CSc 227 — Program Design and Development
Spring 2014 (McCann)

http://www.cs.arizona.edu/classes/cs227/spring14/

Program #4: The CalendarDate Class

Due Date: Tuesday, February 25th, 2014, at 9:00 p.m. MST

Overview: The WordHelper assignment did not go as well as I'd expected. Writing classes and their components is a critical skill in this and other Computer Science classes. Rather than just move forward, I want to give you a chance to try writing a different, hopefully somewhat easier, class, in hopes that doing so will help you better grasp the ideas of objects and classes before we complicate them with additional ideas and functionalities.

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. If a negative year is provided, set the object's year to its absolute value.

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.
- `public CalendarDate yesterday()` — the opposite of `tomorrow()`; that is, `yesterday()` returns a reference to a new `CalendarDate` object representing the day before this object's day. For example, the day before January 1, 2000 is December 31, 1999.
- `public String dayOfTheWeek()` — returns a reference to a `String` object containing the name of the date's day of the week ("Sunday", "Monday", ...). Yes, you may re-use your code from `Prog2b` to help with this method.

(Continued...)

Answers to some expected questions:

- *Can we use regular expressions?* No!
- *Can we use String and StringBuilder methods that don't take regular expressions?* Yes!
- *Can we use 1-D arrays?* Yes! For example, an array of month name strings would be helpful in the `CalendarDate` class.
- *Can we use any of Java's calendar or date classes?* No! You need to do the 'dirty work' yourself.

Data: We are not giving you a sample testing program this time. However, we do expect you to write your own testing program (named `Prog4.java`), and to submit it along with `CalendarDate` as part of this assignment. We found on the WordHelper assignment that students were relying on the sample testing program almost exclusively. You need to get into the habit of writing your own tests, covering a wide variety of input values, to verify that your code works.

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

Turn In: Use the 'turnin' page to electronically submit both of your files (`CalendarDate.java` and `Prog4.java`) to the `cs227p04` directory 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:

- Feel free to add additional *private* instance methods to your class, but do not add any more *public* instance methods. Do not create any `static` methods.
- The section leaders report that one of the major deficiencies in the Program #3 submissions was documentation. As always, we want to see good documentation, indentation, variable names, etc. Remember to try to document as you code; that will make the job *much* less tedious. Here's the web page with details on what we expect to see:
  `http://www.cs.arizona.edu/people/mccann/styletemplates227.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.
- Finally, old news: **Start early!**