| Sub: | Programming Using C# and .Net | | | | | | |
|---|---|---|---|---|---|---|---|
| Date: | 12/10/2019 | Duration: | 90 min's | Max Marks: | 50 | Sem | 5th |

**Note : Answer FIVE FULL Questions, choosing ONE full question from each Module**

1 What is namespace? Explain the steps involved in creating a namespace and illustrate few common namespaces

A namespaces is wrapper that is wrapped around one or more structural elements to make them unique and differentiated from other elements.

To declare namespace in C# .Net has a reserved keyword **"namespace"**. If a new project is created in Visual Studio .NET it automatically adds some global namespaces. These namespaces can be different in different projects. But each of them should be placed under the base namespace **"System"**. In C# all namespaces should import by ***using*** keyword, which can tell the compiler which namespaces and libraries of the code you want to use in the system.

## The System namespace:

Within **System** we can find numerous useful types dealing with built in data, mathematical computations, random number generation, environment variables, and garbage collection, as well as a number of commonly used exceptions and attributes. So **System** is a root namespace.

The following are some of the common namespaces provided by the .NET Framework class library:

| Namespaces | Meaning |
|---|---|
| System.Collections<br>System.Collections.Generic | These namespaces define a number of stock container objects (ArrayList, Queue, and so forth), as well as base types and interfaces that allow you to build customized collections. As of .NET 2.0, the collection types have been extended with generic capabilities |
| System.Data System.Data.Odbc<br>System.Data.OracleClient<br>System.Data.OleDb<br>System.Data.SqlClient | These namespaces are used for interacting with databases using ADO.NET. |
| System.Diagnostics | Here, you find numerous types that can be used to programmatically debug and trace your source code. |
| System.Drawing<br>System.Drawing.Drawing2D<br>System.Drawing.Printing | Here, you findnumerous types wrapping graphical primitives such as bitmaps, fonts, and icons, as well as printing capabilities. |
| System.IO<br>System.IO.Compression<br>System.IO.Ports | Include file I/O, buffering, and so forth. As of .NET 2.0, the IO namespaces now include support compression and port manipulation. |
| System.Net | Contains types related to network programming (requests/responses, sockets, end points, and so on). |
| System.Reflection | Define types that support runtime type discovery as well as dynamic |

| | |
|---|---|
| System.Reflection.Emit | creation of types. |
| System.Runtime.InteropServices | Provides facilities to allow .NET types to interact with "unmanaged code" (e.g., C-based DLLs and COM servers) and vice versa. |
| System.Runtime.Remoting | Defines types used to build solutions that incorporate the .NET remoting layer. |
| System.Security | Security is an integrated aspect of the .NET universe. In the security-centric namespaces you find numerous types dealing with permissions, cryptography, and so on. |
| System.Threading | This namespace defines types used to build multithreaded applications. |
| System.Web<br>System.Web.Security | A number of namespaces are specifically geared toward the development of .NET web applications, including ASP.NET and XML web services. |

2W Write a C# program to demonstrate of Abstract classes and abstract methods.

## Abstract Classes:

Classes can be declared as abstract by putting the keyword **"abstract"** before the class definitions.

The **main purpose** of the ***Abstract classes*** is to make classes that only represent base classes, and don't want anyone to create objects of these class types. An abstract class cannot be instantiated because cannot create an object of the class.

```
public abstract class Shape{
     //Class Definition
}
```

```
Shape obj=new Shape();
//Can't be instantiated
```

An abstract class can contain either *abstract methods* or *non-abstract methods*. Abstract members do not have any implementation in the abstract class, but the same has to be provided in its derived class.

Example:

```
public abstract class
   Shape{ public abstract void
   Draw();
   public void NonAbstractMethod()
     { Console.WriteLine("NonAbstract
     Method");
```

## Abstract methods:

Syntax:

```
public abstract void Draw();
```

Abstract methods have no implementation, so the method definitions is followed by a semicolon instead of a normal method block. Derived classes of the abstract class must implement all abstract methods.

