Homework Assignment 004 (2020)

Root Finder of Quadratic Equation & 4-digit Guessing Game

First, create a win application project named as <yourID><YourName>Ass004 to have a form served as the user interface. Remember to rename the default file name (class name as well) Form1.cs to a meaningful one. The application let the user set up a quadratic equation and get the roots for the user. In the second part, the application generates 4 different numeric digits and let the user to guess them.

Let us start writing an object-oriented program (application) by designing classes to serve the functionalities. For example, a RootFinder class that provides public functions for roots finding for a quadratic equation; a FourDigitGuessing class to randomly generate 4 digits for players to guess. In your main form class, you will instantiate an object of RootFinder class and an object of FourDigitGuessing class to accomplish the task.

Root Finder for a Quadratic Equation

In your RootFinder class, define three **double** data fields (variables) a, b, and c to represent the coefficients of the quadratic polynomial equation $ax^2+bx+c=0$. Define a **public** member function (method) to let users supply their values of the coefficients a, b, and c via function arguments. Then, define another **public** method that will return a **string** to display the roots found and related information. Note that, when $a=0$ and $b=0$, the case is "extremely degenerate." When $a=0$ and $b\neq 0$, the case is "degenerate." In this case the equation reduces to $bx+c$ and it has one root $x = -c/b$. When $a\neq 0$ (the general case), the roots are

$$r_1=\frac{1}{2a}\left(-b+\sqrt{b^2-4ac}\right), \textit{ and } r_2=\frac{1}{2a}\left(-b-\sqrt{b^2-4ac}\right).$$

The expression b^2-4ac under the square root is the *discriminant*. If the discriminant is positive, two real roots exist. If the discriminant is zero, the two roots are real and equal. In this case we say that the equation has multiple real roots. If the discriminant is negative, the roots are complex. Your function should return, as appropriate, the following messages:

```
Extremely degenerate!
```

```
Degenerate: …
Multiple real roots: …
Two complex roots: …
```

along with the computed root(s). For example, if 1, 2, and 3 are passed in for the values of *a*, *b*, and *c* respectively, then

```
    Two complex roots:
Root1 = -1.000000 + 1.414214 i
Root2 = -1.000000 - 1.414214 i
```

should be returned. In your form, prepare a user interface control to display the message.

In your main form class, provide versatile UI (user interface) controls, such as Label, TextBox, NumericalUpDown, Button, etc., to provide a user friendly interface to let user modify the numbers and coefficients and visualize the computation results. Therefore, you need to convert the texts, obtained from user interfacing controls, into double values, before calling any computing functions. An object of the RootFinder class should be defined and instantiated in your form in order to help you find the roots for your users.

Since the result returned (obtained) is a string, you need to provide a result display control to show the outcome. You can either use a Label or RitchTextBox to display the results.

For the root finder class, prepare data fields and value setting method:

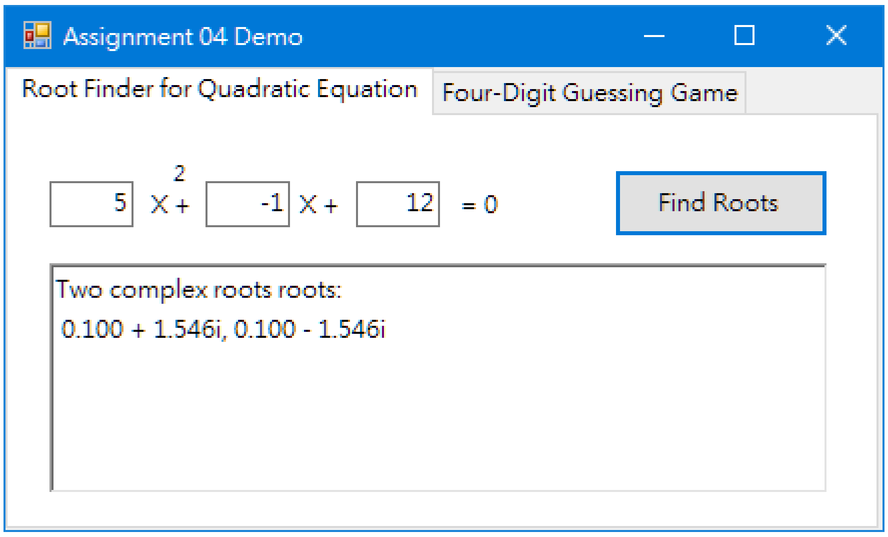
```
double a, b, c;
 public void setQuadraticCoefficients(double aa, double bb, double cc)
 {
     a = aa; b = bb; c = cc;
 }
 // The signature of the root finder function will look like:
 public string findPolynominalRoots()
 {
     string result = "";
     double discriminant = 0.0;
```

```
    double r1, r2, g1, g2;  // Real values and image values

    if (a == 0.0)
    {
        if (b == 0.0) result = ...
        else
        {
        ...
        result = ...
    }
    }
    else
    {
        discriminant = ...
      if (discriminant >= 0.0)
        {
            r1 = ...

            ...
            result = ...
   }
        else
        {
            r1 = ...
            g1 = ...
            result = ...
        }
    }
    return result;
}
```

