USN										

CELEBRATING 25 YEARS CMRIT CMR INSTITUTE OF TECHNOLOGY, BENGALURU. ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 1 – September 2018

Sub:	Dotnet Framework For Application Development	Sub Code:	17/15CS564	Branch:	CSE
Date:	15/10/19	Duration: 90 min's	Max Marks: 50	Sem / Sec:	5/A,B,C

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1 (a)	Explain the concept of params array with an example program.	[3+3]	CO3	L4
(b)	What are Abstract Classes? Describe the concept with an example.	[2+2]	CO3	L2
2 (a)	Describe how to manage system resources by using a Garbage Collector	[06]	CO3	L1
(b)	Examine Sealed Classes and Sealed Methods in brief	[2+2]	CO3	L1
3 (a)	Discuss the properties in C#.	[06]	CO4	L2
(b)	Compare Indexers and Arrays with an example	[04]	CO4	L4
4 (a)	Define Interface? Explain how interfaces are created with an example.	[2+8]	CO3	L1
5 (a)	Write a C# program to implement QUEUE operations Insert and Delete using Generics concepts.	[10]	CO4	L1
6 (a)	Define inheritance? Explain types of inheritance with an example.	[3+7]	CO3	L1
7 (a)	What are collection classes? Explain in detail about collection classes.	[2+8]	CO4	L2

CELEBRATING 25 YEARS
CMRIT
CMR INSTITUTE OF TECHNOLOGY, BENGALURU.
ACCREDITED WITH A+ GRADE BY NAAC

Internal Assessment Test 2 – Oct 2019

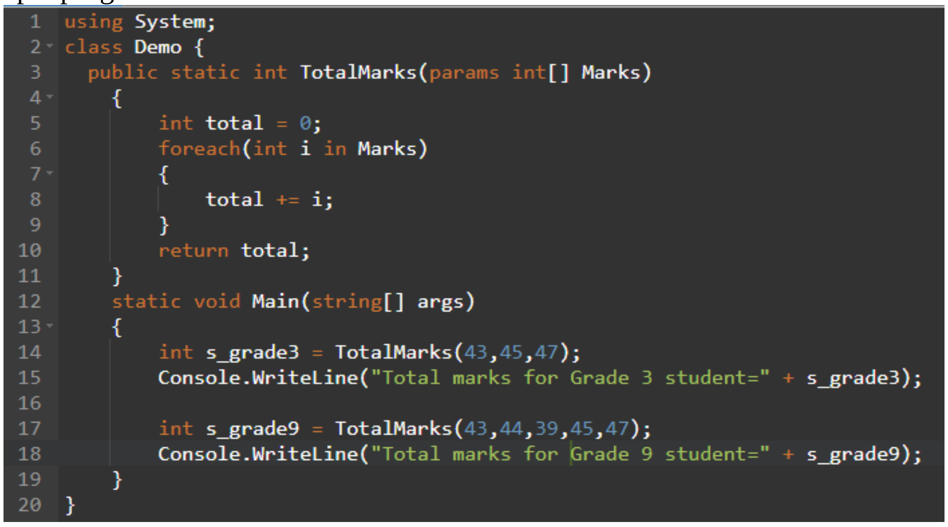
Sub:	Dot Net Framework for application development				Sub Code:	15CS564	Branch:	CSE	
Date:	15/10/19	Duration:	90 mins	Max Marks:	50	Sem / Sec:	V A/B/C		OBE

	Answer any FIVE FULL Questions	MARK S	CO	RB T
1 (a)	**Explain the concept of params array with an example program.**	[06]		

Answer:

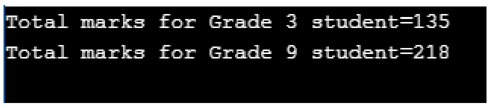
- The "params" keyword in C# allows a method to accept a variable number of arguments. C# params works as an array of objects.
- By using params keyword in a method argument definition, we can pass a number of arguments.
- Instead of using various overloaded methods to pass multiple values, we can simply create an array and pass it as an argument or a comma separated list of values.
- For example, a Student class with a method, TotalMarks that returns the sum of all marks. **Each grade may have a different number of subjects and their respective marks**. The 3rd grade may have 3 subjects only. The 8th grade may have 4 subjects while a 9th grade may have 5 subjects. We can use params in this case and pass 3, 4, and 5 comma separated values.

