# 31927 32998:
# Application Development with .NET

## Week-8 Lecture

Windows Forms in C#: Part - 2

# Outline

- MessageBox in details
  - With buttons
  - With Icons
- Dialogs:
  - File Open
  - Save, etc.
- Handling Multiple Forms
- ToString() method and overriding
- Generics introduction

# MessageBox with Buttons

- `MessageBox.Show(`Over 20 overloaded options`)`
- Will display message on the screen in a dialog box. User clicks OK to close box. Useful for displaying debugging messages, etc.
- MessageBoxButtons is use to specify which buttons to display.

```
enum System.Windows.Forms.MessageBoxButtons
Specifies constants defining which buttons to display on a MessageBox.
```

- The buttons available are show below:

```
MessageBox.Show("Hello", "Messagebox Demo", MessageBoxButtons.);
```

```
MessageBoxButtons.RetryCancel = 5
The message box contains Retry and Cancel buttons.
```

- AbortRetryIgnore
- OK
- OKCancel
- RetryCancel
- YesNo
- YesNoCancel

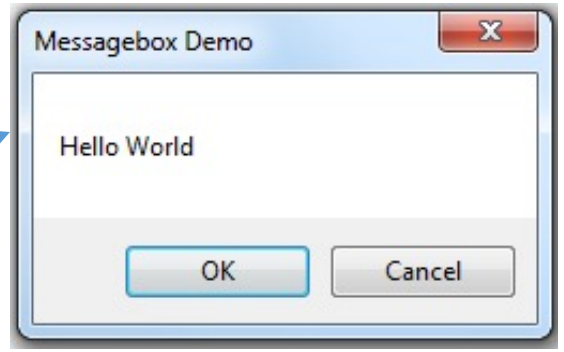
Available button combination

# MessageBox with Buttons

- To check the user input/selection DialogResult is used
- DialogResult is an enum which Specifies identifiers to indicate the return value of a dialog box on the button clicked.

```
enum System.Windows.Forms.DialogResult
Specifies identifiers to indicate the return value of a dialog box.
```

```
DialogResult result = MessageBox.Show("Hello World", "Messagebox Demo", MessageBoxButtons.OKCancel);
```

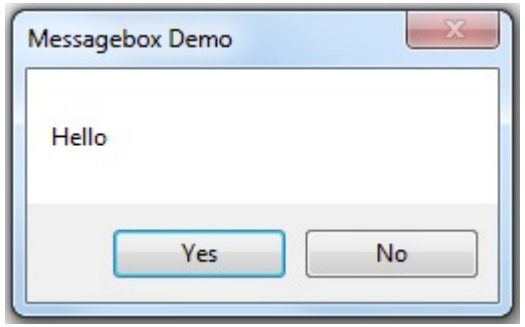



```
// check the user input/selection
// DialogResult is an enum which Specifies identifiers to indicate the return value of a dialog box.
if (result == DialogResult.OK)
{
    // Do Something
}
else if(result == DialogResult.Cancel){
    // Do Something else
}
```

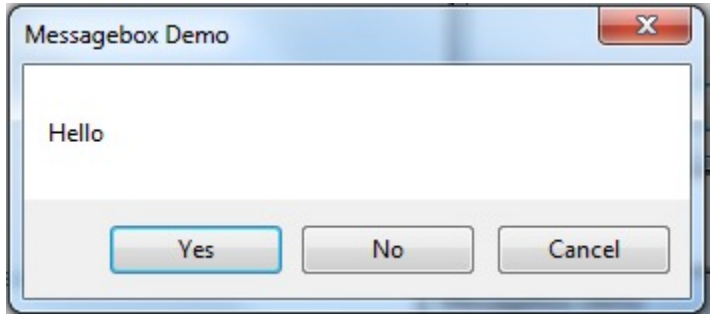
# MessageBox with Buttons

```
MessageBox.Show("Hello", "Messagebox Demo");
```


```
// yes no button
MessageBox.Show("Hello", "Messagebox Demo", MessageBoxButtons.YesNo);
```

