Today's exercise is to define and implement a Rational number class. For this exercise, you are given a C# project with a GUI already created. You are to write the C# code to provide the Rational class and handlers for the GUI buttons that use the Rational class methods.

Use a Web browser to go to the course webpage http://csserver.evansville.edu/~hwang/s08-courses/cs205.html. Under today's date, save the file RationalDemoInClass.zip. Extract the solution folder. Double-click into the folder, then double-click on RationalDemo.sln (the Visual Studio solution file). This will launch Visual Studio with the solution loaded.

The GUI design for the program has been completed. To see it, right-click on Form1.cs in the Solution explorer window and select View Design. It should look like the following:

Currently, there is no handler code for any of the buttons.

This GUI is intended to be used as an interface to test various methods and operations on Rational number objects. More on this during class.

Some GUI notes:

- The TextAlign property of all the textboxes is set to Center. (Default is left.)
- The handlers for the Calculate and Compare buttons should use the textboxes above them. The textboxes are intended to hold the left operand, an operator, and the right operand, respectively. The differences between the expected behavior of each button will be explained in class.
Notes on Rational Numbers

A rational number is a number that can be expressed as an integer or the quotient of an integer divided by a nonzero integer. Expressed as $a/b$, $a$ is called the numerator and $b$ is called the denominator. Results of the binary arithmetic operations on rational numbers represented by $a/b$ and $c/d$ are as follows:

<table>
<thead>
<tr>
<th>Operator</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a/b + c/d$</td>
<td>$(ad + bc) / bd$</td>
</tr>
<tr>
<td>$a/b - c/d$</td>
<td>$(ad - bc) / bd$</td>
</tr>
<tr>
<td>$a/b \times c/d$</td>
<td>$ac / bd$</td>
</tr>
<tr>
<td>$a/b \div c/d$</td>
<td>$ad / bc$, where $c$ not equal to 0</td>
</tr>
</tbody>
</table>

In addition, $a/b = c/d$ when $ad = bc$. From this relationship, the equality and relational operations can be defined. E.g., $a/b < c/d$ when $ad < bc$. This is because $ad$ and $bc$ are the numerators of the left and right operands, respectively, when both are written with the common denominator $bd$.

The beginning of a Rational class for this exercise is in the file Rational.cs, which contains the private attributes, property definitions to get the numerator and denominator of a Rational object, and the empty and explicit-value constructors to create a Rational object. As discussed in last class, one reason for constructors is to ensure that all created Rational objects are valid. In particular, the denominator of a Rational number cannot be 0. The explicit-value constructor checks for this and uses standard C# way of informing the system there has been an error by throwing an Exception object.

Note that the explicit-value constructor calls a private method Reduce that further calls a private method GCD that computes the greatest common divisor of two positive integers. The algorithm that GCD uses was discovered by the ancient mathematician Euclid: For two positive integers $m$ and $n$,

- if $n \leq m$ and $n$ divides $m$, then the greatest common divisor is $n$
- if $n > m$, then swap $m$ and $n$ and start again
- otherwise the greatest common divisor is the greatest common divisor of $n$ and the remainder of $m$ divided by $n$

The GCD method implements this algorithm directly and is a little different than the methods we have seen before because it is recursive. That is, as part of computing its result, it calls itself. This does not cause a problem because eventually, the first case (also called the base case) is reached ending the series of calls.

C# notes
The C# code used for today's exercise is described in the following chapters of the textbook:

- Basic class definition and static methods- Chapter 7
- Defining properties - Chapter 14
- Throwing exceptions when there are errors - pages 103-106 of Chapter 6
- Overriding virtual methods - pages 205-208 of Chapter 12
- Defining overloaded operators and conversion operators - Chapter 19
- Difference between value types and reference (class) types - Chapter 8
Assignment
(10 points) Working in pairs, complete this program by writing the Rational methods and handlers for each of the buttons. We will work on this in class in the following order:

- In order to input and display Rational numbers, we need to define a Parse static method that receives a string and returns the equivalent Rational number, and override the virtual ToString method that returns the equivalent string representation of a Rational number. Both of these are defined in the Rational class. For Parse, we want to be able to accept Rational numbers of the form n/d or just n. For ToString, we want to return a string of the form "n/d" unless d is 1, then it should return just "n".

- Implement the handlers for the Numerator and Denominator buttons by parsing aRational.Text and displaying the Numerator and Denominator properties, respectively.

- Implement the Reciprocal method that returns the reciprocal Rational number and the handler for the Reciprocal button that parses the textbox, calls the method, and displays the result.

- Since Rational numbers are numbers, we would like to be able to use the standard arithmetic operators to create Rational number results. To do this, we overload operator methods. These are static methods with special syntax. Write implementations for operators +, -, *, and /, based on the information given above. Implement the handler for the Calculate button to parse the left and right operands as Rational numbers, then display the result of applying the given operator. The handler should throw an exception if anOp.Text is not a valid operator.

- In addition to the arithmetic operators, Rational numbers should also be comparable using the standard relational and equality operators. Write implementations for operators <, <=, >, >=, ==, and !=. Implement a handler for the Compare button to parse the same left and right operand textboxes, then apply the given operator. The handler should throw an exception if the operator is not valid. (Ignore the warning about overriding Equals and other virtual methods. We'll talk about that later.)

- Finally, it would be good if we could do arithmetic and comparison operations with ints and Rational numbers (just like we do with ints and doubles). To do this we need to be able to convert an int into a Rational number. This is very straightforward as all integers are valid Rational numbers, but it isn't done automatically. C# allows us to write conversion operators. Write an int to Rational conversion operator and implement the handlers for the Calculate - int <op> Rational and the Calculate - Rational <op> int buttons.