USN	1	P	E			M	C	A		

PESIT Bangalore South Campus
Hosur Road, 1km before Electronic City, Bengaluru -560100
Department of Master of Computer Applications

INTERNAL ASSESSMENT TEST I

Date: 23/09/19	**Max Marks:** 40
Subject & Code: Programming Using C#.NET (17MCA52)	**Section:** V MCA
Name of Faculty: Ms. Afshan Rehman	**Time:** 11:30-1:00 PM

Note: ***Answer FIVE full questions. Select one question from each part.***

	Part I	
Q1	Write a C# program to demonstrate abstract classes and abstract methods.	**8**
	OR	
Q2	What are try, catch, throw and finally blocks? Write a C# Program using these blocks to explain check overflow using checked and unchecked statements.	**8**
	Part II	
Q3	How are Delegates used in C#? Explain single cast and multicast delegates?	**8**
	OR	
Q4	Explain the steps involved in creating and using delegate in the program.	**8**
	Part III	
Q5	List and explain the common ADO.NET objects?	**8**
	OR	
Q6	What are Data Adapters? List different Data Adapters in a .NET Framework. Write a C# code to create a dataset from Data Adapter.	**8**
	Part IV	
Q7	Explain group boxes and panel control with events, properties and an example with respect to windows form.	**8**
	OR	
Q8	What are events? Write a C# console application to demonstrate the concept of multiple event handlers for a single event.	**8**
	Part V	
Q9	a) Explain the various mouse events in C# windows application. b) Write a note on Tooltips.	**5** **3**
	OR	
Q10	What are the features of interface? Write a C# program to demonstrate the use of interfaces.	**8**

USN	1	P	E			M	C	A		

PESIT Bangalore South Campus
Hosur Road, 1km before Electronic City, Bengaluru -560100
Department of Master of Computer Applications

Part – 1

1. Write a C# program to demonstrate abstract classes and abstract methods.

Abstraction in C# is the process to hide the internal details and showing only the functionality. The keyword **abstract** is used before the class or method to declare the class or method as abstract.

A method which is declared abstract, has no "body" and declared inside the abstract class only.

Program to demonstrate Abstract classes and abstract methods:

```
using System;
 // declare class 'AreaClass' as abstract
abstract class AreaClass
{
   // declare method 'Area' as abstract
   abstract public int Area();
}
 // class 'AreaClass' inherit in child class 'Square'
class Square : AreaClass
{
   int side = 0;
    // constructor
   public Square(int n)
```

```
  {
     side = n;
  }
   // the abstract method 'Area' is overridden here
  public override int Area()
  {
     return side * side;
  }
}


 class gfg {
   // Main Method
  public static void Main()
  {
     Square s = new Square(6);
     Console.WriteLine("Area  = " + s.Area());
  }
}
```

2. What are try, catch, throw and finally blocks? Write a C# Program using these blocks to explain check overflow using checked and unchecked statements.

- **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.

- **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
- **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **throw** − A program throws an exception when a problem shows up. This is done using a throw keyword.

Checked

The **checked** keyword is used to control the overflow-checking context for integral-type arithmetic operations and conversions. It can be used as an operator or a statement according to the following forms.

Unchecked

The **unchecked** keyword is used to control the overflow-checking context for integral-type arithmetic operations and conversions. It can be used as an operator or a statement according to the following forms.

Example Program:

```
namespace Checked_Unchecked
{
    class Program
```

```
{
    public short a = 30000;
    public short b = 20000;
    public short c;

    public int Add()
    {
        try
        {
            c = checked((short)(a + b));

        }
        catch (System.OverflowException e)
        {
            System.Console.WriteLine(e.ToString());
        }
        return c;
    }

    public int Mul()
    {
        try
        {
            checked
            {
                c = (short)(a * b);
            }
        }
        catch (System.OverflowException e)
        {
            System.Console.WriteLine(e.ToString());
        }
        return c;
    }

    public int Add_Unchecked()
    {
        try
        {
            c = unchecked((short)(a + b));

        }
        catch (System.OverflowException e)
        {
            System.Console.WriteLine(e.ToString());
        }
        return c;
    }

    public int Mul_Unchecked()
    {
        try
        {
            unchecked
            {
                c = (short)(a * b);
            }
        }
        catch (System.OverflowException e)
        {
          System.Console.WriteLine(e.ToString());
        }
        return c;
      }

      static void Main(string[] args)
      {
        Program p = new Program();

        // For checked
        Console.WriteLine("Checked output value is: {0}", p.Add());
        Console.WriteLine("Checked output value is: {0}", p.Mul());
        // For Unchecked
        Console.WriteLine("Checked output value is: {0}", p.Add_Unchecked());
        Console.WriteLine("Checked output value is: {0}", p.Mul_Unchecked());
        Console.ReadKey(true);
      }
    }
}
```

Part – 2

3. How are Delegates used in C#? Explain single cast and multicast delegates?

A delegate is an object which refers to a method or you can say it is a reference type variable that can hold a reference to the methods. Delegates in C# are similar to the function pointer in C/C++. It provides a way which tells which method is to be called when an event is triggered. For example, if you click an *Button* on a form (Windows Form application), the program would call a specific method. In simple words, it is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed.

Syntax : [modifier] delegate [return_type] [delegate_name] ([parameter_list]);

Declaration Example: public delegate int GeeksForGeeks(int G, int F, int G);

Multicasting of delegate is an extension of the normal delegate(sometimes termed as Single Cast Delegate). It helps the user to point more than one method in a single call.

