CMR
INSTITUTE OF
TECHNOLOGY

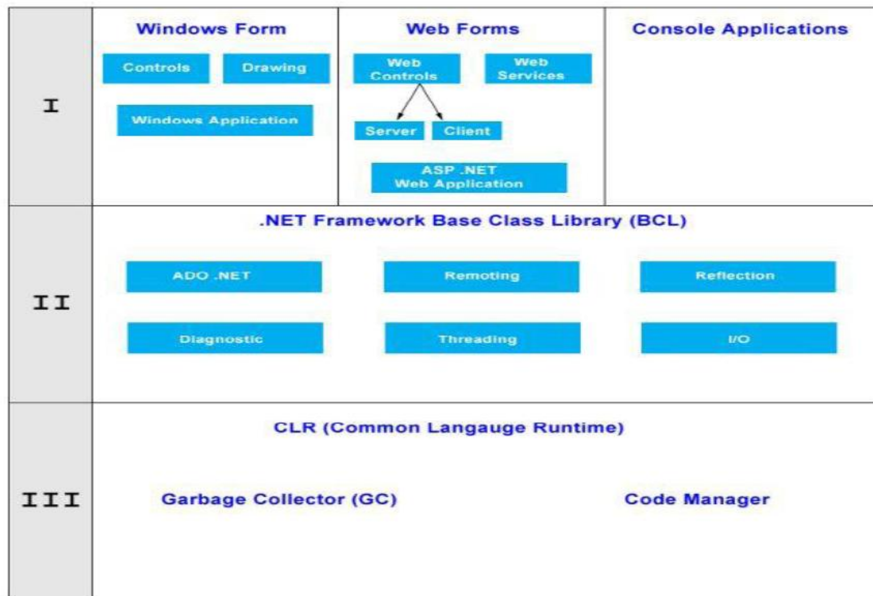
Improvement Test/ Internal Examination – III (Answer Key)

Sub: Programming using C#.NET Code: 13MCA53

Sem: V Branch:MCA

Answer Any **SIX FULL** Questions

1(a) **Illustrate the architecture and components of .NET framework with neat diagram.** [6]



CLR: Provides run time environment to run the code and provide various services to develop the application.
CTS: Specify certain guidelines for declaring using and managing types at runtime.
Base Class library: Reusable types. Classes, interfaces, value types helps in speeding-up application development process.
CLS: Common Language Specification.
Windows forms: is the graphical representation of any windows displayed in an application.
Web application: Uses ASP.NET to build application.
ADO.NET: Provides functionality for database communication.
Programming Languages: C#, VB, J#,VC++ and more are supported in .NET environment.

(b) **Summarize C# Preprocessor directives**

#if : When the C# compiler encounters an **#if** directive, followed eventually by an #endif directive, it will compile the code between the directives only if the specified symbol is defined.

#else create a compound conditional directive
#elif : create a compound conditional directive.

#endif: specifies the end of a conditional directive, which began with the #if directive.
#define: **#define** to define a symbol. Use the symbol as the expression that's passed to the #if directive, the expression will evaluate to **true.**
#undef undefine a symbol, such that, by using the symbol as the expression in a #if directive, the expression will evaluate to **false**.
#warning: generate a level one warning from a specific location in your code
#error: **#error** lets you generate an error from a specific location in your code. For example:
#line lets you modify the compiler's line number and (optionally) the file name output for errors and warnings.
#region: specify a block of code that can be expanded or collapse when using the outlining feature of the Visual Studio Code Editor.
#endregion marks the end of a #region block

2(a) **Explain with example the method of implementing encapsulation using class Properties.**

```
using system;
public class Department
{
private string departname;
public string Departname
{
get
{
return departname;
}
set
{
departname=value;
}
}
}
public class Departmentmain
{
public static int Main(string[] args)
{
Department d= new Department();
d.departname="Communication";
Console.WriteLine("The Department is :{0}",d.Departname);
return 0;
}
}
```

The property has two accessor get and set. The get accessor returns the value of the some property field. The set accessor sets the value of the some property field with the contents of "value". Properties can be made read-only. This is accomplished by having only a get accessor in the property implementation.

