P01 Calendar Printer

Programming II (CS300) Fall 2019
Department of Computer Sciences
University of Wisconsin-Madison

Due: 9:59PM on September 11, 2019
Pair Programming is Allowed

Overview and Learning Objectives:

One of my first steps (as an instructor) in planning for a new semester is to print out a calendar and begin marking it with important dates: like holidays and the dates of our CS300 Exams. If you have not already, please add our exam dates to your own calendar. While doing this in August, it seemed that writing a program to help with this could be both useful and a great exercise for this first CS300 programming assignment.

Through the process of completing this assignment, you will brush up your structured programming skills using Java: practicing the use of control structures, custom static methods, and arrays. This assignment is meant to encourage you to think about the different ways that data can be represented within a computer program. You will also be developing test methods for your code, which is likely to be a new concept for many CS300 students.

Grading Rubric:

5 points	**Pre-Assignment Quiz:** Generally, you will not have access to this write-up without first completing this pre-assignment quiz through Canvas. If you have not already, please complete this quiz as soon as you have access to this course on Canvas.
15 points	**Immediate Automated Tests:** Upon submission of your assignment to Gradescope, you will receive feedback from automated grading tests about whether specific parts of your submission conform to this write-up specification. If these tests detect problems in your code, they will attempt to give you some feedback about the kind of defect that they noticed. Note that passing all of these tests does NOT mean your program is correct. To become more confident in this, you must write and run additional tests of your own.
15 points	**Manual Grading Feedback:** After the deadline for an assignment has passed, the course staff will begin manually grading your submission. We will focus on looking at your algorithms, use of programming constructs, and the style and readability of your code. This grading usually takes about a week from the hard deadline, after which you will find feedback on Gradescope.
15 points	**Supplemental Automated Tests:** When your manual grading feedback appears on Gradescope, you will also see the feedback from these additional automated grading tests. These tests are similar to the Immediate Automated Tests, but check your code against different requirements within this specification.
50 points	**TOTAL**

Additional Assignment Requirements:

The ONLY import statement that you may include in this assignment is **import java.util.Scanner;**

You **CANNOT** make use of java.util.Date, or any classes outside java.lang and the default package.

0. First Assignment Advice, which also applies to future CS300 Assignments.

0.1. Tools and Java Version

If you plan to use your own computer, we encourage you to install the Eclipse IDE with Java 8 for use throughout this course. Our assignment write-ups and TA support assume the use of these tools. We will only grade your source code, and do not prohibit the use of other IDEs. But we may not be able to provide support around the configuration or use of such other tools.

0.2. Available Computers and Backups

In addition to your own computer, students may find it helpful to work on Computers in the CS Labs, and Rental Laptops through either DoIT or UW Libraries. If you ever experience any kind of trouble with your own computer throughout the semester, our expectation is that you use another computer (such as those listed above) to complete your work on time.

We also recommend making regular backups of your progress as you work. You may save backups of your work to your Office 365 One Drive account. Or you can upload your work to Gradescope as early submissions. Some extra benefits of using Gradescope in this way include: 1) getting automated test feedback about your submission and confirmation that you saved the right thing in the right place, and 2) having something in Gradescope for partial credit, in case you miss the final submission deadline.

0.3. Course Style Guide

You should review the course style guide now, otherwise you are likely to lose points for submitting code that does not conform to these requirements on this and future programming assignments. At the end of this style guide are some helpful tips for configuring Eclipse to help with these requirements. Also, pay close attention to the requirements for commenting and the use of JavaDoc style comments in your code.

0.4. Pair Programming

Not every CS300 assignment allows pair programming. When pair programming is permitted, you must register your partner as a part of completing your pre-assignment quiz. If you do not declare a partner at that time, you may NOT work with a partner on the assignment. Partners may be other students enrolled in any CS300 lecture this semester, and may be different for each assignment.

Registered partners may submit their shared work on an assignment through either partner's account, and then add their team member as a partner on that submission using the "+ Add Group Member" button in Gradescope. This should result in an email notification to the partner of the submitted work.

