University of Washington
Computer Science & Engineering 142 (Computer Programming I), Summer 2005

Programming Assignment #8 (Birthday and Date), 20 points
Due: Tuesday, 8/16/2005, 6:00 PM

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 files `Scanner.java` and `TeacherDate.class` from the course web site. Place both of these files in the same folder as your program. Hand in your two program files electronically from the "Assignments" link on the course web page.

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 A (Birthday program):

The first part of this assignment asks you to write a 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 birthdate, 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 Sunday, August 7, 2005. If you run on a different date, you will receive slightly different output.

Example logs of execution (user input underlined):

```
What year, month, and day were you born? 1987 5 8
You were born on 1987/5/8, which was a Friday.
It will be your birthday in 274 days.
```

```
What year, month, and day were you born? 1952 10 8
You were born on 1952/10/8, which was a Wednesday.
It was also a leap year.
It will be your birthday in 62 days.
```

```
What year, month, and day were you born? 1973 8 7
You were born on 1973/8/7, which was a Tuesday.
Happy birthday!  You are now 32 years old.
```

You should implement this program using the `TeacherDate` class available on the course web page. A `TeacherDate` contains all of the methods and behavior described on the next page, exactly matching the Date class you will implement in Part B.

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 >= 1753, month will be between 1 and 12, and day will be between 1 and the number of days in that month).

Part B (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 of the `TeacherDate` class to help implement your `Date` class.

Methods you must implement in your `Date` class (these methods are 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:

```
(int)(System.currentTimeMillis() / 1000 / 60 / 60 / 24)
```

`public boolean equals(Object o)`
Returns whether `o` refers to a `Date` object that represents the same year, month, and day as this one. You may assume that `o` refers to a `Date` object, and cast `o` into a `Date`.

`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 either `"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 and 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 format to match the following:
`"2005/5/24"`

Stylistic Guidelines:

A major portion of this assignment is to demonstrate your understanding of using objects and defining new types of objects. Therefore, for Part A, you should implement the expected behavior using TeacherDate objects. For Part B, 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 Part A, even though the program may be shorter than previous programs, you should still use at least **2 methods other than `main`** to solve the problem. No one method should be overly long. The `main` method in particular should not directly perform file input/output; perform these tasks in other methods. The `main` method should serve as an outline of the behavior of the entire program, showing the major methods that are called.

You should keep in mind the ideas we have been stressing all quarter. You do not want to have redundant code. You do not want to have any one 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.

In Part B, you should use **global constants for the number of days per week (7), the number of days per year (365), and the number of days per leap year (366)** in your Date class. It is unlikely that these constants' values would ever change, but the constant is helpful for documentation purposes, even though it will not make the program particularly flexible. You may wish to have other constants to avoid otherwise "magic" numbers and values in your code.

In Part B, you should properly encapsulate your `Date` objects by making sure that their methods and constructors are `public` and their fields are `private`.

You are required to properly indent your code and will lose points if you make significant indentation mistakes. See section 2.5.3 of the book for an explanation and examples of proper indentation. You should also use white space to make your program more readable, such as between operators and their operands, between parameters, and blank lines between groups of statements or methods.

Give meaningful names to methods and variables in your code. Follow Java's naming standards about the format of ClassNames, methodAndVariableNames, and CONSTANT_NAMES. Localize variables whenever possible -- that is, declare them in the smallest scope in which they are needed.

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 the methods explaining the more complex sections of code.

Hints and Suggestions:

Do Part A before Part B, to get a good understanding of how `Date` objects work. Write Part B in phases:

- Write the `Date(year, month, day)` constructor, `getDay`, `getMonth`, `getYear`, `toString` methods first.
- Then implement `isLeapYear` and `equals`.
- 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 (you could call it something like `Birthday2` or `TestDate`) from Part A to use your `Date` instead of `TeacherDate`. (When you go to turn in your `Birthday` program, please make sure it is using `TeacherDate`.)