

UNIT I: ASP.NET WITH C#

UNIVERSITY QUESTIONS WITH ANSWER

1. What is namespace? Explain with the help of an example. [Nov.2018 Regular]

- A namespace is designed for providing a way to keep one set of names separate from another.
- Namespace are used to organize your programs.
- The class names declared in one namespace does not conflict with the same class names in another.
- Namespace don't correspond to file, directory or assembly names.
- They could be written in separate files and / or separate assemblies and still belong to the same namespace.

Example:

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

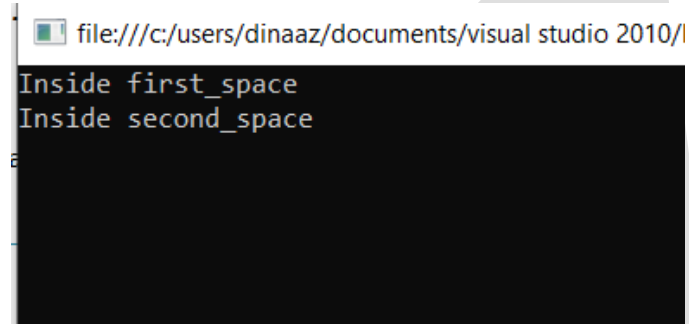
namespace first_space
{
    class namespace_cl
    {
        public void func()
        {
            Console.WriteLine("Inside first_space");
        }
    }
}

namespace second_space
{
    class namespace_cl
    {
        public void func()
        {
            Console.WriteLine("Inside second_space");
        }
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
    }  
    }  
}  
classTestClass  
{  
  
    staticvoid Main(string[] args)  
    {  
        first_space.namespace_cl fc = newfirst_space.namespace_cl();  
        second_space.namespace_clsc = newsecond_space.namespace_cl();  
        fc.func();  
        sc.func();  
        Console.ReadKey();  
    }  
}
```

Output:



```
file:///c:/users/dinaaz/documents/visual studio 2010/...  
Inside first_space  
Inside second_space
```

2. Explain jagged array with an example. [Nov. 2018 Regular]

- Jagged array is an **array of arrays** such that member arrays can be of different sizes.
- In other words, the length of each array index can differ.
- The elements of Jagged Array are reference types and initialized to null by default.
- Jagged Array can also be mixed with multidimensional arrays. Here, the number of rows will be fixed at the declaration time, but you can vary the number of columns.

Declaration

In Jagged arrays, user has to provide the number of rows only. If the user is also going to provide the number of columns, then this array will be no more Jagged Array.

Syntax:

```
data_type[][] name_of_array = new data_type[rows][]
```

Example:

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
int[][] jagged_arr = new int[4][]
```

In the above example, a single-dimensional array is declared that has 4 elements(rows), each of which is a 1-D array of integers.

Initialization : The elements of Jagged Array **must be initialized** before its use. You can separately initialize each array element. There are many ways to initialize the Jagged array's element.

Ex: Consider the following declaration of a jagged array:

```
int [][] x = new int [3][];  
x[0] = new int [4];  
x[1] = new int [3];  
x[2] = new int [5];
```

After this allocation, the jagged array looks like this:

```
x[0] [0] x[0] [1] x[0] [2] x[0] [3]  
x[1] [0] x[1] [1] x[1] [2]  
x[2] [0] x[2] [1] x[2] [2] x[2] [3] x[2] [4]
```

How to store/access each element of a jagged array:

To store the number 15 in x[0][1], we will write: x[0][1] = 15;

Jagged arrays are useful in situations where we need a very large two-dimensional array that is sparsely populated.

Example

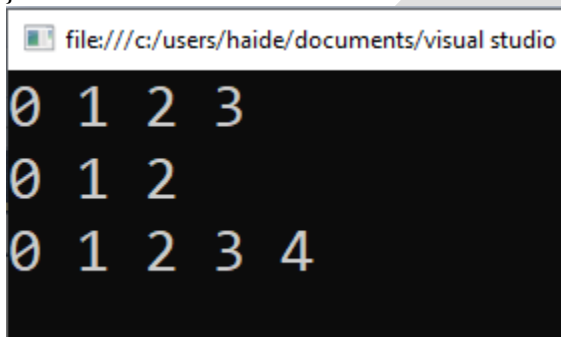
```
using System;  
namespace Jagged_Array_ex  
{  
    class Program  
    {  
        static void Main()  
        {  
            int[][] jagged = new int[3][]; //dec  
            jagged[0] = new int[4]; // 1st row contain 4 column  
            jagged[1] = new int[3];  
            jagged[2] = new int[5];  
            int i;  
            // Store values in first array and display the values.  
            for (i = 0; i < 4; i++)  
            {  
                jagged[0][i] = i;  
                Console.Write(jagged[0][i] + " ");  
            }  
            Console.WriteLine();  
        }  
    }  
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
// Store values in second array and display the values.
for (i = 0; i < 3; i++)
{
    jagged[1][i] = i;

    Console.Write(jagged[1][i] + " ");
}
Console.WriteLine();
// Store values in third array and Display the values.
for (i = 0; i < 5; i++)
{
    jagged[2][i] = i;
    Console.Write(jagged[2][i] + " ");
}

Console.ReadKey();
}
}
```



```
0 1 2 3
0 1 2
0 1 2 3 4
```

