CSC110 Assignment 4: Decision Making

Objectives

Upon completion of this assignment, you need to be able to:

- Follow a control flow based on conditional decisions using Java.
- Use conditional looping to solve a problem.
- Use a simple cumulative sum technique to count.
- Use basic arithmetic to simulate computer *intelligence*.
- Organize the storage of temporary data.
- Continue building good programming skills.

Introduction

Have you ever accused your computer of consciously reacting to you as if it were a friend or enemy? It is not unusual to assume it is actually making decisions based on its relationship to you. However, its *intelligence* is based on its ability to compute at insanely rapid speeds and to store and access huge amounts of data. The topic of artificial intelligence is very popular among computer scientists. Much of the *smart* technology appears to make decisions that was previously thought only capable of humans: self-driving cars, medical assessment, and earthquake prediction are some of currently popular topics that come to mind.

In this assignment, we will create a few useful and simple methods to handle dates in a Gregorian calendar. We will also use these methods to simulate a simple guessing game for the computer. It will guess *the user's* birthday, the user being a person at the keyboard. If it randomly guessed until it got it right, the average number of guesses would be $365/2 = 182.5$, which hardly seems intelligent, so we allow the computer to narrow its search after each guess.

Quick Start

(1) Download this pdf file and store it on your computer.
(2) Create a directory / folder on your computer specific to this assignment. for example, `CSC110/assn4` is a good name.
(3) Save a file called `CalendarGames.java` in your directory. Fill the source code with the class name and the following methods:

```
public static String monthToString(int month)
public static boolean isLeapYear(int year)
public static int numDaysInMonth(int month, int year)
public static void guessMyBirthday()
public static void main(String[] args)
```

Hint: copy and paste the above methods, so you get them exactly. You may change the variable names, which were shortened here to fit on a single line.
(4) Create empty methods to start. For each method that returns a value, return a simple value (such as -1 or `false`, depending on the return type specified).
(5) Compile the source code and make sure you are ready to start programming.
(6) Start with the easiest methods, and for each method:
- Read the specific method details in the specifications. Use these details to create the method description comments above the header in the source code.
- Write the source code that is required to make the method behave as stated.
- Test the method thoroughly in the `main` method with different inputs and print out the results so you can verify that they are correct.

In the case where you need to call another method, you can just call it; if it is finished, the return value will be correct, but if it is not finished, you can deal with the temporary return value. In other words, you are not restricted to the order. If you like to complete the called method first, you are using a *bottom-up* approach; if you like to complete the calling method first, you are using a *top-down* approach.

Detailed Instructions

We offer the following organization of the parts of the programming task. For more details about what each of the method *does*, check out the specification document. In the following subsections, we provide a detailed description of how to approach the implementation of each of the methods named in the `CalendarGames` class:

monthToString:

Most people refer to the months of the year by their names. However, it is easier to do the arithmetic needed to refer to the month by its number: $1, ..., 12$. This method allows us to easily convert the numeric version of a month to its name. You may use `if-else` statements or the `switch` statement to solve this. We recommend you try both techniques for good practice.

isLeapYear:

In a leap year, February has 29 cays in the month, instead of its usual 28. The specifications details for this method have a link to Wikipedia's explanation of leap years. It's simple and requires some `if-else` statements as well.

numDaysInMonth:

The computer should not guess a date that doesn't exist, like September 31, if we don't want it to appear unintelligent, or February 29 if the day the player was born was not a leap year. Once the computer has guessed the month, it needs to access the minimum day (that's 1 in all cases), and the maximum day, which is the number of days in the particular month. This method needs to answer that question. You may use *nested* `if-else` statements; the `switch` statement also works very well here. In this method, you can see the usefulness of the *falling through* property of the `switch` statement.

In case you forget:

> The months that contain 31 days are January, March, May, July, August, October, and December.
> The months that contain 30 days are April, June, September, and November.
> In a non-leap year, February has 28 days, in a leap year, it has 29.

guessMyBirthday

This method makes use of the other methods to play a simple interactive game. It will be familiar from Lab#4 where we implemented a guessing game with a number between 1 and 10.

The computer needs to act intelligently, so variables that store the low and high parts of a range are helpful. You may assume that the user will always give accurate information. Note that during the development and testing phase of your implementation, you may need to stop the program. You can do this by typing `Control-C`, a command that stops

any running program that you started from the command line. This is handy information when you start programming using loops and they cycle into an "infinite" loop.

This method will require a fair number of programming statements. For this reason, please note that your code (especially in this method) meets the following standards:

- It must be easy to read and understand. Little programmer comments help. Note that comments do not need to state the obvious : `\\this is a for-loop` is not necessary. However a small note like `\\determining the correct month`this a helpful way to introduce a block of code.
- It must make a clear distinction between its parts, for instance, guess the month first and then the day.
- Variable declarations should be allocated to specific blocks.
- The interaction between user and computer must follow the example below.
- You will need to use some kind of looping, as the computer repeated makes a guess based on the narrowing range; we recommend the `while` loop.

Example

```
-bash
demo$ java CalendarGames
Is your birthday in June? yes or no: no
Is your birthday after this month? yes or no: yes
Is your birthday in September? yes or no: no
Is your birthday after this month? yes or no: yes
Is your birthday in November? yes or no: no
Is your birthday after this month? yes or no: yes
Is your birthday in December? yes or no: yes
Is your birthday in December 16? yes or no: no
Is your birthday after this day? yes or no: no
Is your birthday in December 8? yes or no: no
Is your birthday after this day? yes or no: yes
Is your birthday in December 12? yes or no: no
Is your birthday after this day? yes or no: no
Is your birthday in December 10? yes or no: no
Is your birthday after this day? yes or no: no
Is your birthday in December 9? yes or no: yes
It took 9 guesses to find your birthday, which is December 9
demo$
```

Submission

Submit the following completed file to the Assignment folder on conneX.

- `CalendarGuesses.java`

Please make sure you have submitted the required file(s) and conneX has sent you a confirmation email. Do not send [`.class`] (the byte code) files. Also, make sure you *submit* your assignment, not just save a draft. Draft copies are not made available to the instructors, so they are not collected with the other submissions. We *can* find your draft submission, but only if we know that it's there.

A note about academic integrity

It is OK to talk about your assignment with your classmates, and you are encouraged to design solutions together, but each student must implement their own solution.

Grading

Marks are allocated for . . .

- No errors during compilation of the source code.
- All method names, return types, and argument lists must be exactly as shown. Variable names in the argument lists do not need to be the same as shown, but the ordering and the data types must be. Extra *helper* methods are acceptable.
- The required methods must behave as detailed in the specification document.
- All the required methods are *called* by another method. Any testing done in `main` should remain. It is fine to comment them out as you progress.
- For the sake of the marker, who must run hundreds of programs, please use a similar prompt and interaction as in the example.
- The number of computer guesses is always 9 or fewer.
- Style of the source code meets the requirements outlined in the Coding conventions document available in the Lab Resources folder of conneX.