**BIT 142**

**Programming Assignment #2: Birthday/Date**

**Due date: See Course Schedule**

|  |
| --- |
| **You must do this assignment BY YOURSELF.**  **You are NOT ALLOWED to work with anyone else. All of your work must be your own, original work.** |

Your instructor would like to thank to Marty Stepp and Hélène Martin at the University of Washington, Seattle, who originally wrote this assignment (for their CSE 142, in Java)

This program focuses on classes and objects. Turn in two files named Birthday.cs and Date.cs. You will also need the support file Date.dll; it is contained in the starter project for this assignment.

The assignment has two parts: a client program that uses Date objects, and a Date class of your own whose objects represent calendar dates. It is meant to be shorter than other recent assignments and is also worth fewer points.

**Part A (Birthday.cs, client program):**

The first part of this assignment asks you to write a client program that uses an existing MyDate class written by the instructor. The goal of Part A is to give you a bit of practice creating and using MyDate objects from a client's perspective and to give you an appreciation for the usefulness MyDate objects in general.

Begin by prompting the user for today's date and for his/her birthday, first by month and then by day. Use this information to print the number of days in the month the user was born, and the number of days from today to the user's birthday. If the user's birthday is today, print a Happy Birthday message.

Below are several example logs of execution from the program; user input is **bold and underlined**. Your program's output should match these examples exactly given the same input.

|  |  |
| --- | --- |
| What is today's year?  **2017**  What is today's month?  **11**  What is today's day?  **4**  What is your birthday year?  **1997**  What is your birthday month?  **9**  What is your birthday day?  **9**  There are 30 days in month #9  2017 is NOT a leap year  Number of days until birthday 9/9: 309  You are 20 years old right now | What is today's year?  **2020**  What is today's month?  **12**  What is today's day?  **15**  What is your birthday year?  **2000**  What is your birthday month?  **12**  What is your birthday day?  **15**  There are 31 days in month #12  2020 IS a leap year  Happy Birthday!  You are 20 years old right now |
| What is today's year?  **2000**  What is today's month?  **8**  What is today's day?  **19**  What is your birthday year?  **1980**  What is your birthday month?  **11**  What is your birthday day?  **30**  There are 30 days in month #11  2000 IS a leap year  Number of days until birthday 11/30: 103  You are 19 years old right now | What is your birthday year?  **2100**  What is today's month?  **10**  What is today's day?  **2**  What is your birthday year?  **2080**  What is your birthday month?  **10**  What is your birthday day?  **1**  There are 31 days in month #10  2100 is NOT a leap year  Number of days until birthday 10/1: 364  You are 20 years old right now |

Notice that the age listed in the lower-left box is 19 (not 20) because the user was born on November 30th and the current date is only August 19th.

Solve this problem using MyDate objects. The methods and behavior of each MyDate object are described on the next page. For Part A you can use an instructor-provided version of MyDate by downloading the starter project from the website. Inside that .ZIP you’ll see a folder named SOLUTION\_FROM\_INSTRUCTOR that contains MyDate.dll. This [Dynamic Link Library (DLL) file](http://en.wikipedia.org/wiki/Dynamic-link_library) contains a sample implementation of the MyDate class (as described below). There are instructions in the Birthday.cs file that tell you how to use the instructor’s version of the MyDate object or your own.

You can construct a MyDate object as follows:

MyDate **name** = new MyDate(**year, month**, **day**);

*To figure out the number of days until the user's next birthday, represent today and the birthday as MyDate objects, then, by advancing one date until it reaches the other and counting, you can determine the number of days between them.*

**You do have to handle leap years in this assignment.** According to <http://www.wwu.edu/skywise/leapyear.html>:  
*“There is a leap year every year whose number is perfectly divisible by four - except for years which are both divisible by 100 and not divisible by 400. The second part of the rule effects century years. For example; the century years 1600 and 2000 are leap years, but the century years 1700, 1800, and 1900 are not. This means that three times out of every four hundred years there are eight years between leap years.”*

You may assume valid input (that the user will always type a month between 1-12 and a day between 1 and the end of that month).

**Part B (MyDate.cs, class of objects):**

The second part of this assignment asks you to implement a class named MyDate, stored in a second file named MyDate.cs. For all methods/constructors shown below, you may assume that any parameter values passed are valid.