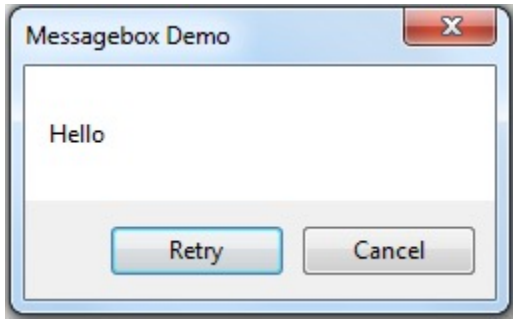



```
// Yes No Cancel button
MessageBox.Show("Hello", "Messagebox Demo", MessageBoxButtons.YesNoCancel);
```

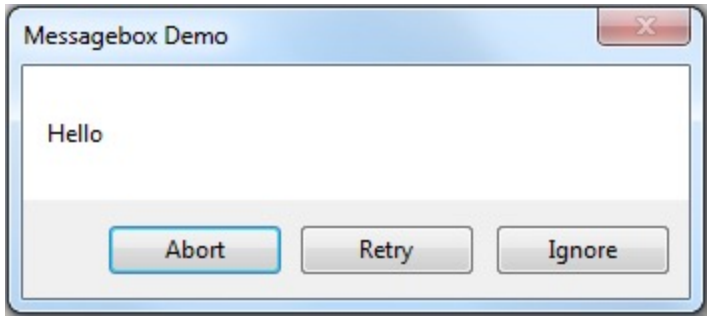


# MessageBox with Buttons

```
// Retry Cancel button
MessageBox.Show("Hello", "Messagebox Demo", MessageBoxButtons.RetryCancel);
```




```
// Abort Retry Ignore button
result = MessageBox.Show("Hello", "Messagebox Demo", MessageBoxButtons.AbortRetryIgnore);
```



# MessageBox with Icons

- MessageBoxIcon is use to specify which icon to display.
- MessageBoxIcon is an enum which Specifies the icon o display.

enum System.Windows.Forms.MessageBoxIcon
Specifies constants defining which information to display.

```
MessageBox.Show("Do you like C# programming?", "FeedBack", MessageBoxButtons.YesNo, MessageBoxIcon.);
```

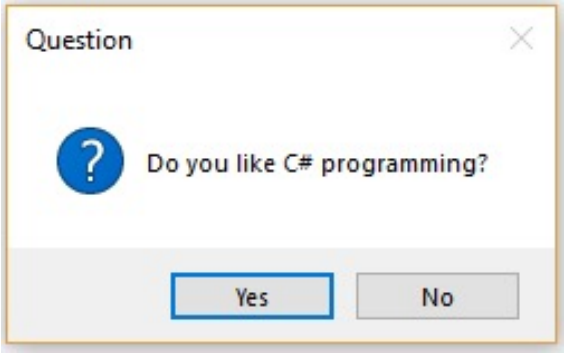
Available Icons

- Asterisk
- Error
- Exclamation
- Hand
- Information
- None
- Question
- Stop
- Warning

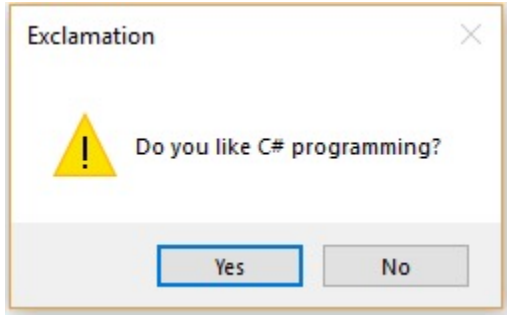
MessageBoxIcon.Stop = 16
The message box contains a symbol consisting of white X in a circle with a red background.

