UNIT – 5

Working with Collections:

• The Collections in C# are a set of predefined classes that are present in the System.Collections namespace that provides greater capabilities and functionalities than the traditional arrays.

• They are reusable, reliable powerful and efficient and have been carefully designed and tested to ensure quality of performance.

• Collections are similar to arrays but provides additional functionalities such as dynamic resizing they automatically increase their size at execution time to accommodate additional elements, inserting of new elements removing of existing elements etc.

• Collection like list, array list, queue, stack, sorted list etc.

• Collection represents group of objects which is used to perform various functions such as sorting, updating, deleting, retrieving, searching and sorting of objects.

List:

The list is declared as **List<T>.** It is a class in c# that represents a strongly typed list of objects. C# list class to add, find, sort, reverse, and search items in a collection of objects using List class methods and properties.

To add value in List using predefined method as Add().

Example:

```
using System;

using System.Collection.Generic;

class ListDemo

{

  static void Main(string[] args)

  {

    String Student = new List();

    //list is created that can store string elements Student.Add(“Our”);

    Student.Add(“Creative”); Student.Add(“Info,”);

    Student.Add(“Karnataka”);

    foreach (string name in Student)

   {
```

```
        Console.WriteLine(name);
    }
    Console.ReadLine();
   }
}
```

Output:

Our Creative Info, Karnataka

Dictionary <TKey, TValue>

•The Dictionary is a generic collection that stores key-value pairs in no particular order.

Dictionary Characteristics

•The Dictionary stores key-value pairs.

•It comes under System.Collections.Generic namespace.

•Implements IDictionary interface.

•Value can be null or duplicate.

Example:

```
using System;
using System.Collections.Generic;
namespace DictionaryDemo
{
 class Program
 {
  static void Main(string[] args)
  {
     Dictionary d = new Dictionary();
     d.Add(1, "amar");
     d.Add(2, "abhi");
     foreach(KeyValuePair obj in d)
     {
         Console.WriteLine(obj);
```

```
    }
  }
 }
}
```

ArrayList: ----

• In C#, the ArrayList is a non-generic collection of objects whose size increases dynamically. It is the same as Array except that its size increases dynamically. ArrayList is a class that is similar to an array, but it can be used to store values of various types.

• An ArrayList doesn't have a specific size.

• Any number of elements can be stored.

Example:

```
using System;
using System.collections;
namespace arraylistDemo
 {
  class Program
 {
   Static void Main(string[] args)
{
    ArrayList Ar = new ArrayList();
    Ar.Add(10); Ar.Add(20);
    foreach (int i in Ar)
    {
    Console.WriteLine(i);
     }
     Console.ReadLine();
     }
  }
}
```

C# Hashtable:

• HashTable is similar to ArrayList but represents the items as a combination of key and value.

• The Hashtable is a non-generic collection that stores key-value pairs, similar to generic Dictionary<TKey, TValue> collection.

Hashtable Characteristics

• HashTable stores key-value pairs.

• It comes under System.Collections namespace.

• Keys must be unique and cannot be null.

• Values can be null or duplicate.

Example:

```
using System;
using System.collections;
namespace HashtableDemo
{
  class Program
{
  static void Main(string[] args)
{
  Hashtable a = new Hashtable();
  a.Add(1,”Arun”);
  a.Add(2,”Akash”);
  Console.WriteLine(a[1]);
  Console.WriteLine(a[2]);
 }
}
}
```

Generic Classes:

• Generic classes define functionalities that can be used for any data type and are declared with a class declaration followed by a type parameter (T) enclosed within angular brackets (<>).

• Generic classes are defined using a type parameter in angle brackets after the class name.

• Generic introduced in C# 2.0.

