COMPUTER SCIENCE 12B-1 (SPRING TERM, 2017)
PROGRAMMING IN JAVA

PROGRAMMING ASSIGNMENT 2

Due Date: Thursday February 09, 2017, 11:55pm

Overview:

This assignment will give you practice with defining new types of objects. Turn in two Java classes named `Birthday` and `Date`, stored in files named `Birthday.java` and `Date.java` respectively. You will also need the support file `TeacherDate.class` from LATTE. Place this file in the same folder as your program's classes. The assignment has two parts: a program that uses date objects to print information about the user's birthday, and a `Date` class of your own whose objects represent calendar dates.

Part 1: Birthday program

The first part of this assignment asks you to write a client Java program that uses an existing date class written by the instructor. Your program should prompt the user for his/her birth year, month, and day. Based on this information, you will print that birth date, what day of the week it fell on, a message if that was a leap year, and the number of days until the user's birthday. If it is the user's birthday, you will print a Happy Birthday message including the user's current age.

The following three logs of execution show the various behaviors of the program. Note that the number of days until the user's birthday depends on when the program was run. The logs of execution below were generated on **Saturday, January 30, 2010**. Since you run the program on a different date, you will receive slightly different output.

Example logs of execution (user input underlined):

```
What month, day, and year were you born? 5 8 1987
You were born on 1987/5/8, which was a Friday.
It will be your birthday in 98 days.
You are 8303 days old.
```

```
What month, day, and year were you born? 1 28 1952
You were born on 1952/1/28, which was a Monday.
1952 was a leap year.
It will be your birthday in 363 days.
You are 21187 days old.
```

```
What month, day, and year were you born? 1 30 1900
You were born on 1900/1/30, which was a Tuesday.
Happy birthday!  You are now age 110.
You are 40177 days old.
```

You should implement this program using the `TeacherDate` class provided to you. A `TeacherDate` contains all of the methods and behavior described on the next page, exactly matching the `Date` class you will implement in Part 2. You can construct a `TeacherDate` object in one of two ways:

```
TeacherDate date = new TeacherDate(year, month, day);  // represents specific date
TeacherDate today = new TeacherDate();                 // represents today
```

You may assume that the user types valid input: the user will type integers, and they will be within the proper ranges (year will be at least 1753, month will be between 1 and 12, and day will be between 1 and the number of days in that month, and that the user's birth date will be a date no later than today (the user will not claim to have been born in the future)). You may also assume that the user's birthday is not on the "leap day" of February 29, because this is an odd case, since that date only occurs approximately once every four years. (Leap day babies usually celebrate their birthdays on February 28 or March 1 on non-leap years).

Part 2: Date class

The second part of this assignment asks you to implement a class named `Date`. Your `Date` class should implement the following behavior. None of the methods below should print any output to the console. You may not call any methods or utilize any behavior from the `TeacherDate` class to help implement your `Date` class (other than `getDaysSinceEpoch`, see below), nor use any of Java's date-related objects such as `GregorianCalendar`.

Methods you must implement in your `Date` class
(these methods are also already present in the `TeacherDate` class):

`public Date(int year, int month, int day)`
Constructs a new Date representing the given year, month, and day. You may assume that the parameter values are valid. Since the modern Gregorian calendar was established in late 1752, you may assume that the year parameter value will be greater than or equal to 1753.

`public Date()`
Constructs a new Date representing today (the date at which the program is run).

It may be helpful for you to know that the following expression returns the number of days that have elapsed since January 1, 1970. (You must import `java.util.*;` to be able to refer to `TimeZone`.)

```
int daysSinceEpoch = (int) ((System.currentTimeMillis() +
    TimeZone.getDefault().getRawOffset()) / 1000 / 60 / 60 / 24);
```

Or, rather than the above complicated expression, you may call the method `TeacherDate.getDaysSinceEpoch()` to see how many days have elapsed since 1/1/1970. (This is the ONLY place where your `Date` code may refer to `TeacherDate`.)

```
int daysSinceEpoch = TeacherDate.getDaysSinceEpoch();
```

`public int getDay()`
Returns this Date's day of the month, between 1 and the number of days in that month (which will be between 28 and 31).

