CSc 127B — Introduction to Computer Science II
Fall 2015 (McCann)

http://www.cs.arizona.edu/classes/cs127b/fall15/

Program #1: The CalendarDate Class

Due Date: Thursday, September 3rd, 2015, at 9:00 p.m. MST

Overview: It will take us a little while to cover enough new material to have a 'real' programming assignment. In the meantime, I want to give you a chance to demonstrate that you remember your Java programming skills.

Assignment: You will create a new Java class called `CalendarDate` in a file named `CalendarDate.java`. Each `CalendarDate` object knows the date it represents – year, month, and day – and allows various instance methods to work with and manipulate that state information.

`CalendarDate` has one constructor:

- `public CalendarDate (int year, int month, int day)` — creates a `CalendarDate` object that represents the given date. If the month is out of range (that is, below 1 or above 12), month is assumed to be 1 or 12, respectively. Similarly, if the day is out of range for the month, set it to the closest legal date for that month. Thus, `new CalendarDate(2014,13,-6)` would create a `CalendarDate` object representing the date December 1, 2014. Britain, and thus the U.S.-to-be, adopted the Gregorian calendar in the middle of 1752. Thus, years less than 1753 are to be set to 1753.

The `CalendarDate` class includes implementations of the following instance methods:

- `public void setDate (int year, int month, int day)` — resets this object's date to the one specified by the parameters. The same parameter range rules apply as for the constructor (see above).
- `public int getYear()` — returns the integer value of this object's year.
- `public int getMonth()` — returns the integer value of this object's month.
- `public String getMonthAsString()` — returns a reference to a `String` representation ("January", "February", etc.) of this object's month.
- `public int getDay()` — returns the integer value of the object's day.
- `public String toString()` — returns a reference to a String representation of the date, in 'month day, year' order, where the month is the name of the month. Thus, a `CalendarDate` object created with `new CalendarDate(2012,6,13)` would return the string "June 13, 2012" from a call to `toString()`.
- `public boolean equals (CalendarDate otherDate)` — returns true if this object and the `CalendarDate` object referenced by the formal parameter both represent the same date.
- `public CalendarDate tomorrow()` — returns a reference to a new `CalendarDate` object that represents the day after this object's day. For example, if this object is currently representing December 31, 1999, `tomorrow()` will return a reference to a `CalendarDate` object representing January 1, 2000.

  For `tomorrow()` to work correctly, it must recognize and deal with leap years to get the correct number of days in February. Years that are not evenly divisible by 4 are never leap years. This year, 2015, is not a leap year for that reason. Years that are evenly divisible by 4 are *usually* leap years. The exception is when the year is also divisible by 100 but not by 400; such years are not leap years.

  Examples really help here. 2016 will be a leap year; 2016 is evenly divisible by 4, but not by 100, so the exception doesn't apply. The year 2000 was also a leap year; 2000 is divisible by 4, 100 and 400; again, the exception did not apply. The exception did apply in 1900; it is divisible by 4 and 100 but not by 400, meaning that 1900, although it is evenly divisible by 4, was not a leap year. 2100 won't be a leap year, either, for the same reason.

(Continued...)

Answers to some expected questions:

- *Can we use any of Java's calendar or date classes?* No! You need to do the 'dirty work' yourself.
- *Can we use Strings, StringBuffers, StringBuilders, and/or 1-D arrays?* Yes! For example, an array of month name strings would be helpful in the `CalendarDate` class.
- *Can we use regular expressions?* No!
- *Wait a minute; what the heck is a 'regular expression'?* Don't worry about it; just don't use any methods that take regular expressions as arguments. You won't use one by accident, we promise.

Data: In the first section activity, you completed two methods and ran a testing program we provided to test that your methods worked correctly. We will be providing a similar testing program for your `CalendarDate` class methods, but we won't be providing it until next week. This is not an acceptable excuse to wait until next week to start! Until the test program is ready, do your own testing!

Output: Because the `CalendarDate` class does not produce any output of its own, you don't need to worry about meeting any output specifications on this assignment.

Turn In: Use 'turnin' to electronically submit your well-tested, completely-documented, and well-structured (`CalendarDate.java` file to the `cs127bsXp01` directory (replace the 'X' with your section's letter) at any time before the stated due date and time.

Want to Learn More?

- If you're curious about the classes Java supplies to work with dates, look at the API pages for the `Date` (the `java.util` version, not the `java.sql` version), `GregorianCalendar`, and `SimpleTimeZone` classes. Remember, you can't use these, or any other Java API time-related classes, in `CalendarDate`.

Hints and Reminders:

- If you feel that it would be helpful to write additional methods for `CalendarDate`, you may, but make them 'private' methods by starting them with the word `private` instead of the word `public`. Do not create any `static` methods. (If you don't know the differences between public and private methods, and between instance and static methods, don't worry; we'll be covering them in this class.)
- We know that writing documentation isn't a lot of fun, but it is important, which is why we allocate about 25% of the points on each programming assignment to code style and commenting. The best advice I can give you is this: Write the block comments first, then write the code (and, at the same time, any intra-code comments). Doing this makes the job *much* less tedious, and forces you to think before you code. Here's the web page with details on what we expect to see:
`http://www.cs.arizona.edu/people/mccann/styletemplates.html`
- When testing your `CalendarDate` class, keep in mind that, once the object has been created, all of the instance methods can be called in any order. Don't assume that a programmer will only use the object in one way.
- Speaking of testing: Test all of the methods, not just the hard-to-code ones. The getters and setters all need to work, too; test them! And, the `equals()` method is handy for testing the methods that return new `CalendarDate` objects.
- My usual final reminder: **Start early!** You may not have written any Java code in months. You'll want to be sure that you have enough time to both relearn what you've forgotten and complete this assignment.