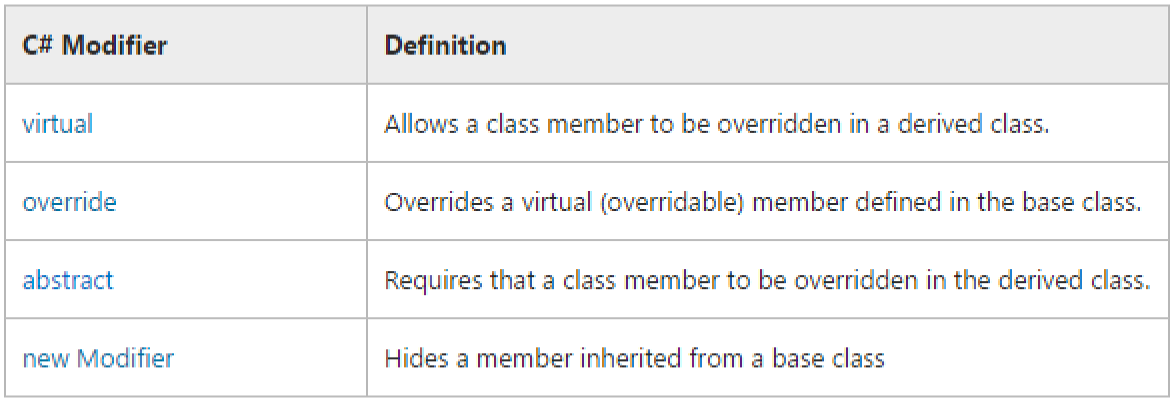
OBJECTIVES:

1. Understanding method overriding, hiding and working with them.
2. Understanding and working with timer control.
3. Understanding and working with List control.
4. Understanding and working with dynamic code generation.
5. Practice activities.

OBJECTIVE 1: Understanding method overriding, hiding and working with them.

Method overriding

⇨ By default, a derived class inherits all members from its base class.
⇨ If you want to change the behavior of the inherited member, you need to override it.
⇨ That is, you can define a new implementation of the method in the derived class.
⇨ The following modifiers are used to control how properties and methods are overridden.



C# Modifier	Definition
virtual	Allows a class member to be overridden in a derived class.
override	Overrides a virtual (overridable) member defined in the base class.
abstract	Requires that a class member to be overridden in the derived class.
new Modifier	Hides a member inherited from a base class

⇨ **Method overriding example:** In this example instead of virtual abstract keyword is used with the method which should be overridden.

```
abstract class ShapesClass
{
    abstract public int Area();
}
class Square : ShapesClass
{
    int side = 0;

    public Square(int n)
    {
        side = n;
    }
    public override int Area()
    {
        return side * side;
    }

    static void Main()
    {
        Square sq = new Square(12);
        Console.WriteLine("Area of the square = {0}", sq.Area());
    }

}
// Output: Area of the square = 144
```

Method hiding

⇨ The **new** keyword explicitly hides a member that is inherited from a base class.
⇨ When you hide an inherited member, the derived version of the member replaces the base class version.
⇨ Although you can hide members without using the **new** modifier, you get a compiler warning. If you use **new** to explicitly hide a member, it suppresses this warning.
⇨

```
public class BaseC
{
    public int x;
    public void Invoke() { }
}
public class DerivedC : BaseC
{
    new public void Invoke() { }
}
```

OBJECTIVE 3: Understanding and working with timer control.

Timer control or Timer class

- ⇨ Implements a timer that raises an event at user-defined intervals.
- ⇨ Whenever the Enabled property is set to **true** and the Interval property is greater than zero, the Tick event is raised periodically according to interval value set.
- ⇨ Start()instance method of Timer class starts the timer.
- ⇨ Stop() instance method of Timer class stops the timer.
- ⇨ Whenever Start() method of Timer control is called, an event of Timer control triggers automatically according to the Interval value defined using Interval property.

OBJECTIVE 4: Understanding and working with ListBox control.