Program to demonstrate use of delegates

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

public delegate void Operation(int p, int q);
namespace prog7
{
class Program
{
public static void Addnum(int p,int q)
{
```

```
Console.WriteLine(p + q);
}
public static void Mulnum(int p, int q)
{
Console.WriteLine(p * q);
}
static void Main(string[] args)
{
Operation op1 = Addnum;
op1 += Mulnum;
op1(10,20);
}
}
}
```

4. Explain the steps involved in creating and using delegate in the program.

Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System.Delegate class.

Following are the four steps to create and use a delegate in program:-

Declaring a delegate

Defining delegate methods

Creating delegate objects

Invoking delegate objects.

Declaring Delegates

Delegate declaration determines the methods that can be referenced by the delegate. A delegate can refer to method, which has the same signature as that of the delegate.

Syntax for delegate declaration is −

delegate <return type> <delegate-name> <parameter list>

For example, consider a delegate −

public delegate void compute(int x, int y);

Defining delegates method

A delegate method is any method whose signature (number, type of parameter and return type) matches the delegate signature exactly.

It can be either a static method or a instance method.

```
Public static void add(int a, int b);
{
    Console.write("sum={0}, a+b);
}
Public void subtract(int a, int b)
{
    Console.write("difference={0}, a-b);
}
```

Creating delegate object

You can create object of a delegate by using following syntax

Delegate-name object_name = new delegate_name (expression);

Where, expression is what you want to encapsulate into the delegate may be a method name or other delegate object

```
Compute cmp1 = new Compute(DelegateTest.add);
DelegateTest dt = new DelagateTest();
Compute cmp2 = new Compute(dt.subtract);
```

Invoking delegate object

A delegate is invoked as a method is invoked. The delegate can be invoked by two ways:

using () operator or using the Invoke() method of delegate

Following syntax is used

Delegate_object (argument_list);

Cmp1(30,20);

Cmp2(30,10);

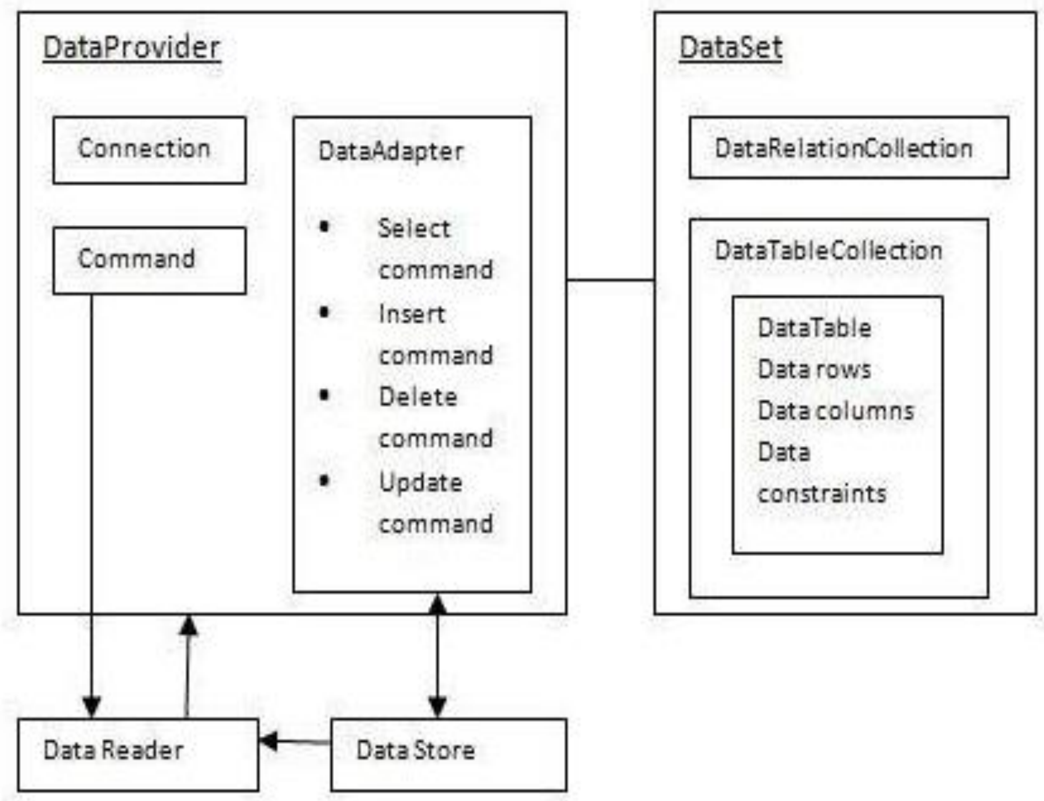
A delegates can be used to change the behavior of a program at runtime.

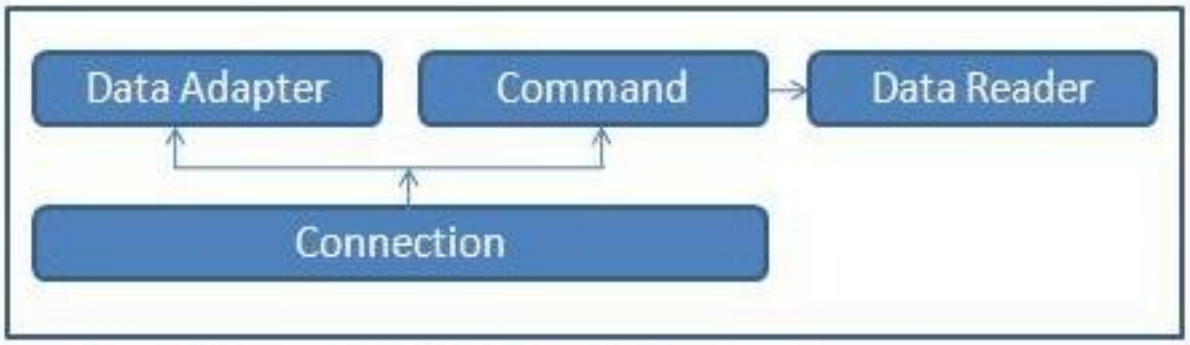
Part - 3

5. List and explain the common ADO.NET objects?

ADO.NET provides a bridge between the front end controls and the back end database. The ADO.NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.

The following figure shows the ADO.NET objects at a glance:





The four Objects from the .Net Framework provide the functionality of Data Providers in ADO.NET.

a) The *Connection Object* provides physical connection to the Data Source. The base class for all Connection objects is the DbConnection class.

b) The *Command Object* uses to perform SQL statement or stored procedure to be executed at the Data Source. The base class for all Command objects is the DbCommand class.

c) The *DataReader Object* is a stream-based , forward-only, read-only retrieval of query results from the Data Source, which do not update the data. The base class for all Data Reader objects is the DbDataReader class.

d) Finally the *DataAdapter Object* , which populate a Dataset Object with results from a Data Source . The base class for all Data Adapter objects is the DbDataAdapter class.

ASP.NET Data Providers

The .Net Framework includes mainly three Data Providers for ADO.NET.

They are the Microsoft SQL Server Data Provider , OLEDB Data Provider and ODBC Data Provider .

a) SQL Server uses the SqlConnection object
b) OLEDB uses the OleDbConnection Object
c) ODBC uses OdbcConnection Object

Dataset

The DataSet object is central to supporting disconnected, distributed data scenarios with ADO.NET. The DataSet represents a complete set of data, including related tables, constraints, and relationships among the tables. The following illustration shows the DataSet object model.

The methods and objects in a DataSet are consistent with those in the relational database model.

The DataSet can also persist and reload its contents as XML, and its schema as XSD schema.

6. What are Data Adapters? List different Data Adapters in a .NET Framework. Write a C# code to create a dataset from Data Adapter.

DataAdapter is a part of the ADO.NET Data Provider. DataAdapter provides the communication between the **Dataset** and the Datasource. We can use the DataAdapter in combination with the DataSet Object. DataAdapter provides this combination by mapping Fill method, which changes the data in the DataSet to match the data in the data source, and Update, which changes the data in the data source to match the data in the DataSet. That is, these two objects combine to enable both data access and data manipulation capabilities.

The **DataAdapter** can perform Select , Insert , Update and Delete SQL operations in the Data Source. The Insert , Update and Delete SQL operations , we are using the continuation of the Select command perform by the DataAdapter. The **SelectCommand** property of the DataAdapter is a Command Object that retrieves data from the data source. The **InsertCommand** , **UpdateCommand** , and **DeleteCommand** properties of the DataAdapter are Command objects that manage updates to the data in the data source according to modifications made to the data in the DataSet. From the following links describes how to use SqlDataAdapter and **OleDbDataAdapter** in detail.

DataAdapter

The DataAdapter constructor has many overloaded forms. You can create a constructor with no arguments, pass a Command object, pass a Command object with Connection object as arguments, or any combination of these. You can also specify a SQL statement as string for querying a particular table or more than one table. You can also specify the connection string or a Connection object to connect to the database.

Listing 5-41 creates a connection, builds a SQL statement using the SELECT query, and passes the SQL string and the connection objects as SqlDataAdapter constructor arguments. I've been using in the previous examples.

Listing 5-41. Executing a SELECT statement using SqlDataAdapter