`public int getMonth()`
Returns this Date's month of the year, between 1 and 12.

`public int getYear()`
Returns this Date's year.

`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 7, 2005 fell on a Sunday, so the return value for a new Date(2005, 8, 7) object would be "Sunday". It may be helpful for you to know that January 1, 1753 was a Monday.

`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()`
Advances this Date to the next day after its current day. Note that this might advance the Date into the next month or year. For example, the next day after August 7 is August 8; the next day after December 31 is January 1; the next day after February 28, 2005 is March 1, 2005, and the next day after February 28, 2004 is February 29, 2004 (because 2004 is a leap year).

`public String toString()`
Returns a String representation of this Date, in a *year/month/day* format to match the following: "2005/5/24"

`public boolean equals(Object o)`
Returns whether the given object is a Date that refers to the same year/month/day as this Date.

Stylistic Guidelines:

A major portion of this assignment is to demonstrate your understanding of using objects and defining new types of objects. Therefore, for `Birthday` program, you should implement the expected behavior using `TeacherDate` objects. For the `Date` program, you should implement your `Date` as a new type of objects, using non-static methods and non-static data fields as appropriate.

We will be grading on your appropriate use of control structures like loops and if/else statements, your ability to avoid redundancy and your ability to break down the overall problem into methods that each solve part of the overall problem. In the `Birthday` program you should have at least **2 static methods other than main** to solve the problem. No one method should be overly long, and each method should perform a coherent task. Your main method should still contain the overall control flow of the program.

You should keep in mind the ideas we have been stressing in class. You do not want to have redundant code. You do not want to have any method be overly long. You want to break the problem down into logical sub-problems so that someone reading your code can see the sequence of steps it is performing.

For your `Date` class, you should properly encapsulate your `Date` objects.

Follow general stylistic guidelines, such as indentation and whitespace, meaningful identifier names, and localization of variables.

Include a comment at the beginning of your program with basic information and a description of the program and include a comment at the start of each method. Also put brief comments inside longer methods explaining the more complex sections of code.

Hints and Suggestions:

Complete your `Birthday` program before you start your own `Date` class, to get a good understanding of how `Date` objects work. Write your `Date` class in phases:

- Write the `Date(year,month,day)` constructor, `getDay`, `getMonth`, `getYear`, `toString`, equals methods first.
- Then implement `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`.)
- Lastly, write `getDayOfWeek` and the `Date`() 'today' constructor. You can use your `nextDay` method to help you write these methods.

You can test your `Date` class by creating a modified version of your Birthday program that uses your Date instead of `TeacherDate`.

Note: How to add the .class file to your project

1. Right click on the project folder
2. Go to "Build Path" → "Configure Build Path"
3. A window will open, select the tab "Libraries" and click on "Add External Class Folder" and choose the folder that contains the `TeacherDate.class` file. Click "OK".

Submission:

Your Java source code (program) should be submitted via Latte the day it is due.

For late policy check the syllabus.

Grading:

You will be graded on

- **External Correctness:** The output of your program should match exactly what is expected. Programs that do not compile will not receive points for external correctness.
- **Internal Correctness:** Your source code should follow the stylistic guidelines shown in class. Also, remember to include the comment header at the beginning of your program.
- **One-on-one interactive grading:** By the end of the day that the assignment is due, please make an appointment with your TA for an interactive 10-15 minute grading session. You will receive an email notifying you of the name of the TA who has been assigned to you for this assignment with further instructions on setting up the appointment. (You will be meeting with a different TA for each assignment). One-on-one interactive grading will help you improve your programming skills and avoid repeating mistakes from one assignment to the next.