- Restricts its implementation in an abstract derived class
- Allows implementation in a non-abstract derived class

- Requires declaration in an abstract class only
- Restrict declaration with `static` and `virtual` keywords
- Allows you to override a virtual method.

```
using System;
namespace Chapter4_Examples{
     abstract class
        absClass{
    //A Non abstract method
    public int AddTwoNumbers(int Num1, int
        Num2){ return Num1 + Num2;
    }
    //An abstract method to be overridden in derived class
    public abstract int MultiplyTwoNumbers(int Num1,int Num2);
  }

  //A Child Class of absClass
  class absDerived:absClass{
     //using override keyword,implementing the abstract method MultiplyTwoNumbers
     public override int MultiplyTwoNumbers(int Num1,int
         Num2){ return Num1 * Num2;
     }
  }
  class AbstractDemo{
    static void Main(string[] args)
      { absDerived calculate = new
      absDerived();
      int added = calculate.AddTwoNumbers(10,20);
      int multiplied = calculate.MultiplyTwoNumbers(10,20);
      Console.WriteLine("Added : {0}, Multiplied : {1}", added, multiplied);
      Console.ReadLine();
    }
  }
}
```

```
Added : 30, Multiplied : 200
```

3 Write a C# program to demonstrate Indexer Overload.

An indexer is used to treat an object as an array. It is used to provide index to an object to obtain values from the object.

- Implementing an indexer requires you to use brackets ([ ]) with an object to get and set a value of the object.
- Indexer are declared as properties, with the difference that in case of indexers, you do not need to provide name to them. You need to use the ***"this"*** keyword to define an indexer.

```
using System;
namespace Class_Demos{
 class
  MyClass{ string[
  ] mydata; int
  arrsize;
  public MyClass(int size){
    arrsize = size;
    mydata = new  string[size];
    for (int i = 0; i < size; i++)
       mydata[i] = "DataValue";
    }
```

```
public string this[int position]
  { set{ mydata[position] = value;
  } get{return (mydata[position]);
  }
}
public string this[string data]{
  set{
   for (int i=0; i<arrsize; i++){
     if (mydata[i] == data)
          mydata[i] = value;
  }
}
```

**Continued...**

```
    get{
      int count = 0;
      for (int i = 0; i < arrsize; i++)
         { if (mydata[i] == data)
            count = count + 1;
      }
      return count.ToString();
    }
  }
 }
 class
   IndOverload{ static
   void Main(){
     int size = 10;
     MyClass obj=new MyClass(size);

     obj[1] =  "Hello"; obj[3]
     = "Good Morning"; obj[7]
     = "Welcome";
     obj["DataValue"] = "Have a nice day";

     for (int i = 0; i < size; i++)
       Console.WriteLine("obj[{0}]: {1}", i, obj[i]);

     Console.WriteLine("\n\nNumber of \"Have a nice day\"entries:{0}",
                                          obj["Have a nice day"]);

     Console.Read();
   }
 }
}
```

4 Write a C# program to explain interface inheritance and implementation of interfaces.

## Interfaces:

***"An interface is a collection of data members and member functions, but it does not implement them".***

Interface are introduced to provide the feature of **multiple inheritance** to classes.

Multiple inheritance is the feature of OOP which allows a class to inherit from multiple classes. The methods defined in an interface do not have implementation, they only specify the parameters that they will take and the types of values they will return.

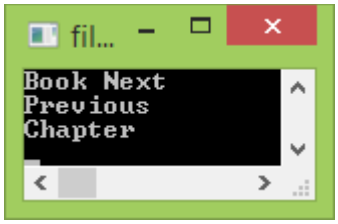
```
interface <Interface_Name> {
        //Abstract method declaration in interface body
}
```

```
public interface Channel
{
    void Next();
}
```

Example:

```
using System;
namespace Class_Demos{
  public interface Channel{
      void Next();
      void Previous();
  }
  public interface Book{
      void Next();
      void Chapter();
  }
```

```
class InterfaceDemo:Channel, Book{
  void Channel.Next(){
     Console.WriteLine("Channel Next");
  }
  void Book.Next()
     { Console.WriteLine("Book
     Next");
  }
  public void Previous()
     { Console.WriteLine("Previous"
     );
  }
  public void Chapter()
     { Console.WriteLine("Chapter"
     );
  }
  static void Main(){
   InterfaceDemo ind = new InterfaceDemo();
   ((Book)ind).Next();//invoking Book method
   ind.Previous();
   ind.Chapter();
   Console.Read();
  }
 }
}
```




### 3.1 Implementation of Interfaces and Inheritance:

When an interface is implemented by a base class, then the derived class of the base class automatically inherits method of the interface. You can initialize an object of the interface by type casting the object of the derived class with the interface itself.

