CSE 142, Autumn 2008
Programming Assignment #8: Birthday/Date (12 points)
Due Tuesday, November 25, 2008, 11:30 PM

This program focuses on classes and objects. Turn in two files named `Birthday.java` and `Date.java`. You will also need the support file `Date.class` from the course web site; place it in the same folder as your program.

The assignment has two parts: a client program that uses `Date` objects, and a `Date` class of your own whose objects represent calendar dates. It is meant to be somewhat shorter than other recent assignments and is also worth fewer points.

Part A (`Birthday.java`, client program):

The first part of this assignment asks you to write a client program that uses an existing `Date` class written by the instructor. The point of Part A is to give you a bit of practice creating and using `Date` objects from a client's perspective and to give you an appreciation for the usefulness `Date` objects in general.

Begin by prompting the user for today's date and for his/her birthday, each as a month-day pair. Use this information to print the number of days in the month the user was born, and the number of days from today to the user's birthday. If the user's birthday is today, print a Happy Birthday message.

Below are several example logs of execution from the program; user input is bold and underlined. Your program's output should match these examples exactly given the same input. See the course web site for more logs with other input.

```
What is today's date (month and day)? 11 4
What is your birthday (month and day)? 9 9
There are 30 days in month #9
Your birthday 9/09 is in 309 days.

<< your birthday fact >>
```

```
What is today's date (month and day)? 12 15
What is your birthday (month and day)? 12 15
There are 31 days in month #12
Happy birthday!

<< your birthday fact >>
```

```
What is today's date (month and day)? 8 19
What is your birthday (month and day)? 11 30
There are 30 days in month #11
Your birthday 11/30 is in 103 days.

<< your birthday fact >>
```

```
What is today's date (month and day)? 10 2
What is your birthday (month and day)? 10 1
There are 31 days in month #10
Your birthday 10/01 is in 364 days.

<< your birthday fact >>
```

The end of the program's output prints a "fun fact" about your own birthday. This can be any message you like, as long is it is nonempty, of reasonable length, and not overly offensive. Try searching Wikipedia and Google for facts about your date of birth. For example, if your birthday is Sep. 19, you could print the following:

```
Did ye know?  Marty's birthday of 9/19 also be International Talk like a Pirate Day.
Arr, me mateys, arr!
```

Solve this problem using `Date` objects. The methods and behavior of each `Date` object are described on the next page. For Part A you can use an instructor-provided version of `Date` by downloading the file `Date.class` from the web site and saving it to the same folder as your `Birthday.java` file. You can construct a `Date` object as follows:

```
Date name = new Date(month, day);
```

To figure out the number of days until the user's birthday, represent today and the birthday as `Date` objects. By advancing one date until it reaches the other and counting, you can determine the number of days between them.

You do not have to worry about leap years at all for this assignment. Assume that February always has 28 days and that the year is always 365 days long.

(A leap year occurs roughly every 4 years and adds a 29th day to February, making the year 366 days long. But we will ignore that possibility for this assignment. You may also assume that neither today nor the user's birthday is the rare "leap day" of February 29. Leap day babies usually celebrate their birthdays on February 28 or March 1 on non-leap years.)

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

Part B (`Date.java`, class of objects):

The second part of this assignment asks you to implement a class named `Date`, stored in a second file named `Date.java`. Your `Date` class should implement the following behavior:

- `public Date(int month, int day)`
  Constructs a new `Date` representing the given month and day. You may assume that the parameter values passed are valid.

- `public int getMonth()`
  This method should return a `Date` object's month of the year, between 1 and 12. For example, if the Date object represents August 17, this method should return 8.

- `public int getDay()`
  This method should return a `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 the Date object represents August 17, this method should return 17.

- `public String toString()`
  This method should return a `String` representation of a `Date` in a *month/day* format. The day should always be shown as two digits; if the day is between 1 and 9, it should be preceded by a 0. For example, if the `Date` object represents March 24, return `"5/24"`. If this `Date` object represents December 3, return `"12/03"`. Note that this method *returns* the string; it does not print any output.