```
string ConnectionString = "Integrated Security = SSPI;" +
"Initial catalog = Northwind; " + " Data Source = localhost; ";
string SQL = "SELECT CustomerID, CompanyName FROM Customers";
SqlConenction conn = new SqlConnection(ConnectionString);

// open the connection
conn.Open();

// Create a SqlDataAdapter object
SqlDataAdapter adapter = new SqlDataAdapter(SQL, conn);
```

As discussed earlier, there is no difference in creating OleDb, Sql, or ODBC data adapters. The only difference is the connection string. For example, the following code snippet shows you how to create an OleDbDataAdapter object.

Listing 5-42 uses the Access2000 North wind database and accesses all records of the Orders table by using a SELECT *SQL query.

Listing 5-42. Executing a SELECT statement using OleDbDataAdapter

```
// create a connection object
string ConnectionString = @"Provider =Microsoft.Jet.OLEDB.4.0; " +
"Data source = c:\\ Northwind.mdb";
string SQL = "SELECT * FROM orders";
OleDbConnection conn = new OleDbConnection(ConnectionString);
```

```
// open the connection
conn.Open ( );

// create an OleDbDataAdapter objecto
OleDbDataAdapter adapter = new OleDbDataAdapter(SQL, conn);
```

You can also use DataAdapter's Command properties by using the Command object with OleDbDataAdapter. For example, the following code uses OleDbCommand to set the SelectCommand property of the data adapter. You can also see that OleDbDataAdapter has no arguments as its constructor:

```
// Create an OleDbDataAdapter object
OleDbDataAdapter adapter = new OleDbDataAdapter();
adapter.SelectCommand = new OleDbCommand (SQL, conn);
```

Part - 4

7. Explain group boxes and panel control with events, properties and an example with respect to windows form.

GroupBoxes and Panels

Group Boxes and panels arrange controls on a GUI. They are used to group several controls of similar functionality or if they are related in a GUI.

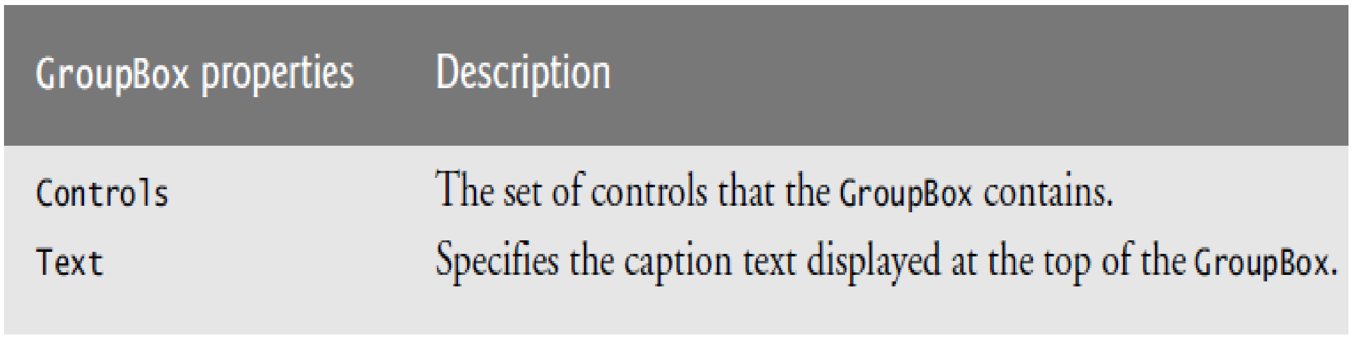
It can be used to show or hide a set of controls at once by setting its visibility properties

The primary difference between these two controls is that group boxes can display captions and do not include scroll bars whereas panels include scrollbars and panels do not include caption

Group boxes have thin border by default and panel can be set to have different borders style using their borderstyle property.

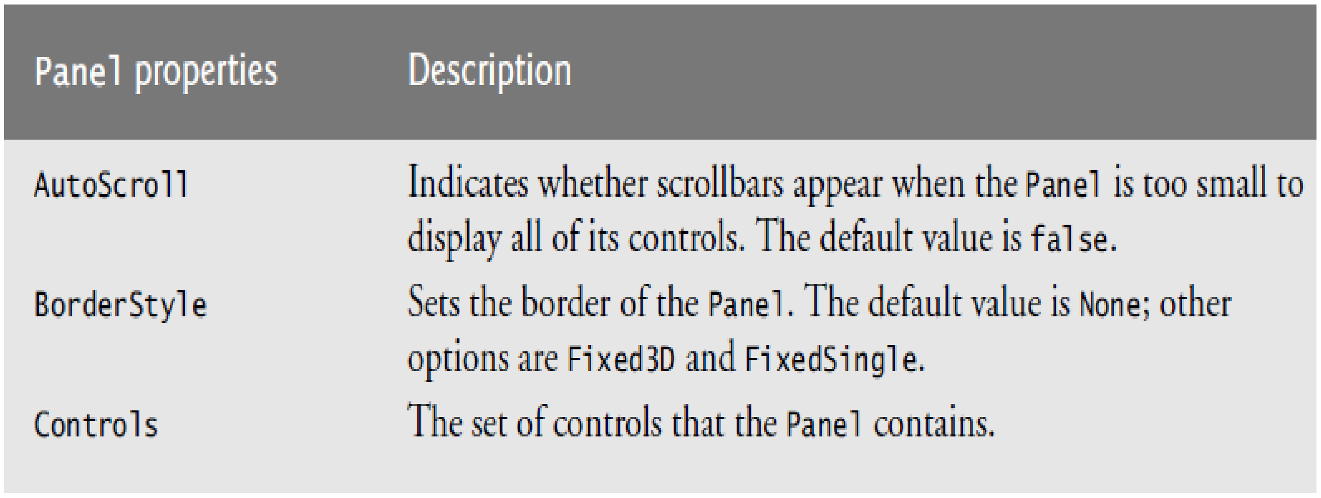
Panels and Group boxes can contain other panels and group boxes for more complex layouts.

GroupBox Properties

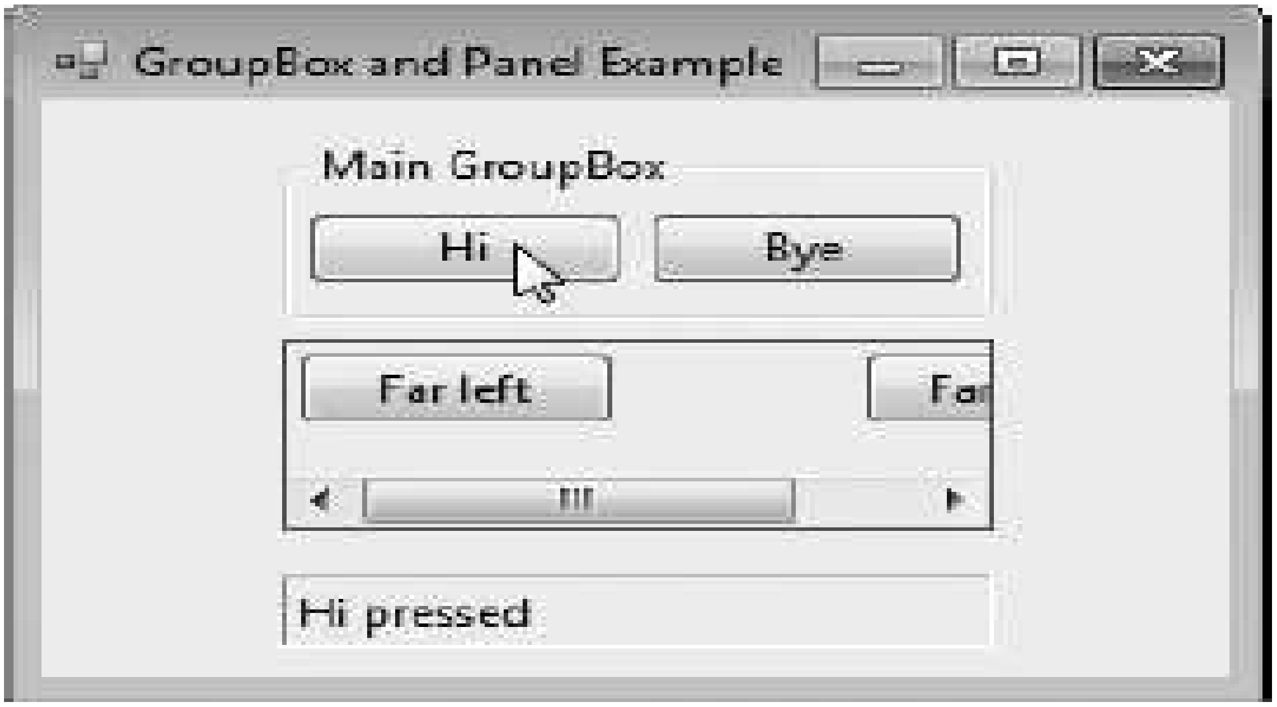


GroupBox properties	Description
`Controls`	The set of controls that the `GroupBox` contains.
`Text`	Specifies the caption text displayed at the top of the `GroupBox`.

Panel Properties



`Panel` properties	Description
`AutoScroll`	Indicates whether scrollbars appear when the `Panel` is too small to display all of its controls. The default value is `false`.
`BorderStyle`	Sets the border of the `Panel`. The default value is `None`; other options are `Fixed3D` and `FixedSingle`.
`Controls`	The set of controls that the `Panel` contains.



8. What are events? Write a C# console application to demonstrate the concept of multiple event handlers for a single event.

It may be necessary to use a single event handler for multiple events or have multiple events perform the same procedure. For example, it is often a powerful time-saver to have a menu command raise the same event as a button on your form does if they expose the same functionality. You can do this by using the Events view of the Properties window in C# or using the Handles keyword and the **Class Name** and **Method Name** drop-down boxes in the Visual Basic Code Editor.

To connect multiple events to a single event handler in Visual Basic

1. Right-click the form and choose **View Code**.
2. From the **Class Name** drop-down box, select one of the controls that you want to have the event handler handle.
3. From the **Method Name** drop-down box, select one of the events that you want the event handler to handle.
4. The Code Editor inserts the appropriate event handler and positions the insertion point within the method. In the example below, it is the Click event for the Button control.

VBCopy

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click
' Add event-handler code here.
End Sub
```