(b) **Classify between runtime and compile time polymorphism**

Static or Compile Time Polymorphism

In static polymorphism, the decision is made at compile time.
Which method is to be called is decided at compile-time only.
Method overloading is an example of this.
Compile time polymorphism is method overloading, where the compiler knows which overloaded method it is going to call.
Method overloading is a concept where a class can have more than one method with the same name and different parameters.
Compiler checks the type and number of parameters passed on to the method and decides which method to call at compile time and it will give an error if there are no methods that match the method signature of the method that is called at compile time.

Dynamic or Runtime Polymorphism

Run-time polymorphism is achieved by method overriding.
Method overriding allows us to have methods in the base and derived classes with the same name and the same parameters.
By runtime polymorphism, we can point to any derived class from the object of the base class at runtime that shows the ability of runtime binding.
Through the reference variable of a base class, the determination of the method to be called is based on the object being referred to by reference variable.
Compiler would not be aware whether the method is available for overriding the functionality or not. So compiler would not give any error at compile time. At runtime, it will be decided which method to call and if there is no method at runtime, it will give an error.

3(a) **What are delegates? Explain with code example, the concept of multicasting with delegates**

Delegate: A delegate is a special type of object that contains the details of a method rather than data.
In C# delegate is a class type object, which is used to invoke the method that has been encapsulated into it at the time of its creation. A delegate can be used to hold the reference to a method of any class.
Delegate contains 3 important piece of information

1. The name of the method on which it makes calls
2. Argument of this method
3. Return value of this method

Creating and using delegate:

1. Declaring a delegate
2. Defining delegate methods
3. Creating delegate objects
4. Invoking delegate objects

Declaring a delegate
Access-modifier delegate return-type delegate-name (parameter-list);
Public delegate void compute(int x, int y);
Defining Delegate Methods
Public static void Add(int a, int b)
{
Console.WriteLine("Sum={0}",a +b);
}

Creating Delegate Objects:
Delegate-name object-name=new delegate-name(expression);
Invoking Delegate object
Delegate-object(argument-list)
Cmp1(30,20);

A delegate object can hold reference of and invoke multiple methods.

```
using System;
delegate void CustomDel(string s);
class TestClass
{
    static void Hello(string s)
    {
        System.Console.WriteLine("  Hello, {0}!", s);
    }
    static void Goodbye(string s)
    {
        System.Console.WriteLine("  Goodbye, {0}!", s);
    }
    static void Main()
    {
        CustomDel hiDel, byeDel, multiDel, multiMinusHiDel;
        hiDel = Hello;
        byeDel = Goodbye;
        multiDel = hiDel + byeDel;
        multiMinusHiDel = multiDel - hiDel;
        Console.WriteLine("Invoking delegate hiDel:");
        hiDel("A");
        Console.WriteLine("Invoking delegate byeDel:");
        byeDel("B");
        Console.WriteLine("Invoking delegate multiDel:");
        multiDel("C");
        Console.WriteLine("Invoking delegate multiMinusHiDel:");
        multiMinusHiDel("D");
        Console.ReadLine();
    }
}
```

(b) **What are Events? Explain with code Event Source and Event Handling.**

Events:
An event is a delegate type class member that is used by an object or a class to provide notification to another objects that an event as occurred.
Event in itself is an action that is generated by a user or a computer.
In C#, we can use events to keep objects notified of the current state of another object or condition.
The event keyword can be used to declare an event.

Access-modifier event type event-name;
Class-object.event-name+=new class-object.delegate-name (method-name);

Event Sources:

Event always has two sides

1. Event Source
2. Event Handler

Event Source: pay attention to firing events and then detects the timing when an event should be fired.

Event Handler: deals with receiving information that an event has fired and verifies the information that the event is present.

An event source is an object that notifies other objects or tasks that something has happened. Event notification takes the form of callbacks.

Event source sends out the general message any object interested in the event can interpret it. Event handling is in the form of publisher subscriber.

```
using System;
public delegate void DivBySevenHandler(object o, DivBySevenEventArgs e);
public class DivBySevenEventArgs : EventArgs
{
  public readonly int TheNumber;
  public DivBySevenEventArgs(int num)
  {
    TheNumber = num;
  }

}
public class DivBySevenListener
{

  public void ShowOnScreen(object o, DivBySevenEventArgs e)
  {
    Console.WriteLine(
      "divisible by seven event raised!!! the guilty party is {0}",
      e.TheNumber);
  }
}

public class sample
{
  public static event DivBySevenHandler EventSeven;

  public static void Main()
  {
    DivBySevenListener dbsl = new DivBySevenListener();
    EventSeven += new DivBySevenHandler(dbsl.ShowOnScreen);
    GenNumbers();
    Console.ReadLine();
  }
```

```
public static void OnEventSeven(DivBySevenEventArgs e)
{
  if (EventSeven != null)
    EventSeven(new object(), e);
}

public static void GenNumbers()
{
  for (int i = 0; i < 99; i++)
  {
    if (i % 7 == 0)
    {
      DivBySevenEventArgs e1 = new DivBySevenEventArgs(i);
      OnEventSeven(e1);
    }
  }
}

}
```