Example We have created two interface `BaseInterface` and `DerivedInterface`. The `BaseInterface` interface is inherited by the `DerivedInterface` interface. Then the `InterfaceImplemeter` class implements the `DerivedInterface` interface

```
using System; namespace
Class_Demos{
   interface BaseInterface{ void
      GetPersonalDetail(); void
      GetContactDetail();
   }
   interface
        DerivedInterface:BaseInterface{ void
        ShowDetail();
   }
 class InterfaceImplementer :
    DerivedInterface{ string name;
    long phonenum;
    public void GetPersonalDetail()
        { Console.WriteLine("Enter your Name");
        name = Console.ReadLine();
    }
    public void GetContactDetail()
       { Console.WriteLine("Enter your Phone Number");
       phonenum = int.Parse(Console.ReadLine());
    }
    public void ShowDetail(){ Console.WriteLine("\nYour
       Details:"); Console.WriteLine("Name: " + name);
       Console.WriteLine("Phone Number: " + phonenum);
    }
 }
 class
    InterfaceDemo1{ stati
    c void Main(){
      InterfaceImplementer Myobj = new InterfaceImplementer();
      Myobj.GetPersonalDetail();
      Myobj.GetContactDetail();
      Myobj.ShowDetail();

      Console.ReadLine();
    }
 }
```

}

5 What are delegates?Explain the concepts of multicast delegates with an example

## 1.1 Creating and using Delegates:

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

i. Declaring a delegate
ii. Defining delegate methods
iii. Creating delegate objects
iv. Invoking delegate objects

### i. Declaring a delegate:

A delegate represents a class type, it can be declared at any place where a class can be defined – outside all classes or inside a class.

Syntax:

Where,

```
Access-modifier delegate <return-type>
                    delegate_name(arg1, arg2, … argn);
```

**Access-modifier** –that controls the accessibility of the delegate and can be *public, private, protected* or *internal*.

**delegate** – is a keyword indicates the declaration belongs to a delegate

**return-type** – return type of the delegate

**delegate-name** – is the name of the delegate

**Parameter-list** – the list of parameter that the delegate takes. Example: **public delegate void Compute(int x, int y);**

## Multicasting with Delegates:

A delegate object to hold references of and invoke multiple methods. Such objects are called ***multicast*** delegates or ***combinable*** delegates.

```
sample s1=new sample(m.add);
s1 += m.sub;
s1 += m.mul;
```

```
using System; namespace
Class_Demos{
        delegate void sample(int a,int b);
    class MCDelegates{
        void add(int n1, int n2){
            Console.WriteLine(n1+ "+" +n2+ "=" + (n1 + n2));
        }
        void mul(int n1, int n2) {
            Console.WriteLine(n1 + "*" + n2 + "=" + (n1 * n2));
        }
        void sub(int n1, int n2){
```

```
            Console.WriteLine(n1 + "-" + n2 + "=" + (n1 - n2));
        }
        static void Main(){
            MCDelegates m=new MCDelegates();
            sample s1=new sample(m.add); s1 +=
            m.sub;
            s1 += m.mul;
            s1(10, 20);
            Console.ReadKey();
        }
    }
}
```

6 Explain how custom exceptions will be created in C# with suitable example.

## Custom Exception(ApplicationException):

The **ApplicationException** is thrown by a user program, not by the common language runtime. If you are designing an application that needs to create its **own exceptions**.

```
public class ApplicationException :
Exception{
        //Various constructors.
}
```

To create your own exception class, here are some important recommendations:

- Give a meaningful name to your Exception class and end it with Exception.
- Throw the most specific exception possible.
- Give meaningful messages.
- Do use InnerExceptions.
- When wrong arguments are passed, throw an ArgumentException or a subclass of it, if necessary.

## Building Custom Exceptions, Take One:

Any custom exception we create needs to derive from the **System.Exception** class. You can either derive directly from it or use an intermediate exception like **SystemException** or **ApplicationException** as base class.

```
using System;
namespace Chapter5_Examples {
  class TestException :
    ApplicationException{ public override
    string Message{
       get{
        return "This exception means something bad happened";
      }
    }
  }
```

```
   class CusException
      { static void Main(){
        try{
           throw new TestException();
        }
        catch(TestException ex)
           { Console.WriteLine(ex);
        }
        Console.ReadLine();
      }
   }
}
```