- T supports all data types involved in C#.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication5
{
class GenericDemo
{
T student;
public GenericDemo(T s)
{
  this.student = s;
 }
   public T getstudent()
 {
  return this.student;
 }
}
 class Program
{
   static void Main(string[] args)
   {
         GenericDemo gd = new GenericDemo(10);
         GenericDemo gd1 = new GenericDemo("BCA");
         GenericDemo gd2 = new GenericDemo(5.5f);
         Console.WriteLine(gd.getstudent());
         Console.WriteLine(gd1.getstudent());
```

```
        Console.WriteLine(gd2.getstudent());
        Console.ReadLine();
    }
  }
}
```

Output:

10

BCA

5.5

Comparable:

- It is class under a collection namespace.
- The comparable class can performe only to compare the values.
- To define in program we must declare namespace (i.e using System.Collection.Generic;)
- In this comparable class only compare source value to any destination value.

Sorting:

- This method can be used to sort the particular list/value.
- But this sorting method uses when we declare a class comparable otherwise not possible to sort the list/value.
- If you declare the sort method without a comparable class then your output window becomes a blank screen.

Example:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace ConsoleApplication6
 {
 public class student : IComparable<student>
{
```

```
public int Sid {get; set;}
public string Name {get; set;}
 public int CompareTo(student other)
{
if(this.Sid>other.Sid)
return 1;
else if(this.Sid<other.Sid)
return -1;
else
return 0;
}
}
class compareStudent
{
static void Main(string[] args)
{
student s1 = new student { Sid = 2, Name = "Amit" };
student s2 = new student { Sid = 1, Name = "Arun" };
student s3 = new student { Sid = 5, Name = "Kiran" };
student s4 = new student { Sid = 3, Name = "Suman" };
student s5 = new student { Sid = 4, Name = "Sohan" };
List students = new List() { s1, s2, s3, s4, s5 };
students.Sort();
foreach (student s in students)
{
Console.WriteLine(s.Sid + " " + s.Name);
}
Console.ReadLine();
}
}
```

}

OUTPUT:

1 Arun

2 Amit

3 Suman

4 Sohan

5 Kiran

WinForms: (Introduction)

• Windows Forms is a Graphical User Interface(GUI) class library that is bundled in .Net Framework.

• Its main purpose is to provide an easier interface to develop applications for desktops, tablets, and PCs.

• It is also termed the WinForms. The applications which are developed by using Windows Forms or WinForms are known as the Windows Forms Applications that run on the desktop computer.

• WinForms can be used only to develop Windows Forms Applications not web applications. WinForms applications can contain different types of controls like labels, list boxes, textboxes, etc.

Create a project

First, you'll create a C# application project. The project type comes with all the template files you'll need before you've even added anything.

1. Open Visual Studio.

2. On the start window, select **new project**.

3. In New Project Window Select Windows Forms Application and give a name to the project and Click **OK**

Controls:

The Control class implements very basic functionality required by classes that display information to the user. It handles user input through the keyboard and pointing devices.

C# Control Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client-side Windows applications.

Control & Description:

Button:

Fires an event when a mouse click occurs or the Enter or Esc key is pressed. Represents a button on a form. Its text property determines the captiondisplayed on the button's surface.

CheckBox:

Permits a user to select one or more options.Consists of a check box with text or an image beside it. The check box can also be represented as a buttonby setting: checkBox1.Appearance = Appearance.Button.

CheckedListBox:

Displays list of items. ListBox with checkbox preceding each item in list. ComboBox Provides TextBox and ListBox functionality. Hybrid control that consists of a textbox and a drop-down list. It combines properties from boththe TextBox and the ListBox.

Label:

Adds descriptive information to a form. Text that describes the contents of control or instructions for using a control or form.

ListBox:

Displays a list of items—one or more of which may be selected. May contain simple text or objects. Its methods, properties, and events allowitems to be selected, modified, added, and sorted.

MenuStrip:

Adds a menu to a form. Provides a menu and submenu system for a form. It supersedes the MainMenu control.

RadioButton:

Permits user to make one choice among a group ofoptions. Represents aWindows radio button.

TextBox:

Accepts user input. Canbe designed to accept single- or multi-line input. Properties allow it to mask input for passwords, scroll, set letter casingautomatically, and limit contents to read-only.