4(a) Explain in detail multitier application architecture

Information Tier

The **information tier** (also called the **bottom tier**) maintains the application's data. This tier typically stores data in a relational database management system. For example, a retail store might have a database for storing product information, such as descriptions, prices and quantities in stock. The same database also might contain customer information, such as user names, billing addresses and credit card numbers. This tier can contain multiple databases, which together comprise the data needed for an application.

Business Logic

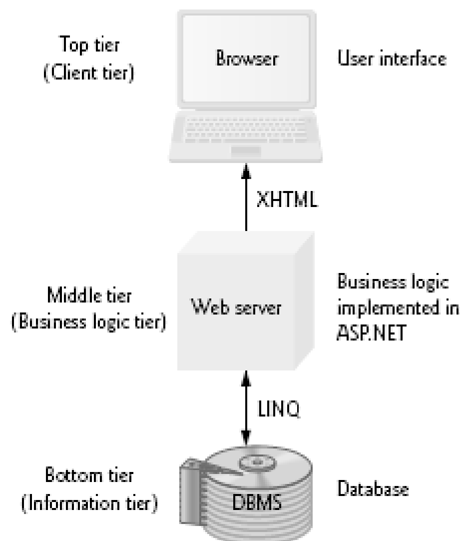
The **middle tier** implements **business logic, controller logic** and **presentation logic** to control interactions between the application's clients and its data. The middle tier acts as an intermediary between data in the information tier and the application's clients. The middle-tier controller logic processes client requests (such as requests to view a product catalog) and retrieves data from the database. The middle-tier presentation logic then processes data from the information tier and presents the content to the client. Web applications typically present data to clients as web pages.

Business logic in the middle tier enforces *business rules* and ensures that data is reliable before the server application updates the database or presents the data to users. Business rules dictate how clients can and cannot access application data, and how applications process data. For example, a business rule in the middle tier of a retail store's web-based application might ensure that all product quantities remain positive. A client request to set a negative quantity in the bottom tier's product information database would be rejected by the middle tier's business logic.

Client Tier

The **client tier**, or **top tier**, is the application's user interface, which gathers input and displays output. Users interact directly with the application through the user interface (typically viewed in a web browser), keyboard and mouse. In response to user actions (for example, clicking a hyperlink), the client tier interacts with the middle tier to make requests and to retrieve data from the information tier. The client tier then displays to the user the data retrieved from the middle tier. The client tier never directly interacts with the information tier.

Information Tier
The **information tier** (also called the **bottom tier**) maintains the application's data. This tier typically stores data in a relational database management system. For example, a retail store might have a database for storing product information, such as descriptions, prices and quantities in stock. The same database also might contain customer information, such as user names, billing addresses and credit card numbers. This tier can contain multiple databases, which together comprise the data needed for an application.



5(a) What are cookies? Explain session management using cookies.

Cookies: Cookie is a piece of data stored by web browsers in a small text file on the users computer. A cookie maintain information about the client during and between browsers session. The first time the user visits the website the users computer might receive a cookie from the server this cookie is then reactivated each time the user revisits the site.

Session tracking:

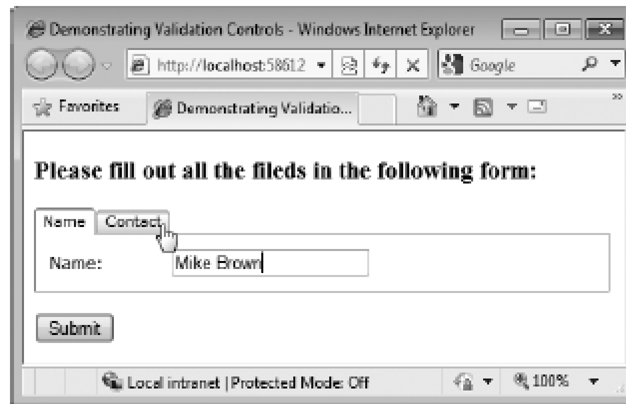
Personalization,
Privacy
Recogonizing clients

