Ajay Kumar Garg Engineering College, Ghaziabad

Department of MCA

Sessional Test-2 (Model Solution)

Course:	MCA	Semester:	V
Session:	2017-18	Section:	MCA-1&2
Subject:	Dot Net Framework ~~and C#~~	Sub Code:	NMCA-513
Max Marks:	50	Time:	2 hour

SECTION-A

Q. 1. Differentiate between Const and read-only.

Ans. 'Const' :
- cant be static
- value is evaluated at compile time
- initialized at declaration only.

read-only:
- can be either instance-level or static
- value is evaluated at run time.
- can be initialized in declaration or by code in the constructor.

Q 2. What are the uses of Indexers?

Ans. Indexers as a property is a smart field. As an smart array it allows an object be indexed thrugh get and set accessor methods. It enables to easily index into an object for purpose of setting or retriening values.

Q 3. What is namespace?

Ans. Name spaces are a meas for organizing types. They organize large code projects. The namespace keyword is used to declare a scope that contains a set of related object.

Q.4. What is Boxing and unboxing?

Ans. Boxing is the process of converting a value type to the type object or to any interface type implemented by this value type. Unboxing extracts the value type from the object. Boxing is implicit, unboxing is explicit.

Q.5. Mention any two properties of checkbox control.

Ans. A checkBox control allows users to select a single or multiple options from a list of options. Some properties are:

- Font
- Checked

SECTION - B

Q.6. What are delegates? Give an example in C# to implement multicast delegates.

Ans. A delegate in c# is similar to a function pointer in c or c++. The delegate allows programmer to encapsulate a refesence to a method inside a delegate object delegate are object-oriented, type-safe and secure.

A delegate definition defines a type that encapsulates a method with a particular set of arguments and return type. For static methods, a delegate object encapsulates the method to be called. For instance method, a delegate object encapsulates both an instance and a method on the instance.

Syntax for delegate declaration is:

Delegate <return type> <delegate name> < parameter list>;

e.g.

public delegate int Mydelegate (String a);

Program:

```
delegate void Del (String s);
class TestClass {
Static void Hello (String s) {
 System. Console. WriteLine (" Hello, {0}! ", s);
}
Static void Goodbye (String s) {
System. Console. WriteLine (" Goodbye, {0}! ", s);
}
Static void Main () {
 Del a, b, c, d;
 a = Hello;
 b = Goodbye;
 c = a + b
 d = c - a;
 System. Console. WriteLine (" Invoking a: ");
 a ("A");
 System. Console. WriteLine ("Invoking b: ");
 b ("B");
 System. Console. WriteLine (" Invoking c: ");
 c ("(");
 System. Console. WriteLine (" Invoking d: ");
```

cl ("A"); } }

87. What is operator overloading? write a code in C# for overloading '+' operator to add two matrix objects.

Ans. C# gives you the ability, via operator overloading to add the standard mathematical operators to a class so that you can write more intuitive code by using those operators. Operator overloading permits user-defined operator implementations to be specific for operations where one or both of the operands are of user-defined class or struct type.

Program:

```
Using System;
namespace ConsoleApplication1
 {
   class Program
   {
     static void Main (String[] args)
     {
        Matrix mat1 = new Matrix();
        Matrix mat2 = new Matrix();
        Console.WriteLine ("Matrix 1:");
        mat1.PrintMatrix();
        Matrix mat3 = mat1 + mat2;
        Console.WriteLine();
        Console.WriteLine ("Matrix 1 + Matrix 2 =");
        mat3.PrintMatrix();
        Console.ReadLine();
   }}
```

```
class Matrix
{
   public const int DimSize = 3;
   Private int [,] m_matrix = new int [DimSize, DimSize];
   public static Matrix operator + ( Matrix mat1, Matrix mat2)
    {
       Matrix new_Matrix = new Matrix(0);
       for (int x = 0; x < DimSize; x++)
         for (int y = 0; y < DimSize; y++)
            new_Matrix.m_matrix [x, y] = mat1.m_matrix[x, y]
                                        + mat2.m_matrix[x, y];
       return new_Matrix;
    }
   public  Matrix()
    {
       Console.WriteLine ("Enter 9 Elements:");
       for (int x = 0; x < DimSize; x++)
         for (int y = 0; y < DimSize; y++)
            m_matrix [x, y] = int.parse (Console.ReadLine());
    }
  public Matrix (int v)
    {
      for ( int x = 0; x < DimSize; x++)
        for( int y = 0; y < DimSize; y++)
           m_matrix [x, y] = v;
    }
Public  void  PrintMatrix()
    {
      Console.WriteLine();
```

```
for (int x=0; x<DimSize; x++)
  {
    Console.Write("[");
    for (int y=0; y<DimSize; y++)
      {
        Console.Write(m_matrix[x,y]);
        if ((y+1)%2)<3)
            Console.Write(", ");
      }
      Console.WriteLine("]");
  }
    Console.WriteLine();
  }
}
```

