W. P. CAREY
SCHOOL *of* BUSINESS
ARIZONA STATE UNIVERSITY

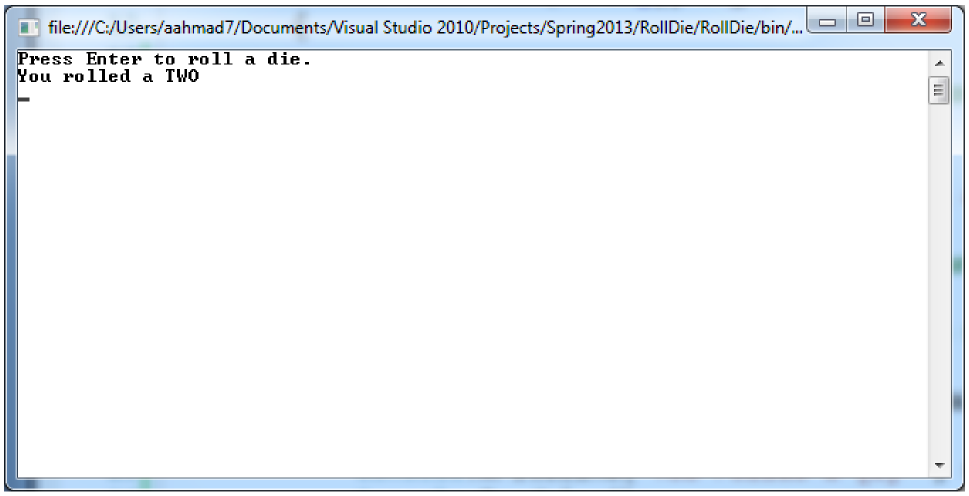
CIS 345 – Business Information Systems Development II – Fall 2014

In-class Exercise 2: Basic Methods (Die Rolling App)

Due on Blackboard: Tomorrow, Tuesday, September 9 by 10:00 PM

Objective: This exercise is designed for you to start developing methods and gain proficiency in working with method calls, arguments, and parameters as well as distinguishing between the usage of static and non-static methods.

Design a basic Die Rolling application that generates a random number from 1 to 6 and outputs the number in text format, e.g. "You rolled a ONE."


file:///C:/Users/aahmad7/Documents/Visual Studio 2010/Projects/Spring2013/RollDie/RollDie/bin/...

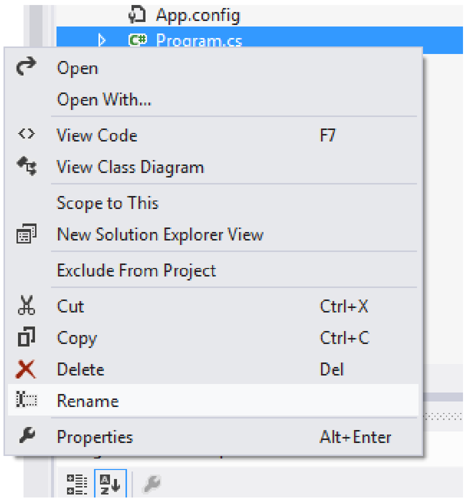
```
Press Enter to roll a die.
You rolled a TWO
_
```

For this exercise, you will need to generate a random number within a method and then create another method that can accept an integer and returns a string that represents the number spelled out in English. Finally, you will call both the methods in the right sequence passing return values from one to the other as needed and displaying the final output on the screen.

Note: All methods in this project will be static.

1. Create a project called DieRollApp.

2. Rename your class name from Program.cs to RollDie.cs by right-clicking on the filename in Solution Explorer and clicking rename. Accept its recommendation to change the name of the class throughout the project code.



3. Create a NumbertoText method. *Make sure you put the method inside the class block. Methods must always be within the class block but they cannot be placed inside another method i.e. methods cannot be nested.*

   All methods will be in the structure:

   ```
   private static ReturnValueType MethodName (ParameterType ParameterName )
    {
       // Insert in the body of the method, statements as described in the method descriptions.
     }
   ```

   Do not start working on the Main method first. The Main method must be completed last. Since Main calls other method, we must write those methods first.

   Method name: NumberToText
   Purpose: This method accepts an integer and returns a string representation of that number. If it receives 1, it returns "ONE". If it receives 2, it returns "TWO", and so on.
   Input Parameter: integer, called number
   Return value type: string

   In the body of the method:

   Do not use a WriteLine anywhere within this method. It should only return a string by using the return keyword.

   - declare a string, NumberText
     - Initialize it with a blank string as a value i.e. ""
   - Use the switch control statement on the parameter variable (we called it 'number' here)
     - case 1: assign "ONE" to NumberText
     - case 2: assign "TWO" to NumberText
     - Complete the rest of the cases
   - After the end of the switch, return NumberText using the 'return' keyword. *Make sure the return statement is AFTER the body of the entire switch statement. It should not be in the body of the switch statement.*

4. Create a GenerateRandomNumber method in the class block.

   Chapter 7 introduces you to the Random class, which allows you to generate random numbers. Figure 7.6 and 7.7 show samples of using the Random class. There are samples of working with the Random object as well as switch statements within Chapter 7 of the book e.g. pages 245-248.

   Alternatively, look up Microsoft's documentation for the Random class on MSDN. You will find descriptions of all methods as well as sample code.

   http://msdn.microsoft.com/en-us/library/2dx6wyd4%28v=vs.110%29.aspx

   The method you write is supposed to generate a random number from 1 to 6 and return the value as an integer.