If you ever wish to dissolve a partnership for any reason, or if you unexpectedly receive a partner notification from Gradescope, please email your instructor immediately, to let them know.

0.5. Academic Conduct Expectations

All assignment submissions must be representations of your own individual work, or of work that was collaboratively developed by both you and your registered partner (when partners are allowed).

You are encouraged to discuss general algorithms, behavior of java constructs, and debugging strategies with anyone. But you are NOT allowed to view or share any code or solutions (buggy, complete, or in part) with anyone outside of the CS300 course staff. AVOID leaving any of your work anywhere that is accessible to others: publicly accessible websites, computers, storage devices, etc.

1. Project Files and Setup

Create a new Java Project in Eclipse called: P01 Calendar Printer. Then create two Java classes / source files within that project's src folder (both inside the default package) called CalendarPrinter and CalendarTester. Each of these classes should include their own main method. For a reminder or introduction to creating projects with source files in Eclipse, please see these instructions.

Appendix II at the end of this assignment lists some methods in the Java API that are likely to be helpful in this assignment. Please make sure you are familiar with these methods before you begin.

2. Creating First Test Method

To practice good structured programming, we will be organizing our CalendarPrinter implementation into several easy-to-digest sized methods. We want to test these methods before writing code that makes use of calling them. When we do find bugs in the future, we will add additional tests to demonstrate whether those defects exist in our code. This will help us see when a bug is fixed, and it will help us notice if similar bugs surface or return in the future.

Here are the JavaDoc style comments and signature for the first method for which we would like to write a test. Remember that we will be implementing this public static method inside our CalendarPrinter class, but only after we finish writing its test method inside CalendarTester.

```
/**
 * Calculates the number of centuries (rounded down) that is represented by
 * the specified year (ie. the integer part of year/100).
 * @param year to compute the century of (based on the Gregorian Calendar AD)
 *   String must contain the digits of a single non-negative int for year.
 * @return number of centuries in the specified year
 */
public static int getCentury(String year)
```

A common trap when writing tests is to make the test code as complex or even more complex than the code that it is meant to test. This can lead to there being more bugs and more development time required for testing code, than for the code being tested. To avoid this trap, we aim to make our test code as dead-simple as possible. You can copy these getCentury() tests into your CalendarTester class as an example.

```
public static boolean testGetCentury() {
  if(CalendarPrinter.getCentury("2") != 0) return false;
  if(CalendarPrinter.getCentury("2019") != 20) return false;
  if(CalendarPrinter.getCentury("44444") != 444) return false;
  return true;
}
```

You should strive to keep all of your tests as simple as possible. If you would like to print out more specific feedback when tests fail (before returning false), that can also be helpful. Notice that this test method does not test the behavior when a string like "year of independence" is passed to the method. This is because the method specifications indicates that this method only works correctly when it is passed a string containing a non-negative int. Since we have no specification or expectation for what should happen under other circumstances, we should not bother writing tests for such circumstances.

In CS300, we generally will NOT worry about the possibility of overflowing ints in cases like this either, unless the write-up specification explicitly mentions such concerns.

Notice how this test helps clarify the requirements and expected behavior of the getCentury() method. When you encounter a problem or question about your code, start by creating a test like this to verify your understanding of what is happening, versus what should be happening when your code runs. Sharing tests like this with course staff who are helping you throughout the semester is a great way to help the course staff help you more efficiently. If you haven't already, you should now implement the getCentury() method, and make sure that it passes your own test.

Now is a great time to try submitting your assignment for automated feedback on your progress. Skip down to step 5 for instructions to do this. You can expect several automated tests to fail right now (don't worry, there is no grade penalty), since there are still many un-implemented features in your code. As you complete this assignment, continue to make periodic submissions to back-up your work, to get feedback on your progress, and to improve your recorded grade.