Your MyDate class should implement the following behavior:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| * public MyDate(int year, int month, int day) Constructs a new MyDate representing the given year, month, and day. Month should be between 1 and 12 (if it isn’t just use 1), and day should be a valid value for that year and month (if it isn’t then just use 1) * public MyDate(MyDate otherDate) Constructs a new MyDate by copying the year, month, and day from the otherDate object into the newly created object. For this reason this is sometimes referred to as a “copy constructor”. Because the otherDate object should only contain valid data (month between 1 and 12, for exmple) you can just copy the fields over without having to check that they’re valid. * public int getYear() This method should return the year of the MyDate object on which it was called. For example, if the MyDate object represents August 17, 2017, this method should return 2017. * public int getMonth() This method should return the month of the MyDate object on which it was called, between 1 and 12. For example, if the MyDate object represents August 17, 2017, this method should return 8. * public int getDay() This method should return the day of the month of the MyDate object on which it was called, between 1 and the number of days in that month (which will be between 28 and 31). For example, if the MyDate object represents August 17. 2017, this method should return 17. * public void setDate(int year, int month, int day) Modifies the state of the MyDate object on which it was called to represent the given month and day. If the parameters are not valid then your code should react the same way that the MyDate() constructor does. * public String toString() This method should return a String representation of the MyDate object on which it was called in a *month/day* format. For example, if the MyDate object represents March 24, 2007, return "3/24/2007". If this MyDate object represents December 3, 2100, return "12/3/2100". This method *returns* the string; it does not print output. * public bool isLeapYear() This method should return true if the MyDate object on which it was called represents a date in a leap year. * public bool equals(MyDate d) This method should return true when the MyDate object on which it was called represents the same date as the given MyDate parameter, or false otherwise. * public bool equalsIgnoreYear(MyDate d) This method should return true when the MyDate object on which it was has the same month and day values as the given MyDate parameter, or false otherwise. Examples:  When comparing August 17, 2017 and August 17, 2017, this method should return true. When comparing August 17, 2017 and August 17, 201**5**, this method should still return true (since the month and day are the same, even though the years are different) When comparing August 17, 2017 and August 18, 2017, this method should return false. * public bool earlierThan(MyDate d) This method should return true when the MyDate object on which it was called occurs earlier in time than the date given as the MyDate parameter, or false otherwise. Some examples:  August 17. 2017 is earlier than August 18. 2017 August 17. 2017 is earlier than December 18. 2017  August 17, 2000 is earlier than December 18. 2001. August 17, 2000 is NOT earlier than August 17. 2000. August 17, 2001 is NOT earlier than August 17. 2000. * public int daysInMonth() This method should return the number of days in the month represented by the MyDate object on which it was called. The following table lists the number of days in each of the twelve months of the year:  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | | Name | Jan | Feb | Mar | Apr | May | Jun | Jul | Aug | Sep | Oct | Nov | Dec | | Days | 31 | 28 or 29  (see below) | 31 | 30 | 31 | 30 | 31 | 31 | 30 | 31 | 30 | 31 |   For example, if the MyDate object represents August 17, 1997, this method should return 31. If this MyDate object represents February 14, 1991 (which is NOT a leap year), you should return 28. If this MyDate object represents February 14, 2000 (which is a leap year), you should return 29.   * public void nextDay() This method should modify the state of the MyDate object on which it was called by advancing it 1 day in time. For example, if the MyDate object represents September 19, a call to this method should modify   the MyDate object's state so that it represents September 20. Note that depending on the date, a call to this method might advance the MyDate object into the next month or year. For example, the next day after August 17 is August 18; the next day after February 28 is March 1; and the next day after December 31 is January 1. |

None of the methods listed above should print any output to the console. You may not utilize any behavior from the instructor-provided MyDate class to help implement your own MyDate class, nor use any of C#’s date-related classes such as System.DateTime.

**Testing Your Program:**

You can test your MyDate program by running your Birthday.cs program from Part A with it, by compiling your MyDate.cs.

Birthday.cs is not a great testing program; it might not call all of your MyDate methods or may not call them in a very exhaustive way that tests all cases and combinations. Therefore you might want to create another small client program of your own to help **test** other aspects of your MyDate class's behavior. You might also use the **Immediate Window** in Visual Studio Pro (Debug 🡪 Windows 🡪 Immediate) to test MyDate objects, call the various methods, and look at the results.