3. What is .NET Framework? Explain its architecture in brief. [Nov.2018 Regular]

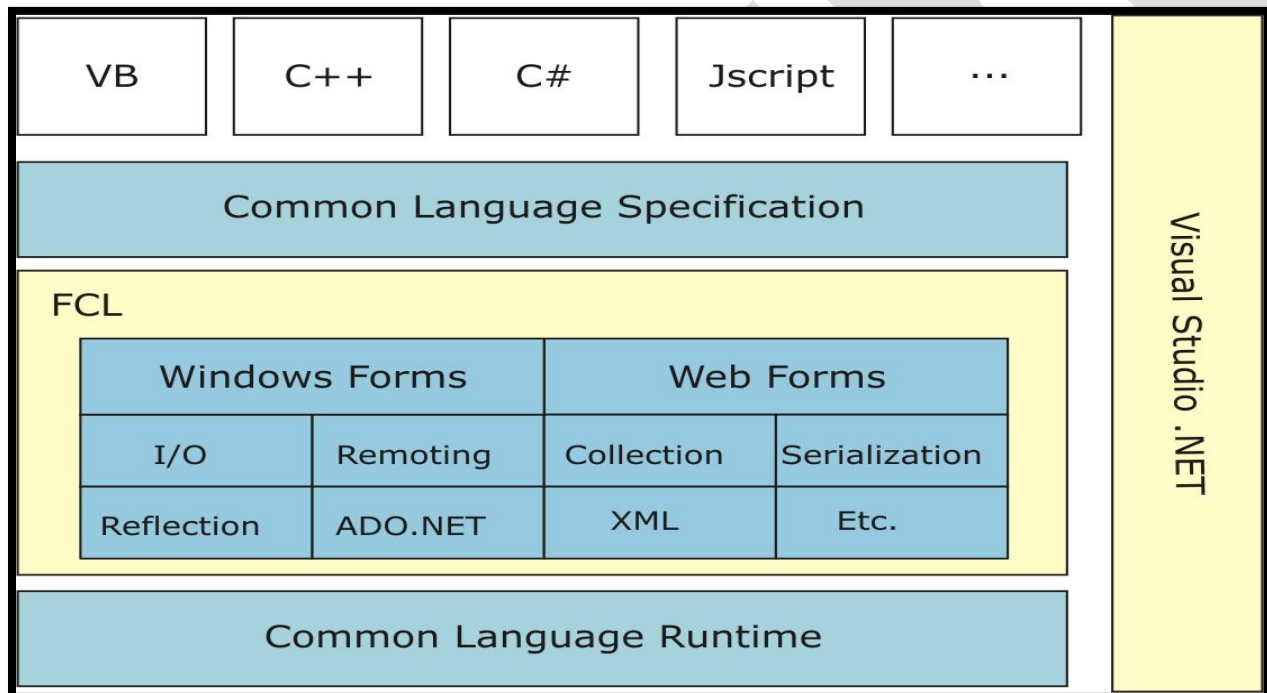
Draw and explain .NET Framework architecture. [May 2019 ATKT]

The .NET Framework is a new and revolutionary platform created by Microsoft for developing applications.

- The .NET framework is one of the tools provided by the .NET infrastructure and tools component of the .NET platform.
- The .NET platform provides a new environment for creating and running robust, scalable and distributed applications over the Web.
- C# derives much of its power from the .NET framework on which it runs.
- .Net is a platform introduced by Microsoft to develop, host, deploy, maintain and execute applications. In other words, it caters to the entire development life cycle of application making developers' life more comfortable.

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

- These applications could be, database driven client-server application, desktop or windows applications, web application like Online shopping web site, Online banking portal, smart application and web service.
- The .NET framework provides an environment for building, deploying and running web services and other applications. It consists of three distinct technologies.
 1. Common Language Runtime (CLR)
 2. Framework Base Classes
 3. User and program interfaces (ASP .NET and Windows forms).



The CLR is the core of the .NET framework and is responsible for loading and running C# programs. Base classes provide basic data types, collection classes and other general classes for use by C# and other .NET languages. The top layer contains a set of classes for developing web services and to deal with the user interface.

The Common Language Runtime

The Common Language Runtime is a runtime environment in which programs written in C# and other .NET languages are executed. It also supports cross-language interoperability.

The CLR provides a number of services that include:

- Loading and execution of programs
- Memory isolation for applications
- Verification of type safety
- Compilation of IL into native executable code
- Providing metadata

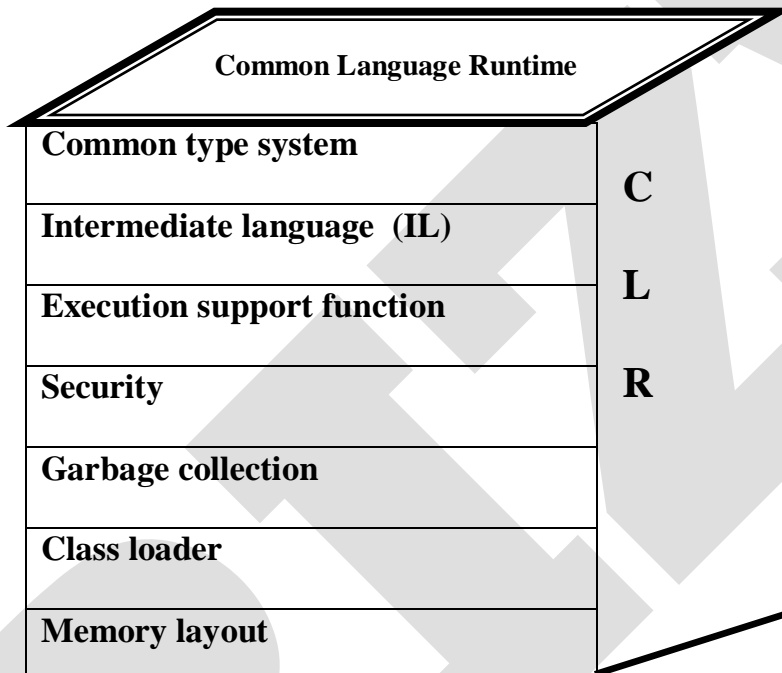
B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

- Memory management (automatic garbage collection)
- Enforcement of security
- Interoperability with other systems
- Managing exceptions and errors
- Support for tasks such as debugging and profiling.