5. Append the other events you would like handled to the Handles clause.

VBCopy

```
Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button1.Click, Button2.Click
' Add event-handler code here.
End Sub
```

6. Add the appropriate code to the event handler.

USN	1	P	E			M	C	A		

PESIT Bangalore South Campus
Hosur Road, 1km before Electronic City, Bengaluru -560100
Department of Master of Computer Applications

Part – 5

9a) Explain the various mouse events in C# windows application.

b) Write a note on Tooltips.

Mouse – Event Handling

Mouse events such as - clicks and moves, which are generated when the user interacts with a control via the mouse.

Mouse events can be handled for any control that derives from class System.Windows.Forms.Control.

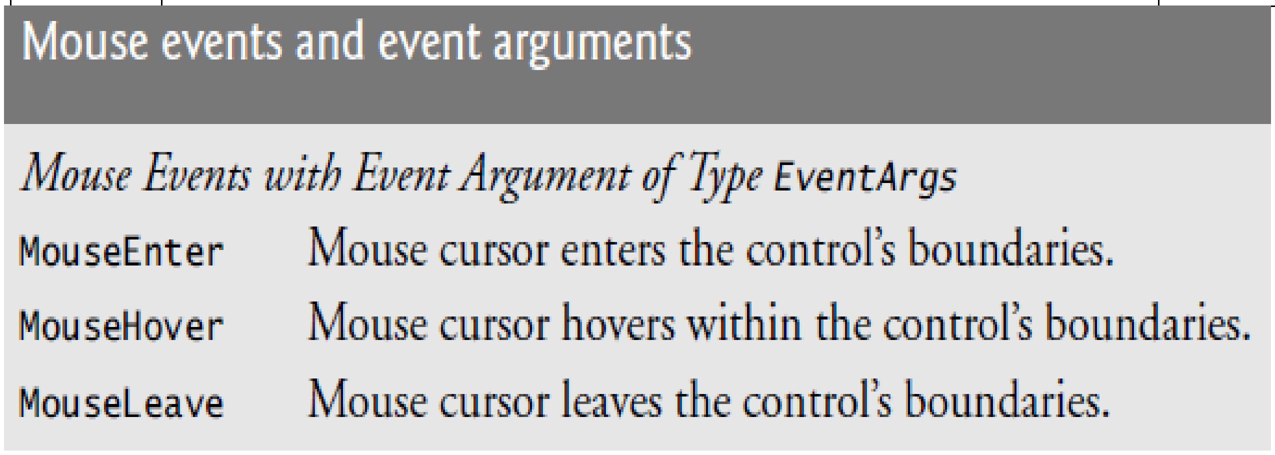
Information about the event is passed to the event handling method through an object of class MouseEventArgs, and the delegate used to create the mouse-event handler is MouseEventHandler.

Each mouse event handling method for these events requires an object and a mouseEventArgs object as argument.

Class MouseEventArgs contain information related to the mouse like :- mouse pointer's x- and y-coordinates, the mouse button pressed (left, right, center), and the number of times the mouse was clicked.

