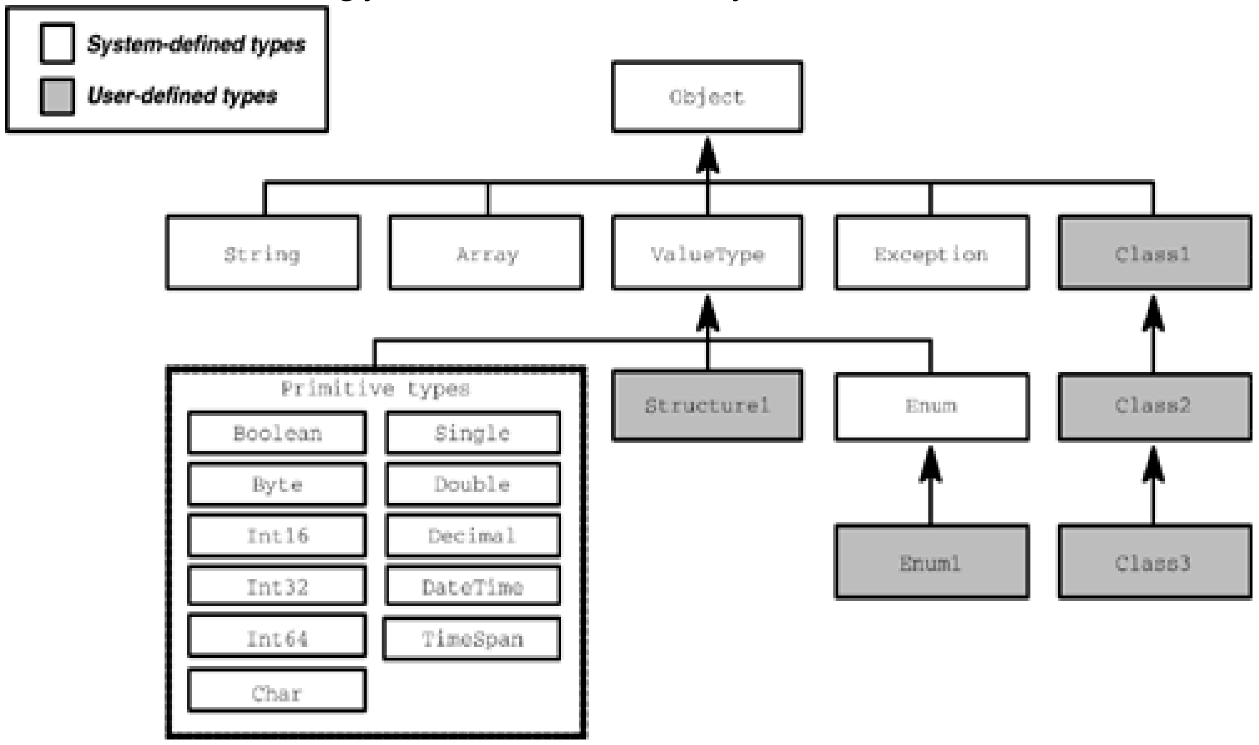
VARIABLES and CONSTANTS

Variable
It is a named storage location in computer's memory that can contain data that can be modified during program execution. A variables has a name, the word used to refer to the value the variable contains and a data type (which determine the kind of data the variable can store)

Data Types
The CTS (*Common Type System* (CTS)) and CLS (Common Language Specification) are standards that must be conformed to by all .NET languages

- CTS defines the basic data types that IL understands. Each .Net compliant language should map its data types to these standard data types. This makes it possible for the supported languages to communicate with each other by passing/receiving parameters to and from each other
- The Common Language Specification (CLS) works with the CTS to ensure language interoperability. It is generally a set of minimum standards that all compilers targeting .NET must support.

Data Types – The Type Inheritance Hierarchy
The CTS is based on a singly rooted inheritance hierarchy.