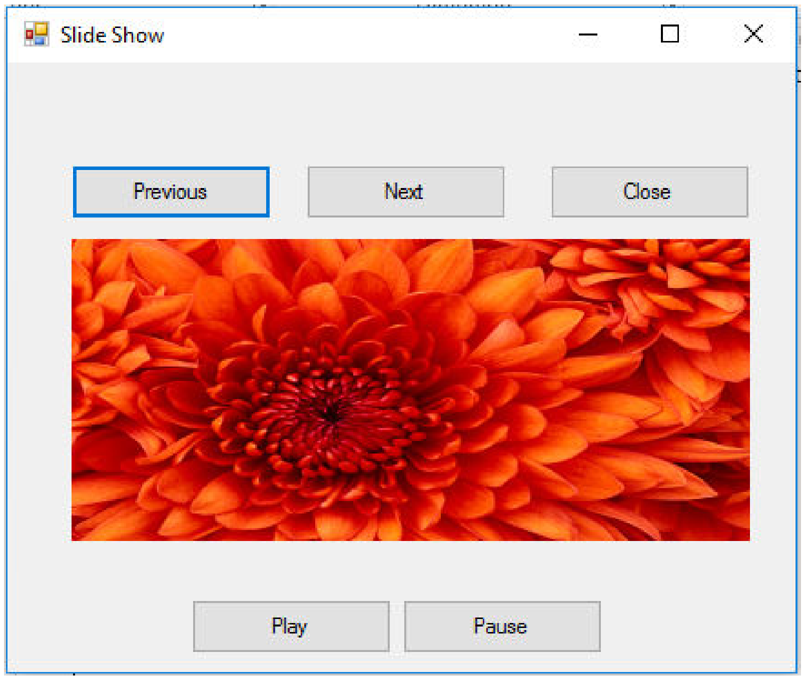
ListBox control or ListBox class

- ⇨ A list box is a control window that contains a simple list of items from which the user can choose.
- ⇨ Items collection is used to hold the items of ListBox.
- ⇨ Add(item) method of Items collection is used to add item in it.
- ⇨ Items collections indexing is zero based.

ACTIVITIES SECTION

ACITVITY 1: STEPS

- ⇨ Extending Activity 3 of Lab 4.
- ⇨ Create a windows forms application named PictureViewer.
- ⇨ Design interface of form according to interface given.
- ⇨ In this interface two new buttons have been introduced.
  - o btnPlay will start the slide show of images.
  - o btnPause will stop the slide show of images.



- ⇨ Conditions for program.
  - o After form loading, default flow of slide show is displaying the next image in slide show.
  - o If user has pressed the Next button before starting the slide show then flow will be same as default.
  - o If user has pressed the previous button before starting the slide show then flow of previous button will be the flow of slide show.
  - o If the flow is default and slide show reaches the last image, then it should start from the first image automatically.
  - o If the flow is not default and if slide show reaches the first image then slide show should continue from the last image.
  - o If user presses the Pause button for stopping the slide show then order should be kept no matter what order was there.
  - o If after pressing Pause button user clicks any of the button from previous or next then flow should be according to buttons.

[Note]: You can use Timer control to periodically call method that you want.

- ⇨ Start() instance method of Timer control start the Timer.
- ⇨ Stop() instance method of Timer control stops the Timer.
- ⇨ Interval property of Timer control sets interval in milliseconds for Timer to call its event periodically.
- ⇨ When Timer control starts its event will trigger periodically, so for invoking user defined method you should call your method in its event.