Example program:



```
using System;
class Demo {
  public static int TotalMarks(params int[] Marks)
    {
        int total = 0;
        foreach(int i in Marks)
        {
            total += i;
        }
        return total;
    }
    static void Main(string[] args)
    {
        int s_grade3 = TotalMarks(43,45,47);
        Console.WriteLine("Total marks for Grade 3 student=" + s_grade3);

        int s_grade9 = TotalMarks(43,44,39,45,47);
        Console.WriteLine("Total marks for Grade 9 student=" + s_grade9);
    }
}
```

Output:



```
Total marks for Grade 3 student=135
Total marks for Grade 9 student=218
```

(b)	**What are Abstract Classes? Describe the concept with an example.**	[04]		

Answer:

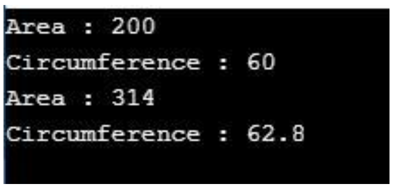
- An abstract class is a special type of class that cannot be instantiated and acts as a base class for other classes.
- Abstract class members marked as abstract must be implemented by derived classes.
- The purpose of an abstract class is to provide a common definition of the base class that multiple derived classes can share and can be used only as a base class and never want to create the object of this class.
- Any class can be converted into an abstract class by adding the abstract modifier to it.

Example program:

```
using System;
abstract class Shape1
   {
      public abstract float Area();
      public abstract float Circumference();
   }

   class Rectangle1 : Shape1
   {
      float L=10;
      float B=20;
      public override float Area()
      {
         return L * B;
      }

      public override float Circumference()
      {
         return 2 * (L + B);
      }
   }

   class Circle1 : Shape1
   {

      float R=10;
      public override float Area()
      {
         return 3.14F * R * R;
      }
      public override float Circumference()
      {
         return 2 * 3.14F * R;
      }
   }

   class Program
   {
      public static void Calculate(Shape1 S)
      {

         Console.WriteLine("Area : {0}", S.Area());
         Console.WriteLine("Circumference : {0}", S.Circumference());
      }
      public static void Main(string[] args)
      {
         Rectangle1 R = new Rectangle1();
         Calculate(R);
         Circle1 C = new Circle1();
```

```
            Calculate(C);
        }
    }
```

Output:



```
Area : 200
Circumference : 60
Area : 314
Circumference : 62.8
```

2 (a) **Describe how to manage system resources by using a Garbage Collector.** [06]

The garbage collector (GC) manages the allocation and release of memory. The garbage collector serves as an automatic memory manager.

- You do not need to know how to allocate and release memory or manage the lifetime of the objects that use that memory.
- An allocation is made any time you declare an object with a "new" keyword or a value type is boxed. CLR (*common language runtime*) allocates memory for the object from heap.
- When there isn't enough memory to allocate an object, the GC must collect and dispose of garbage memory to make memory available for new allocations.
- This process is known as garbage collection.

GC works on a block of memory to store objects called *managed heap*.
When garbage collection process is put in motion, it checks for dead objects and the objects which are no longer used. Then, it compacts the space of live object and tries to free more memory.
Heap is managed by different '*Generations*':

- **0 Generation (Zero):** This generation holds short-lived objects, e.g., Temporary objects. GC initiates garbage collection process frequently in this generation.
- **1 Generation (One):** This generation is the buffer between short-lived and long-lived objects.
- **2 Generation (Two):** This generation holds long-lived objects that needs to be persisted for a certain amount of time like a static and global variable,
- Objects which are not collected in generation Zero, are then moved to generation 1, such objects are known as *survivors*, similarly objects which are not collected in generation One, are then moved to generation 2 and from there onwards objects remain in the same generation.

There are *no specific timings* for GC to get triggered. GC automatically starts operation on the following conditions:

- When virtual memory is running out of space.
- When allocated memory is suppressed acceptable threshold (when GC found if the survival rate (living objects) is high, then it increases the threshold allocation).
- When we call **GC.Collect()** method explicitly, as GC runs continuously, we actually do not need to call this method.

The GC class controls the garbage collector of the system. Some of the methods in the GC class are given as follows:

1. **GC.GetGeneration() Method :** This method returns the generation number of the target object. It requires a single parameter i.e. the target object for which the generation number is required.

2. **GC.GetTotalMemory() Method :** This method returns the number of bytes that are allocated in the system. It requires a single boolean parameter where true means that the method waits for the occurrence of garbage collection before

returning and false means the opposite.