(b) How to test an ASP.NET Ajax applications

Testing the Application in Your Default Web Browser
To test this application in your default web browser, perform the following steps:

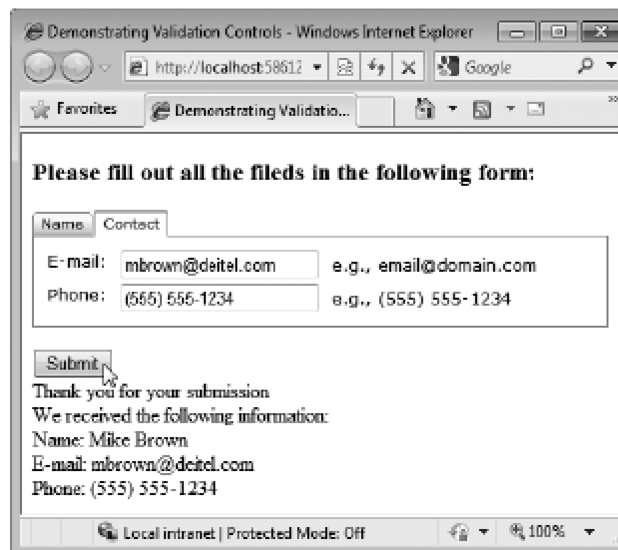
1. Select Open Web Site... from the Visual Web Developer File menu.
2. In the Open Web Site dialog, select File System, then navigate to this chapter's examples, select the `ValidationAjax` folder and click the Open `Button`.
3. Select `Validation.aspx` in the Solution Explorer, then type *Ctrl* + *F5* to execute the web application in your default web browser.

a) Entering a name on the Name tab then clicking the Contact tab



b) Entering an e-mail address in an incorrect format and pressing the *Tab* key to move to the next input field causes a callout to appear informing the user to enter an e-mail address in a valid format

c) After filling out the form properly and clicking the Submit button, the submitted data is displayed at the bottom of the page with a partial page update



6(a) Write a C# program to demonstrate Indexer Overload

```
using System;
namespace IndexerApplication
{
  class IndexedNames
  {
```

```
private string[] namelist = new string[size];
static public int size = 10;
public IndexedNames()
{
  for (int i = 0; i < size; i++)
  {
    namelist[i] = "N. A.";
  }
}

public string this[int index]
{
  get
  {
    string tmp;

    if( index >= 0 && index <= size-1 )
    {
      tmp = namelist[index];
    }
    else
    {
      tmp = "";
    }

    return ( tmp );
  }
  set
  {
    if( index >= 0 && index <= size-1 )
    {
      namelist[index] = value;
    }
  }
}
public int this[string name]
{
  get
  {
    int index = 0;
    while(index < size)
    {
      if (namelist[index] == name)
      {
       return index;
      }
      index++;
    }
    return index;
  }

}

static void Main(string[] args)
```

```
    {
      IndexedNames names = new IndexedNames();
      names[0] = "Zara";
      names[1] = "Riz";
      names[2] = "Nuha";
      names[3] = "Asif";
      names[4] = "Davinder";
      names[5] = "Sunil";
      names[6] = "Rubic";

      //using the first indexer with int parameter
      for (int i = 0; i < IndexedNames.size; i++)
      {
        Console.WriteLine(names[i]);
      }

      //using the second indexer with the string parameter
      Console.WriteLine(names["Nuha"]);
      Console.ReadKey();
    }
  }
}
```

(b) Explain the following:

i) **MDI Windows:**

MDI child forms are an essential element of Multiple-Document Interface (MDI) Applications, as these forms are the center of user interaction.

1. reate a new Windows Forms project. In the Properties Windows for the form, set its IsMdiContainer property to true, and its WindowsStateproperty to Maximized. This designates the form as an MDI container for child windows.
2. From the Toolbox, drag a MenuStrip control to the form. Set its Text property to File.
3. Click the ellipses (…) next to the Items property, and click Add to add two child tool strip menu items. Set the Text property for these items toNew and Window.
4. In Solution Explorer, right-click the project, point to Add, and then select Add New Item.
5. In the Add New Item dialog box, select Windows Form (in Visual Basic or in Visual C#) or Windows Forms Application (.NET) (in Visual C++) from the Templates pane. In the Name box, name the form Form2. Click the Open button to add the form to the project.

ii) **Event Driven GUI**

In Visual C#, we can use either the **Windows Form Designer** or the **Windows Presentation Foundation (WPF) Designer** to quickly and conveniently create user interfaces. For information to help you decide what type of application to build, see Overview of Windows-based Applications. There are three basic steps in creating user interfaces:

- Adding controls to the design surface.
- Setting initial properties for the controls.
- Writing handlers for specified events.

Programs with graphical user interfaces are primarily event-driven. They wait until a user

does something such as typing text into a text box, clicking a button, or changing a selection in a list box. When that occurs, the control, which is just an instance of a .NET Framework class, sends an event to your application. We have the option of handling an event by writing a special method in our application that will be called when the event is received.

7(a) List the difference between Properties and Indexers with example.

Property	Indexer
Allows methods to be called as if they were public data members.	Allows elements of an internal collection of an object to be accessed by using array notation on the object itself.
Accessed through a simple name.	Accessed through an index.
Can be a static or an instance member.	Must be an instance member.
A get accessor of a property has no parameters.	A **get** accessor of an indexer has the same formal parameter list as the indexer.
A set accessor of a property contains the implicit**value** parameter.	A **set** accessor of an indexer has the same formal parameter list as the indexer, and also to the value parameter.
Supports shortened syntax with Auto-Implemented Properties (C# Programming Guide).	Does not support shorte

(b) List and explain the Access specifiers in C#:

All types and type members have an accessibility level, which controls whether they can be used from other code in your assembly or other assemblies. We can use the following access modifiers to specify the accessibility of a type or member when you declare it:

public:The type or member can be accessed by any other code in the same assembly or another assembly that references it.

```
class SampleClass
{
     public int x; // No access restrictions.
}
```

private:The type or member can be accessed only by code in the same class or struct.

```
class Employee
{
 private int i;
 double d;   // private access by default
}
```

protected:The type or member can be accessed only by code in the same class or struct, or

in a class that is derived from that class.

```
class A
{
   protected int x = 123;
}

class B : A
{
 static void Main()
 {
   A a = new A();
   B b = new B();

   b.x = 10;
 }
}
```

internal:The type or member can be accessed by any code in the same assembly, but not from another assembly.

```
public class BaseClass
{
  // Only accessible within the same assembly
 internal static int x = 0;
}
```

8(a) Explain static construct or with sample code.

Static Constructor:

- A static constructor does not take access modifiers or have parameters.
- A static constructor is called automatically to initialize the class before the first instance is created or any static members are referenced.
- A static constructor cannot be called directly.
- The user has no control on when the static constructor is executed in the program.
- A typical use of static constructors is when the class is using a log file and the constructor is used to write entries to this file.
- Static constructors are also useful when creating wrapper classes for unmanaged code, when the constructor can call the LoadLibrary method.
- If a static constructor throws an exception, the runtime will not invoke it a second time, and the type will remain uninitialized for the lifetime of the application domain in which your program is running.

```
class SimpleClass
{
  // Static variable that must be initialized at run time.
  static readonly long baseline;

  // Static constructor is called at most one time, before any
  // instance constructor is invoked or member is accessed.
```

```
    static SimpleClass()
    {
      baseline = DateTime.Now.Ticks;
    }
}
```

```
System Namespace:
```

The System namespace contains fundamental classes and base classes that define commonly-used value and reference data types, events and event handlers, interfaces, attributes, and processing exceptions.

```
Namespace namespacename
{
//code declaration
}
```

(b) Explain the steps involved in creating and using delegate in C# program

Steps involved in creating and using delegate:

1. Declaring a delegate
2. Defining delegate methods
3. Creating delegate objects
4. Invoking delegating objects.

```
using System;
public delegate double Conversion(double from);
class DelegateDemo
{
    public static double FeetToInches(double feet)
    {
        return feet * 12;
    }

          static void Main()
          {

              Conversion doConversion = new Conversion(FeetToInches);
              Console.Write("Enter Feet:  ");
              double feet = Double.Parse(Console.ReadLine());
              double inches = doConversion(feet);
              Console.WriteLine("\n{0} Feet = {1} Inches.\n", feet, inches);
              Console.ReadLine();
          }
      }
```