# MessageBox with Icons

```
MessageBox.Show("Do you like C# programming?", "FeedBack", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
```

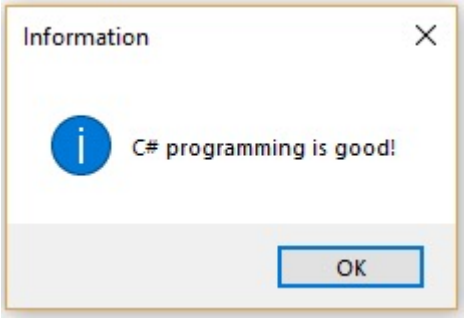



```
MessageBox.Show("Do you like C# programming?", "Exclamation", MessageBoxButtons.YesNo, MessageBoxIcon.Exclamation);
```

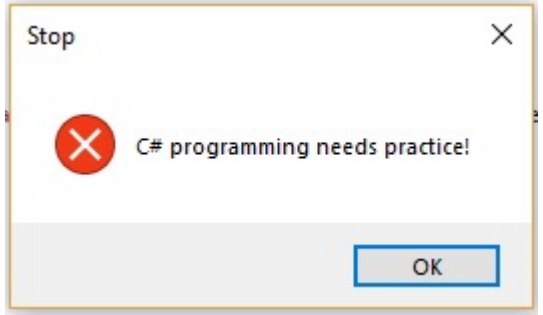


# MessageBox with Icons

```
MessageBox.Show("C# programming is good!", "Information", MessageBoxButtons.OK, MessageBoxIcon.Information);
```

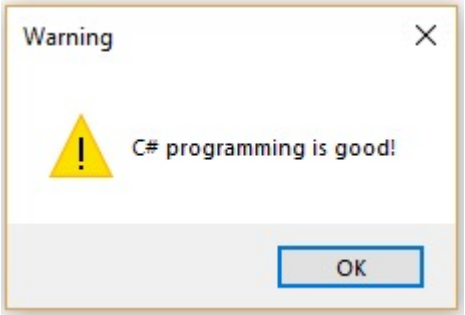



```
MessageBox.Show("C# programming needs practice!", "Stop", MessageBoxButtons.OK, MessageBoxIcon.Stop);
```

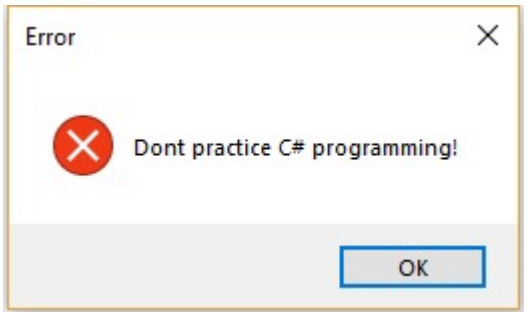


# MessageBox with Icons

```
MessageBox.Show("C# programming is good!", "Warning", MessageBoxButtons.OK, MessageBoxIcon.Warning);
```




```
MessageBox.Show("Dont practice C# programming!", "Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
```



# Common Dialogs boxes:

- Dialog boxes are type of windows, which is used to initiate communication or dialog between a computer and its user.
- A dialog box is most often used to implement a command or to respond to a question.
- Windows.Form is a base class
- Dialog boxes are of two types, which are given below.
  - Modal dialog box: Temporarily halts the application and the user cannot continue until the dialog has been closed/completed
  - Modeless dialog box: Used when the requested information is not essential to continue and the dialog can be left open while continuing the work.