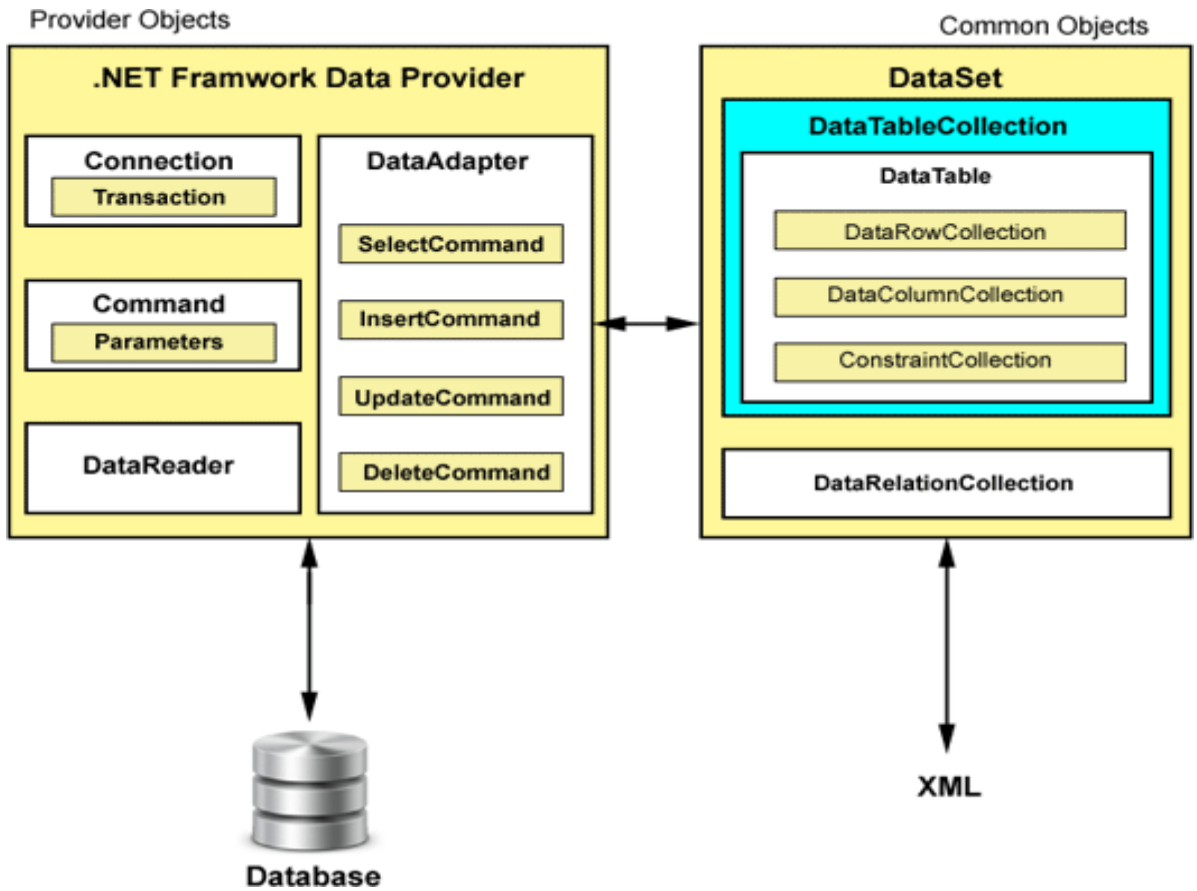
7 Describe the architecture of ADO.NET with a neat diagram.

**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 architectures:

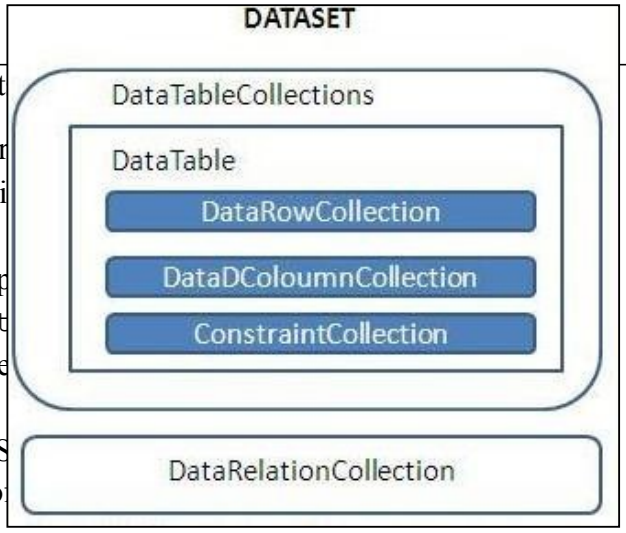
- **Connected architecture:** the application remains connected with the database throughout the processing.
- **Disconnected architecture:** the application automatically connects/disconnects during the processing. The application uses temporary data on the application side called a **DataSet**.




