Why a Class?

Imagine that the company you work for is going to create a series of lessons on fractions. It might be useful to create a Fraction class that all of the fraction lessons could use. A fraction has a numerator and denominator. For instance 1/2 has a numerator of 1 and a denominator of 2. The denominator should never be 0.

The string version of a fraction is the numerator, "/", and the denominator.

We can convert a fraction to decimal by dividing the numerator by the denominator (making sure that the denominator is not zero, of course!)

We can create a Fraction class with all of the features above incorporated. This Fraction class can then be used by anyone who writes one of the lessons on Fractions.

The Fraction Class

Start a new Visual Basic program called **TestFraction**. We will thoroughly test the class before we let other programmers use it.
From the menu select Project-> Add Class. Name the class **Fraction**.
Write the code for the class as shown below. Notice that we never allow the denominator to be zero.

```
Public Class Fraction
   Private pNumerator As Integer = 0
   Private pDenominator As Integer = 1
   Public Sub New()
      pNumerator = 0
      pDenominator = 1
   End Sub

   Public Sub New(ByVal numerator As Integer, ByVal denominator As Integer)
      pNumerator = numerator
      If denominator <> 0 Then
         pDenominator = denominator
      End If
   End Sub

   Public Overrides Function ToString() As String
      Return "" & pNumerator & "/" & pDenominator
   End Function

   Public Function ToDouble() As Double
      Return pNumerator / pDenominator
   End Function
End Class
```

Explanation
Notice that pNumerator and pDenominator are both private, with default values of 0 and 1. Making these members private allows us to restrict the values assigned to them. In particular, we want to make sure that the denominator is never zero.

New: The New method is called a constructor. When we create a New Fraction, the constructor function is called. Our Fraction class has two constructors: New with no arguments and New with 2 arguments, the numerator and denominator.

ToDouble: Because it is impossible for the denominator to ever be zero, we can convert it to a decimal without checking whether the denominator is 0.

ToString: Every class has a ToString method. We want to write our own definition of what ToString returns. We must override that predefined function with our own definition that will return the numerator, a slash, and the denominator.

Now in the code for the form try each example below to create a new instance of the fraction class:

Example 1:

```
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
      Dim myFraction As New Fraction()
      Me.Text = myFraction.ToString
   End Sub
End Class
```

When you run this example you will see 0/1 in the text of the form. This example uses the constructor New Fraction with no arguments. Notice that the default value for the numerator is 0 and the numerator has a default value of 1. If we change the code to
`Me.Text = myFraction.ToDouble` we will see 0 in the text for the form.

Example 2:

```
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
      Dim myFraction As New Fraction(1, 0)
      Me.Text = myFraction.ToString
   End Sub
End Class
```

Can you guess what will display? Look at the code. The denominator is given a default value of 1 and the zero is NOT accepted so our fraction is 1/1.

Example 3:

```
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
      Dim half As New Fraction(1, 2)
      Me.Text = half.ToString & "=" & half.ToDouble
   End Sub
End Class
```

This time it will display **1/2 = 0.5**

Try some additional examples such as 3/4 or 1/3. Try to guess what the output will be.
You can also change the value of the numerator or denominator using the set methods.
Make sure you understand all of the code before going on to the next example.

To Do:
Add a text box for the numerator and denominator. Add labels to display the string; the decimal value.

To Think About:
Every industry has objects that they deal with everyday: transactions, packages, customers, boxes, printers, etc. How would you define any of these things as classes?

Get and Set
Usually private members have Get and Set methods. Add the following code to the Fraction class:

```
    Public Property Numerator() As Integer
        Get
            Return pNumerator
        End Get
        Set(numerator As Integer)
            pNumerator = numerator
        End Set
    End Property

    Public Property Denominator() As Integer
        Get
            Return pDenominator
        End Get
        Set(denominator As Integer)
            If denominator <> 0 Then
                pDenominator = denominator
            End If
        End Set
    End Property
```

This code will let you set and get the numerator as shown below:

```
Public Class Form1
    Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Dim myFraction As New Fraction(1, 4)
        myFraction.Numerator = 5
        Me.Text = myFraction.ToString & "=" & myFraction.ToDouble
    End Sub
End Class
```

When you run this you will see 5/4=1.25 A fraction that has a numerator that is bigger than its denominator is called an improper fraction.

To Do:
Write a Boolean method Improper that returns True if the fraction is improper. Return False otherwise.
Add a method **ToStringProper** that would return "1 1/4" instead of "5/4" *(This is not an override.)*