- `public int daysInMonth()`
  This method should return the number of days in the month represented by a `Date` object. The following table lists the number of days in each of the twelve months of the year:

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Name	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Days	31	28	31	30	31	30	31	31	30	31	30	31

  For example, if the `Date` object represents August 17, this method should return 31. If this `Date` object represents February 14, you should return 28. You do not need to worry about leap years.

- `public void nextDay()`
  Modifies the state of a `Date` object by advancing it 1 day in time. For example, if the `Date` object represents September 19, a call to this method should modify the `Date` object's state so that it represents September 20. Note that depending on the date, a call to this method might advance the `Date` object into the next month or year. For example, the next day after August 17 is August 18; the next day after February 28 is March 1; and the next day after December 31 is January 1.

None of the methods listed above should print any output to the console. You may not utilize any behavior from the instructor-provided `Date` class to help implement your own `Date` class, nor use any of Java's date-related classes such as `java.util.GregorianCalendar` or `java.util.Date`.

You can test your `Date` program by running your `Birthday.java` program from Part A with it. By compiling your `Date.java` file you will overwrite our provided `Date.class` with your own. (If necessary you can always revert to our `Date.class` by re-downloading it or backing it up.) `Birthday.java` is not a great testing program; it might not call all of your `Date` methods or may not call them in a very exhaustive way that tests all cases and combinations. Therefore you may want to create another small client program of your own to help test other aspects of your `Date` class's behavior.

You may put additional behavior in your `Date` class if you like, but we will still test your `Birthday` program with the instructor-provided `Date` class, so it should still run correctly with that class and not only when used with your `Date`.

Development Strategy and Hints:

Complete Part A before Part B, to get a good understanding of how `Date` objects work from the client's perspective.

Write Part B in phases:

- Write the constructor and `getDay` and `getMonth` methods first.
- Then implement `toString` and `daysInMonth`.
- Last, write `nextDay`.

We encourage you to build your `Date` class incrementally, writing a small amount of code at a time and testing it. It is possible to test an incomplete `Date` class by writing some of its methods and then creating a small client program to call just those methods.

Recall that code in one of an object's methods is able to call any of the object's other methods if so desired. Specifically, when implementing `nextDay` you may want to consider calling other methods within the `Date` object to help you.

Since objects can be difficult to visualize and understand, we strongly recommend that you use the jGRASP debugger to step through your code to understand each method's behavior, especially in Part B. If you are stuck on a particular method, we also recommend using temporary debugging `println` statements from inside the `Date` class to see what is going on. For example, printing the state of the current `Date` object from inside the `daysInMonth` or `nextDay` method can help you find bugs.

You may want to compare two `Date` objects to see if they represent the same month and day. Normally you can compare objects for equality by calling the `equals` method. But the `Date` class does not have its own `equals` method, so the behavior will be incorrect. You will have to manually compare dates by examining their state of months and days.

There is a method called `String.format` that you might find helpful to optionally use on this assignment. It behaves just like `System.out.printf` but returns a formatted string rather than printing it. For example:

```
double tuition = 23456.7;
String formattedTuition = String.format("My tuition is $%.2f per quarter", tuition);

// formattedTuition stores "My tuition is $23456.70 per quarter"
```

Style Guidelines:

For Part A, you are to solve the problem by creating and using `Date` objects as much as possible. This is because a major goal of this assignment is to demonstrate understanding of using objects and defining new classes of objects. Part A is not required to have any static methods besides `main`, though you may have additional methods if you like.

For Part B, implement your `Date` as a new type of objects, 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.

On both Parts of the assignment, you should follow general past style guidelines such as: appropriately using control structures like loops and if/else statements; avoiding redundancy using techniques such as methods, loops, and `if/else` factoring; properly using indentation, names, types, variables; and not having lines longer than 100 characters. You should properly comment your code with a proper heading in each file, a description on top of each method, and on any complex sections of your code. Specifically, place a comment heading at the top of each method of the `Date` class, written in your own words, describing that method's behavior, parameters, and return values if any.

You are limited to features in Chapters 1 through 8. For reference, our `Birthday.java` solution is around 30-40 lines including comments, and our `Date.java` solution is around 50-60 lines.