3. Additional Helper Methods to Implement and to Test

The next helper methods that you will implement and test will benefit from making use of the following constants. Define these constants near the top of your CalendarPrinter class (outside of all methods):

```
private final static String[] DAYS_OF_WEEK = {"MON", "TUE", "WED", "THU", "FRI",
    "SAT", "SUN"};
private final static String[] MONTHS_OF_YEAR = {"JAN", "FEB", "MAR", "APR", "MAY",
    "JUN", "JUL", "AUG", "SEP", "OCT", "NOV", "DEC"};
```

Below are JavaDoc method headers and signatures for the helper methods that you must implement and test, similar to the getCentury() method described in the previous step.

```
/**
 * Calculates the number of years between the specified year, and the first
 * year in the specified year's century.  This number is always between 0 - 99.
 * @param year to compute the year within century of (Gregorian Calendar AD)
 *   String must contain the digits of a single non-negative int for year.
 * @return number of years since first year in the current century
 */
public static int getYearWithinCentury(String year)

/**
 * This method computes whether the specified year is a leap year or not.
 * @param yearString is the year that is being checked for leap-year-ness
 *   String must contain the digits of a single non-negative int for year.
 * @return true when the specified year is a leap year, and false otherwise
 */
public static boolean getIsLeapYear(String yearString)
// Note implementation tips in Appendix I below.

/**
 * Converts the name or abbreviation for any month into the index of that
 * month's abbreviation within MONTHS_OF_YEAR.  Matches the specified month
 * based only on the first three characters, and is case in-sensitive.
 * @param month which may or may not be abbreviated to 3 or more characters
 * @return the index within MONTHS_OF_YEAR that a match is found at
```

```
 *         and returns -1, when no match is found
 */
public static int getMonthIndex(String month)

/**
 * Calculates the number of days in the specified month, while taking into
 * consideration whether or not the specified year is a leap year.
 * @param month which may or may not be abbreviated to 3 or more characters
 * @param year of month that days are being counted for (Gregorian Calendar AD)
 *   String must contain the digits of a single non-negative int for year.
 * @return the number of days in the specified month (between 28-31)
 */
public static int getNumberOfDaysInMonth(String month, String year)

/**
 * Calculates the index of the first day of the week in a specified month.
 * The index returned corresponds to position of this first day of the week
 * within the DAYS_OF_WEEK class field.
 * @param month which may or may not be abbreviated to 3 or more characters
 * @param year of month to determine the first day from (Gregorian Calendar AD)
 *   String must contain the digits of a single non-negative int for year.
 * @return index within DAYS_OF_WEEK of specified month's first day
 */
public static int getFirstDayOfWeekInMonth(String month, String year)
// Note implementation tips in Appendix I below.

/**
 * Creates and initializes a 2D String array to reflect the specified month.
 * The first row of this array [0] should contain labels representing the days
 * of the week, starting with Monday, as abbreviated in DAYS_OF_WEEK.  Every
 * later row should contain dates under the corresponding days of week.
 * Entries with no corresponding date in the current month should be filled
 * with a single period.  There should not be any extra rows that are either
 * blank, unused, or completely filled with periods.
 * For example, the contents for September of 2019 should look as follows,
 * where each horizontal row is stored in different array within the 2d result:
 *
 * MON TUE WED THU FRI SAT SUN
 *   .   .   .   .   .   .   1
 *   2   3   4   5   6   7   8
 *   9   10  11  12  13  14  15
 *   16  17  18  19  20  21  22
 *   23  24  25  26  27  28  29
 *   30  .   .   .   .   .   .
 *
 * @param month which may or may not be abbreviated to 3 or more characters
 * @param year of month generate calendar for (Gregorian Calendar AD)
 *   String must contain the digits of a single non-negative int for year.
 * @return 2d array of strings depicting the contents of a calendar
 */
public static String[][] generateCalendar(String month, String year)
```

In addition to implementing each of these methods, be sure to create a different test method for each. Each of these tests should call the specific method that they are testing AT LEAST THREE TIMES with

different inputs. In order for our automated grading tests to find and call these methods, be sure that each of these methods are named testX where X is the UpperCamelCase name of the method being tested (as you saw between getCentury and testGetCentury). It is also important that all seven of these test methods are defined within your CalendarTester class as public static methods that take no arguments and return a boolean: true when a test passes and false otherwise.

