Homework 2

This homework assignment is designed to help you learn about building Java classes and to observe the decomposition of a complicated task into simple subtasks. This is an individual assignment; you may not share code with other students.

Your task is to fill in the implementation of a class that manipulates dates. Do not use any of the built-in operations on dates provided in the Java library in your solution. The overall task is broken down into subtasks, which we suggest you implement in a bottom-up order, so that you can easily test as you go. The grading test cases will give partial credit for the more basic operations, even if some of the higher level operations do not work properly.

Please observe these notes on grading.

1) Your program must compile without errors to receive any partial credit on this assignment. If only one or two of your methods work, remove any code that causes problems for "javac" before submitting your solution. However, don't remove any of the method declarations that appear in the skeletal Date.java we give you.
2) We have provided a main method in the Date class that tests some of your methods. You are welcome to modify the main method as you please, perhaps to add further tests of your own. We will not be testing or grading the main method in this assignment. (It does, of course, need to compile.)
3) You are welcome to add new methods to the Date class. Since they will presumably be "helping" methods, declare them "private", not "public".
4) Do not change the prototype (interface) of any method. If you change the arguments or the return type, or you change a method from static to non-static, your program will not compile with our test cases, and will not receive credit.
5) Do not have any extraneous print statements in your program, including error messages. Your program should print out exactly what is specified and nothing else. (If the comment prefixing a method does not mention printing, the method should not print anything.) The only exception here is the main method, which can do anything you like, so long as it compiles.
6) Although some test cases are provided in the main method, we will add trickier ones to our grading test suite, which won't be run until _after_ the due date. It is your responsibility to ensure that your methods work correctly on any input, not just the test cases. So you might want to add more tests.

The file Date.java contains a skeleton, plus some test code, for a Date class. Your job is to fill in the implementations of the methods. We have specified most or all of the methods you'll need, including some helper methods.

Part I

Implement the basic helper methods listed below. These methods, like the main method, are declared "static." They are also declared "public" so we can test them from another class. Don't change that.

The Unix "cal" command will remind you of the number of days in each month. February contains 28 days most years, but 29 days during a leap year. A leap year is any year divisible by 4, except that a year divisible by 100 is not a leap year, except that a year divisible by 400 is a leap year after all. Hence, 1800 and 1900 are not leap years, but 1600 and 2000 are. (Implement this rule in your program even if you know information to the contrary.)

```
  /** Checks whether the given year is a leap year.
   *  @return true if and only if the input year is a leap year.
   */
  public static boolean isLeapYear(int year) {
    ...
  }

  /** Returns the number of days in a given month.
   *  @param month is a month, numbered in the range 1...12.
   *  @param year is the year in question, with no digits omitted.
   *  @return the number of days in the given month.
   */
  public static int daysInMonth(int month, int year) {
    ...
  }

  /** Checks whether the given date is valid.
   *  @return true if and only if month/day/year constitute a valid date.
   *
   *  Years prior to A.D. 1 are NOT valid.
   */
  public static boolean isValidDate(int month, int day, int year) {
    ...
  }
```

Part II

Define the internal state that a "Date" object needs to have by declaring some data fields (all private) within the Date class. Define the basic constructor specified below. A Date should be constructed only if the date is valid. If a caller attempts to construct an invalid date, the program should halt after printing an error message of your choosing. To halt the program, include the line:

```
  System.exit(0);


  /** Constructs a date with the given month, day and year.   If the date is
   *  not valid, the entire program will halt with an error message.
   *  @param month is a month, numbered in the range 1...12.
   *  @param day is between 1 and the number of days in the given month.
   *  @param year is the year in question, with no digits omitted.
   */
  public Date(int month, int day, int year) {
    ...
  }

  /** Returns a string representation of this date in the form month/day/year.
   *  The month, day, and year are expressed in full as integers; for example,
   *  12/7/2006 or 3/21/407.
   *  @return a String representation of this date.
   */
  public String toString() {
    ...
  }
```

Part III

Implement the following methods.

```
  /** Determines whether this Date is before the Date d.
   *  @return true if and only if this Date is before d.
   */
  public boolean isBefore(Date d) {
    ...
  }

  /** Determines whether this Date is after the Date d.
   *  @return true if and only if this Date is after d.
   */
  public boolean isAfter(Date d) {
    ...
  }

  /** Returns the number of this Date in the year.
   *  @return a number n in the range 1...366, inclusive, such that this Date
   *  is the nth day of its year.  (366 is used only for December 31 in a leap
   *  year.)
   */
  public int dayInYear() {
    ...
  }

  /** Determines the difference in days between d and this Date.  For example,
   *  if this Date is 12/15/2012 and d is 12/14/2012, the difference is 1.
   *  If this Date occurs before d, the result is negative.
   *  @return the difference in days between d and this date.
   */
  public int difference(Date d) {
    ...
  }
```

Hint 1: once you've implemented isBefore(), it's possible to implement isAfter() with just one line of code. You need to think carefully, though: "return !isBefore(d)" is incorrect. Can you see why?

Hint 2: all the methods in the Date class can read all the private fields in _any_ Date object (not just "this" Date object).

Part IV

Implement the final missing piece of your class, a second constructor that takes a String argument.

```
  /** Constructs a Date object corresponding to the given string.
   *  @param s should be a string of the form "month/day/year" where month must
   *  be one or two digits, day must be one or two digits, and year must be
   *  between 1 and 4 digits.  If s does not match these requirements or is not
   *  a valid date, the program halts with an error message of your choice.
   */
  public Date(String s) {
    ...
  }
```

We're flexible on how you handle dates that are "almost correct". For example, the string " 011/4/2010 AD" is technically not valid because of the spaces and letters and leading zero, but it's your choice whether you treat it the same as "11/4/2010" or halt with an error message. We won't be pedantic about this or make it gratuitously difficult, so please don't worry about these "edge cases" (and please don't ask tons of clarification questions on Piazza--we're not going to have trick test cases).

But your Date constructor definitely should not accept "11/31/2009" or "12/4" or "hey dude". These aren't ambiguous cases; they're clearly wrong.

Hint: use the online Java API to familiarize yourself with all the methods available to you in the String class.

Submitting your solution

WARNING: make sure your code _compiles_ and _runs before you submit it.

After submitting, if you realize your solution is flawed, you may fix it and submit again. You may submit as often as you like. Only the last version you submit before the deadline will be graded.