ACTIVITY 2: STEPS

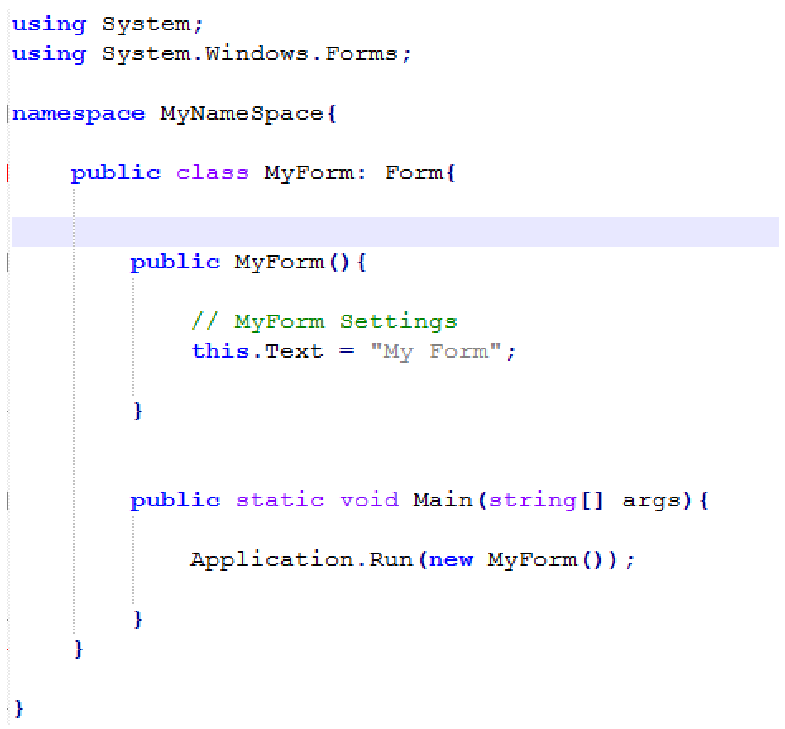
- ⇨ Create a file named MyForm.cs in a separate folder without using visual studio.

- ⇨ Place given code in the file, run the file and observe the results.
- ⇨ Execute the file using console by csc.
  - ○ csc MyForm.cs   // for compilation
  - ○ MyForm   // for execution

- ⇨ For path setting

```
D:\CSharp\csharpproject>set path="C:\Windows\Microsoft.NET\Framework64\v4.0.30319\";
```

- ⇨ **Version 1 of program**



```
using System;
using System.Windows.Forms;

namespace MyNameSpace{

    public class MyForm: Form{


        public MyForm(){

            // MyForm Settings
            this.Text = "My Form";

        }


        public static void Main(string[] args){

            Application.Run(new MyForm());

        }
    }

}
```

- ⇨ **Version 2 of program**
- ⇨ Run the program and observe the results.

```
using System;
using System.Windows.Forms;
namespace MyNameSpace{
    public class MyForm: Form{

        private Button btnLoad;
        private PictureBox pboxPhoto;
        public MyForm(){
            // MyForm Settings
            this.Text = "My Form";

            // btnLoad Settings
            btnLoad = new Button();
            btnLoad.Text = "&Load";
            btnLoad.Width = this.Width/2;
            btnLoad.Height = 20;
            btnLoad.Left = (this.Width-btnLoad.Width)/2;
            btnLoad.Top  =  50;
            btnLoad.UseMnemonic = true;

            // pboxPhoto Settings
            pboxPhoto = new PictureBox();
            pboxPhoto.BorderStyle = BorderStyle.Fixed3D;
            pboxPhoto.Width = this.Width/2;
            pboxPhoto.Height = this.Height/2;
            pboxPhoto.Left = (this.Width - pboxPhoto.Width)/2;
            pboxPhoto.Top = (this.Height - pboxPhoto.Height)/2;

            // Adding controls to the form
            this.Controls.Add(btnLoad);
            this.Controls.Add(pboxPhoto);
        }

        public static void Main(string[] args){
            Application.Run(new MyForm());
        }
    }
```

- ⇨ **Version 3 of program**
- ⇨ Run the program and observe the results.

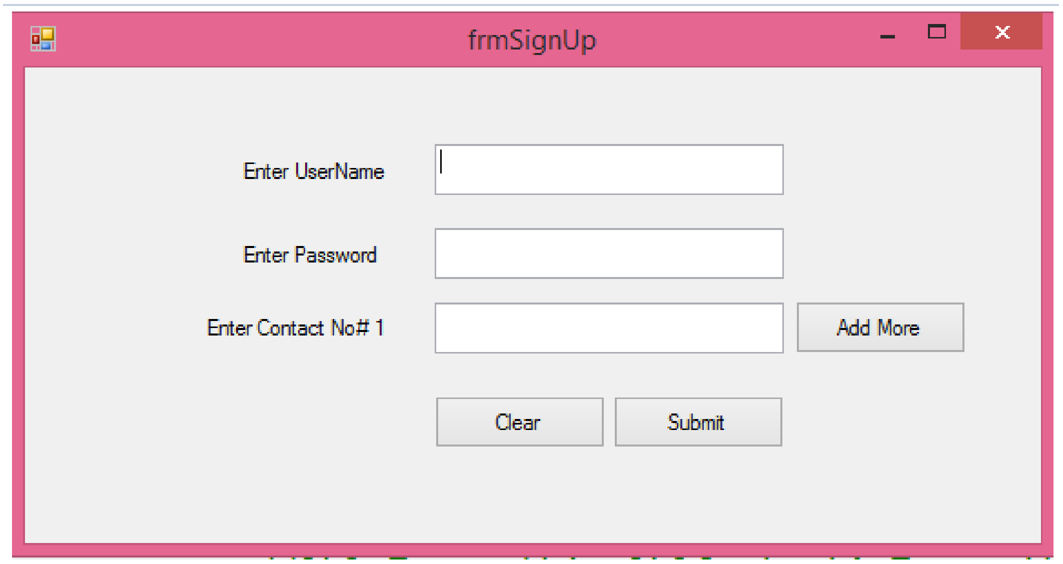
```
using System;
using System.Windows.Forms;
namespace MyNameSpace{
    public class MyForm: Form{
        private Button btnLoad;
        private PictureBox pboxPhoto;
        public MyForm(){
            // MyForm Settings
            this.Text = "My Form";
            // btnLoad Settings
            btnLoad = new Button();
            btnLoad.Text = "&Load";
            btnLoad.Width = this.Width/2;
            btnLoad.Height = 20;
            btnLoad.Left = (this.Width-btnLoad.Width)/2;
            btnLoad.Top  =  50;
            btnLoad.UseMnemonic = true;
            btnLoad.Click += new System.EventHandler(this.OnLoadClick);
            // pboxPhoto Settings
            pboxPhoto = new PictureBox();
            pboxPhoto.BorderStyle = BorderStyle.Fixed3D;
            pboxPhoto.Width = this.Width/2;
            pboxPhoto.Height = this.Height/2;
            pboxPhoto.Left = (this.Width - pboxPhoto.Width)/2;
            pboxPhoto.Top = (this.Height - pboxPhoto.Height)/2;
            // Adding controls to the form
            this.Controls.Add(btnLoad);
            this.Controls.Add(pboxPhoto);
        }
        private void OnLoadClick(object sender, System.EventArgs e){
            this.pboxPhoto.ImageLocation = @"D:\vb-pics\1.jpg";
        }
        public static void Main(string[] args){
            Application.Run(new MyForm());
        }
    }
}
```