Place of CLR



Components of CLR:



Common type System (CTS) - The .NET framework provides multiple languages support using the feature known as Common Type System that is built in to the CLR. The CTS support variety of types and operations found in most programming languages and therefore calling one language from another does not require type conversions. CTS supports two categories of types which derived from the base type called System. Object.

1. **Value types** directly contain the data. Actual data is always stored in stack.
2. **Reference types** store a reference to the memory address on the stack but the actual value is stored on heap.

Common Language Specification (CLS) - The Common Language Specification defines a set of rules that enables interoperability on the .NET platform. The CLS is a

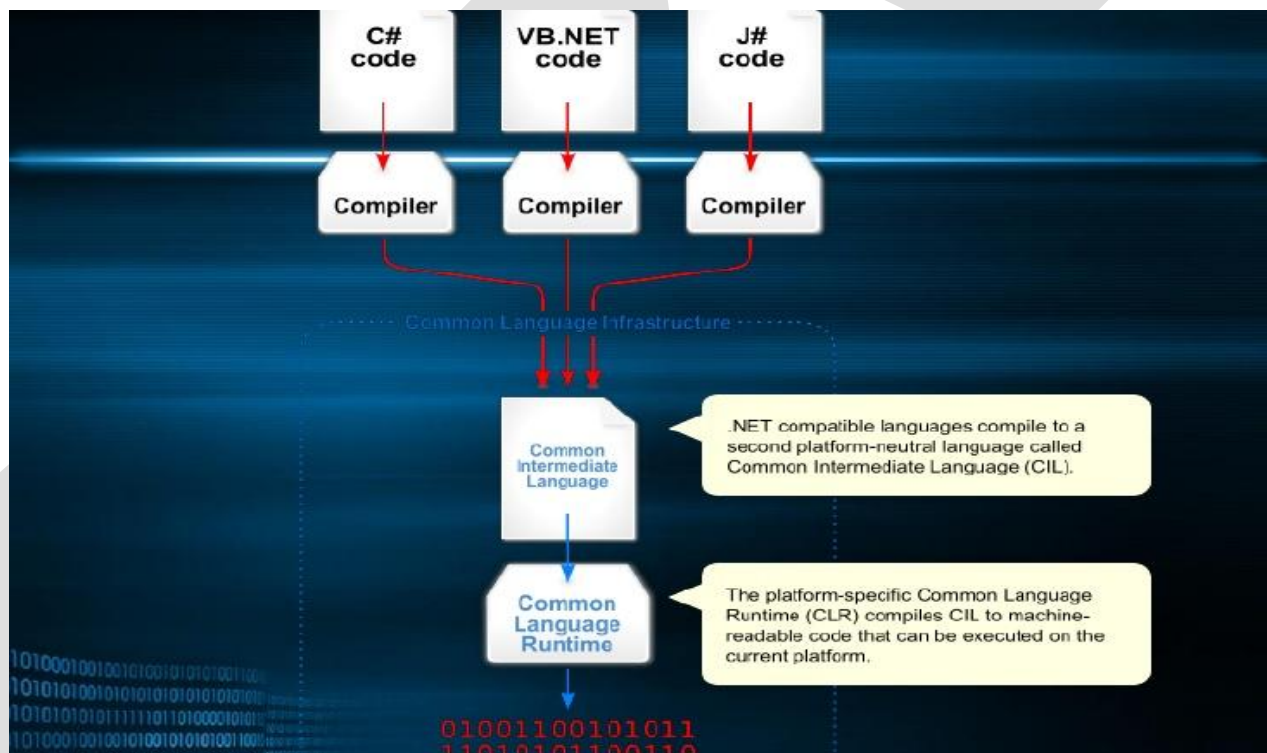
B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

subset of CTS and therefore the languages supporting the CLS can use each other's class libraries as if they are their own.

Microsoft Intermediate Language (MSIL) -The Common Language Runtime manages the execution of .NET code. Here is how it works: When you compile a C# program, the output of the compiler is not executable code. Instead, it is a file that contains a special type of pseudocode called *Microsoft Intermediate Language* (MSIL). MSIL defines a set of portable instructions that are independent of any specific CPU. MSIL or simply IL is an instruction set into which all the .NET programs are compiled. In essence, MSIL defines a portable assembly language. One other point: although MSIL is similar in concept to Java's bytecode, the two are not the same.

Any program compiled to MSIL can be run in any environment for which the CLR is implemented. This is part of how the .NET Framework achieves portability.

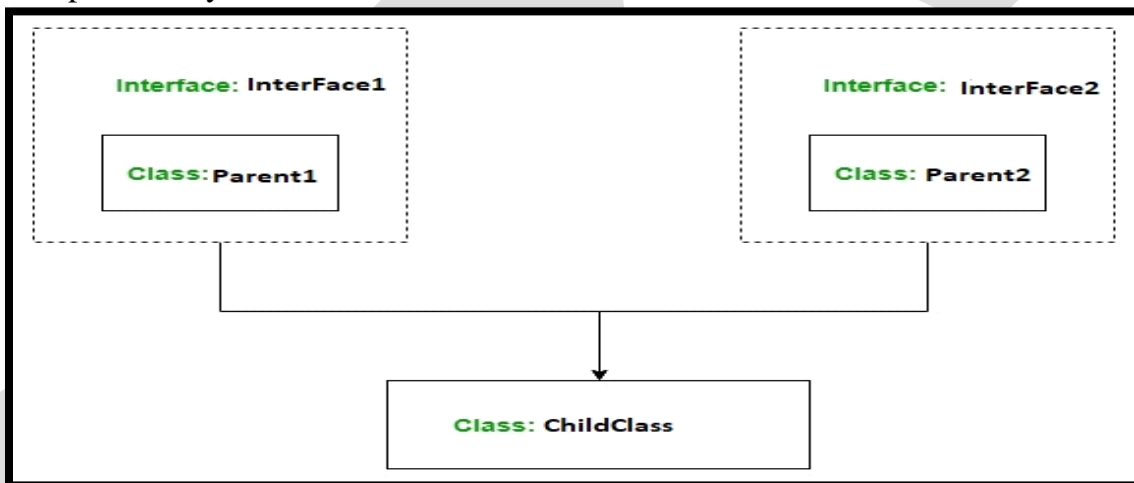
Microsoft Intermediate Language is turned into executable code using a *JIT compiler*. "JIT" stands for "Just-In-Time." The process works like this: When a .NET program is executed, the CLR activates the JIT compiler. The JIT compiler converts MSIL into native code on demand as each part of your program is needed.