Note that the String[][] returned by generateCalendar contains a lot of data. Rather than comparing the output of this method against a hard coded copy of the expected output, you are welcome to check whether a few representative samples from the output are correct. You could check whether a specific date or two are stored in the correct positions, check a day of week label, check the size of the array dimensions, etc.

4. Creating a Driver Application

Using the helper methods that you have previously implemented and tested, you will now develop a console application to display the calendar for any month of any year. Here is a log from a sample run, which demonstrates how your application should look:

```
Welcome to the Calendar Printer.
================================
Enter the month to print: Februruru-ary
Enter the year to print: 2020
MON TUE WED THU FRI SAT SUN
 .   .   .   .   .   1   2
 3   4   5   6   7   8   9
 10  11  12  13  14  15  16
 17  18  19  20  21  22  23
 24  25  26  27  28  29  .
================================
Thanks, and have a nice day.
```

Aim to make your program's output match the example above as closely as possible. Our automated grading tests will NOT detect differences in spacing on any given line, but it will be good practice to match this output as closely as you are able. Your program will NOT be tested with any invalid user inputs, so there is no need to worry about that. Note that "Februruru-ary" is a valid month, since the helper methods above key off only the first three letters.

You may organize the functionality of this driver application into methods of your own choosing and design. But avoid designing overly complex methods. This specific application code should be broken down into sub-tasks that are organized across at least three different methods. All of these driver methods should be defined within your CalendarPrinter class, and executed by running main.

5. Assignment Submission

Congratulations on completing your first CS300 assignment! The only two files that you should submit for this assignment are CalendarPrinter.java and CalendarTester.java. If you do not already have a gradescope.com account or cannot find your password for it, please use your wisc.edu email address to update your password using this link: https://www.gradescope.com/reset_password. Your score for this assignment is based on your submission that is marked "active" and is made prior to the hard deadline.

6. Appendix I: Implementation Tips for getIsLeapYear, and getFirstDayOfWeekInMonth

6.1. Tips for calculating getIsLeapYear

The following excerpt from Wikipedia is preserved here for use in this assignment (https://en.wikipedia.org/wiki/Leap_year#Algorithm):

The following pseudocode determines whether a year is a leap year or a common year in the Gregorian calendar (and in the proleptic Gregorian calendar before 1582). The year variable being tested is the integer representing the number of the year in the Gregorian calendar.

if (year is not divisible by 4) then (it is a common year)
else if (year is not divisible by 100) then (it is a leap year)
else if (year is not divisible by 400) then (it is a common year)
else (it is a leap year)

6.2. Tips for calculating getFirstDayOfWeekInMonth

The following excerpt from Wikipedia is preserved here for use in this assignment (https://en.wikipedia.org/wiki/Zeller%27s_congruence#Implementation_in_software). Note that this formula is more general than is needed for this assignment, because it calculates the day of week for any day of any month, whereas your program only ever needs to compute the first day of the month:

$$h = \left(q + \left\lfloor \frac{13(m+1)}{5} \right\rfloor + K + \left\lfloor \frac{K}{4} \right\rfloor + \left\lfloor \frac{J}{4} \right\rfloor + 5J \right) \mod 7,$$

where

- h is the day of the week (0 = Saturday, 1 = Sunday, 2 = Monday, ..., 6 = Friday)
- q is the day of the month
- m is the month (3 = March, 4 = April, 5 = May, ..., 14 = February)
- K the year of the century (year mod 100).
- J is the zero-based century (the integer part of year/100) For example, the zero-based centuries for 1995 and 2000 are 19 and 20 respectively (to not be confused with the common ordinal century enumeration which indicates 20th for both cases).
- $\lfloor \cdots \rfloor$ is the floor function or integer part
- mod is the modulo operation or remainder after division

NOTE: In this algorithm January and February are counted as months 13 and 14 of the previous year. E.g. if it is 2 February 2010, the algorithm counts the date as the second day of the fourteenth month of 2009 (02/14/2009 in DD/MM/YYYY format)

7. Appendix II: Helpful Methods from the Java API

Integer.parseInt
PrintStream.print
Scanner.nextLine
String.equals
String.length
String.substring
String.toUpperCase