USN	1	P	E			M	C	A		

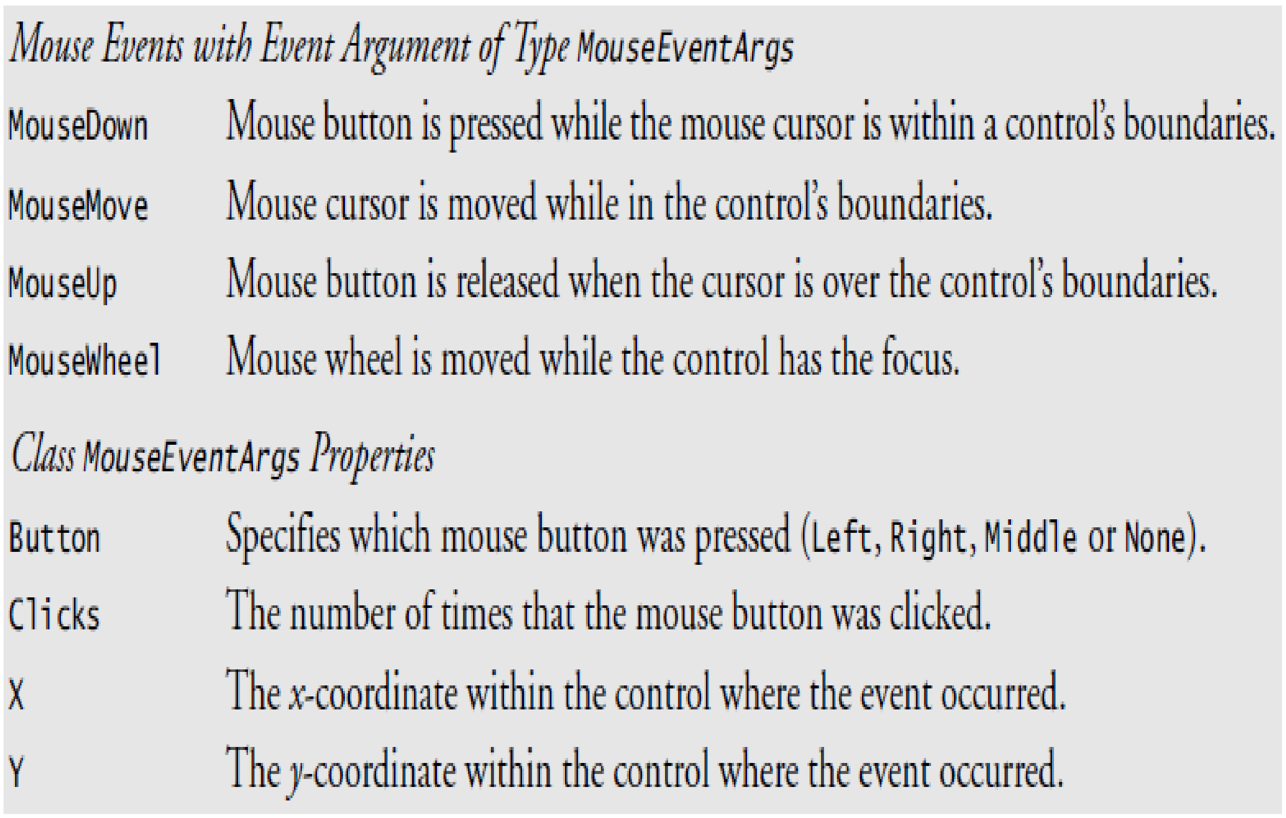
PESIT Bangalore South Campus
Hosur Road, 1km before Electronic City, Bengaluru -560100
Department of Master of Computer Applications



Mouse events and event arguments

Mouse Events with Event Argument of Type `EventArgs`

`MouseEnter`	Mouse cursor enters the control's boundaries.
`MouseHover`	Mouse cursor hovers within the control's boundaries.
`MouseLeave`	Mouse cursor leaves the control's boundaries.

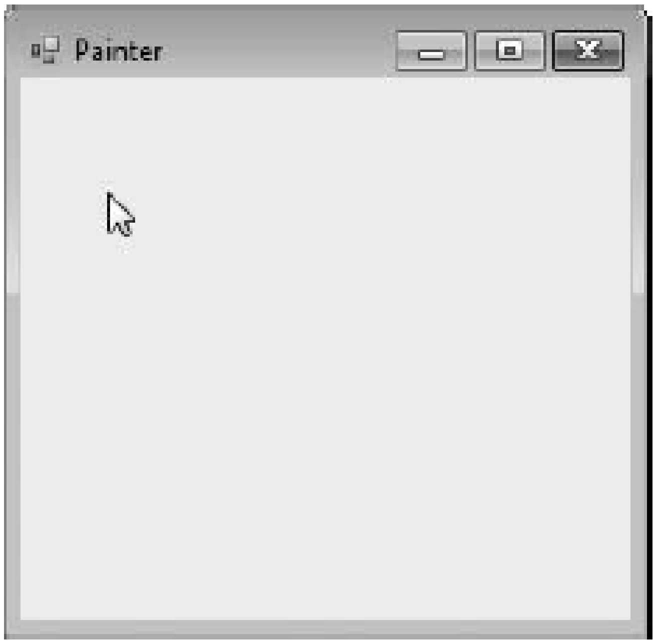


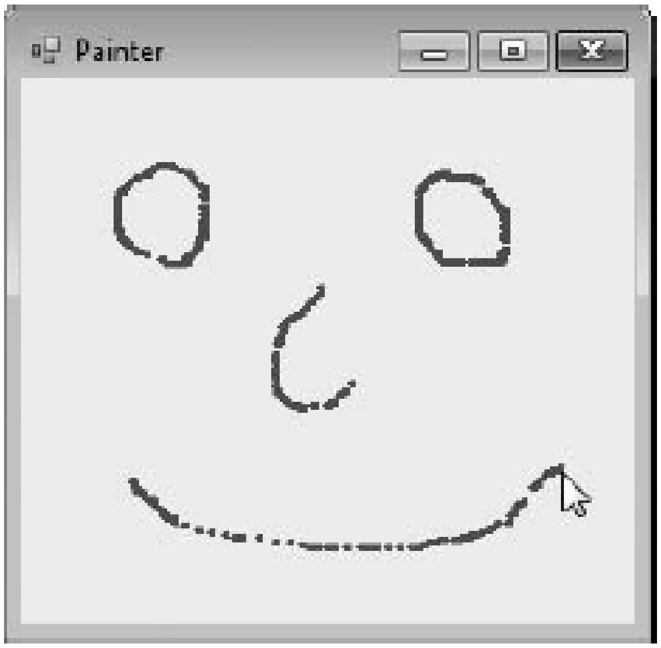
Mouse Events with Event Argument of Type `MouseEventArgs`

`MouseDown`	Mouse button is pressed while the mouse cursor is within a control's boundaries.
`MouseMove`	Mouse cursor is moved while in the control's boundaries.
`MouseUp`	Mouse button is released when the cursor is over the control's boundaries.
`MouseWheel`	Mouse wheel is moved while the control has the focus.

