object).

Style Guidelines:

Implement your Date as a new type of object, using non-static methods, non-static data fields, constructors, etc. as appropriate. You should also properly encapsulate your Date objects by making their methods and constructors public and their data fields private. As much as possible you should avoid redundancy and repeated logic within the Date class. Avoid unnecessary fields: Use fields to store the important data of your Date objects but not to store temporary values that are only used within a single call to one method.

Comment your code including: (1) with Course Name, Program name, File name, Date, Programmer Name and Program Description, and (2) descriptive comments throughout the source code.

Submit using Canvas online assignment submission (Please note that late submissions beyond the last day of classes are not accepted):

General assignment requirements:

- **Reflections:** Reflect briefly on each program you write, as a comment at either the beginning or end of each main file. Comment on things like: how hard was the program and why? What did you learn from it? Where did you need help? How long did it take to complete? What was the best part of the assignment and why? These reflections count towards a part of your assignment score.
- **Identification:** You must include a header comment with your name, this course number (CSC142), the date, and the assignment number in every java file you write.
- **Compilation:** Source code (*.java) must compile in standard Java; Remember that you will receive very low score for programs that do not compile.
- **Online Submission:** Submit an electronic copy of all assignments via Canvas. You may submit on Canvas multiple times for any assignment till the time it is due - I will only grade the last (most recent) submission unless you request otherwise.

Submit the following:

- source code files (Date.java and DateClient.java)

Make sure you keep a copy of everything that you submit for your records, in case of disk failures and server crashes etc.