CMR
INSTITUTE OF
TECHNOLOGY

USN	1	C								

Internal Assessment Test 2 – November 2020

Sub:	**Programming using C#.NET**							**Code:**	18MCA51
Date:	02-11-20	Duration:	90 mins	Max Marks:	50	**Sem:**	V A & B	**Branch:**	**MCA**

Note: Answer any 5 questions. All questions carry equal marks. Total marks: 50

			Marks	**OBE**	
				CO	**RBT**
1.	a	How delegates are used in C#? Discuss single cast and multicast delegates with an example.	**10**	CO2	L3

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.
A Delegate can be defined as a delegate type. Its definition must be similar to the function signature. A delegate can be defined in a namespace and within a class.
A delegate cannot be used as a data member of a class or local variable within a method.
Delegate declarations look almost exactly like abstract method declarations, you just replace the abstract keyword with the delegate keyword.
Delegates are especially used for implementing events and the call-back methods. All delegates are implicitly derived from the System. Delegate class.

In C#, delegate is a reference to the method. It works like function pointer in C and C++. But it is objected-oriented, secured and type-safe than function pointer.

For static method, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.

The best use of delegate is to use as event.

Internally a delegate declaration defines a class which is the derived class of System.Delegate.

Singlecast Delegate

This is a kind of delegate that can refer to single method at one time. SingleCast Delegates refer to a single method with matching signature. SingleCast Delegates derive from the System.Delegate class.

Single cast delegate program

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

namespace delegatefunction
```

```
{
   // Delegate definition
   public delegate int delefunc(int x, int y);

   class Program
   {
      static int add(int a, int b)
      {
         return a + b;
      }
      public static void Main(string[] s)
      {
         // instantiate the delegate
         // delegatename obj = new delegatename(classname.methodname)
         delefunc d1 = new delefunc(Program.add);
         // pass the values and print output

