CMR
INSTITUTE OF
TECHNOLOGY

Internal Assesment Test – II(Answer Key)

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

Branch: MCA Sem: V

		Marks	OBE CO	OBE RBT
1(a)	**What is namespace? Explain the steps involved in creating a namespace and illustrate few common namespaces.**	[10]	CO1	L4

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. System.Security

		Marks	OBE CO	OBE RBT
2(a)	**Define partial class and explain with program.**	[05]	CO2	L2 L5

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() { }
    }
}
```


(b) **Define sealed class and explain with program.** [05] CO1 L2 L5

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);
    }
}
```

3(a) **Name and explain the access modifiers for structs in C#** [05] CO1 L5

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

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

protected
The type or member can be accessed only by code in the same class or struct, or in a class that is derived from that class.

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

class Employee
{
 private int i;
```

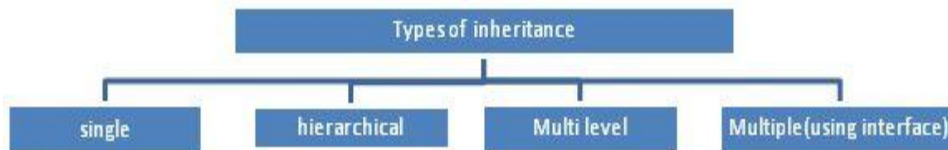
double d;   // private access by default
}

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

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

   b.x = 10;
 }
}
```

(b) **Explain the types of inheritance in C#**	[05]	CO1	L5



4(a) **What is Polymorphism? Explain in detail Compile Time polymorphism and Runtime Polymorphism**	[10]	CO1	L1 L5

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.

5(a) **Explain the characteristics of abstract classes and abstract methods** [10] CO1 L4

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

6(a) **Illustrate boxing and unboxing using C# program** [10] CO1 L3

Boxing is used to store value types in the garbage-collected heap. Boxing is an implicit conversion of a value type to the type **object** or to any interface type implemented by this value type. Boxing a value type allocates an object instance on the heap and copies the value into the new object.
Consider the following declaration of a value-type variable: int i = 123;\

Unboxing is an explicit conversion from the type **object** to a value type or from an interface type to a value type that implements the interface.

An unboxing operation consists of:

- Checking the object instance to make sure that it is a boxed value of the given value type.
- Copying the value from the instance into the value-type variable.

```
int i = 123;      // a value type
object o = i;     // boxing
int j = (int)o;   // unboxing
```

7(a) **Explain the following operators: 1. Using?? 2. Using the :: 3. is and as operator**	[10]	CO1	L4

The ?? operator is called the null-coalescing operator. It returns the left-hand operand if the operand is not null; otherwise it returns the right hand operand.

```
class NullCoalesce
{
    static int? GetNullableInt()
    {
        return null;
    }

    static string GetStringValue()
    {
        return null;
    }

    static void Main()
    {
        int? x = null;

        // Set y to the value of x if x is NOT null; otherwise,
        // if x = null, set y to -1.
        int y = x ?? -1;

        // Assign i to return value of the method if the method's
result
        // is NOT null; otherwise, if the result is null, set i to
the
        // default value of int.
        int i = GetNullableInt() ?? default(int);

        string s = GetStringValue();
        // Display the value of s if s is NOT null; otherwise,
        // display the string "Unspecified".
        Console.WriteLine(s ?? "Unspecified");
    }
}
```

The :: operator is used to execute a parent class method from within a subclass method.

```
namespace NamespaceA{
    int x;
    class ClassA {
    public:
        int x;
    };
}

int main() {

    // A namespace name used to disambiguate
    NamespaceA::x = 1;

    // A class name used to disambiguate
    NamespaceA::ClassA a1;
    a1.x = 2;

}
```

is and as operator

The is operator in C# is used to check the object type and it returns a bool value: **true** if the object is the same type and **false** if not.

```
namespace IsAndAsOperators
{
  // Sample Student Class
  class Student
  {
    public int stuNo { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }

  }
  // Sample Employee Class
  class Employee
  {
    public int EmpNo { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public double Salary { get; set; }

  }
  class Program
  {

    static void Main(string[] args)
    {
      Student stuObj = new Student();
      stuObj.stuNo = 1;
      stuObj.Name = "Siva";
      stuObj.Age = 15;

      Employee EMPobj=new Employee();
```

```
            EMPobj.EmpNo=20;
            EMPobj.Name="Rajesh";
            EMPobj.Salary=100000;
            EMPobj.Age=25;

            // Is operator

            // Check Employee EMPobj is Student Type

            bool isStudent = (EMPobj is Student);
            System.Console.WriteLine("Empobj is a Student ?: {0}",
isStudent.ToString());

            // Check Student stiObj is Student Typoe
            isStudent = (stuObj is Student);
            System.Console.WriteLine("Stuobj is a Student ?: {0}",
isStudent.ToString());

            stuObj = null;
            // Check  null object Type
            isStudent = (stuObj is Student);
            System.Console.WriteLine("Stuobj(null) is a Student ?: {0}",
isStudent.ToString());
            System.Console.ReadLine();
        }
    }
```

		[Marks]	[CO]	[RBT]
8(a)	**Explain Checked and unchecked statement**	[05]	CO2	L4

C# statements can execute in either checked or unchecked context. In a checked context, arithmetic overflow raises an exception. In an unchecked context, arithmetic overflow is ignored and the result is truncated.

- checked Specify checked context.
- unchecked Specify unchecked context.

If neither **checked** nor **unchecked** is specified, the default context depends on external factors such as compiler options.

The following operations are affected by the overflow checking:

- Expressions using the following predefined operators on integral types:
  ++ -- - (unary) **+ - * /**
- Explicit numeric conversions between integral types.

The /checked compiler option lets you specify checked or unchecked context for all integer arithmetic statements that are not explicitly in the scope of a **checked** or **unchecked** keyword.

```
checked
{
    int i3 = 2147483647 + ten;
    Console.WriteLine(i3);
}

unchecked
{
    int1 = 2147483647 + 10;
```

```
}
int1 = unchecked(ConstantMax + 10);
```

(b) **Explain the following statements. 1. try 2. catch 3. Finally** [05] CO2 L4

Exceptions provide a way to transfer control from one part of a program to another. C# exception handling is built upon four keywords: try,catch, finally, and throw. try: A try block identifies a block of code for which particular exceptions is activated. It is followed by one or morecatch blocks.

```
using System;

class Exercise
{
	static void Main()
	{
		double Number1, Number2;
		double Result = 0.00;
		char Operator;

	Console.WriteLine("This program allows you to perform an operation on two
numbers");

		try
		{
			Console.WriteLine("To proceed, enter");

			Console.Write("First Number:  ");
			Number1 = double.Parse(Console.ReadLine());

			Console.Write("An Operator (+, -, * or /): ");
			Operator = char.Parse(Console.ReadLine());
			if( Operator != '+' && Operator != '-' &&
				Operator != '*' && Operator != '/' )
				throw new Exception(Operator.ToString());

			Console.Write("Second Number: ");
			Number2 = double.Parse(Console.ReadLine());

			if( Operator == '/' )
				if( Number2 == 0 )
			throw new DivideByZeroException("Division by zero is not
allowed");

			Result = Calculator(Number1, Number2, Operator);
		Console.WriteLine("\n{0} {1} {2} = {3}", Number1, Operator,
Number2, Result);
		}
		catch(FormatException)
		{
			Console.WriteLine("The number you typed is not valid");
		}
		catch(DivideByZeroException ex)
```

```
        {
                Console.WriteLine(ex.Message);
        }
        catch(Exception ex)
        {
        Console.WriteLine("\nOperation Error: {0} is not a valid operator",
ex.Message);
        }
    }

    static double Calculator(double Value1, double Value2, char Symbol)
    {
        double Result = 0.00;

        switch(Symbol)
        {
            case '+':
                Result = Value1 + Value2;
                break;

            case '-':
                Result = Value1 - Value2;
                break;

            case '*':
                Result = Value1 * Value2;
                break;

            case '/':
                Result = Value1 / Value2;
                break;
        }

        return Result;
    }
}
```

Course Outcomes		PO1	PO2	PO3	PO4	PO5	PO6	PO7	PO8
CO1:	Understand C# and client-server concepts using .Net Frame Work Components.	3	2	1	-	-	-	-	1
CO2:	Apply delegates, event and exception handling to incorporate with ASP, Win	3	3	3	1	-	-	1	2

	Form, and ADO.NET.								
CO3:	Analyze the use of .Net Components depending on the problem statement.	1	3	1	-	-	-	-	1
CO4:	Implement & develop a web based and windows based application with Database connectivity.	3	3	3	2	-	1	2	2

Cognitive level	KEYWORDS
L1	List, define, tell, describe, identify, show, label, collect, examine, tabulate, quote, name, who, when, where, etc.
L2	summarize, describe, interpret, contrast, predict, associate, distinguish, estimate, differentiate, discuss, extend
L3	Apply, demonstrate, calculate, complete, illustrate, show, solve, examine, modify, relate, change, classify, experiment, discover.
L4	Analyze, separate, order, explain, connect, classify, arrange, divide, compare, select, explain, infer.
L5	Assess, decide, rank, grade, test, measure, recommend, convince, select, judge, explain, discriminate, support, conclude, compare, summarize.

PO1 –Apply *knowledge*; PO2- *Problem analysis*; PO3- *Design/development of solutions*; PO4 – Team work ; PO5 – *Ethics* ; PO6 -Communication; PO7- *Business Solution*; PO8 – Life-long learning ;