B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

4. Write a program in C# to demonstrate multiple inheritance using interfaces. [Nov. 2018 Regular]

- C# does not support multiple inheritance. However, you can use interfaces to implement multiple inheritance.
- An interface is a collection of public instance (that is, non static) methods and properties that are grouped together to encapsulate specific functionality. After an interface has been defined, you can implement it in a class.
- This means that the class will then support all of the properties and members specified by the interface.



- Interfaces cannot exist on their own. You can't "instantiate an interface" as you can a class. In addition, interfaces cannot contain any code that implements its members; it just defines the members themselves.
- The implementation must come from classes that implement the interface.

Interfaces are declared by using the **interface** keyword. Here is a simplified form of an interface declaration:

```
interface name
{
    ret-type method-name1(param-list);
    ret-type method-name2(param-list);
    // ...
}
```


B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

ret-type method-nameN(param-list);
}

The following program demonstrates this:

```
using System;
interface abc
{
    void xyz();
}
interface def
{
    void xyz();
}
class Demo : abc, def
{
    public static void Main()
    {
        Console.WriteLine("Hello Interfaces");
        Demo ObjDemo = new Demo();
        abc objabc = ObjDemo;
        objabc.xyz();
        def objdef = ObjDemo;
        objdef.xyz();
        Console.Read();
    }
    void abc.xyz()
    {
        Console.WriteLine("In abc.xyz");
    }
    void def.xyz()
    {
        Console.WriteLine("In def.xyz");
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
Hello Interfaces  
In abc.xyz  
In def.xyz
```

5.Explain various types of constructors in C#. [Nov. 2018 Regular]

“C# supports a special type of method called constructor ,that enables an objects to initialize itself when it is created.”

- Constructor is used to initialize an object (instance) of a class.
- Constructor is a like a method without any return type.
- Constructor has same name as class name.
- Constructor follows the access scope (Can be private, protected, public, Internal and external).
- Constructor can be overloaded.

Types of Constructors in C#

There are 5 types of constructor in C# as listed below

1. **Default Constructor**
2. **Parameterized Constructor**
3. **Copy Constructor**
4. **Static Constructor**
5. **Private Constructor**

Default Constructor :-

A constructor without any parameters is called as default constructor means a constructor which does not have any input parameter is known as default constructor.

Syntax :

```
class User  
{  
    // Default Constructor  
    User()  
{  
    // Your Custom Code  
}  
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

PROGRAM :-

using System;

namespace ConstructorEX

{

class C1

{

public int a, b;

public C1() // Default Constructor

{

a = 10;

b = 20;

}

public void display()

{

Console.WriteLine("Value of a is " + a);

Console.WriteLine("Value of b is " + b);

}

}

class Demo

{

static void Main()

{

C1 obj = new C1(); // constructor to initialize object

obj.display();

Console.Read();

}

}

}

file:///E:/ASP.NET 2016-17/7-6-16/OOPS Progr

```
Value of a is 10
Value of b is 20
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

Parameterized Constructor :-

A constructor having one or more parameters is called as parameterized constructor means a constructor which is having single input parameter or multiple input parameters of same data types or different data types are known as parameterized constructor.

PROGRAM :-

```
using System;
class c1
{
    int a, b;

    public c1(int x, int y) //Parameterized Constructor
    {
        a = x;
        b = y;
    }

    public void displayed()
    {
        Console.WriteLine(" Value of a is " + a);
        Console.WriteLine(" Value of b is " + b);
    }
}

class b
{
    public static void Main(string[] args)
    {
        c1 obj1 = new c1(45, 98);
        obj1.displayed();

        Console.WriteLine("\n\n");

        c1 obj2 = new c1(32, 90);
        obj2.displayed();

        Console.Read();
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

file:///E:/ASP.NET 2016-17/7-6-16/

```
Value of a is 45  
Value of b is 98
```

```
Value of a is 32  
Value of b is 90
```

Copy Constructor :-

A constructor that contains a parameter of same class type is called as copy constructor. C# does not provide a copy constructor. A copy constructor enables you to copy the data stored in the member variables of an object of the class into another new object means it helps to copy data stored in one object into another new object of the same instance.

PROGRAM :-

```
using System;  
namespace ConsoleApplication3  
{  
    class Sample  
    {  
        public string param1, param2;  
        public Sample(string x, string y)  
        {  
            param1 = x;  
            param2 = y;  
        }  
        public Sample(Sample obj) // Copy Constructor  
        {  
            param1 = obj.param1;  
            param2 = obj.param2;  
        }  
    }  
    class Program  
    {  
        static void Main(string[] args)  
        {  
            Sample obj = new Sample("Welcome", "abc"); // Create instance to class Sample
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
Sample obj1=new Sample(obj); // Here obj details will copied to obj1
Console.WriteLine(obj1.param1 +" to " + obj1.param2);
Console.ReadLine();
}
}
}
```

OUTPUT :-

Welcome abc

Static Constructor :-

A static constructor should be parameter less means it should not contain any input parameter. Program will not execute if static constructor is having any input parameter. Static constructor can be invoked once for any number instances are created and it is invoked only during the first initialization of instance. It is used to initialize static fields of the class Static constructor is created using a static keyword as shown below.