You may put additional behavior in your MyDate class if you like, but we will still test your Birthday program with the instructor-provided MyDate class, so it should still run correctly with that class and not only when used with your MyDate. In other words, your Birthday.cs should not rely on any MyDate behavior that is not described in this specification.

**Development Strategy and Hints:**

Complete Part A before Part B, to get a good understanding of how MyDate objects work from the client's perspective. Write Part B in phases:

• Write the first constructor, getDay, getMonth, getYear, and isLeapYear methods first.  
(To get started you might want to have the first constructor just blindly copy the parameters into the object)

• Then write the setDate method (and fix up the first constructor so that it uses setDate)

• Then implement toString, equals, and earlierThan, and copy constructor.

• Last, write daysInMonth and nextDay.

Build your MyDate class incrementally, writing a small amount of code at a time and testing it. It is possible to test an incomplete MyDate class by writing some of its methods and then creating a small client program to call just those methods.

Recall that code in one of an object's methods is able to call any of the object's other methods if so desired. Specifically, when implementing nextDay you may want to consider calling other methods within the MyDate object to help you.

Since objects can be difficult to visualize and understand, we strongly recommend that you use the Visual Studio debugger to step through your code to understand each method's behavior, especially in Part B. You can also use temporary debugging Console.WriteLine statements from inside the MyDate class to see what is going on. For example, printing the state of the current MyDate object from inside the daysInMonth or nextDay method can help you find bugs.

**Style Guidelines:**

For Part A, you are to solve the problem by creating and using MyDate objects as much as possible. This is because a major goal of this assignment is to demonstrate understanding of using objects and defining new classes of objects. Part A is not required to have any methods besides main, though you may have additional methods if you like.

For Part B, implement your MyDate as a new type of object, using methods, instance variables / data fields, constructors, etc. as appropriate. You should also properly **encapsulate** your MyDate objects by making their methods and constructors public and their data fields private. As much as possible you should avoid redundancy and repeated logic within the MyDate class. Avoid unnecessary fields: use fields to store the important data of your MyDate objects but not to store temporary values that are only used within a single call to one method.

## Group Work, Commenting:

             You are not allowed to work in groups for this assignment.  You should start, finish, and do all the work on your own.  If you have questions, please contact the instructor.

             Additionally, you should comment your code, paying particular attention to areas that are difficult to understand.  If you found something to be tricky when you wrote it, make sure to comment it so that the next person (the instructor, who's grading you) understands what your code is doing.  It is not necessary to comment every single line.

The purpose of this requirement is to both help you understand, and have you demonstrate, a thorough understanding of exactly how your program works.

*Every file that you turn in should have:*

* At the top of each file that you normally edit, you should put your name (first and last), the name of this class (“BIT 142”), and the year and quarter, and the assignment number, including the revision number, which starts at 0 (“A2.0”).  If you’re handing this in again for a regrade, make sure to increase the minor version number by one (from “A2.0”, to “A2.1").  
  You normally edit the C# source code files (.CS files), and any Word documents that you're handing in (if any).  
  You do not normally edit the .SLN or .CSPROJ files, and so you should not try to put this identifying information in those files.

In general, you should make sure to do the following before handing in your project:

* All variables used should have meaningful names.
* The code should be formatted consistently, and in an easy to read format.

**What to turn in:**

 ·        A single electronic folder (a directory).  This folder should contain:

* The source code for the program – all the .CS files in your project.  
  I would prefer that you include the project files – stuff ending in .SLN and .VCPROJ, so I can build your project more easily.  If you can save these files (the .SLN / . VCPROJ) into a file format that can be opened by VS.Net 2003, that would be great.
* You have to name the folder with your last name, then first name, then the assignment number (both the major version – 2, and the minor (revision) number – 0).  Example: "Panitz, Mike, A2.0"
* You **should not** include the **bin**, **obj**, or **Debug** directories, or anything from it.  I will dock you a couple points if you do.  Also, you don't need to include your .NCB file, if it's present.

**How to electronically submit your homework:**

There's a link on the homework page to the document that guides you through handing in your work.