Value Types and Reference Types
A data type can be characterized as a value type or a reference type

i). Value types store data in the memory allocated to the variable and includes primitive types as well as structures and enumerators i.e. Value variables represent an allocation of memory that holds values

ii). Reference types store a 32-bit memory address which references a value or a collection of values i.e. a reference variable never holds the instance itself (or a value). It just holds a reference to an instance of an object thus; the term *pointer* is often used to describe a reference type. The object class and hence all other classes are reference types

Types of Data
VB provides numerous data types to allow programmers to optimize the use of computer memory. The following table describes the various data types used in Visual Basic:

Data Type	**Description of data stored**	**Memory storage in bytes**
	Text Data Storage	
String	Alphanumeric data such as letters of the alphabet, digits that are not treated as numbers, and other special characters.	Size varies
Char	Stores single Unicode characters (supports any international language).	2
	Numeric Data Storage – Fixed Point	
Decimal	Decimal numeric values – often used to store dollars/cents. The Decimal Type is almost perfect for storing (modeling) money.	16
	Numeric Data Storage – Floating Point	
Double	Double-precision numeric values with 14 digits of accuracy.	8
Single	Single-precision numeric values with 6 digits of accuracy.	4
	Numeric Data Storage – Whole Numbers (no decimal point)	
Short	Whole numeric values in the range -32,768 to 32,767.	2
Integer	Whole numeric values in the range -2,147,483,648 to +2,147,483,647.	4
Long	Whole numeric values that are very, very large.	8
	Special Data Types	
Boolean	True or False.	2
Byte	Stores binary data of values 0 to 255 – can be used to store ASCII character code values.	1
Date	Stores dates in the form 1/1/0001 to 12/31/9999.	8
Object	Stores data of any type.	4

Note: Strings are stored as Unicode characters in two bytes as an unsigned short (UShort). The first 127 values store letters, digits, and special characters. The values 128 to 255 are for special characters such as keyboard arrows while the values 256 to 65,536 are for international characters and diacritical marks

Naming Rules for Variables and Constants

The programmer decides what to name variables and constants, however there are technical rules that must be followed when creating variable and constant names.

- Names can include letters, digits, and the underscore, but **must** begin with a letter.
- Names cannot contain spaces or periods.
- Names cannot be **VB reserved words** such as **LET**, **PRINT**, **DIM**, or **CONST**.
- Names are **not** case sensitive. This means that **`TotalInteger`**, **`TOTALINTEGER`**, and **`totalinteger`** are all **equivalent** names.
- For all intensive purposes, a name can be as long as you want (the actual limit is 16,383 characters in length).

Naming Conventions

Naming variables and constants follows the **Camel Casing** naming convention. Use the following guidelines:

1. Create meaningful names – do not name a variable or constant **X** or **Y** or **XInteger**. Instead use names such as **StudentNameString**, **CountStudentsInteger**, and **AmountDueDecimal**.
2. Begin each name with an uppercase letter and capitalize each successive word in the name such as **AmountDueDecimal**
3. Use Uppercase for constant names such as **TAX_RATE_SINGLE**

Declaring Variables (Dimensioning Variables)

Variable declarations ensure that appropriate memory space is reserved for the variables depending on the data type of the variable.

Local variables are declared with the **Dim** statement while module-level variables are declared with the **Private** statement. When declaring variables you need to specify a variable/constant name and data type

1. Specifying an initial value for a variable is optional -- do so if necessary.
2. If an initial value is not assigned, then a string stores the "empty string" and all numeric variable types store the value zero.

Syntax:

```
Dim VariableName As Type
Private VariableName As Type
```

Examples:

```
Dim StudentNameString As String
Dim AccountBalanceDecimal As Decimal = 100D
```

You can also declare more than one variable in a **Dim** statement as follows

```
Dim SubtotalDecimal, TotalDecimal, TaxAmountDueDecimal As Decimal
```

Declaring Constants

Constants are values that are stored to memory locations; however, a constant cannot have its value change during program execution. Constants are declared with the **Const** statement.

VB has two different types of constants.

1. **Intrinsic Constants** – these are defined as **enumerations** such as **Color.Red** and **Color.Blue**. These are called **intrinsic** because they are predefined in VB and always exist for your use.
2. **Named Constants** – these are constants you define with a **Const** statement. These constants are specific to your programming application.

Examples:

```
Const SALES_TAX_RATE_SINGLE As Single = 0.0725F
Const TITLE_STRING As String = "Data Entry Error"
Const MAX_SIZE_INTEGER As Integer = 4000
```

Follow the following rules for assigning numeric values to constants:

- You can use numbers, a decimal point, and a plus (+) or minus (-) sign.
- Do not include special characters such as a comma, dollar sign, or other special characters.
- Append one of these characters to the end of the **numeric constant or variable** to denote a data type declaration. If you do not use these, a whole number is assumed to be **Integer** and a fractional value is assumed to be **Double**.