```
using System;
namespace ConsoleApplication3
{
    class Sample
    {
        public string param1, param2;
        static Sample()
        {
            Console.WriteLine("Static Constructor");
        }
        public Sample()
        {
            param1 = "Sample";
            param2 = "Instance Constructor";
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Here Both Static and instance constructors are invoked for first instance
            Sample obj=new Sample();
            Console.WriteLine(obj.param1 + " " + obj.param2);
        }
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
// Here only instance constructor will be invoked
Sample obj1 = new Sample();
Console.WriteLine(obj1.param1 + " " + obj1.param2);
Console.ReadLine();
}
}
}
```

OUTPUT :-

Static Constructor

Sample Instance Constructor

Sample Instance Constructor

Private Constructor :-

In c#, Private Constructor is a special instance constructor and it is used in a classes that contains only static members. If a class contains one or more private constructors and no public constructors, then the other classes are not allowed to create an instance for that particular class except nested classes.

```
using System;
namespace ConsoleApplication3
{
    public class Sample
    {
        public string param1, param2;
        public Sample(string a,string b)
        {
            param1 = a;
            param2 = b;
        }
        private Sample() // Private Constructor Declaration
        {
            Console.WriteLine("Private Constructor with no prameters");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Here we don't have chance to create instace for private constructor
            Sample obj = new Sample("Welcome","toxyz");
            Console.WriteLine(obj.param1 + " " + obj.param2);
            Console.ReadLine();
        }
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
}  
}  
}
```

OUTPUT :-

Welcome to xyz

6.What is delegate? Explain multicast delegate with an example. [Nov. 2018Regular]

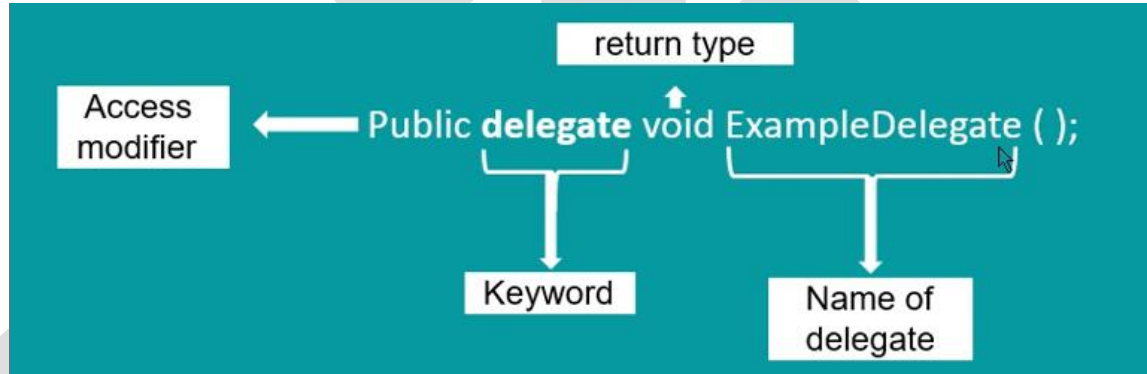
Delegate:

C# delegates are similar to pointers to functions, in C or C++. A **delegate** is a reference type variable that holds the reference to a method. ... **Delegates** are especially used for implementing events and the call-back methods. All **delegates** are implicitly derived from the System.**Delegate** class.

Declaring a Delegate using delegate keyword

```
public delegate typeofdelegate delegatename();
```

Example:



Multicast Delegate:

- A delegate can invoke only one method,(whose reference is encapsulated into the delegate).
- It is possible for certain delegate to hold & invoke multiple methods.
- Such delegates are called as Multicast delegate.
- They are also known as combinable delegates.
- Must satisfy the following conditions:
 1. Return type of delegate must be void.

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

2. None of the parameters of the delegate type can be declared as output parameter , using out keyword.

They are derived from **System.MulticastDelegate** class.

Example

```
using System;
namespace multicast_delegate_example
{
    delegate void MutiCastDelegate();
    class multiCastdelegateDemo
    {
        public static void Display()
        {
            Console.WriteLine("ZAIDI");
            Console.WriteLine();
        }
        public static void Print()
        {
            Console.WriteLine("RIZVI");
            Console.WriteLine();
        }
    }
} // class multiCastdelegateDemo close
class Program
{
    static void Main(string[] args)
    {
        MutiCastDelegate m1 = new
MutiCastDelegate(multiCastdelegateDemo.Display);
        MutiCastDelegate m2 = new MutiCastDelegate(multiCastdelegateDemo.Print);
        MutiCastDelegate m3 = m1 + m2;
        MutiCastDelegate m4 = m2 + m1;
        MutiCastDelegate m5 = m3 - m2;
        m3();
        m4();
        m5();
        Console.Read();
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

ZAIDI

RIZVI

RIZVI

ZAIDI

ZAIDI

7.Elaborate Array memory representation with an example. [May 2019 ATKT]

1. Array Memory Representation in One-Dimensional Array [1D].

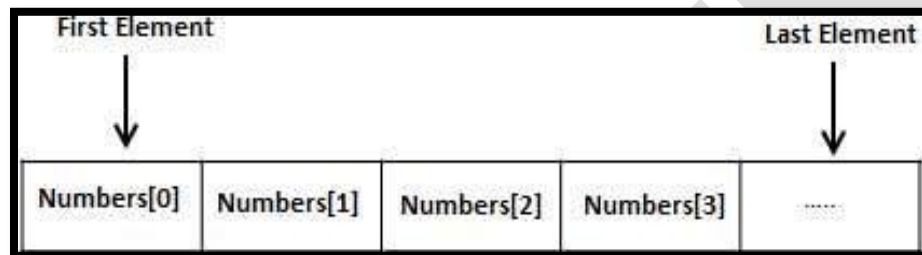
- An array stores a fixed-size sequential collection of elements of the same "Data Type". An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type stored at contiguous/Linear Sequences memory locations.
- Instead of declaring individual variables, such as:
 - `int [] number0 = new int[10];`
 - `int [] number1 = new int[10];`
- User can declare one array variable such as `numbers` and use `numbers[0]`, `numbers[1]`, and ..., `numbers[99]` to represent individual variables. A specific element in an array is accessed by an index.