TreeView:

Displays data as nodes in a tree. Features include the ability to collapse or expand, add, remove, and copy nodes in a tree.

Button:

Button control is used to perform a click event in Windows Forms, and it can be clicked by a mouse or by pressing Enter keys. It is used to submit all queries of the form by clicking the submit button or transferring control to the next form.

Buttons are one of the simplest controls and are mostly used for executing some code when the user wants.

Button Properties

BackColor: It is used to set the background color of the Button.

BackgroundImage: It is used to set the background image of the button control.

ForeColor: It is used to set or get the foreground color of the button.

Image: It is used to set or gets the image on the button control that is displayed.

Text: It is used to set the name of the button control.

Button Events

BackColorChanged: A BackColorChaged event is found in button control when the Background property is changed.

BackgroundImage: Changed A BackgoundImageChanged event is found in button control when the value of the BackgoundImage property is changed.

Click: A Click event is found in the button control when the control is clicked.

DoubleClick: When the user makes a double-click on the button, a double-click event is found in the button control.

TextChanged: It is found in the button control when the value of the text property is changed.

Example:

code for the MessageBox there:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
{
   MessageBox.Show("Wellcome to Controls");
}
```

TextBox:

Text box controls are used for entering text at runtime on a form. A single line of text is taken in by default. But, you can change the settings to accept multiple texts. Also, we can even add scroll bars to it. TextBoxes allows the user to input data into the program.

TextBox Properties

TextAlign: It is used for setting text alignment.

ScrollBars: It is used for adding scrollbars, both vertical and horizontal.

Multiline: It is used to set the TextBox Control to allow multiple lines.

MaxLength: It is used for specifying the maximum character number the TextBox Control will accept.

Font: It is used to display text in the control.

Example:

the button click event to use the text of the textbox:

```
private void cmdShowMessage_Click(object sender, EventArgs e)
```

```
{
   string UserText = txtUserMessage.Text; MessageBox.Show(UserText);
}
```

Label:

Label control is used as a display medium for text on Forms. Label control does not participate in user input or capture mouse or keyboard events.

Label Properties

Name: Name property represents a unique name of a Label control. It is used to access the control in the code.

Font: Font property represents the font of text of a Label control.

Text: Text property of a Label represents the current text of a Label control.

TextAlign: The TextAlign property represents text alignment that can be Left, Center, or Right.

MaxLength: The TextLength property returns the length of a Label's contents.

Menus And Context menus

Menus

• An imperative part of the user interface in a Windows-based application is the menu.

• A menu ona form is created with a Menu object, which is a collection of MenuItem objects.

• You can add menus to Windows Forms at design time by adding the Menu control andthen adding menu items to it using the Menu Designer.

• Menus can also be added programmatically by creating one or more Menu controls objects in to a form and adding MenuItem objects to the collection.

Different types of menus available in C# are

➢ MenuStrip - used to create normal horizontal or vertical menus.

➢ ContextMenu - used to create popup menus. Menus appears when the mouse rightclicked and disappears once selection is made.

➢ ToolStrip - used to create menus when there is less space and these are called as dropdown menus.

➢ StatusStrip - used to create status bar where status of formcontrols can be displayed.

Context Menu:

A context menu is a group of commands or menu items that can be accessed by right-clicking on the control surface. It usually contains some frequently used commands for example Cut, Copy and Paste in a text editor.

• In Windows Forms, a context menu is created using the ContextMenuStrip control and its command or menu items are ToolStripMenuItem objects.

• The Image property of ToolStripMenuItem is used to add an image to a menu item. In design view, select the menu item and go to its property and click the ellipsis next to the Image property and select image from the Local Resource or Project Resource File.

• A context menu is also known as a popup menu. A context menu appears when you right click on a Form or on a control.

Type of items in ContextMenuStrip

1. MenuItem (ToolStripMenuItem): It is used to give a simple menu item like "Exit" inthe above example.

2. ComboBox(ToolStripComboBox): It is used to insert a ComboBox in the context menu where the user can select or type an item in the ComboBox.

3. Separator (ToolStripSeparator): It is used to put a horizontal line (ruler) betweenmenu items.

4. TextBox (ToolStripTextBox): It is used to put a TextBox in the context menu where the user can enter an item.

MenuStrip

• MenuStrip adds a menu bar to Windows Forms program.

• With this control, we add a menu area and then add the default menus or create custommenus directly in Visual Studio.

• We can create a menu using MenuStrip control at design-time or using the MenuStrip classin code at run-time or dynamically.

• Once a MenuStrip isonthe Form, you can add menu items and set its properties and events.

Type of items in MenuStrip

• MenuItem (ToolStripMenuItem): It is used to give a simple menu item like "Exit".

• ComboBox(ToolStripComboBox): It is used to insert a ComboBox in the context menuwhere the user can select or type an item in the ComboBox.

• Separator (ToolStripSeparator): It is used to put a horizontal line (ruler) between menuitems.

• TextBox (ToolStripTextBox): It is used to put a TextBox in the context menu where theuser can enter an item.

Toolstrip

• ToolStrip control provides functionality of Windows toolbar controls in Visual Studio 2010.

• This menu is also called as dropdown menu because in the form only an arrow will be appearing when we click on arrow menu will appear.

• To create aToolStrip control at design-time, simply drag and drop a ToolStrip control fromToolbox onto a Form.

• At run time this can be created by creating the object of toolStrip class.

Type of items in toolstrip

• Button: Used to create a toolbar button that supports both text and images. Represents a selectable ToolStripItem.

• Separator: Represents a line used to group items. Use this to group related items on a menu or ToolStrip. The Separator is automatically spaced and oriented horizontally or vertically to accord with its container.

• Label: Used to create a label that can render text and images that can implement the ToolStripItem.

• DropDownButton: It looks like ToolStripButton, but it shows a drop-down area when the user clicks it.

• SplitButton: Represents a combination of a standard button on the left and a drop□down button on the right, or the other way around if the value of RightToLeft is Yes.

• ComboBox: Displays an editing field combined with a ListBox, allowing the user to select from the list or to enter new text. By default, a ToolStripComboBox displays an edit field with a hidden drop-down list.

MDI

• MDI stands for Multiple Document Interface.

• It is an interface design for handling documents within a single application.

• When application consists of an MDI parent form containing all other window consisted by app, then MDI interface can be used.

• Switch focus to specific document can be easily handled in MDI. For maximizing alldocuments, parent window is maximized by MDI

SDI

• SDI stands for Single Document Interface.

• It is an interface designed for handling documents within a single application.

• SDI exists independently from others and thus is a stand-alone window.

• SDI supports one interface means you can handle only one application at a time.

• For grouping,SDI uses special window managers.

Key Difference

SDI	MDI
Stands for “Single Document Interface”.	Stands for “Multiple Document Interface”
One document per window is enforced in SDI	Child windows per document are allowed inMDI.
SDI is not a container control	MDI is a container control
SDI contains one window only at a time	MDI contains multiple documents at a time that appeared as a child window
SDI supports one interface means you can handleonly one application at a time.	MDI supports many interfaces means we can handle many applications at a time according to the user's requirements.
SDI uses Task Manager for switching betweendocuments	For switching between documents MDI uses a special interface inside the parent window
SDI grouping is possible through special window managers.	MDI grouping is implemented naturally
In SDI it is implemented through a special code or window manager.	For maximizing all documents, the parent window is maximized by MDI
It is difficult to implement in SDI.	Switch focus to a specific document can be easily handled.

Dialog boxes

Dialog boxes are of two types, which are given below.

1. Modal dialog box

➢ A dialog box that temporarily halts the application and the user cannot continue until the dialog has been closed is called a modal dialog box.

➢ The application may require some additional information before it can continue or may simply wish to confirm that the user wants to proceed.

➢ The application continues to execute only after the dialog box is closed, until then the application halts.

For example, when saving a file, the user gives a name of an existing file; a warning is shown that a file with the same name exists, whether it should be overwritten or be saved with a different name. The file will not be saved unless the user selects “OK” or “Cancel”.

2. Modeless dialog box

➢ It is used when the requested information is not essential to continue.

➢ The Window can be left open, while work continues somewhere else.

For example, when working in a text editor, the user wants to find and replace a particular word. This can be done, using a dialog box, which asks for the word to be found and replaced. The user can continue to work, even if this box.

Example: Demonstrate Dialog box (Modal and Modeless)

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace DailogBoxExample

 {

static class Program { static void Main()

{

DialogResult result = MessageBox.Show("Do you want to continue?",
"Confirmation",MessageBoxButtons.YesNo, MessageBoxIcon.Question);

  if (result == DialogResult.Yes)

 {

   MessageBox.Show("You clicked Yes");

 }

 else

 {

   MessageBox.Show("You clicked No");

 }

 }

}
```