         Console.WriteLine("Addition of numbers = {0}", d1(20, 30));
         Console.ReadKey();
      }
   }
}
```

Multicast Delegate

A delegate that holds a reference to more than one method is called multicasting delegate. A Multicast Delegate is a delegate that holds the references of more than one function. When we invoke the multicast delegate, then all the functions which are referenced by the delegate are going to be invoked. If you want to call multiple methods using a delegate then all the method signature should be the same.

Multicast Delegate Program 1

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

namespace multicastedelexamole
{
   // declaring the delegate
   public delegate void MyDel(int num1, int num2);
   class Sample
   {
      // Method Add is the first method called by the delegate MyDel
  static void Add(int num1, int num2)
      {
         Console.WriteLine("\tAddition: " + (num1 + num2));
      }
           // Method Sub is the second method called by the delegate MyDel
      static void Sub(int num1, int num2)
      {
         Console.WriteLine("\tSubtraction: " + (num1 - num2));
      }
           // Method Mul is the third method called by the delegate MyDel
```

```
        static void Mul(int num1, int num2)
        {
            Console.WriteLine("\tMultiplication: " + (num1 * num2));
        }

        static void Main()
        {
            int num1 = 0;
            int num2 = 0;
                    // instantiating the delegate with first method as parameter
            MyDel del = new MyDel(Add);
                    // input the values to be passed as arguments
            Console.Write("Enter the value of num1: ");
            num1 = int.Parse(Console.ReadLine());

            Console.Write("Enter the value of num2: ");
            num2 = int.Parse(Console.ReadLine());
                    // second method is appended to the delegate object
            del += new MyDel(Sub);
                    // third method is appended to the delegate object
            del += new MyDel(Mul);

            Console.WriteLine("Call 1:");
// the methods will be executed one after the other and output will be displayed.
            del(num1, num2);

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

2.	a	"Catching on exceptions programmatically is good and necessary mechanism" justify with suitable examples.	**10**	CO2	L3

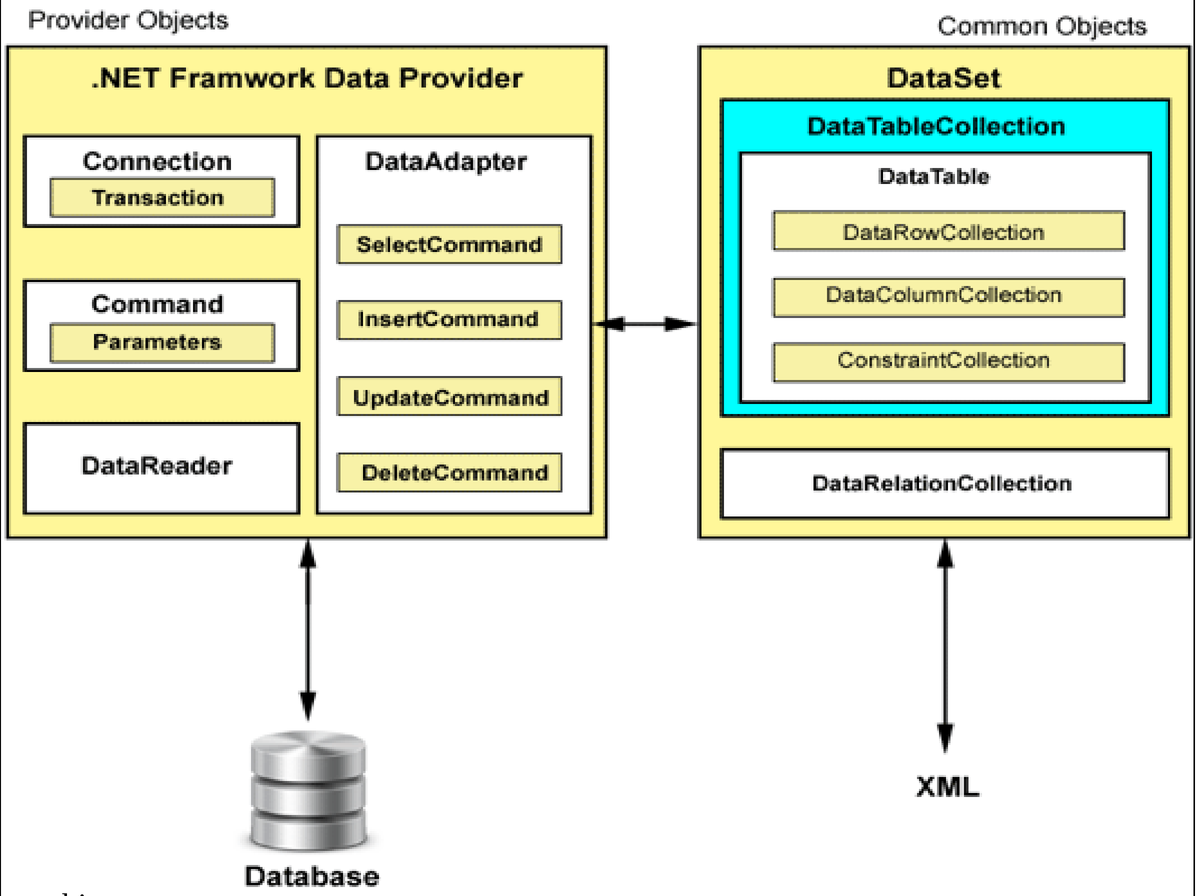
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace GMT_TCF
{
class Program {
static void Main(string[] args)
{
Console.WriteLine(" Enter the dividend");
int m= Convert.ToInt32(Console.ReadLine());
Console.WriteLine(" Enter the divisor");
int n = Convert.ToInt32(Console.ReadLine());
try { int k = m / n;
Console.WriteLine("Output is:" + k.ToString());
 }
catch (DivideByZeroException e) {
Console.WriteLine("Exception Caught:" + e.Message);
} finally {
Console.ReadLine();
 } }
```

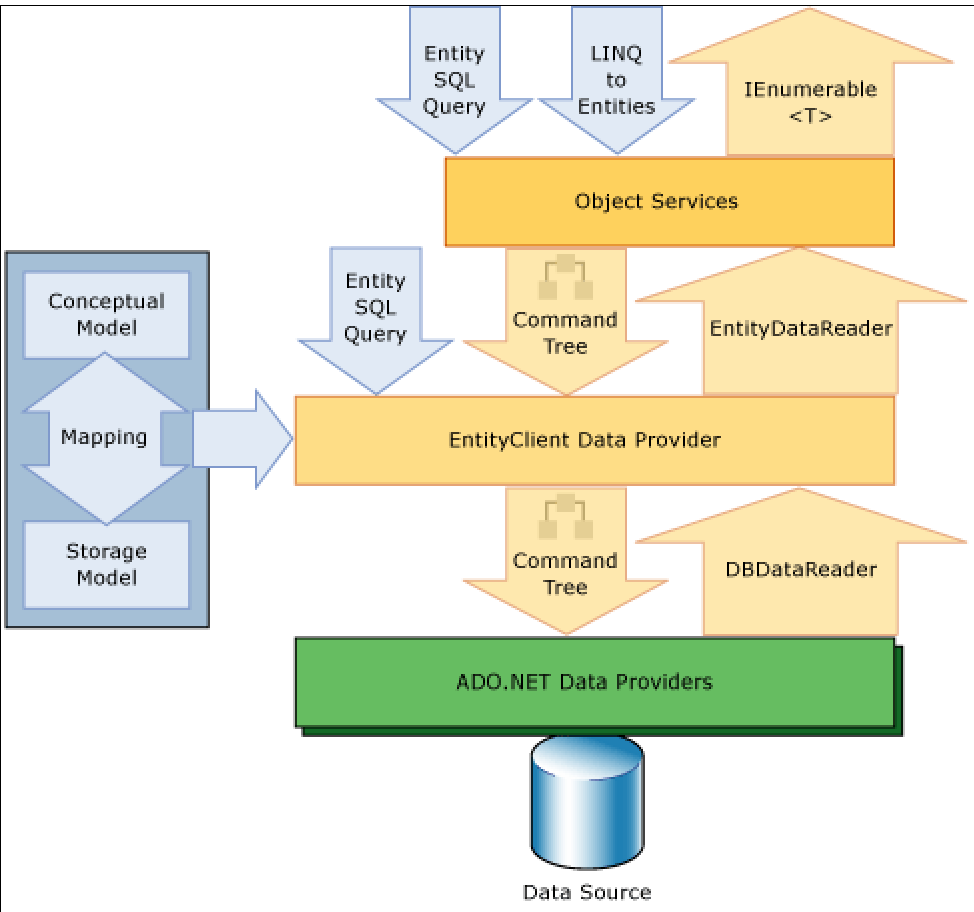
3.	a	Write a C# Program to Illustrate Exception Handling for Invalid Typecasting in UnBoxing.	**10**	CO2	L4

```
class TestUnboxing
{
    static void Main()
    {
        int num = 123;
        object obj = num;
        try
        {
            int j = (short)obj;
            System.Console.WriteLine("Unboxing");
        }
        catch (System.InvalidCastException e)
        {
            System.Console.WriteLine("{0} Error: Incorrect unboxing", e.Message);
        }
        System.Console.Read();
    }
}
```