Sample UI for the root finder:



```
Two complex roots roots:
 0.100 + 1.546i, 0.100 - 1.546i
```

Helpful classes of .NET Framework and primary methods:

`Convert`	**`ToDouble()`**
`Math`	**`Sqrt()`**
`string`	**`Format()`**

C# Operators

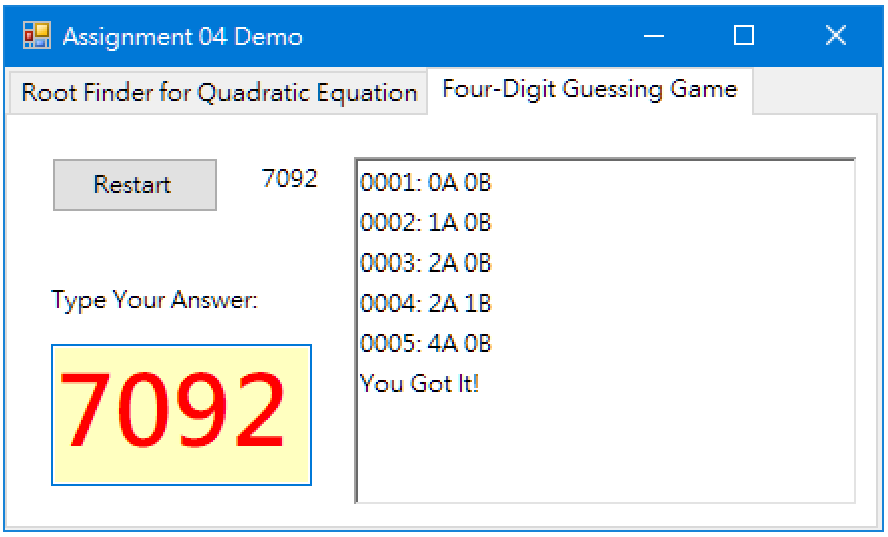
==	equality operator
!=	not equal operator
>	larger than operator
>=	larger than or equal operator
<	less than operator
<=	less than or equal operator
&&	and Boolean operator
||	or Boolean operator

```
C# if Statement
if( BooleanExpression )
{
    // true statements
}
else
{
  // false statements
}
```

Four-Digit Guessing Game

In your FourDigitGuessing class, define an array of **int** to store the four generated numerical digits, where they are all different. An object of **Random** class is used to randomly assign digits for the game.

To play the game, a user types in 4 digits, hit return key. The class check whether each digit is exactly match with the correct answer. If a guessing digit matches the value of the correct digit in the same position, then count an A. Otherwise, if the value matches one correct digit but the position is not hit, count a B. If the value of the guessing digit does not match any correct digit, count nothing. Therefore, at most four counts of A and B will be reported for the player to keep guessing the digits. The game is solved when 4A is reported.



```
0001: 0A 0B
0002: 1A 0B
0003: 2A 0B
0004: 2A 1B
0005: 4A 0B
You Got It!
```

```
class DigitGuessing
    {
        int[] digits = new int[4];
        Random rnd = new Random();

        public DigitGuessing()
        {
            // constructor
              ….
            ResetDigits();
        }

        public void ResetDigits()
        {
            … rnd.Next(10)…
        }

        public string GetCorrectAnswer()
        {
            string s = "";
            for (int i = 0; i < numberOfDigits; i++)
                …
            return s;
        }

        public string CheckCorrectness( string answer )
        {
            int countA = 0, countB = 0;
            for( int i = 0; i < numberOfDigits; i++ )
            {

                if (digits[i].ToString() == answer[i].ToString() )
              {
               …
```

```
             }
              else
              {
                 …
              }
           }
           if ( … )
             return $"{countA}A {countB}B\nYou Got It!";
           return $"{countA}A {countB}B";
       }
}
```