`Decimal`	`D`	`40.45D`
`Double`	`R`	`12576.877R`
`Integer`	`I`	`47852I`
`Long`	`L`	`9888444222L`
`Short`	`S`	`2588S`
`Single`	`F`	`0.0725F`

Scope of Variables and Constants
Each variable (or constant) has a finite lifetime and visibility (scope). The **variable lifetime** is how long the variable exists before the computer operating system garbage-collects the memory allocated to the stored value. Variable scope refers to the statements that can use a variable
There are four levels of scope.

i). **Namespace** (use a **Public Shared** declaration instead of **Dim**) – the variable is visible within the entire project (applicable to a project with multiple forms).
  - The variable is project-wide and can be used in any procedure in any form in the project.
  - The variable memory allocation is garbage-collected when application execution terminates.

ii). **Module level** (usually use **Private** to declare a variable; use **Const** to declare a constant) – a variable/constant can be used in any procedure on a specific form – it is not visible to other Forms.
  - Use module-level variables when the values that are stored in their memory locations are used in more than one procedure (event handlers or any other type of procedure).
  - A module-level variable or constant is created (allocated memory) when a form loads into memory and the variable or constant remains in memory until the form is unloaded.

iii). **Local** (use **Dim/Static** to declare a variable; use **Const** to declare a constant) – a variable/constant is declared and used only within a single procedure.
Syntax
```
[Dim | Static] varname As type [=initexpr]
```

  - The **Dim** keyword declares a local variable that is created when the containing procedure (sub or function) is invoked. The variable and any value it might have is destroyed when execution of the containing procedure ends
  - The **Static** keyword declares a local variable that its value persist from one procedure invocation to the next

iv). **Block** (use **Dim** to declare the variable; use **Const** to declare the constant) – the variable/constant is only visible within a small portion of a procedure – Block variables/constants are rarely created.

Data Types and Type Conversion
Type conversion is used to convert data from one data type to another

1. **Converting Input Data Types**
As part of the **Input** phase of the **Input-Process-Output** model, you must convert values from the **Text** property of a TextBox and store the converted values to memory variables. **Text** property always stores **string** values, even if the string looks like a number.
Parse method – converts a value from a Text property to an equivalent numeric value for storage to a numeric variable. Parse means to examine a string character by character and convert the value to another format such as decimal or integer.
In order to parse a string that contains special characters such as a decimal point, comma, or currency symbol, use the **Globalization** enumeration shown in the coding examples below.

- If you don't specify the Globalization value of **Globalization.NumberStyles.Currency**, then a value entered into a textbox control such as **$1,515.95** will **NOT** parse to Decimal.
- The Globalization value **Globalization.NumberStyles.Number** will allow the **Integer.Parse** method to parse a textbox value that contains a comma, such as **1,249**.

Example1 – This example shows you how to declare numeric variables then store values to them from Textbox controls.

```
'Declare variables
Dim PriceDecimal As Decimal
'Convert values from textbox controls to memory
PriceDecimal = Decimal.Parse(PriceTextBox.Text,
Globalization.NumberStyles.Currency)
```

Example2 – This example shows you how to declare numeric variables and store values to them from Textbox controls using a single assignment statement in one step.

```
Dim QuantityInteger As Integer = Integer.Parse(QuantityTextBox.Text,
Globalization.NumberStyles.Number)
```

Older versions of VB used **named functions** to convert values. Examples are the **CDec** (convert to Decimal) and **CInt** (convert to Integer) functions as shown below:

```
'Converts to decimal and Integer
PriceDecimal = CDec(PriceTextBox.Text)
QuantityInteger = CInt(QuantityTextBox.Text)
```

Notice that the TextBox's **Text** property value of **$100.00** will **NOT** generate an error if you use the **CDec** function to convert the value as shown above—the data will convert satisfactorily. The functions are also faster and easier to type.