Q 8 What is collections in C#? Discuss with example.

Ans: For large variety of applications Dot Net Framework provide prepacked data structure classes. These claues are known as collection classes - They store collections of data. Each instance of one of these classes is a collection of items. Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index etc. The System.Collections namespace has many classes for the individual purpose.

- ArrayList
- BitArray

- Stack
- Queue
- Comparer
- Hash Table
- SortedList

ArrayList - ArrayList is a dynamic array; it will increase the size of the storage location as required. It stores the value as object. The allocation of the ArrayList can be achieved through the TrimtoSize property.

```
ArrayList oArrayList = new ArrayList()
oArrayList.Add("Aditya");
oArrayList.Add("MCA");
oArrayList.Add("027");
oArrayList.Remove("027");
oArrayList.Sort();
```

Q 9. Describe the use of properties with example.

Ans. Properties are attributes associated with objects. A property is a member that provides a flexible mechanism to read, write, or compute the value of a privatefield. Properties can be used as if they are public members, but they are actually special methods called accessors. This enables data to be accessed easily and still helps promote the safty and flexibility of methods. Properties

enables a class to expose a public way of getting and setting values, while hiding implementation or verification code. A get property accessor is used to return the property value, and a set property accessor is used to assign a new value. The value keyword is used to define the value being assigned by the set accessor. Properties can be read-write, read-only, write-only.

```
class TimePeriod
{
    private double seconds;

    public double Hours
    {
        get { return seconds/3600; }
        set { if value <0 || value >24)
                throw new ArgumentOutOfRangeException(
            "must be between 0 to 24");
            seconds = value *3600;
        }
    }}
```

Q 10 What is an event? Give an example

Ans. Events are user action such as keypress. Applications need to respond to events when they occur. The events are declared and raised in a class and associated with the event handlers using delegates within the same class or some other class.

The class containing event is used to publish the event. This is called publisher class. Some other class that accepts this event is called Subscriber class.

To declare an event inside a class, first a delegate type for the event must be declared.

e.g

```
public delegate string MyDel(string str);
```

Next the event itself is declared, using the event keyword:

```
even MyDel MyEvent;
```

Program

```
using System;
namespace SampleApp {
  public delegate string MyDel(string str);
  class EventProgram
  {
    event MyDel MyEvent;
    public EventProgram() {
        this.MyEvent += new MyDel(this.welcomeUser);
    }
    public string welcomeUser(string username) {
        return "welcome" + username;
    }
  Static void Main(string[] args) {
     Event Prygrm obj1 = new Event Prygrm();
```

```
String result = obj1.MyEvent ("Tutorials Point");
Console.WriteLine (result);
  }
 }
}
```

SECTION-C

Q.11. Explain Multithreading in C#. How can we create and start Thread? How can we set the priorities of threads, illustrate through an example?

Ans. Thread is a lightweight process. With the help of threads we can increase the responsetime of the application. To use multithreading we have to use Threading namespace. In multithreading a single program can create multiple threads ~~with which~~ and these threads can perform concurrently.