eg:

- `int [] number = new int[10];`
- `number[0]=12;`
- `number[1]=23;`
- `number[2]=34;`
- `number[10]=45;`

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

- All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element. In arrays, index number starts from 0 to N^{th} number.



```
CODE: //One-Dimensional Array
using System;
class MyArray
{
    static void Main(string[] args)
    {
        int[] number = new int[10]; // number is an array of 10 integers.
        int i, j;                    // integer variables are declared.

        // initialize elements of array number in for loop.
        for (i = 0; i < 10; i++)
        {
            number[i] = i + 100;
        }

        // output each array element's value is retrieved using for loop.
        for (j = 0; j < 10; j++)
        {
            Console.WriteLine("Number{" + j + "}: " + number[j]);
        }
        Console.Read();
    }
}
```

OUTPUT:

```
CODE: //Using Foreach Loop
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
Number{[0]}: 100  
Number{[1]}: 101  
Number{[2]}: 102  
Number{[3]}: 103  
Number{[4]}: 104  
Number{[5]}: 105  
Number{[6]}: 106  
Number{[7]}: 107  
Number{[8]}: 108  
Number{[9]}: 109
```

```
using System;  
class MyArray  
{  
    static void Main(string[] args)  
    {  
        int[] number = new int[10]; // number is an array of 10 integers.  
        //integer variables are declare.  
  
        // initialize elements of array number in for loop.  
  
        for (i = 0; i < 10; i++)  
        {  
            number[i] = i + 100;  
        }  
        // output each array element's value is retrieve using Foreach loop.  
        foreach (int j in number)  
        {  
            i = j;  
  
            Console.WriteLine("Number{[\"+j+\"]} = "+i);  
        }  
    }  
}  
OUTPUT:
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
Number{[100]} = 100
Number{[101]} = 101
Number{[102]} = 102
Number{[103]} = 103
Number{[104]} = 104
Number{[105]} = 105
Number{[106]} = 106
Number{[107]} = 107
Number{[108]} = 108
Number{[109]} = 109
```

2. Array Memory Representation in Two-Dimensional Array [2D].

- C# allows multidimensional arrays. Multi-dimensional arrays are also called rectangular array. You can declare a 2-dimensional array of strings as –
-string [,] names;
- or, a 3-dimensional array of int variables as – -int [, ,] m;
- The simplest form of the multidimensional array is the 2-dimensional array. A 2-dimensional array is a list of one-dimensional arrays.
- A 2-dimensional array can be thought of as a table, which has x number of rows and y number of columns. Following is a 2-dimensional array, which contains 3 rows and 4 columns

	Column 0	Column 1	Column 2	Column 3
Row 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

- Thus, every element in the array a is identified by an element name of the form a[i , j], where a is the name of the array, and i and j are the subscripts that uniquely identify each element in array a.

CODE: *//Two-Dimensional Array*

```
using System;
```

```
class MyArray
```

```
{
```

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

```
{
```

```
//an array with 5 rows and 2 columns.
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
int[, ] a = newint[5, 2] { { 0, 0 }, { 1, 2 }, { 2, 4 }, { 3, 6 },  
    { 4, 8 } };  
inti, j;  
  
// output each array element's value using Nested for loop.  
for (i = 0; i < 5; i++)  
    {  
    for (j = 0; j < 2; j++)  
        {  
        Console.WriteLine("a[{0}+{1}"] = "a[i, j]);  
        }  
    }  
Console.Read();  
}
```

OUTPUT:

```
a[[0], [0]] = 0  
a[[0], [1]] = 0  
a[[1], [0]] = 1  
a[[1], [1]] = 2  
a[[2], [0]] = 2  
a[[2], [1]] = 4  
a[[3], [0]] = 3  
a[[3], [1]] = 6  
a[[4], [0]] = 4  
a[[4], [1]] = 8
```

8.Explain any five properties / methods of Math Class. [May 2019 ATKT]

The System.Math class offers many constant fields and static methods that you can use to do trigonometric, logarithmic, and other mathematical calculations.

Methods

METHOD	DESCRIPTION
<u>Abs()</u>	Returns the absolute value of a specified number.
<u>Acos()</u>	Returns the angle whose cosine is the specified number.
<u>Acosh()</u>	Returns the Inverse hyperbolic cosine of the specified number.
<u>Asin()</u>	Returns the angle whose sine is the specified number.