2. **Converting Variable Values to Output Data Types**
In order to display numeric variable values as output the values must be converted from numeric data types to string in order to store the data to the **Text** property of a TextBox control using the **ToString** method. These examples show converting strings to a numeric representation with 2 digits to the right of the decimal (**N2**) and currency with 2 digits to the right of the decimal (**C2**) as well as no digits to the right of a number

```
SubtotalTextBox.Text = SubtotalDecimal.ToString("N2")
SalesTaxTextBox.Text = SalesTaxDecimal.ToString("C2")
QuantityTextBox.Text = QuantityInteger.ToString("N0")
```

Types of Conversion:

i). **Implicit Conversion(**Widening conversions**)** – this is conversion from a narrower to wider data type (less memory to more memory) – this is done automatically as there is no danger of losing any precision. In this example, an integer (4 bytes) is converted to a double (8 bytes):

```
        BiggerNumberDouble = SmallerNumberInteger
```

ii). **Explicit Conversion (**Narrowing conversions**)** – this is also called **Casting** and is used to convert between numeric data types that do not support implicit conversion.

Performing Explicit Type Conversion
The members of the **System.Convert** class explicitly convert data from one type to another. Methods exist for each primary data type
The table below shows the **Convert** method that is used to convert from one numeric data type to another numeric data type. Note that fractional values are rounded when converting to integer.

`Decimal`	`NumberDecimal = Convert.ToDecimal(ValueSingle)`
`Single`	`NumberSingle = Convert.ToSingle(ValueDecimal)`
`Double`	`NumberDouble = Convert.ToDouble(ValueDecimal)`
`Short`	`NumberShort = Convert.ToInt16(ValueSingle)`
`Integer`	`NumberInteger = Convert.ToInt32(ValueSingle)`
`Long`	`NumberLong = Convert.ToInt64(ValueDouble)`

Notice that VB uses a wider data type when calculations include unlike data types. This example produces a decimal result.

```
    AverageSaleDecimal = TotalSalesDecimal / CountInteger
```

Arithmetic Operators
It refers to operators used for mathematical calculations and are the same as in many other programming languages. The operands might be either literal values, object properties, constants or variables
The following is a list of binary arithmetic operators.

Operator	Operation
*	Multiplication
/	Division
\	Integer Division
+	Addition
-	Subtraction
^	Exponentiation
Mod	Modulus Division

Exponentiation – This raises a number to the specified power – the result produced is data type **Double**.
Example: `ValueCubedDouble = NumberDecimal ^ 3`

Integer Division – Divide one integer by another leaving an integer result and discarding the remainder, if any.
Example: If the variable **MinutesInteger = 130**, then this expression returns the value of **2 hours**.

```
        HoursInteger = MinutesInteger \ 60
```

Modulus Division – This returns the remainder of a division operation .Using the same value for MinutesInteger **= 500**, this expression returns the value **20 minutes** and can be used to calculate the amount of overtime worked for an 8-hour work day.

```
        MinutesInteger = MinutesInteger Mod 60
```

Order of Precedence
The **order of precedence** for expressions that have more than one operation is the same as for other programming languages. Evaluate values and calculation symbols in this order:

(1). Values enclosed inside parentheses
(2). Exponentiation
(3). Multiplication and Division
(4). Integer Division
(5). Modulus Division
(6). Addition and Subtraction

Examples
Assuming that: **X=2**, **Y=4**, and **Z=3, the** following table shows the computations of the expressions

Problem	Result
X + Y ^ Z	66
16 / Y / X	2
X * (X + 1)	6
X * X + 1	5
Y ^ X + Z * 2	22

This table shows example of mathematical notation and the equivalent VB expression.

Mathematical Notation	VB Expression
2X	2 * X
3(X + Y)	3 * (X + Y)
πr^2	3.14 * r ^ 2

Assignment Operator
The **(=)** is the assignment operator. It is used store the value of the expression on the right side of the operator to the variable on the left side of the operator. Examples:

```
ItemValueDecimal = QuantityInteger * PriceDecimal
HoursWorkedSingle = MinutesWorkedSingle / 60F
```

The updation assignment (plus symbol combined with the assignment operator) allows you to **accumulate** a value in a memory variable. Examples:

```
TotalSalesDecimal += SaleAmountDecimal
```

is equivalent to statement

```
TotalSalesDecimal = TotalSalesDecimal + SaleAmountDecimal
```

Option Explicit and Option Strict
These options change the behavior of your coding editor and the program compiler.

Option Explicit option is **ON** by default in VB.NET.