3. **GC.Collect() Method :** Garbage collection can be forced in the system using the *GC.Collect() method*. This method requires a single parameter i.e. number of the oldest generation for which garbage collection occurs.

2 (b) **Examine Sealed Classes and Sealed Methods in brief** [04]

Answer:

- Sealed classes are used to restrict the inheritance feature of object oriented programming.
- Once a class is defined as a sealed class, this class cannot be inherited.
- In C#, the sealed modifier is used to declare a class as sealed.
- If a class is derived from a sealed class, compiler throws an error.

- When you define new methods or properties in a class, you can prevent deriving classes from overriding them by not declaring them as virtual.
- It is an error to use the abstract modifier with a sealed class, because an abstract class must be inherited by a class that provides an implementation of the abstract methods or properties.
- When applied to a method or property, the sealed modifier must always be used with override.
- Because structs are implicitly sealed, they cannot be inherited.

```
sealed class SealedClass
{
   public int x;
   public int y;
}

class SealedTest2
{
   static void Main()
   {
      var sc = new SealedClass();
      sc.x = 110;
      sc.y = 150;
      Console.WriteLine($"x = {sc.x}, y = {sc.y}");
   }
}

// Output: x = 110, y = 150
```

3 (a) **Discuss the properties in C#.** [06]

Answer:

- Properties are the special type of class members that provides a flexible mechanism to read, write, or compute the value of a private field.
- Properties can be used as if they are public data members, but they are actually special methods called ***accessors***. Example: get, set
- This enables data to be accessed easily and help to promote the flexibility and safety of methods.

Syntax:

```
<access_modifier> <return_type> <property_name>
{
        get { // body }
        set { // body }
}
```

- **Read and Write Properties:** When property contains both get and set methods.
- **Read-Only Properties:** When property contains only get method.
- **Write Only Properties:** When property contains only set method.

Example for get accessor:

```
class Student {

    // Declare roll_no field
    private int roll_no;

    // Declare roll_no property
    public int Roll_no
     {

       get
         {
           return roll_no;
         }

       set
         {
           roll_no = value;
         }
    }
}
```

Example for set accessor:

```
  class Student {
          // Declare roll_no field
          private int roll_no;

          // Declare roll_no property
          public int Roll_no
          {

             get
               {
                return roll_no;
               }

          }
     }
```

4 (a)	**Define interface. Explain how interfaces are created with an example**
	
	Answer:
	• An interface contains definitions for a group of related functionalities that a class or a struct can implement. Interfaces specify what a class must do and not how.
	• Like a class, ***Interface*** can have methods, properties, events, and indexers as its members.
	• Interfaces will contain only the declaration of the members.
	• The implementation of interface's members will be given by class who implements the interface implicitly or explicitly.
	• By default, all the members of Interface are public and abstract.
	• The interface is defined by using the keyword '***interface***'.
	• Interface cannot contain fields because they represent a particular implementation of data.
	• It is used to provide total abstraction.
	
	Syntax to declare interface
	
	interface <interface_name >
	
	{
	
	        // declare Events
	
	        // declare indexers
	
	        // declare methods
	
	        // declare properties
	
	}
		[10]		