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

Asinh()	Returns the Inverse hyperbolic sine of the specified number.
Atan()	Returns the angle whose tangent is the specified number.
Atan2()	Returns the angle whose tangent is the quotient of two specified numbers.
Atanh()	Returns the Inverse hyperbolic tangent of the specified number.
BigMul()	Produces the full product of two 32-bit numbers.
Cbrt()	Returns the cube root of a specified value.
Ceiling()	Returns the smallest integral value greater than or equal to the specified number.
Cos()	Returns the cosine of the specified angle.
Cosh()	Returns the hyperbolic cosine of the specified angle.
DivRem()	Calculates the quotient of two numbers and also returns the remainder in an output parameter.
Exp()	Returns e raised to the specified power.
Floor()	Returns the largest integral value less than or equal to the specified number.
IEEERemander()	Returns the remainder resulting from the division of a specified number by another specified number.
Log()	Returns the logarithm of a specified number.
Log10()	Returns the base 10 logarithm of a specified number.
Max()	Returns the larger of two specified numbers.
Min()	Returns the smaller of two numbers.
Pow()	Returns a specified number raised to the specified power.
Round()	Rounds a value to the nearest integer or to the specified number of fractional digits.
Sign()	Returns an integer that indicates the sign of a number.
Sin()	Returns the sine of the specified angle.
Sqrt()	Returns the square root of a specified number.
Tan()	Returns the tangent of the specified angle.
Truncate()	Calculates the integral part of a number.

```
using System;
namespace ConsoleApplication5
{
    class Program
    {
        static void Main(string[] args)
        {
            // To find E constant values
            double e = Math.E;
            // Print result
            Console.WriteLine("Math.E = " + e);
            // To find PI constant values
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
double pi_value = Math.PI;
// Print result
Console.WriteLine("Math.PI = " + pi_value);
// Taking decimal values
decimal[] deci = {Decimal.MinValue, 45.14M, 0M, -17.47M, Decimal.MaxValue};
// using foreach loop
foreach (decimal value in deci)

    // Displaying the result
    Console.WriteLine("Absolute value of {0} = {1}", value, Math.Abs(value));
double x = 81;

// Input positive value, Output square root of x
Console.WriteLine(Math.Sqrt(x));
double a = 70;
// converting value to radians
double b = (a * (Math.PI)) / 180;

// using method and displaying result
Console.WriteLine(Math.Cos(b));
Console.WriteLine(Math.Asin(b));
Console.WriteLine(Math.Atan(b));
Console.WriteLine(Math.Sin(b));
Console.Read();
}
}
```

```
Math.E = 2.71828182845905
Math.PI = 3.14159265358979
Absolute value of -79228162514264337593543950335 = 79228162514264337593543950335
Absolute value of 45.14 = 45.14
Absolute value of 0 = 0
Absolute value of -17.47 = 17.47
Absolute value of 79228162514264337593543950335 = 79228162514264337593543950335
90.342020143325669
NaN
0.884869583149728
0.939692620785908
```

9. Give details about type conversion. [May 2019 ATKT]

- Type conversion is converting one type of data to another type.
- It is also known as Type Casting in C#, type casting has two forms.
 1. Implicit type conversion.

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

2. Explicit type conversion

Implicit Type Conversion - These conversions are performed by C# in a type-safe manner. For example, are conversions from smaller to larger integral types and conversions from derived classes to base classes.

Explicit type conversion - These conversions are done explicitly by users using the pre-defined functions. Explicit conversions require a cast operator.

Following table shows the implicit types of conversion that is supported by C# :

CONVERT FROM DATA TYPE	CONVERT TO DATA TYPE
byte	short, int, long, float, double
short	int, long, float, double
int	long, float, double
long	float, double
float	double

The following example shows an **Explicit** type conversion -

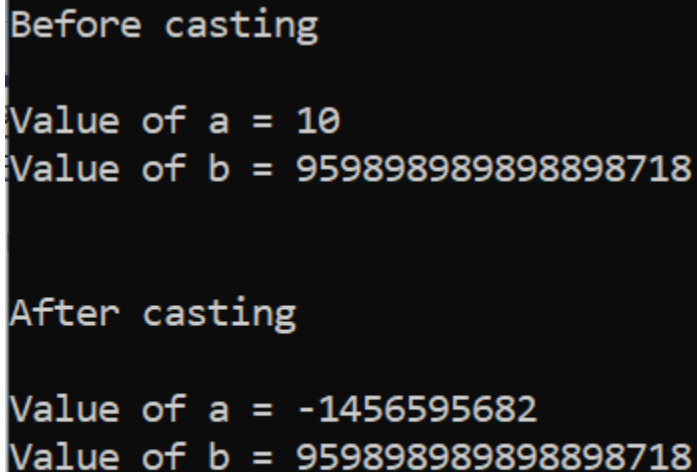
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace casting_example
{
    class Program
    {
        static void Main(string[] args)
        {
            int a = 10;
            long b = 959898989898898718;
            Console.WriteLine("Before casting \n");
            Console.WriteLine("Value of a = " + a);
            Console.WriteLine("Value of b = " + b);
            Console.WriteLine("\n");
            //b = a; // Implicit Type Conversion
            a = (int)b; // Explicit type conversion
            Console.WriteLine("After casting \n");
        }
    }
}
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
Console.WriteLine("Value of a = " + a);  
Console.WriteLine("Value of b = " + b);  
Console.Read();
```

```
}  
}  
}
```



```
Before casting  
  
Value of a = 10  
Value of b = 959898989898898718  
  
After casting  
  
Value of a = -1456595682  
Value of b = 959898989898898718
```

10. Write short notes on Value and Reference types. [May 2019 ATKT]

Value Type:

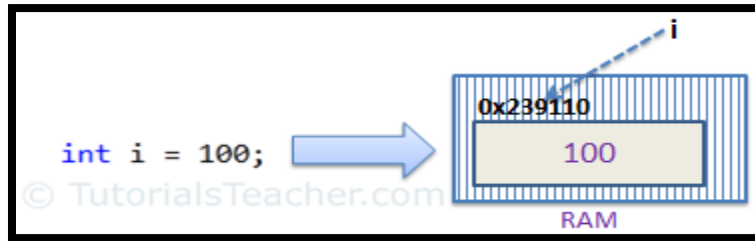
A data type is a value type if it holds a data value within its own memory space. It means variables of these data types directly contain their values. For example, consider integer variable:

```
int i=100;
```

The system stores 100 in the memory space allocated for the variable 'i'.

The following image illustrates how 100 is stored at some hypothetical location in the memory (0x239110) for 'i':

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE



The following data types are all of value type:

Bool	byte	Char
Decimal	double	Enum
float	int	Long
Sbyte	short	Struct
UInt	ulong	Ushort

Passing parameter by value

- By default, parameters are passed by value. In this method a duplicate copy is made and sent to the called function. There are two copies of the variables. So if you change the value in the called method it won't be changed in the calling method.
- Value parameter is also called in parameter. A parameter declared with no modifiers is a value parameter.
- A value parameter corresponds to a local variable that gets its initial value from the corresponding argument supplied in the method invocation. A method is permitted to assign new values to a value parameter.
- Such assignments only affect the local storage location represented by the value parameter-they have no effect on the actual argument given in the method invocation.

We use this process when we want to use but don't want to change the values of the parameters passed.

Practical demonstration of passing parameter by value