   Method name: GenerateRandomNumber
   Purpose: This method generates and return a random number.
   Input Parameter: None.
   Return value type: integer

   In the body of the method:

   - Declare and instantiate a random object as shown above or in the book. This is the part that says "Random randomNumbers = new Random()" in the book.
   - Generate a new random number from 1 to 6 by calling the *Next* method of the object of type Random. Store the return value of the method in the variable DieFace (make sure you also declare the variable).

     Note that the *Next* method is not static. There you will not type "Random.Next" but will rather say "randomNumbers.Next" For the *Next* method, the upperbound value is exclusive. So, you must specify you want to go up to 7 even though you only want numbers up to 6.

     See the italicized portion in part (5) on the next page if you need instructions on how to store return values.

   - Return DieFace using the 'return' keyword.

5. Complete the Main method, in which you will utilize the object of type Random to generate a random number. *Even though the main method runs first, you should write it last, so the methods it calls are already written.*

   In the body of the Main method:

   - Tell the user that pressing enter will roll a die. Use ReadLine() to insert a pause.
   - Call GenerateRandomMethod. Store its return value in a variable, DieValue (you will need to declare the variable within the Main method).

     To call methods, you need to write their name, put in the paratheses, and within the parantheses, provide it with any parameter values that are required. Then, IF the method returns a value, you need to assign the entire method to a variable.

     e.g. string input = Console.ReadLine();

     In the example above, the ReadLine method is called. No parameters are provided. ReadLine returns a string as its return value. The entire method is assigned to the variable, input. The left-hand side data type, string, is the same as the right-hand side return value datatype (also a string).

   - Call NumberToText and supply it with DieValue as an argument within the parantheses. Store the return value in a string variable. You will need to declare it as well (call it FaceText).
   - Display the variable, FaceText, using a WriteLine statement. Use placeholders or string concatenation (joining strings using "+" sign).

     Example of using placeholders

     Console.WriteLine("Value of x is {0} and y is {1}", x, y);

     The statement above will take the value of x and y and replace it at locations {0} and {1} within the string. X is the 0th variable, and y is the 1st. If you want to format it, you can add more formatting instructions for decimals or even variable preceding/trailing spaces. Refer to the book for examples.

<u>Other things to try</u> *(We will do a demonstration of this in class when we come back)*

- Use the Watch window to keep track of your random number variable
    - Keep a watch on the values of specific variables you are interested in monitoring
    - Use the variables as part of a condition, e.g. DieValue == 3;
- Use the Locals window to see values of your local variables.
- Learn how to use the Immediate window.
    - Put a breakpoint on the line that declares NumbertoText.
    - Open the immediate window and change the value of the parameter once you enter the method, e.g., number = 5. This will modify your program on the fly and is a good way to test your program's behavior based on varying values.
    - Try modifying parameter as well as argument variables. Are you able to do so?

- Interesting computing behavior:

  Try looping the random number generator and printing out the random numbers to the console. On "slower" machines in the past, the "random" numbers were the same! Test how the numbers look today. Are they equally random when fewer iterations vs. having a large number of iterations? Also, compare the result with the person next to you. Different computers have been known to exhibit different behavior with the same code (when using the Random object).

- Practice more loops:

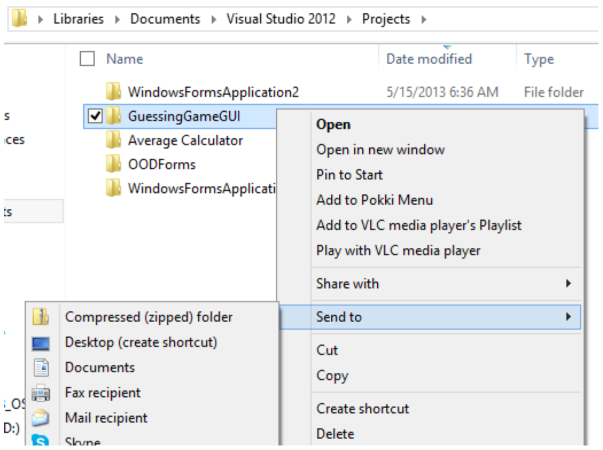
  Add a sentinel controlled loop that repeatedly generates and outputs the random number unless the user presses "X".

Submission Instructions

NOTE CHANGE OF SUBMISSION PROCEDURE BELOW

Submission should be made using a zip file that contains all of the Visual Studio C# project files. You will need to ***zip the entire project folder*** along with the .sln and .suo files. The folder will automatically contain the class source files as well as the executable file that is generated in \RollDieApp\bin\Debug folder. Upload file to the Blackboard assignment drop box.

Zip the entire top-level folder by right-clicking the folder and selecting Send to | Compressed (zipped) folder.



Using built-in windows zip tools: http://windows.microsoft.com/en-us/windows/compress-uncompress-files-zip-files

Make sure you check the following. Your grade is dependent on all these criteria being met.

- You have included your name as a comment within your class.
  - e.g. " // Inclass 2, Jane C. Smith, CIS 345, Tuesday 9:00 AM"
- Class file is called RollDie.cs (rename from Program.cs).
- Your Visual Studio project is called RollDieApp.
- Zip filename is: **FirstNameLastName_Inclass2.zip**

Verify your zip file before you submit

- Check for actual class files being present in the folder before you zip it.
- Check your zip file size after zipping – if it is 1K, it likely contains only a shortcut.
- Uncompress your zip file before submitting and verify that files are present.
- Download your zip file after submitting, uncompress, and again verify that your files are present.