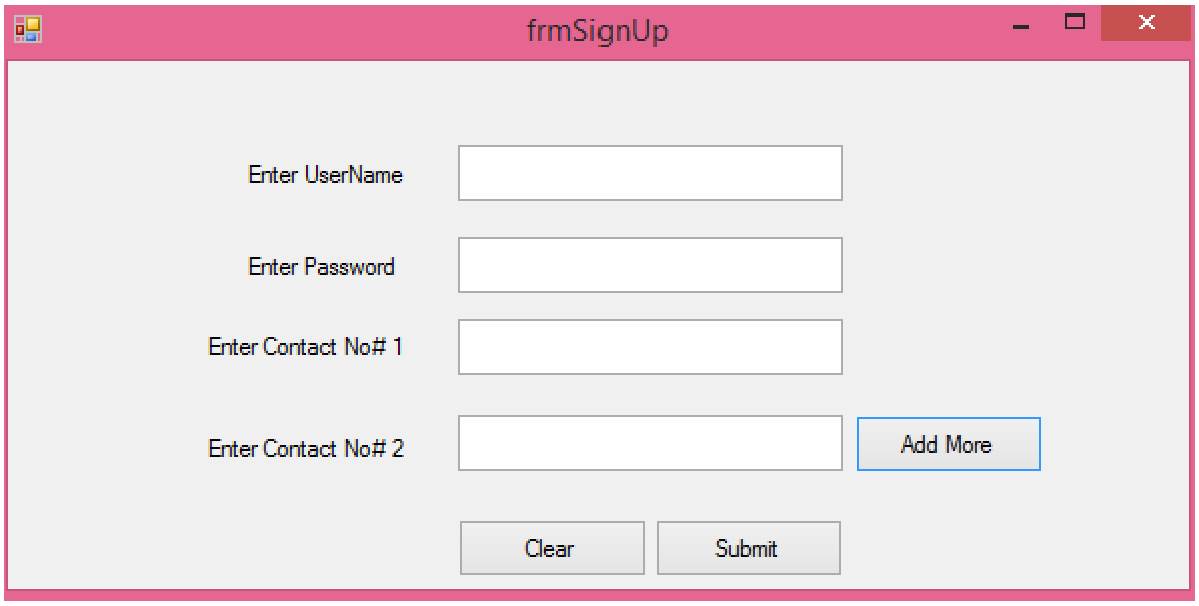
ACTIVITY 3: STEPS

- Create a Form for user registration named frmSignUp.
- Add following components on form.
    - **txtUsername** a text box.
    - **lblUsername** a label for user name.
    - **txtPassword** a text box.
    - **lblPassword** a label for password.
    - **txtContact1** a text box.
    - **lblContact1** a label for Contact1.
    - **btnSubmit** a button.
    - **btnClear** a button.
    - **btnAddMore** a button.

- Click on btnAddMore button second text box should be added to a form named **txtContact2** along with label named **lblContact2**.

- On click event of btnSubmit show all gathered information in a message box.





[Note]

- ⇨ Use **Controls** property of form to add new controls on form.
- ⇨ For example

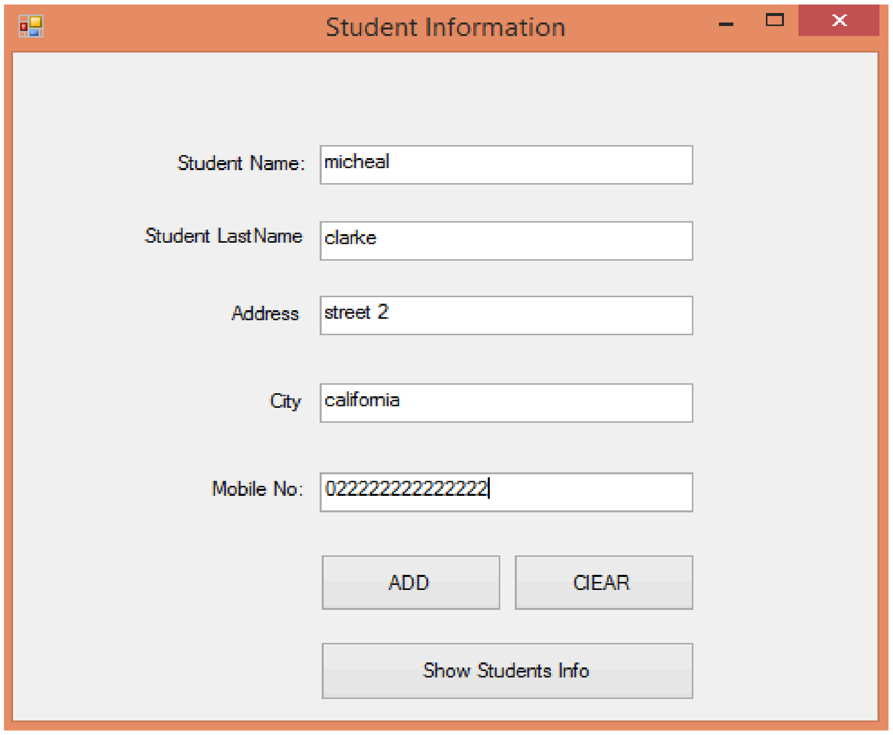
```
TextBox obj1 = new TextBox();
Obj1.Name = “txtContact1”;
This.Controls.Add(obj1);
```

For accessing value of dynamically added text box there are many ways.

1. Use object created for textbox to access its value.
2. User controls property of form control to access value of textbox added.
   `string contact1 = this.Controls[“txtContact1”].Text;`

ACTIVITY 4: STEPS

- ⇨ Create windows forms application named StudentManagmenet.
- ⇨ Create a form in it named frmStudentInfo.
- ⇨ Create given interface of form.



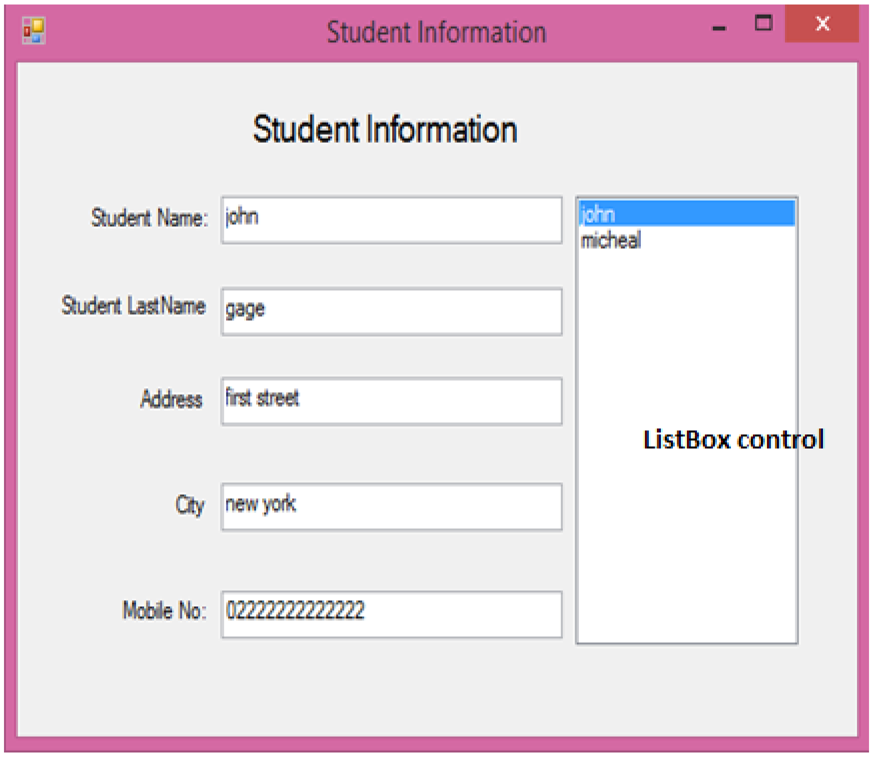
Student Name:	micheal
Student LastName	clarke
Address	street 2
City	california
Mobile No:	022222222222222

- ⇨ **Conditions for program.**
- ⇨ Save obtained information from form into object of StudentClass that's a model class when **btnAdd** is clicked.
- ⇨ If there is no information into the fields then appropriate message should be displayed to user.

- ⇨ **btnClear** will be used to clear information from all text fields.
- ⇨ **btnShowStudents** will be used to show form which will be used to display all student records added up to now.

- ⇨ Create a class named Database having static array of type StudentClass.
- ⇨ For demonstration purpose add two records of StudentClass in array of StudentClass.
- ⇨ After every object creation clear the information of textboxes, so that new record could be added to form.
- ⇨ After adding both records Click **btnShowStudents** to display second form and to hide current form.

- ⇨ Create another form named frmShowStudents
- ⇨ Create following interface of form.
- ⇨ One additional control is added into this form named lstStudents which is ListBox control.



Student Name:	john
Student LastName	gage
Address	first street
City	new york
Mobile No:	02222222222222

john
micheal

- ⇨ Add student records into the ListBox.
- ⇨ On selecting any student from ListBox, student's information should be displayed into the fields.

- ⇨ **Some useful properties and methods of ListBox.**

  - ListBox.Items.Add(item): will add specified item in ListBox
  - Item can be string value, integer value or any object.

  - Default event of ListBox is nameOfListBox_SelectIndexChanged
  - nameOfListBox.SelectedIndex will return the index of currently selected item from list box.

  - nameOfListBox.SelectedItem will return the selected item itself.