```
using System;
namespace PassByValue
{
    public class XX
    {
        public int sum(int a, int b)
        {
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
a = a + 100;
b = b + 200;
return (a + b);
}
}
class Program
{
    static void Main(string[] args)
    {
        int a = 100, b = 200;
        XX obj = new XX();

        Console.WriteLine("Sum of a and b is : " + obj.sum(a, b));
        Console.WriteLine();
        Console.WriteLine("Value of a is : " + a);
        Console.WriteLine();
        Console.WriteLine("Value of b is : " + b);
        Console.WriteLine();
        Console.ReadLine();
    }
}
```

Sum of a and b is : 600
Value of a is : 100
Value of b is : 200

Reference Type

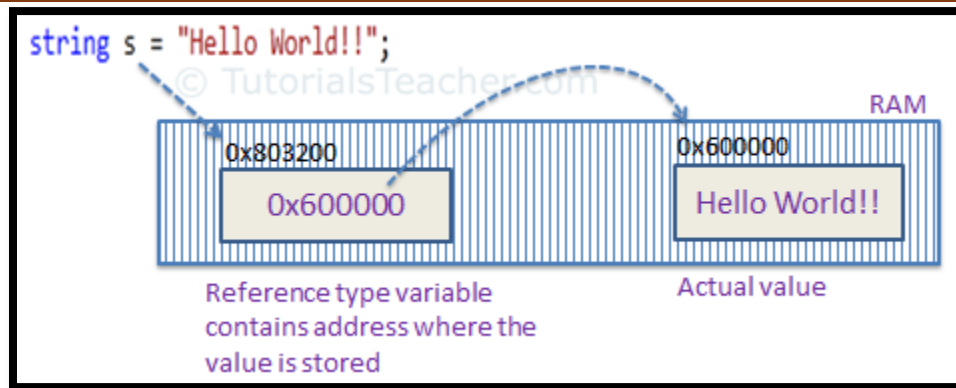
Unlike value types, a reference type doesn't store its value directly. Instead, it stores the address where the value is being stored. In other words, a reference type contains a pointer to another memory location that holds the data.

For example, consider following string variable:

```
String s="Hello World!!";
```

The following image shows how the system allocates the memory for the above string variable.

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE



As you can see in the above image, the system selects a random location in memory (**0x803200**) for the variable 's'. The value of a variable is **0x600000** which is the memory address of the actual data value. Thus, reference type stores the address of the location where the actual value is stored instead of value itself.

The following data types are of reference type:

string
All arrays, even if their elements are value types
Class
Delegates

Passing parameter by ref

Passing parameters by ref uses the address of the actual parameters to the formal parameters. It requires ref keyword in front of variables to identify in both actual and formal parameters.

A parameter declared with a ref modifier is a reference parameter. Contrastingly to value parameter, a reference parameter does not make a new storage location. Instead, a reference parameter represents the same storage location as the variable given as the argument in the method invocation. Within a method, a reference parameter is always considered definitely assigned.

The process of ref is bidirectional i.e. we have to supply value to the formal parameters and we get back processed value.

We use this process when we want to use or change the values of the parameters passed.

The ref keyword passes arguments by reference. It means any changes made to this argument in the method will be reflected in that variable when control returns to the calling method.

using System;

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

```
namespace PassByRef
{
    class XX
    {
        public int sum(ref int a, ref int b)
        {
            a = a + 100;

            b = b + 200;

            return (a + b);
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            int a = 100, b = 200;

            XX obj = new XX();

            Console.WriteLine("sum of a and b is : " + obj.sum(ref a, ref b));

            Console.WriteLine("Value of a is : " + a);

            Console.WriteLine("Value of b is : " + b);
            Console.ReadLine();
        }
    }
}
```

Output:

Sum of a and b is : 600

Value of a is : 200

Value of b is : 400

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

11.Explain static members and partial class. [May 2019 ATKT]

Static Members :-

- A class can be static, and it can have static members, both functions and fields.
- A static class can't be instantiated, so in other words, it will work more as a grouping of related members than an actual class.
- You may choose to create a non-static class instead, but let it have certain static members.
- A non-static class can still be instantiated and used like a regular class, but you can't use a static member on an object of the class. A static class may only contain static members.

First, here is an example of a static class:

PROGRAM :-

```
public static class MyStaticClass
{
    public static int myStaticMyStaticClass
    public static void MyStaticMethod()
    {
        Console.WriteLine("This is a static method.");
    }
    public static int MyStaticProperty { get; set; }
}
class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("MyStaticClass.myStaticVariable");
        MyStaticClass.MyStaticMethod();
        MyStaticClass.MyStaticProperty = 100;
        Console.WriteLine("MyStaticClass.MyStaticProperty");
    }
}
```

OUTPUT :-

```
0
This is a static method.
100
```

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

Partial class :-

- There are several situations when splitting a class definition is desirable:
- When working on large projects, spreading a class over separate files enables multiple programmers to work on it at the same time. When working with automatically generated source, code can be added to the class without having to recreate the source file.

keyword modifier, as shown here:

Syntax :

```
public partial class Employee
{
    public void DoWork()
    {
    }
}
public partial class Employee
{
    public void GoToLunch()
    {
    }
}
```

The partial keyword indicates that other parts of the class, struct, or interface can be defined in the namespace. All the parts must use the partial keyword. All the parts must be available at compile time to form the final type. All the parts must have the same accessibility, such as public, private, and so on.

B.Sc - IT/CS DEPARTMENT OF RIZVI COLLEGE OF ARTS, SCIENCE AND COMMERCE

Diagram:

