**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 month?  **11**  What is today's day?  **4**  What is your birthday month?  **9**  What is your birthday day?  **9**  There are 30 days in month #9  Number of days until birthday 9/9: 309 | What is today's month?  **12**  What is today's day?  **15**  What is your birthday month?  **12**  What is your birthday day?  **15**  There are 31 days in month #12  Happy Birthday! |
| What is today's month?  **8**  What is today's day?  **19**  What is your birthday month?  **11**  What is your birthday day?  **30**  There are 30 days in month #11  Number of days until birthday 11/30: 103 | What is today's month?  **10**  What is today's day?  **2**  What is your birthday month?  **10**  What is your birthday day?  **1**  There are 31 days in month #10  Number of days until birthday 10/1: 364 |

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(**month**, **day**);

*To figure out the number of days until the user's birthday, represent today and the birthday as MyDate objects. By advancing one date until it reaches the other and counting, you can determine the number of days between them. Absolute days (from Homework #4) should* ***not*** *be used to calculate the days until the user’s birthday.*

**You do not have to worry about leap years at all for this assignment.** Assume that February always has 28 days and that the year is always 365 days long. (A leap year occurs roughly every 4 years and adds a 29th day to February, making the year 366 days long. But we will ignore that possibility for this assignment.)

Do NOT assume valid input (the user may type in a month that’s less than 1 or greater than 12, and may type in a day # that’s zero, negative, or larger than the number of days in 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 month, int day) Constructs a new MyDate representing the given month and day. * 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, 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, this method should return 17. * public void setDate(int month, int day) Modifies the state of the MyDate object on which it was called to represent the given month and day. You may NOT assume that the parameter values passed are valid. If the month isn’t valid then use the value 1 instead; if the day isn’t valid *for that particular* month then use 1 instead. * 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, return "3/24". If this MyDate object represents December 3, return "12/3". This method *returns* the string; it does not print output. * public boolean equals(int otherDatesMonth, int otherDatesDay) This method should return true when the MyDate object on which it was called represents the same date as the given parameters, or false otherwise. * 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 | 31 | 30 | 31 | 30 | 31 | 31 | 30 | 31 | 30 | 31 |   For example, if the MyDate object represents August 17, this method should return 31. If this MyDate object represents February 14, you should return 28. You do not need to worry about leap years.   * 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.

You’ll notice a method named TestMyDate(), located at the bottom of the Birthday.cs file. This contains many, many blocks of code that will help test your class for you. Two very important points about this code:

1. **Your MyDate class MUST compile and run with this test code.**If it doesn’t you will receive a point penalty.  
   For example: the equals() method is spelled with all lower case letters. If you decide to make an Equals() method instead (note the uppercase ‘E’) then your MyDate code will no longer compile with the provided test code. If this happens then you will receive a point penalty
2. Passing the tests is NOT a guarantee that your code works 100% correctly!  
   It’s possible that you’ve got an error in your code that wasn’t tested.  
   For example: there are two tests for the ‘daysInMonth’ class – one for February and one for another month. If your code produces the wrong result for, say, June these tests won’t detect that.  
   **You must make sure that your code works correctly, regardless of what these tests say.**

I believe that you should be able to figure what the tests do on your own, but feel free to ask questions if you have any trouble figuring out what the tests are checking, or why your code isn’t working with the tests (please post your questions in a Canvas discussion forum).

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 and getMonthmethods 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 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.