- This option requires you to declare all variables and constants.
- If set to Off, you do not need to declare any variables.
- Sometimes programmers will turn this option off with the command shown below, but it is a **bad practice** because it can cause you to spend many hours trying to find errors in variable names that **Option Explicit On** will find for you.

```
Option Explicit Off
```

Option Strict option is **OFF** by default in VB, and enables or disables strict type checking.

- This option is used to convert from wider data types to narrower ones (ones using less memory).
- Helps avoid the mistake of mixing data types within an expression, for example: trying to add a string value to an integer value.
- With **Option Strict Off**, you can write the following assignment statement to store a value from a textbox to a memory variable – VB will automatically convert the string data in the textbox to integer data for storage in the variable:

```
QuantityInteger = QuantityTextBox.Text
```

- With **Option Strict On**, you must write the following – VB will not automatically convert the data from string to integer – you must parse the data:

```
                    QuantityInteger = Integer.Parse(QuantityTextBox.Text)
```

  Use of **Option Strict On** is a good practice i.e. enable strict type checking to prevent hard to find type conversion errors. To use **Option Strict On** in you program, place the command after the general comments at the top of the program as the first line of code as shown below.

```
    'Project: ComputeApplication
    'Today's Date
    Option Strict On
```

Rounding Numbers
Use the **Decimal.Round** method to round decimal values to a desired number of positions to the right of the decimal. Always specify the number of digits to round – the default is to round to the nearest whole number. Always round when multiplying and dividing or when using exponentiation as these operations can result in rounding errors. Simple subtraction and addition do not require rounding. Example:

```
SalesTaxDecimal = Decimal.Round(Convert.ToDecimal(SubtotalDecimal *
SALES_TAX_RATE_SINGLE), 2)
```

Formatting Data for Output
Data to be formatted for output will often use the **ToString** method. The following are additional examples of formatting the output:

Example1: This shows formatting a decimal value to string for display in a textbox control – the output is formatted as currency (dollar sign, commas, 2 decimal points – the default is to format numeric output with 2 digits to the right of the decimal point).

```
SalesTaxTextBox.Text = SalesTaxDecimal.ToString("C")
```

Example2: This shows formatting as currency, but with no digits to the right of the decimal point.

```
TotalDueTextBox.Text = TotalDueDecimal.ToString("C0")
```

The following are the formatting codes:
i). **C** or **c** – The number is formatted as currency value, and a leading currency symbol appears. the number is formatted with a thousand and decimal operator
ii). **F** or **f** – fixed-point, to format a string of digits, no commas, and minus sign if needed.
iii).**N** or **n** – formats a number with commas, 2 decimal place values, and minus sign if needed.
iv). **D** or **d** – formats integer data types as digits to force a specific number of digits to display.
v). **P** or **p** – formats percent value rounded to 2 decimal place values.
Add a digit such as 0 to format with that number of decimal place values, e.g., **C0** or **N0** produces no digits to the right of the decimal whereas **C4** or **N4** would produce 4 digits to the right of the decimal point.

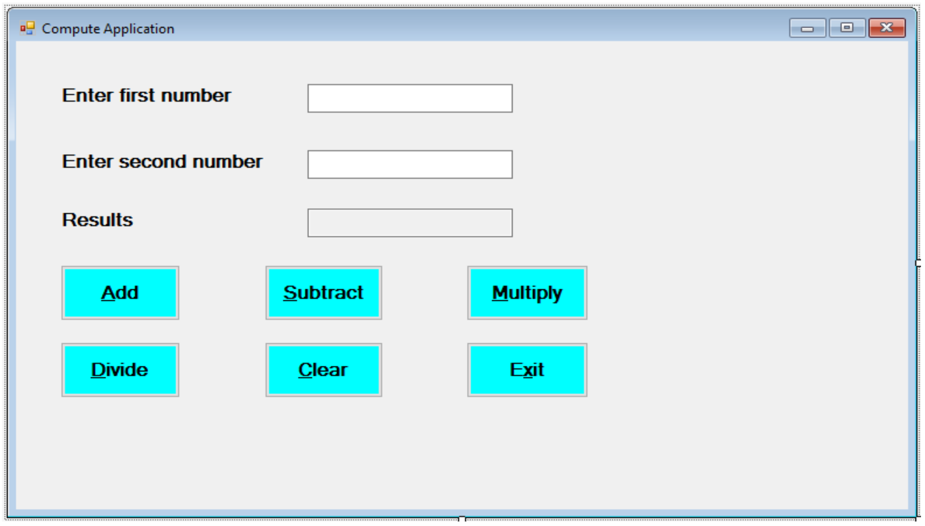
The Imports Statement
- Use the **Imports** statement to eliminate the need for fully qualified references to method names
- It's possible to import any number of namespaces or classes and the order in which **Imports** statements appear is not significant
- Types can also be imported using the Project Properties dialog box which shows a list of all imported namespaces for a Project
  E.g. the following statement import the **System.Convert** class

```
Option Explicit On
Option Strict On
Imports System.Convert
Public Class frmMain
' statements
End Class
```

