USN | | | | | | | | | | |

Internal Test I – September 2018

Sub: Programming Using C# & .Net | Sub Code: 16MCA52 | Branch: MCA

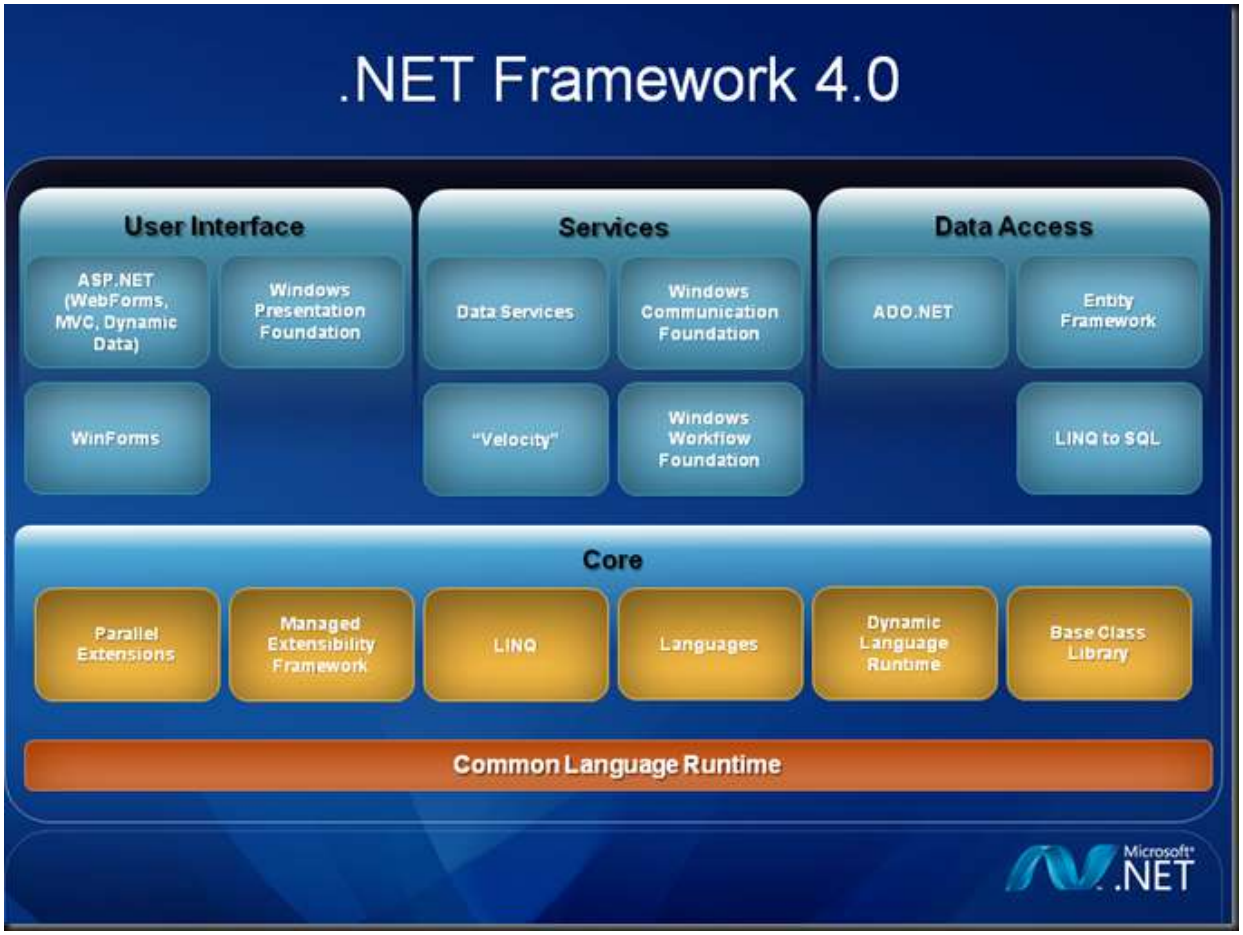
Date: 7/09/18 | Duration: 90 min's | Max Marks: 50 | Sem / Sec: A & B | OBE

Answer ONE FULL Question from each part.

		MARKS	CO	RBT
Part I-1 (a)	Explain the architecture of .NET framework 4.0 and describe its component.	[10]	CO3	L2
2 (a)	What is namespace? Explain the steps involved in creating a namespace and illustrate few common namespaces.	[10]	CO3	L2
Part II-3 (a)	Write a C# program to demonstrate Boxing and Unboxing	[6]	CO2	L2
(b)	List and explain the Access specifiers in C#.	[4]	CO2	L2
4 (a)	Name and explain the access modifiers for Structs in C# .	[4]	CO2	L2
(b)	Explain the types of Inheritance in C#. Also discuss the order of execution of constructor in inheritance.	[6]	CO2	L2
Part III-5(a)	Explain the characteristics of Abstract classes and abstract methods	[5]	CO2	L2
(b)	Write Short notes on Metadata & Assemblies	[5]	CO3	L2
6(a)	Define the following class and explain with program. Partial Classes , Sealed Classes	[10]	CO2	L2
Part IV-7(a)	Explain with example the method of implementing encapsulation using class Properties & Assessors and Mutators.	[10]	CO2	L2
8	What is Polymorphism? Explain in detail Compile Time polymorphism and Runtime Polymorphism	[10]	CO2	L2
Part V-9(a)	List the difference between Properties and Indexers with example.	[6]	CO2	L2
(b)	Explain the steps involved in creating and using delegate in C# program	[4]	CO2	L1
10(a)	Discuss static class and static members.	[5]	CO2	L2

1(a) Explain the architecture of .NET framework 4.0 and describe its component.

Ans:



.NET provides the framework that helps in developing portable, executable, scalable and robust applications. The applications developed in .NET framework 4.0 can be executed in a distributed environment. .NET framework is designed to address the latest needs of the developers.

Tthe components of .Net framework

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.

Managed Code:
The resource, which is within our application domain is, managed code. The resources that

are within domain are faster.

The code, which is developed in .NET framework, is known as managed code. This code is directly executed by CLR with help of managed code execution. Any language that is written in .NET Framework is managed code.

Managed code uses CLR which in turns looks after your applications by managing memory, handling security, allowing cross - language debugging, and so on.

Unmanaged Code:
The code, which is developed outside .NET, Framework is known as unmanaged code.

2(a) What is namespace? Explain the steps involved in creating a namespace and illustrate few common namespaces.
Ans:
Namespace allows to group different entities such as classes, objects and functions under a common name. Start -> All Programs ->Microsoft Visual Studio 2010 -> Microsoft Visual Studio 2010. The visual studio 2010 IDE appears. Select File -> New ->projects on the menu bar. The new project dialog box appears. Select Visual C# -> Windows from the installed templates pane Select the Class Library template from the middle pane Type the name of the namespace and enter the path where it has to be saved and click ok. Common Namespace:
1. System
2. System.Collections
3. System.Data.OLEDB
4. System.Dynamic
5. 5. System.Security

3(a) Write a C# program to demonstrate Boxing and Unboxing.

Ans: **Boxing**
It is defined as the process of explicitly converting a value type into a corresponding reference type by storing the variable in a System.Object.
Consider an example a variable of type short:
short s = 25;
If, during the course of your application, you wish to represent this value type as a reference type, you would “box” the value as follows:
// Box the value into an object reference.
object objShort = s;
Unboxing
It is the process of converting the value held in the object reference back into a corresponding value type on the stack. The unboxing operation begins by verifying that the receiving data type is equivalent to the boxed type, and if so, it copies the value back into a local stack-based variable.
The following unboxing operation works successfully, given that the underlying type of the objShort is indeed a short.
// Unbox the reference back into a corresponding short.
short anotherShort = (short)objShort;
a) Boxing and Unboxing

```
namespace Prg_2a
{
class Program
{
static void box(object obj)
{
Console.WriteLine("Value : " + obj);
}
```

```
static void Main(string[] args)
{
Object o;
int a = 10;
double d = 4.4;
o = a; //boxing integer
Console.WriteLine("Passing integer");
box(a);
Console.WriteLine("Passing Object");
box(o);
int x = (int)o;//Unboxing
Console.WriteLine("");
Console.WriteLine("Unboxing");
Console.WriteLine("a= " + x);
o = d; //boxing double
Console.WriteLine("Passing double");
box(d);
Console.WriteLine("Passing Object");
box(o);
double dd = (double)o; //Unboxing
Console.WriteLine("d= : " + dd);
Console.ReadLine();
}
}
}
```

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

Access Modifier Basics

An access modifier is a keyword of the language that is used to specify the access level of members of a class. C#.net supports the following access modifiers.

Public: When Members of a class are declared as public, then they can be accessed

1. Within the class in which they are declared.
2. Within the derived classes of that class available within the same assembly.
3. Outside the class within the same assembly.
4. Within the derived classes of that class available outside the assembly.
5. Outside the class outside the assembly.