The **four Objects** from the .Net Framework provides the functionality of Data Providers in the ADO.NET. They are

i. The **Connection Object** provides physical connection to the Data Source.
ii. The **Command Object** uses to perform SQL statement or stored procedure to be executed at the Data Source.
iii. 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.
iv. The **DataAdapter Object**, which populate a Dataset Object with results from a Data Source

| OBJECT | DESCRIPTION |
|---|---|
| Connection | • Creates connection to the data source.<br>• The base class for all the **Connection** objects is the **DbConnection** class.<br>• The Connection object has the methods for opening and closing connection and beginning a transaction.<br>Provides **three** types of connection classes:<br>• **SqlConnection** object: to connect to Microsoft SQL Server<br>• **OleDbConnection** object: to connect to Microsoft Access<br>• **OdbcConnection** object: to connect to Oracle |

| | | | |
|---|---|---|---|
| | | Command | • Executes a command against the data source and retrieve a DataReader or DataSet.<br>• It also executes the INSERT, UPDATE or DELETE command against the data source.<br>• The base class for all the **Command** objects is the **DbCommand** class. |
| 8 | Explain the procedure of getting connected to a database and running the following queries with relevant example:<br>(i) Insert record to a table.<br>string cs1 = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\Downloads\daya\Dot net\Dot net-2019\c#_examples\dbconnectivity1\test.accdb";<br>OleDbConnection con1 = new OleDbConnection(cs1);<br>con1.Open();<br>OleDbCommand cmd = new OleDbCommand();<br>cmd.Connection = con1;<br>cmd.CommandText="insert into t1 values('"+ TextBox1.Text + "', '"TextBox2.Text+"');<br>cmd.ExecuteNonQuery();<br>Response.Write("value inserted");<br>(ii) Delete records from a table.<br>string cs1 = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\Downloads\daya\Dot net\Dot net-2019\c#_examples\dbconnectivity1\test.accdb";<br>OleDbConnection con1 = new OleDbConnection(cs1);<br>con1.Open();<br>OleDbCommand cmd = new OleDbCommand();<br>cmd.Connection = con1;<br>cmd.CommandText="delete from t1 where sno='+Textbox1.Text+"';<br>cmd.ExecuteNonQuery();<br>Response.Write("value deleted"); | | |
| 9 | What is DataSet? Explain Various Component<br>`DataSet` is a very useful in-memory repr of data and acts as the core of a wide vari data based applications. A `DataSet` considered as a local copy of the relevant p the database. The data in the `DataSet` manipulated and updated independent of the You can load the data in the `DataSet` valid source, such as the MicrosoftS database, Oracle Database, or Microso database.<br>  <br> | | |
| | **DataSet Components** | | **Description** |

| | | | |
|---|---|---|---|
| | | DataTable | ✎ Consists of `DataRow` and `DataColumn` and stores data in the table row format.<br>✎ The `DataTable` is the central object of the ADO.NET library and similar to a table in a database.<br>✎ The maximum number of rows that a DataTable can contain is fixed at 16,777,216. |
| | | DataView | ✎ Represents a customized view of `DataTable` for sorting, filtering, searching, editing and navigation.<br>✎ allows to create a `view` on a `DataTable` to see a subset of data based on a preset condition specified in the `RowStateFilter` property.<br>✎ used to present a subset of data from the `DataTable`. |
| | | DataColumn | ✎ Consists of a number of columns that comprise a `DataTable`.<br>✎ A `DataColumn` is the essential building block of the `DataTable`.<br>✎ A `DataType` property of `DataColumn` determines the kind of data that a column holds. |
| | | DataRow | ✎ Represents a row in the `DataTable.`<br>✎ use the `DataRow` object and its properties and methods to retrieve, evaluate, insert, delete and update the values in the `DataTable.`<br>✎ `NewRow()` method of the `DataTable` to create a new `DataRow` and the `Add()` method to add the new `DataRow` to the `DataTable.`<br>✎ also delete `DataRow` using `Remove()` method. |
| | | DataRelation | ✎ Allows you to specify relations between various tables.<br>✎ used to relate two `DataTable` objects to each other through `DataColumn` objects.<br>✎ The relationships are created between matching columns in the parent and child tables. |
| 10 | Write a C# program to Display Records on GridView Control.<br>string cs1 = @"Provider=Microsoft.ACE.OLEDB.12.0;Data Source=D:\Downloads\daya\Dot net\Dot net-2019\c#_examples\dbconnectivity1\test.accdb";<br>OleDbConnection con1 = new OleDbConnection(cs1);<br>con1.Open();<br>OleDbDataAdapter d1=new OleDbDataAdapter("select * from t2",con1);<br>DataSet ds = new DataSet();<br>d1.Fill(ds, "[t2]");<br>DataView dv = new DataView(ds.Tables["[t2]"]);<br>//DataView dv = new DataView(ds.Tables[0]);<br>dv.Sort = "name2 desc";<br>GridView5.DataSource = dv;<br>GridView5.DataBind();<br>con1.Close(); | | |