Example
You are required to write a program that will enable user to calculate the sum, product, division and difference of two numbers entered through textboxes after a click of the respective buttons. The results should be displayed by use text box control. The program should enable the user to clear the textboxes ready for the next input.

i) Sketch your graphical user interface assigning appropriate properties to your form and other controls
ii) Write the code for your program and declare you variables appropriately

Solution
i) Appropriate GUI with appropriate properties for the form and controls



Properties

- The first two TextBox controls are used for data entry – use these names: **Number1TextBox** and **Number2Box**
- The third TextBox control is used to display the output. Set these properties:
    - **ReadOnly** property = **True**,
    - Name property = **ResultsTextBox**.
- Name the buttons **AddButton**, **SubtractButton**, **MultiplyButton**, **DivideButton, ClearButton** and **ExitButton**.

ii) Code

```
'Project:  ComputeApplication
'Author name
'Today's Date
Option Explicit On
Option Strict On

Public Class ComputeForm
    'Declare module-level variables
    Private NumberOneInteger, NumberTwoInteger As Integer

    Private Sub AddButton_Click(sender As Object, e As EventArgs) Handles AddButton.Click
        'Declare local-level variable
        Dim SumInteger As Integer
        'Convert values from TextBox controls to numeric types
        NumberOneInteger = Integer.Parse(Number1TextBox.Text,
        Globalization.NumberStyles.Number)
        NumberTwoInteger = Integer.Parse(Number2TextBox.Text,
        Globalization.NumberStyles.Number)
        SumInteger = NumberOneInteger + NumberTwoInteger
        ResultsTextBox.Text = SumInteger.ToString("N0")
    End Sub

    Private Sub SubtractButton_Click(sender As Object, e As EventArgs) Handles
    SubtractButton.Click
        Dim SubtractInteger As Integer
        NumberOneInteger = Integer.Parse(Number1TextBox.Text,
        Globalization.NumberStyles.Number)
        NumberTwoInteger = Integer.Parse(Number2TextBox.Text,
        Globalization.NumberStyles.Number)
        SubtractInteger = NumberOneInteger - NumberTwoInteger
        ResultsTextBox.Text = SubtractInteger.ToString("N0")
    End Sub

    Private Sub MultiplyButton_Click(sender As Object, e As EventArgs) Handles
    MultiplyButton.Click
        Dim MultiplyInteger As Integer
        NumberOneInteger = Integer.Parse(Number1TextBox.Text,
        Globalization.NumberStyles.Number)
        NumberTwoInteger = Integer.Parse(Number2TextBox.Text,
        Globalization.NumberStyles.Number)
        MultiplyInteger = NumberOneInteger * NumberTwoInteger
        ResultsTextBox.Text = MultiplyInteger.ToString("N0")
    End Sub

    Private Sub DivisionButton_Click(sender As Object, e As EventArgs) Handles
    DivisionButton.Click
        Dim DivisionInteger As Double
        NumberOneInteger = Integer.Parse(Number1TextBox.Text,
        Globalization.NumberStyles.Number)
        NumberTwoInteger = Integer.Parse(Number2TextBox.Text,
        Globalization.NumberStyles.Number)
        DivisionInteger = NumberOneInteger / NumberTwoInteger
        ResultsTextBox.Text = DivisionInteger.ToString("N2")
    End Sub

    Private Sub ClearButton_Click(sender As Object, e As EventArgs) Handles
    ClearButton.Click
        Number1TextBox.Clear()
        Number2TextBox.Clear()
        ResultsTextBox.Text = String.Empty
        'Set the focus to the first text box control
        Number1TextBox.Focus ()
    End Sub

    Private Sub ExitButton_Click(sender As Object, e As EventArgs) Handles
    ExitButton.Click
        Me.Close()
    End Sub
End Class
```