Internal: When Members of a class are declared as internal, then they can be accessed

1. Within the class in which they are declared.
2. Within the derived classes of that class available within the same assembly.
3. Outside the class within the same assembly.

Protected: When Members of a class are declared as protected, then they can beaccessed

1. Within the class in which they are declared.
2. Within the derived classes of that class available within the same assembly.
3. Within the derived classes of that class available outside the assembly.

Protected internal: When Members of a class are declared as protected internal, then theycan be accessed

1. Within the class in which they are declared.
2. Within the derived classes of that class available within the same assembly.
3. Outside the class within the same assembly.
4. Within the derived classes of that class available outside the assembly.

Private: Private members of a class are completely restricted and are accessible only withinthe class in which they are declared.

```
class Employee {
private int i;
double d; // private access by default
}
class A {
protected int x = 123;
}
 class B : A {
 static void Main() {
 A a = new A();
 B b = new B();
 b.x = 10;
}
}
```

4(a) Name and explain the access modifiers for Structs in C#.

Ans: **Access Modifier Basics**

An access modifier is a keyword of the language that is used to specify the access level of members of a class. C#.net supports the following access modifiers.

Public: When Members of a class are declared as public, then they can be accessed

1. Within the class in which they are declared.
2. Outside the structure within the same assembly.
3. Outtside the structure outside the assembly.

Internal: When Members of a structure are declared as internal, then they can be accessed

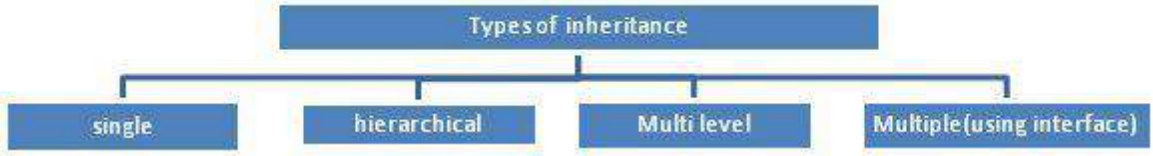
1. Within the structure in which they are declared.
2. Outside the structure within the same assembly.

Protected: Not Allowed

Protected internal: Not Allowed

Private: Private members of a structure are completely restricted and are accessible only within the structure in which they are declared.

4(b) Explain the types of Inheritance in C#. Also discuss the order of execution of constructor in inheritance.
Ans:



5(a) Explain the characteristics of Abstract classes and abstract methods.
Ans: Characteristics of Abstract class:
1. Restricts instantiation, implying that we cannot create object of an abstract class
2. Allows us to define abstract as well as non-abstract members in it.
3. Requires atleast one abstract method
4. Restrict the use of Sealed keyword
5. Possess public access specifier
Characteristics of Abstract methods:
1. Restricts its implementation in an abstract class
2. Allows implementation in a non-abstract derived class
3. Requires declaration in an abstract class only
4. Allows us to override a virtual method
5. Restricts declaration with static and virtual keywords

5(b) Write Short notes on Metadata & Assemblies.
Ans: Metadata is binary information that describes the programs stored in programme Executable file.

When the compilation of code takes place the metadata is inserted in on part of file while the code is converted into Integrated Language.

Metadata contains the following:

Assembly Information: name, version, culture, public key, types of assemblies
Type Information: name, visibility, base class, interface used
Attribute information: that modifies the types & members of class.
Assembly:

An assembly is a logical unit that consists of 4 elements.

Manifest, type metadata, IL code, Resources

Assembly manifest: Every assembly contains data

Assembly name
Version number
Culture
Strong name information
List of all files in the assembly
Type reference information
Information on referenced assemblies.

6(a) Define the following class and explain with program.
Partial Classes , Sealed Classes
Ans:
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. class Container { partial class Nested {

```
void Test() { }

 }
partial class Nested
 {
 void Test2() { }
 }
}
```

The **sealed** modifier prevents other classes from inheriting from it.

```
 sealed class SealedClass {
public int x; public int y;
 }
class SealedTest2 {
 static void Main() {
 SealedClass sc = new SealedClass();
 sc.x = 110; sc.y = 150;
Console.WriteLine("x = {0}, y = {1}", sc.x, sc.y); } }
```

7(a) Explain with example the method of implementing encapsulation using class Properties & Assessors and Mutators.
Ans: Using Property:

```
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.

Using Accessors and Mutators:

```
1.  using system;
2.  public class Department {
3.      private string departname;.....
4.      // Accessor.
5.      public string Get() {
6.          return departname;
7.      }
8.      // Mutator.
9.      public void Set(string a) {
10.         departname = a;
11.     }
12. }
13. Public class program{
14. public static int Main(string[] args) {
15.     Department d = new Department();
16.     d.Set("ELECTRONICS");
17.     Console.WriteLine("The Department is :" + d.Get());
18.     return 0;
19. }
20. }
```

8(a) What is Polymorphism? Explain in detail Compile Time polymorphism and Runtime Polymorphism.
Ans: **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.

9(a) List the difference between Properties and Indexers with example.
Ans:

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

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

Ans: **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.

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();
            }
         }
```

10(a) Discuss static class and static members.