4	a	Explain the implementation of ComboBox in C# by populating the ComboBox with values at runtime.	**10**	CO3	L4

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }
        private void Form1_Load(object sender, EventArgs e)
        {
            comboBox1.Items.Add("weekdays");
            comboBox1.Items.Add("year");
        }
        private void comboBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            comboBox2.Items.Clear();
            if (comboBox1.SelectedItem == "weekdays")
            {
                comboBox2.Items.Add("Sunday");
                comboBox2.Items.Add("Monday");
                comboBox2.Items.Add("Tuesday");
            }
            else if (comboBox1.SelectedItem == "year")
            {
                comboBox2.Items.Add("2012");
                comboBox2.Items.Add("2013");
                comboBox2.Items.Add("2014");
            }
        }
    }
```

		}			
5	a	Explain the components of ADO.NET entity framework. ADO.NET stands for ActiveX Data Objects and is part of the .NET framework technology that allows to access and modify data from different data sources. It supports many types of data sources such as Microsoft SQL Server, MySQL, Oracle, and Microsoft Access. The .NET Framework provides a number of data providers that you can use. These data providers are used to connect to a data source, executes commands, and retrieve results. Various Connection Architectures: There are the following two types of connection  Provider Objects Common Objects .NET Framwork Data Provider Connection Transaction Command Parameters DataReader DataAdapter SelectCommand InsertCommand UpdateCommand DeleteCommand DataSet DataTableCollection DataTable DataRowCollection DataColumnCollection ConstraintCollection DataRelationCollection Database XML architectures: • Connected architecture: the application remains connected with the database throughout the processing. • Disconnected architecture: the application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a DataSet. The following diagram illustrates the Entity Framework architecture for accessing data:	**10**	CO3	L2

		 Entity SQL Query LINQ to Entities IEnumerable <T> Object Services Conceptual Model Mapping Storage Model Entity SQL Query Command Tree EntityDataReader EntityClient Data Provider Command Tree DBDataReader ADO.NET Data Providers Data Source			
6	a	How DataAdapter is used to build database applications. Write a program to demonstrate DataAdapter using DataSet and DataTable. A data adapter aobject serves as a bridge between a data set object and Data Source such as a database to retrieve and save the data. Data adapter contains a set of database commands and a database connection, which we use to fill a dataset object and update the Data Source. .NET makes two primary data adapters available for use with the databases. Other data adapters can also be integrated with Visual Studio .NET. Primary Data Adapters are mentioned below. • OleDbData Adapter, which is suitable for use with certain OLE DB providers. • SQLDataAdapaters, which is specific to a Microsoft SQL server. This is faster than the OleDBDataAdapter. because it works directly with SQL servers and does not go through an OLE Db Layer. **Data adapter properties** We use data adapter objects to act on records from a Data Source. We can also specify, which action we want to perform by using one of following four data adapter properties, which executes a SQL statement. The properties are given below. • Select command retrieves rows from Data Source • Insert command writes inserted rows from data set into Data Source • Update command writes modified rows from data set into Data Source.	**10**	**CO3**	**L4**

- Delete command deletes rows from Data Source.

Methods used by a data adapter

Actually, we use data adapters to fill or to make changes in a data set table to a data store. These methods comprises of following.

- *Fill*
  Use this method of a SQL data adapter to add or refresh row from a Data Source and place them in a Data Set table. The fill method uses Select statement, which is specified in the Select command property

- *Update*
  Use this method of data adapter object to transmit the changes to a dataset table to the corresponding Data Source. This method calls the corresponding insert, delete or update command for each specified row in a data table in a data set.

- *Close*
  Use this method for the connection to a database.

- *Creating Data Adapter with Example*
  The examples given below use a SQLDataAdapter object to define a query in the database.

```
// Assumes that customerConnection is a valid SqlConnection object.

// Assumes that orderConnection is a valid OleDbConnection object.

SqlDataAdapter custAdapter = new SqlDataAdapter(

  "SELECT * FROM dbo.Customers", customerConnection);

OleDbDataAdapter ordAdapter = new OleDbDataAdapter(

  "SELECT * FROM Orders", orderConnection);



DataSet customerOrders = new DataSet();



custAdapter.Fill(customerOrders, "Customers");

ordAdapter.Fill(customerOrders, "Orders");



DataRelation relation = customerOrders.Relations.Add("CustOrders",

  customerOrders.Tables["Customers"].Columns["CustomerID"],

  customerOrders.Tables["Orders"].Columns["CustomerID"]);
```

		foreach (DataRow pRow in customerOrders.Tables["Customers"].Rows)
		
		{
		
		Console.WriteLine(pRow["CustomerID"]);
		
		foreach (DataRow cRow in pRow.GetChildRows(relation))
		
		Console.WriteLine("\t" + cRow["OrderID"]);
		
		}			

7	a	Discuss the components of DataSet and state the difference between DataSet and DataTable.
		A dataset is a structured collection of data generally associated with a unique body of work.The dataset and all of its components. The dataset consists of three main parts: (1) Metadata; (2) UI events; (3) Network traces. Metadata includes the network type (cellular or WiFi), information of GPS, network names and signal strength. UI data is a collection information on the touch screen which includes tapping locations and times. Network traces is also the whole information about the network, including server and client IP addresses, port numbers, packet counts, etc.
		
		A DataTable is an in-memory representation of a single database table which has collection of rows and columns whereas a DataSet is an in-memory representation of a database-like structure which has collection of DataTables. Whenever you want to fetch data from database, it connects indirectly to the database and create a virtual database in local system and then disconnected from database.
		
		DataTable object is lighter than DataSet object since it contains data from single table whereas DataSet is heavier object that can contain data from multiple tables.
		
		DataTable example
		SqlDataAdapter adp = new SqlDataAdapter("select * from SampleTable", con);
		DataTable dt = new DataTable();
		adp.Fill(dt);
		GridView1.DataSource = dt;
		GridView1.DataBind();
		
		DataSet Example:
		SqlDataAdapter adp= new SqlDataAdapter("select * from SampleTable", con);
		DataSet ds = new DataSet();
		adp.Fill(ds);
		GridView1.DataSource = ds.Tables[0];
		GridView1.DataBind();
			10	**CO3**	**L3**

8	a	Explain different validation control with suitable examples supported by ASP.NET by demonstrating through WebForms.	10	CO3	L3

Validation controls are used to,

- Implement presentation logic.
- To validate user input data.
- Data format, data type and data range is used for validation.

Validation is of two types

1. Client Side
2. Serve Side

Client side validation is good but we have to be dependent on browser and scripting language support.

Client side validation is considered convenient for users as they get instant feedback. The main advantage is that it prevents a page from being postback to the server until the client validation is executed successfully.

For developer point of view serve side is preferable because it will not fail, it is not dependent on browser and scripting language.

You can use ASP.NET validation, which will ensure client, and server validation. It work on both end; first it will work on client validation and than on server validation. At any cost server validation will work always whether client validation is executed or not. So you have a safety of validation check.

For client script .NET used JavaScript. WebUIValidation.js file is used for client validation by .NET

Validation Controls in ASP.NET

An important aspect of creating ASP.NET Web pages for user input is to be able to check that the information users enter is valid. ASP.NET provides a set of validation controls that provide an easy-to-use but powerful way to check for errors and, if necessary, display messages to the user.

There are six types of validation controls in ASP.NET

1. RequiredFieldValidation Control
2. CompareValidator Control
3. RangeValidator Control
4. RegularExpressionValidator Control
5. CustomValidator Control
6. ValidationSummary

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
Style="top: 98px; left: 367px; position: absolute; height: 26px; width: 162px"
ErrorMessage="password required" ControlToValidate="TextBox2">
</asp:RequiredFieldValidator>
```

CompareValidator Control

The CompareValidator control allows you to make comparison to compare data entered in an input control with a constant value or a value in a different control.