}

From Inheritance

• In order to understand the power of OOP, consider, for example, form inheritance, a new feature of .NET that lets you create a base form that becomes the basis for creating more advanced forms. The new "derived" forms automatically inherit all the functionality contained in the base form. This design paradigm makes it easy to group common functionality and, in the process, reduce maintenance costs.

• When the base form is modified, the "derived" classes automatically follow suit and adopt the changes. The same concept applies to any type of object.

Example:

Visual inheritance allows you to see the controls on the base form and to add new controls. In this sample we will create a base dialog form and compile it into a class library. You will import this class library into another project and create a new form that inherits from the base dialog form. During this sample, you will see how to:

• Create a class library project containing a base dialog form.

• Add a Panel with properties that derived classes of the base form can modify.

• Add typical buttons that cannot be modified by inheritors of the base form

Developing Custom Controls

• Use Custom Panel/Panel as a border of a Frame for change the size of Form such as Width,Height etc. and use Custom Buttons as a Control Box of a Frame.

• Use Custom Panel as a Top layered Title border of a frame and use label to display text of a frame.

• To display icon on a frame we will set background image to added panel.

• Applying Mouse Events to panels and change properties of Form.

Composite and Extended Controls

• Composite controls provide a means by which custom graphical interfaces can be created and reused. A composite control is essentially a component with a visual representation.

• As such, it might consist of one or more Windows Forms controls, components, or blocks of code that can extend functionality by validating user input, modifying display properties, or performing other tasks required by the author.

• Composite controls can be placed on Windows Forms in the same manner as other controls.

• In the first part of this walkthrough, you create a simple composite control called ctlClock.

• In the second part of the walkthrough, you extend the functionality of ctlClock throughinheritance.

Add Windows Controls and Components to the Composite Control

• A visual interface is an essential part of your composite control.

• This visual interface is implemented by the addition of one or more Windows controls tothe designer surface.

• In the following demonstration, you will incorporate Windows controls into your compositecontrol and write code to implement functionality.

To add a Label and a Timer to your composite control

• In Solution Explorer, right-click ctlClock.cs, and then click View Designer.

• In the Toolbox, expand the Common Controls node, and then double-click Label.

• A Label control named label1 is added to your control on the designer surface.

Click labell. In the properties window

• In the Toolbox, expand the Components node, and then double-click Timer.

• Because a Timer is a component, it has no visual representation at run time.

• Therefore, it does not appear with the controls on the designer surface, but rather in theComponent Designer (a tray at the bottom of the designer surface).

• In the Component Designer, click timer1, and then set the Interval property to 1000 and theEnabled property to true.

• The Interval property controls the frequency with which the Timer component ticks. Eachtime timer1 ticks, it runs the code in the timer1_Tick event.

• The interval represents the number of milliseconds between ticks. In the Component Designer, double-click timer1 to go to the timer1_Tick event for ctlClock.

Thank You

(www.ourcreativeinfo.in)