Ans:

A static class is basically the same as a non-static class, but there is one difference: a static class cannot be instantiated. In other words, you cannot use the new keyword to create a variable of the class type. Because there is no instance variable, you access the members of a static class by using the class name itself.

The following list provides the main features of a static class:

- Contains only static members.
- Cannot be instantiated.
- Is sealed.
- Cannot contain Instance Constructors.

A non-static class can contain static methods, fields, properties, or events. The static member is callable on a class even when no instance of the class has been created. The static member is always accessed by the class name, not the instance name. Only one copy of a static member exists, regardless of how many instances of the class are created.

```
public static class TemperatureConverter
{
   public static double CelsiusToFahrenheit(string temperatureCelsius)
   {
      // Convert argument to double for calculations.
      double celsius = Double.Parse(temperatureCelsius);

      // Convert Celsius to Fahrenheit.
      double fahrenheit = (celsius * 9 / 5) + 32;

      return fahrenheit;
   }

   public static double FahrenheitToCelsius(string temperatureFahrenheit)
   {
      // Convert argument to double for calculations.
      double fahrenheit = Double.Parse(temperatureFahrenheit);

      // Convert Fahrenheit to Celsius.
      double celsius = (fahrenheit - 32) * 5 / 9;

      return celsius;
   }
}

class TestTemperatureConverter
{
   static void Main()
   {
      Console.WriteLine("Please select the convertor direction");
      Console.WriteLine("1. From Celsius to Fahrenheit.");
      Console.WriteLine("2. From Fahrenheit to Celsius.");
      Console.Write(":");

      string selection = Console.ReadLine();
      double F, C = 0;

      switch (selection)
      {
         case "1":
```

```
            Console.Write("Please enter the Celsius temperature: ");
            F = TemperatureConverter.CelsiusToFahrenheit(Console.ReadLine());
            Console.WriteLine("Temperature in Fahrenheit: {0:F2}", F);
            break;

        case "2":
            Console.Write("Please enter the Fahrenheit temperature: ");
            C = TemperatureConverter.FahrenheitToCelsius(Console.ReadLine());
            Console.WriteLine("Temperature in Celsius: {0:F2}", C);
            break;

        default:
            Console.WriteLine("Please select a convertor.");
            break;
    }

    // Keep the console window open in debug mode.
    Console.WriteLine("Press any key to exit.");
    Console.ReadKey();
  }
}
/* Example Output:
   Please select the convertor direction
   1. From Celsius to Fahrenheit.
   2. From Fahrenheit to Celsius.
   :2
   Please enter the Fahrenheit temperature: 20
   Temperature in Celsius: -6.67
   Press any key to exit.
  */
```

10(b) Write a program to add two complex number using operator overloading.

Ans:

```
class Prg3
{
class Complex
{
private int x, y;
public Complex() { }
public Complex(int i, int j)
{
x = i;
y = j;
}
public void ShowXY()
{
Console.WriteLine(Convert.ToString(x) + " " + Convert.ToString(y));
}
public static Complex operator -(Complex c)
{
Complex temp = new Complex();
temp.x = -c.x;
temp.y = -c.y;
return temp;
}
public static Complex operator +(Complex c1, Complex c2)
{
Complex temp = new Complex();
temp.x = c1.x + c2.x;
temp.y = c1.y + c2.y;
```

```
return temp;
}
}
static void Main(string[] args)
{
Complex c1 = new Complex(10, 20);
Complex c2 = new Complex(30, 40);
Complex c3, c4 = new Complex();
Console.Write("Complex Number1 :");
c1.ShowXY();
Console.Write ("Complex Number2 :");
c2.ShowXY(); c3 = -c1;
Console.Write("Changing the sign of Complex Number1 :");
c3.ShowXY();
c4 = c1 + c2;
Console.Write("Addition of two complex Numbers 1 and 2:");
c4.ShowXY();
Console.ReadLine();
}
  }
```