Class `MouseEventArgs` *Properties*

`Button`	Specifies which mouse button was pressed (`Left`, `Right`, `Middle` or `None`).
`Clicks`	The number of times that the mouse button was clicked.
`X`	The *x*-coordinate within the control where the event occurred.
`Y`	The *y*-coordinate within the control where the event occurred.


Painter


Painter

ToolTips

Tooltips are helpful text that appears when the mouse hovers over an item in a GUI. It serves as useful reminders for each icon's functionality.

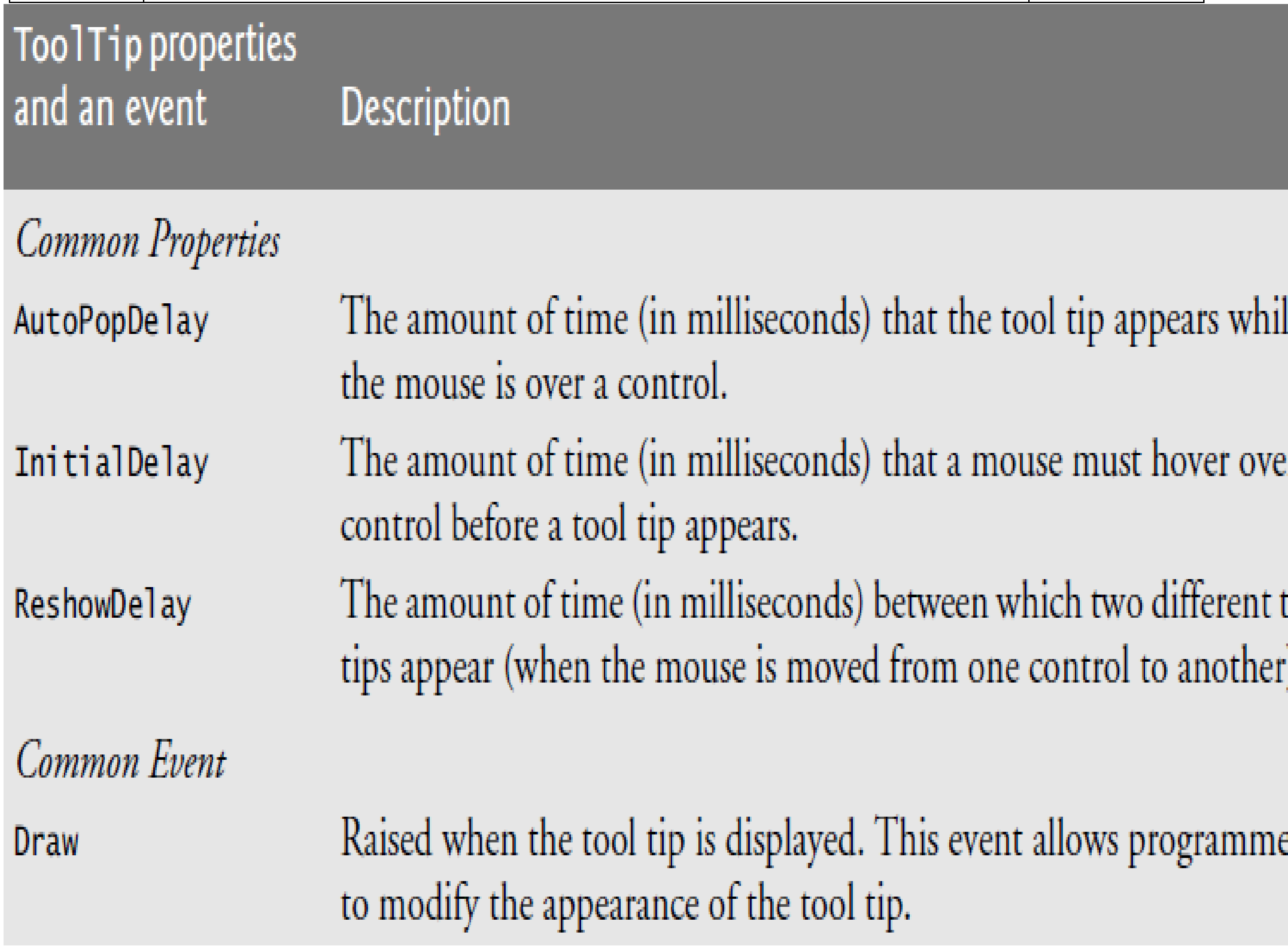
Many programs use tooltips to remind user of each control's purpose.

When we add tooltip component from the toolbox, it appears in the component tray - the region below the form in design mode.

Once a tooltip is added to a form, a new property appears in the properties window.

USN	1	P	E			M	C	A		

PESIT Bangalore South Campus
Hosur Road, 1km before Electronic City, Bengaluru -560100
Department of Master of Computer Applications