Implementing a ThreadStart delegate

```
using System.Threading
public class ThreadExample {
    public static void RunTo () {
        for (int i=0; i<1000; i++)
        { console.Write ("x");
          Thread.Sleep (10);
        }
}
```

Creating Thread with delegate to method RunTo and starting it

```
public static void Main()
{
    Thread t1 = new Thread(new ThreadStart(
                                RunTo));
    t1.Start();
}.
```

The thread priority can be assigned to give preference to a particular thread. The thread with higher priority will get a CPU cycles earlier. The following is the priorities for Thread

```
public enum ThreadPriority {
    Highest,
    AboveNormal,
    Normal,
    BelowNormal,
    Lowest,
}
```

Example

```
public class MyThread {
    public void Thread1() {
        for (int i=0; i<10; i++) {
            Thread thr = Thread.CurrentThread;
            Console.WriteLine(thr.Name + "=" + i);
```

```
            Thread.Sleep(1);
        }
    }
}

public class Myclass {
    Public static void Main(){
        Console.WriteLine("Before Start Thread");
        MyThread t1 = new MyThread();
        MyThread t2 = new MyThread();

        Thread tid1 = new Thread(new ThreadStart(t1.Thread1));
        Thread tid2 = new Thread(new ThreadStart(t2.Thread1));

        tid1.Priority = ThreadPriority.Highest;
        tid2.Priority = ThreadPriority.Lowest;
        try {
            tid1.Start();
            tid2.Start();
        }
        Catch (ThreadException e) {__}
    }
}
```

Q.12. Explain the exception handling mechanism in C#. Write user defined exception for overflow and Stack empty exception in a class created for Stack data Structure.

Ans. Exceptions are runtime errors. These exceptions cause the abrupt termination of execution of application.

The C# language exception handling features help a developer to deal with any unexpected or exceptional situations that occur when a program is running. Exception handling uses try, catch and finally keywords to try actions that may not succeed, to handle failure when developer decide that it is reasonable to do so, and to clean up resources afterward. Exceptions can be generated by CLR and or application libraries or application code. Exceptions are created by using the throw keyword.

try - A try block identifies a block of code for which particular exception is activated. It is followed by one or more catch blocks.

Catch - A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.

finally - The finally block is used to execute a given set of statements whether an exception is thrown or not.

throw - A program throws an exception when a problem shows up. This is done using throw keyword.

```
Using System;
namespace UserException.
{
   class AStack . {
      private object[] elements;
      Private int nextPush;
      Private int MaxSize;
      Public AStack (int MaxSize){
            elements = new Object [MaxSize];
            nextPush = 0;
            this.MaxSize = MaxSize;
         }
      Public void Push (object el){
         if (! Full) {
            elements [nextPush] = el;
            nextPush++;
         }
         else throw new StackOverflowException();
      }
      Public void Pop() {
         if (! Empty)
            nextPuss --;
         else throw new StackUnderflowException();
      }
      Public object Top {
         get {
            if (! Empty)
               return elements [nextPush-1];
            else throw new StackUnderflowException();
         }
      }
      Public bool Full {
         get { return nextPush >= MaxSize; }
      }
      Public bool Empty {
         get { return nextPush == 0; } }
```

```
public int Size {
    get { return nextPush; } }
public override String ToString() {
    String res = "Stack:";
    for (int i=0; i< nextPush; i++) {
        res += elements[i] + " ";
      }
    return res; }
Public void ToggleTop() {
    if (size >=2)
      { object topB11 = Top; Pop();
        Object topB12 = Top; Pop();
        Object Push(topB11); Push(topB12);
      }
    else throw new StackUnderflowException();
  }
}
Public class StackOverflowException : Exception
    { }
public class StackOverflowException : Exception
    { }
}
public class Test
  {
    Public Static void Main()
      {
        AStack S = new AStack(3);
        try {
            S.push(5); S.Push(6); S.Push(7);
            S.Pop(); S.Pop(); S.Pop(); }
    Catch (StackOverflowException)
      { Console.WriteLine("stack overflow"); }
```

```
Catch (StackUnderflowException)
    {
    Console.WriteLine ("Stack Overflow");
    }
}
```