Reference: https://www.c-sharpcorner.com/article/dialog-boxes-in-c-sharp/

# Common Dialogs boxes:

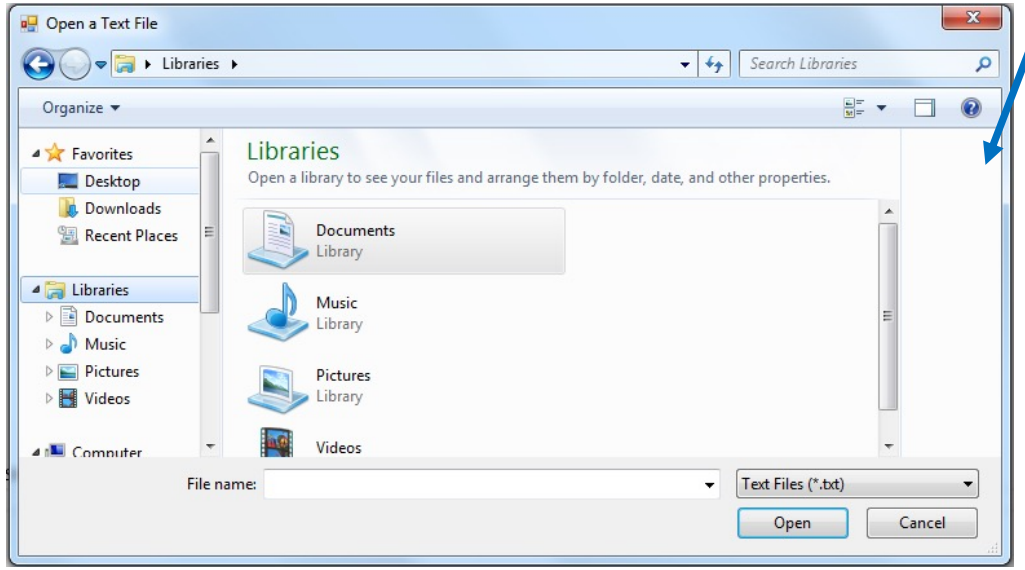
- Common Dialogs boxes: The dialog boxes which are common to all windows applications.
- Performs common tasks such as open a file, save a file, selecting font etc.
- Steps to use Common Dialog boxes:
  1. Create an instance of the required common dialog box,
  2. Set the properties as required,
  3. Call the ShowDialog() method to invoke it.

  ShowDialog() returns a the enum called DialogResult.

Reference: https://www.c-sharpcorner.com/article/dialog-boxes-in-c-sharp/

# Common Dialogs boxes:

1. OpenFileDialog: Allows to choose a file to be opened in an application.

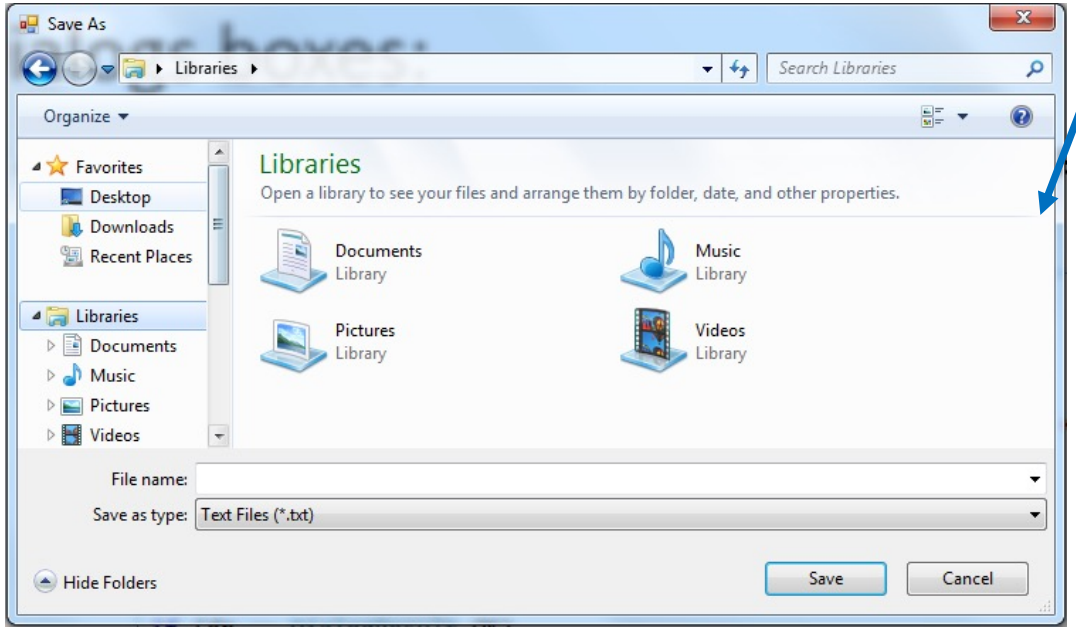



```
//Step 1: Create an instance of the OpenFileDialog
OpenFileDialog openFileDialog1 = new OpenFileDialog();

// Step 2: Set properties
openFileDialog1.Title = "Open a Text File";
//Filter the file type to open
openFileDialog1.Filter = "Text Files (*.txt) | *.txt | All Files(*.*) | *.*";

// Step 3: Call the ShowDialog() method to show the dialog box.
DialogResult dr = openFileDialog1.ShowDialog();

// Check the user response
if (dr == DialogResult.OK)
{
    string filename = openFileDialog1.FileName;
    MessageBox.Show("The File Selected was:" + filename);

    // Do Something to read the file content
}
```

Reference: https://www.c-sharpcorner.com/article/dialog-boxes-in-c-sharp/

# Common Dialogs boxes:

2. SaveFileDialog: Allows the user to select a location for saving a file..



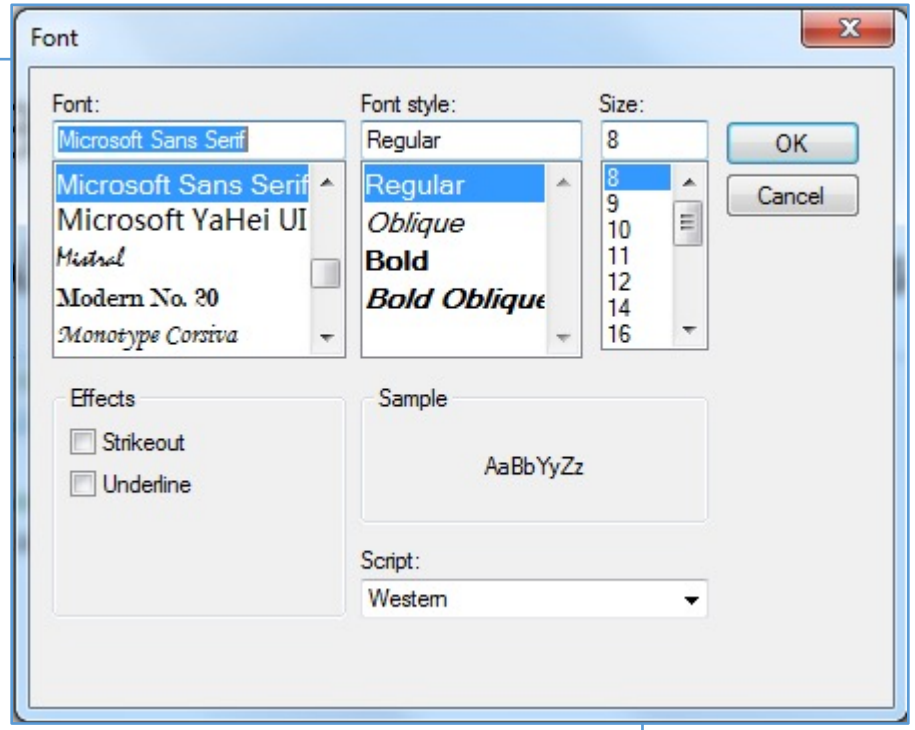

```
//Step 1: Create an instance of the OpenFileDialog
SaveFileDialog saveFile = new SaveFileDialog();

// Step 2: Set properties,  Filter the file types
saveFile.Filter = "Text Files (*.txt) | *.txt | All Files(*.*) | *.*";

// Step 3: Call the ShowDialog() method to show the dialog box.
DialogResult dr = saveFile.ShowDialog();

// Check the user response
if (dr == DialogResult.OK)
{
    // Do Something to save the file
}
```

Reference: https://www.c-sharpcorner.com/article/dialog-boxes-in-c-sharp/

# Common Dialogs boxes:

3. FontDialogBox: Allows the user to select font settings.

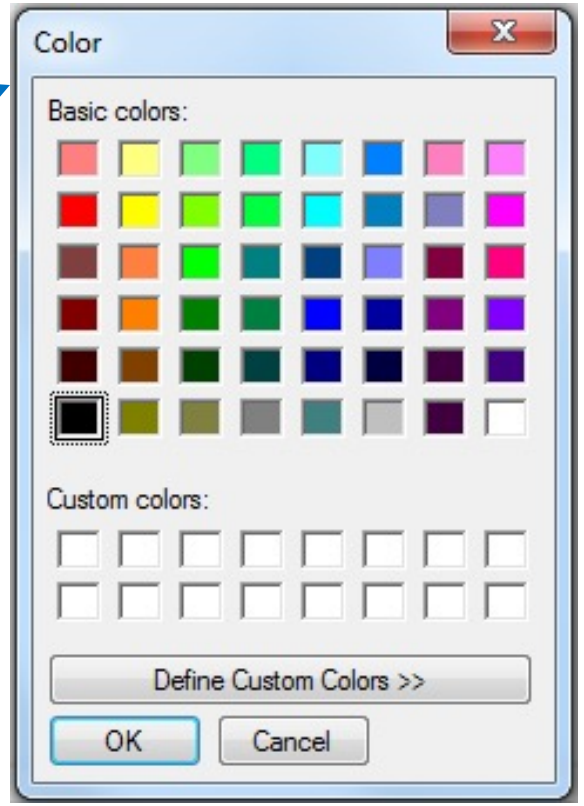
```
//Step 1: Create an instance of the FontDialog
FontDialog fontDialog1 = new FontDialog();

//Call the ShowDialog() method to show the dialog box
DialogResult dr = fontDialog1.ShowDialog();

// Check the user response
if (dr == DialogResult.OK)
{
    string fontName;
    FontStyle  fontStyle;
    float fontSize;

    // Get the use selections
    fontName = fontDialog1.Font.Name;
    fontStyle = fontDialog1.Font.Style;
    fontSize = fontDialog1.Font.Size;

    // Display the user selection
    MessageBox.Show("Fontname: " + fontName + "\nFont Style:"+ fontStyle.ToString()+ "\nFont Size: " + fontSize.ToString());
}
```



# Common Dialogs boxes:

4. ColorDialog: Allows the user to select a color.

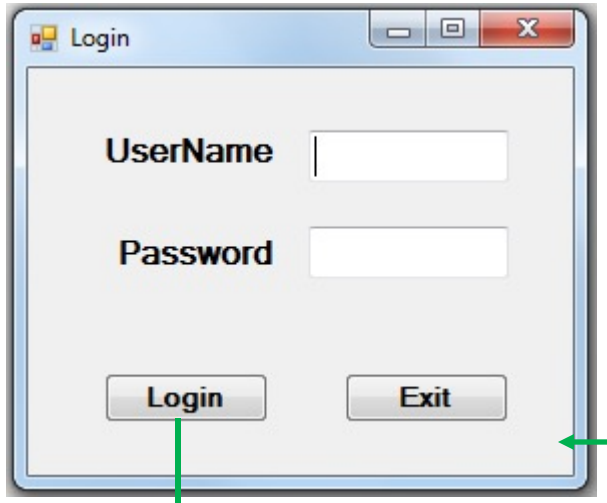
```
//Step 1: Create an instance of the ColorDialog
ColorDialog colorDialog1 = new ColorDialog();

//Call the ShowDialog() method to show the dialog box
DialogResult dr = colorDialog1.ShowDialog();

// Check the user response
if (dr == DialogResult.OK)
{
    this.BackColor = colorDialog1.Color;
    // Do Something
}
```



# Handling Multiple Forms

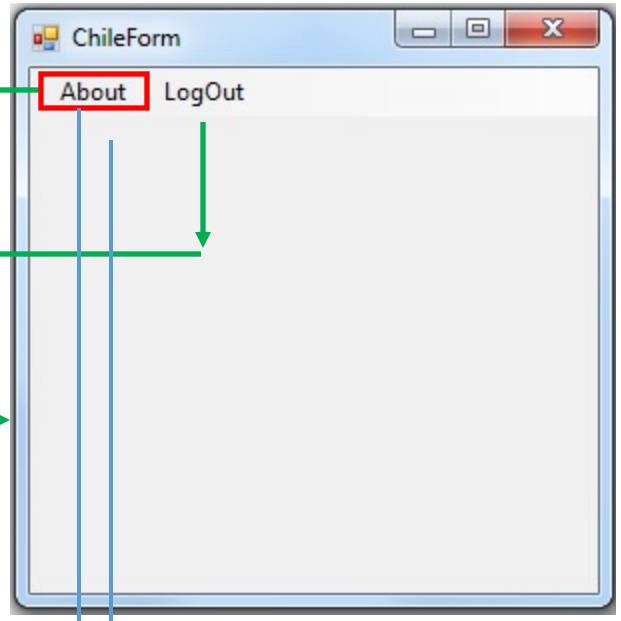
- Majority of the Windows applications uses multiple forms.
- Applications usually have a main form which loads at the start up
- Other forms are accessible from the main form based on the program requirement.
- Example:
  - Main Form: Login Form
  - Other forms: Profile pages, data entry page etc.