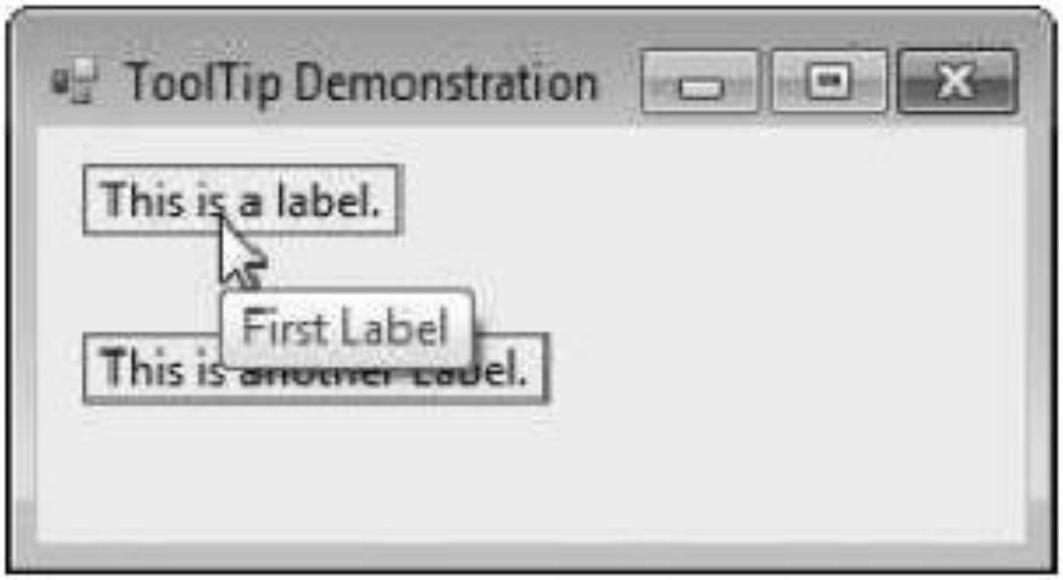


ToolTip properties and an event	Description
Common Properties	
AutoPopDelay	The amount of time (in milliseconds) that the tool tip appears whil the mouse is over a control.
InitialDelay	The amount of time (in milliseconds) that a mouse must hover ove control before a tool tip appears.
ReshowDelay	The amount of time (in milliseconds) between which two different t tips appear (when the mouse is moved from one control to another
Common Event	
Draw	Raised when the tool tip is displayed. This event allows programme to modify the appearance of the tool tip.

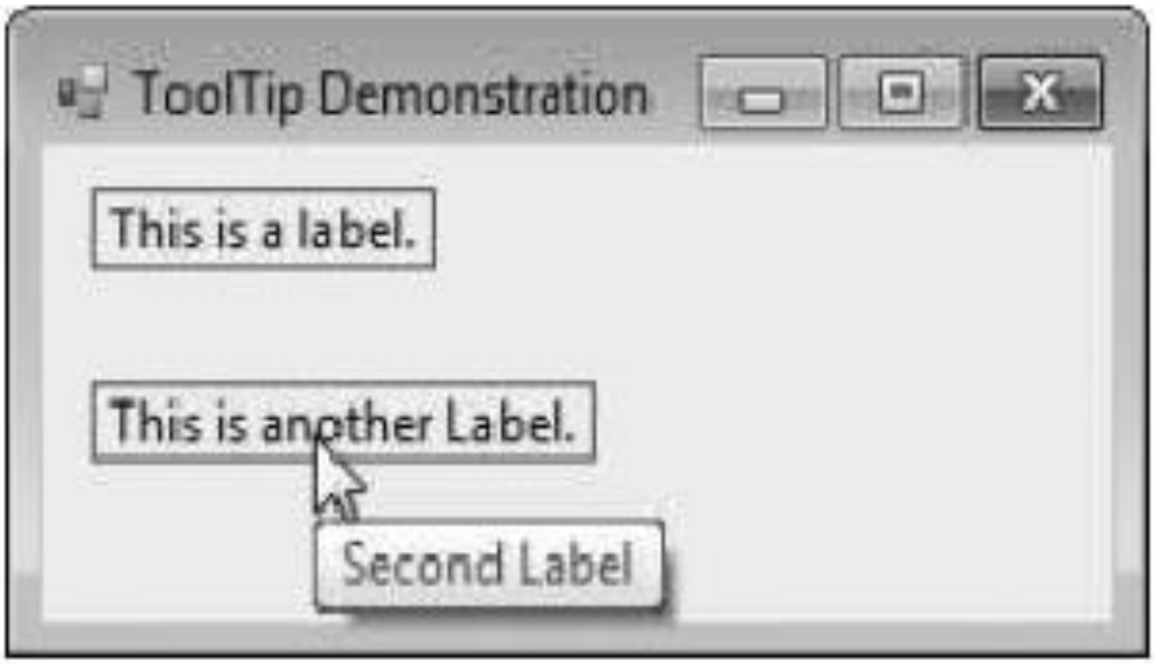
USN	1	P	E			M	C	A		

PES INSTITUTIONS	**PESIT Bangalore South Campus** Hosur Road, 1km before Electronic City, Bengaluru -560100 **Department of Master of Computer Applications**	

a)



b)



10. What are the features of interface? Write a C# program to demonstrate the use of interfaces.

Like a class, ***Interface*** can have methods, properties, events, and indexers as its members. But interfaces will contain only the declaration of the members. The implementation of interface's members will be given by class who implements the interface implicitly or explicitly.

Syntax for Interface Declaration:

```
interface  <interface_name >

{

   // declare Events

   // declare indexers

   // declare methods

   // declare properties

}
```

Syntax for Implementing Interface:

```
class class_name : interface_name
```

To declare an interface, use *interface* keyword. It is used to provide total abstraction. That means all the members in the interface are declared with the empty body and are public and abstract by default. A class that implement interface must implement all the methods declared in the interface.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Prog10
{
public interface IShape
{
void Cal();
void Disp();
}
class Circle:IShape
{
private double Area;
private double Radius;
public Circle()
{
this.Radius = 0;
}
public Circle(double s)
{
this.Radius = s;
}
public void Cal()
{
this.Area = 3.146 * this.Radius * this.Radius;
}
public void Disp()
{
Console.WriteLine("Area of Circle is:" + this.Area);
}
}
class Program
{
static void Main(string[] args)
{
IShape[] shape = { new Circle(30) };
for (int i = 0; i < shape.Length; i++)
{
shape[i].Cal();
shape[i].Disp();
}
Console.ReadKey();
}}
```

USN	1	P	E			M	C	A			

	PESIT Bangalore South Campus Hosur Road, 1km before Electronic City, Bengaluru -560100 **Department of Master of Computer Applications**	