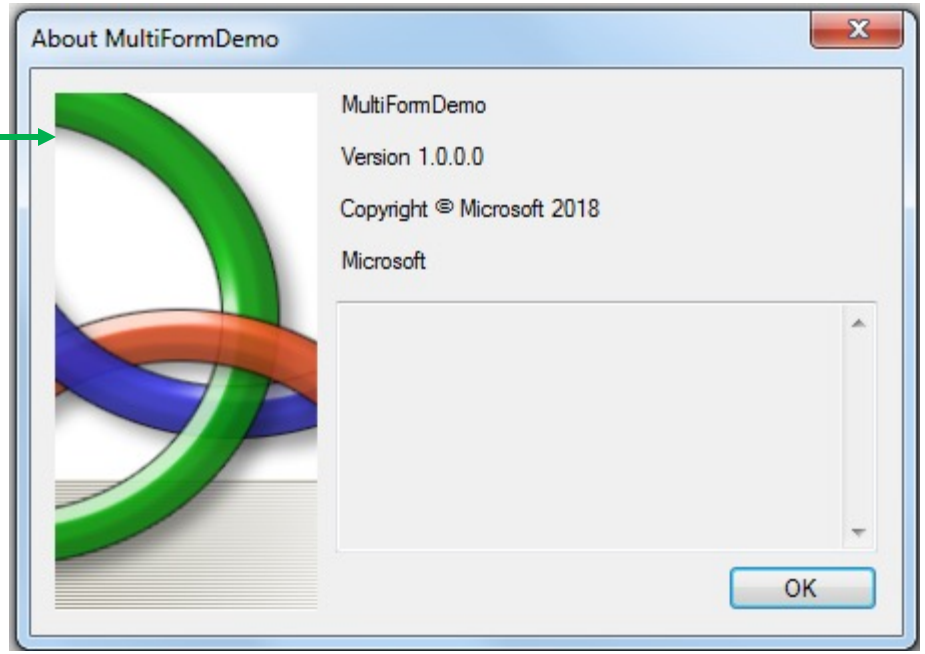
# Handling Multiple Forms - Example




MainForm

ChildForm




About Form

```
// Instantiate the childform
Form2 childForm = new Form2();

// Show the form
childForm.Show();

// Hide the current form
this.Hide();
```

Show About form

```
AboutBox1 aboutForm = new AboutBox1();
aboutForm.Show();
```

Logout

```
this.Hide();
Form1 loginForm = new Form1();
loginForm.Show();
```

# ToString() method and overloading

- What is ToString(): Returns a String Representation of the current object
- Object.ToString is the major formatting method in the .Net Framework
- System.Object is the base class of all reference in the .Net Framework.
- Through inheritance, behaviors/methods in the System.Object class are also available to every type .Net FrameWork.
- Hence ToString() method is available to all the types in the .Net Framework.

```
public override string ToString()
```

# ToString() method and overriding

**Example:**

```
namespace ToStringDemo
{
    0 references
    class Program
    {
        2 references
        public class Employee
        {
            // Properties of the class
            1 reference
            public string firstName { get; set; }
            1 reference
            public string lastName { get; set; }

        }

        0 references
        static void Main(string[] args)
        {
            int salary = 1000;
            // Displays the string representation of INT
            Console.WriteLine("Salary is :{0}", salary.ToString());

            // Create an object of the Employee class
            Employee e1 = new Employee();

            e1.firstName = "George";
            e1.lastName = "Bush";

            // Will display the namespace and the class name without custom implementation of ToString()
            // String represetation of the object!
            Console.WriteLine(e1.ToString());

            Console.ReadKey();
        }
    }
}
```

**Output**

```
Salary is :1000
ToStringDemo.Program+Employee
```

# ToString() method and overriding

```
2 references
public class Employee
{
    // Properties of the class
    2 references
    public string firstName { get; set; }
    2 references
    public string lastName { get; set; }

    // Overiding the ToString method
    2 references
    public override string ToString()
    {
        return ("First Name: " + firstName + " Last Name: " + lastName);
    }
}
```

**Example:**

```
namespace ToStringDemo
{
    0 references
    class Program
    {
        2 references
        public class Employee...

        0 references
        static void Main(string[] args)
        {
            int salary = 1000;
            // Displays the string representation of INT
            Console.WriteLine("Salary is :{0}", salary.ToString());

            // Create an object of the Employee class
            Employee e1 = new Employee();

            e1.firstName = "George";
            e1.lastName = "Bush";

            // Will display the namespace and the class name
            // without custom implementation of ToString()
            // String represetation of the object!
            Console.WriteLine(e1.ToString());

            Console.ReadKey();
        }
    }
}
```

**Output**

```
Salary is :1000
First Name: George Last Name: Bush
```

# Generics

- Allows to design classes and methods decoupled from data types
- Generic classes are widely used by the collection classes available in the System.Collections.Generic namespace.
- Generics allow us to create a parameterized type
- They allow us to create code that is independent of the specific type and rely on the properties of the type.
- Particularly useful for cases where there are multiple sections of code performing the same duty but on different data types

# Generics

**Task:** Check whether two values are equal

This solution works for integer values only!

Can it work for string or other type?

**Output**

```
Number are not equal
```

**Example: Version1**

```
namespace GenericsDemo
{
    1 reference
    public class Calculator
    {
        // Works only for integer type!
        1 reference
        public static bool AreEqual (int number1, int number2)
        {
            return number1 == number2;
        }
    }

    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            // Check equality of two values
            bool checkEquality = Calculator.AreEqual(10, 15);

            if (checkEquality)
            {
                Console.WriteLine("Number are equal");
            }
            else
            {
                Console.WriteLine("Number are not equal");
            }

            Console.ReadKey();
        }
    }
}
```

# Generics

**Task:** Check whether two values of any type are equal

This solution works! And can be used with any types

**Issues:**

1. All type are inherited from System.Object
2. Performance degradation due to boxing and unboxing
3. Its is not strongly typed any more!

**Output**

```
Values are equal
```

**Example: Version2**

```
namespace GenericsDemo
{
    1 reference
    public class Calculator
    {
        // Works only for integer type!
        1 reference
        public static bool AreEqual (object value1, object value2)
        {
            return value1 == value2;
        }
    }

    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            // Check equality of two values
            bool checkEquality = Calculator.AreEqual("ABC", "ABC");

            if (checkEquality)
            {
                Console.WriteLine("Values are equal");
            }
            else
            {
                Console.WriteLine("Values are not equal");
            }

            Console.ReadKey();
        }
    }
}
```

# Generics

**Task**: Check whether two values of any type are equal

This solution works and solves the previous issues!

**Example: Version3**

```
namespace GenericsDemo
{
    2 references
    public class Calculator
    {
        // Works on any type of values using Generics
        2 references
        public static bool AreEqual <T> (T value1, T value2)
        {
            return value1.Equals(value2);
        }
    }

    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            // Check equality of two values
            bool checkEquality = Calculator.AreEqual<string>("ABC", "ABC");
            Console.WriteLine("1. Values are equal: {0}", checkEquality.ToString());

            checkEquality = Calculator.AreEqual<int>(12, 15);
            Console.WriteLine("2. Values are equal: {0}", checkEquality.ToString());

            Console.ReadKey();
        }
    }
}
```

**Output**

```
1. Values are equal: True
2. Values are equal: False
```