Syntax to inherit interface

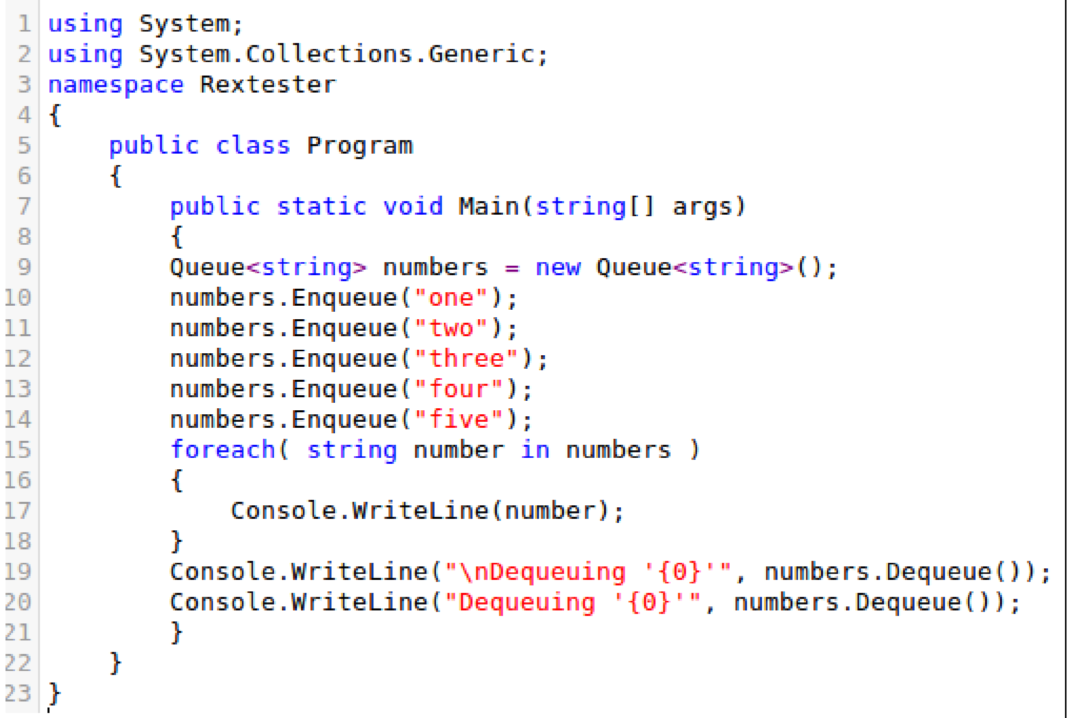
class class_name : interface_name

Example program:

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

namespace InterfaceApplication {

  public interface ITransactions {
          // interface members
          void showTransaction();
          double getAmount();
  }
  public class Transaction : ITransactions {
          private string tCode;
          private string date;
          private double amount;

          public Transaction() {
          tCode = " ";
          date = " ";
          amount = 0.0;
          }
          public Transaction(string c, string d, double a) {
          tCode = c;
          date = d;
          amount = a;
          }
          public double getAmount() {
          return amount;
          }
          public void showTransaction() {
          Console.WriteLine("Transaction: {0}", tCode);
          Console.WriteLine("Date: {0}", date);
          Console.WriteLine("Amount: {0}", getAmount());
          }
  }
  class Tester {

          static void Main(string[] args) {
          Transaction t1 = new Transaction("001", "8/10/2012", 78900.00);
          Transaction t2 = new Transaction("002", "9/10/2012", 451900.00);

          t1.showTransaction();
          t2.showTransaction();
          Console.ReadKey();
          }
  }
}
```

5 (a)	**Write a C# program to implement QUEUE operations Insert and Delete using Generics concepts.**	[10]		

Answer:



```
using System;
using System.Collections.Generic;
namespace Rextester
{
    public class Program
    {
        public static void Main(string[] args)
        {
        Queue<string> numbers = new Queue<string>();
        numbers.Enqueue("one");
        numbers.Enqueue("two");
        numbers.Enqueue("three");
        numbers.Enqueue("four");
        numbers.Enqueue("five");
        foreach( string number in numbers )
        {
            Console.WriteLine(number);
        }
        Console.WriteLine("\nDequeuing '{0}'", numbers.Dequeue());
        Console.WriteLine("Dequeuing '{0}'", numbers.Dequeue());
        }
    }
}
```

6 (a) **Define inheritance. Explain types of inheritance with an example.** [10]

Answer:

- Inheritance is a feature of object-oriented programming languages that allows you to define a base class that provides specific functionality (data and behavior) and to define derived classes that either inherit or override that functionality.
- The class whose members are inherited is called the base class.
- The class that inherits the members of the base class is called the derived class.
- A class can only inherit from a single class.
- Inheritance is transitive. For example, type D can inherit from type C, which inherits from type B, which inherits from the base class type A.
- As a result, the members of type A are available to type D.

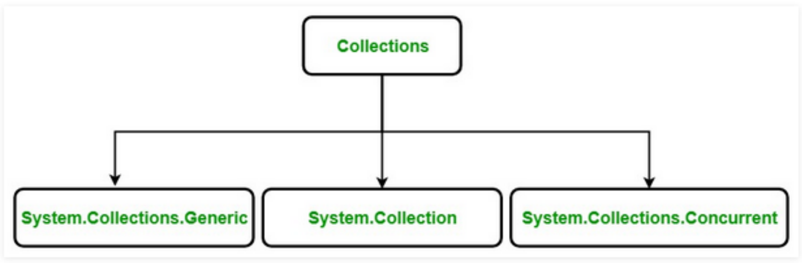
Types of inheritance:

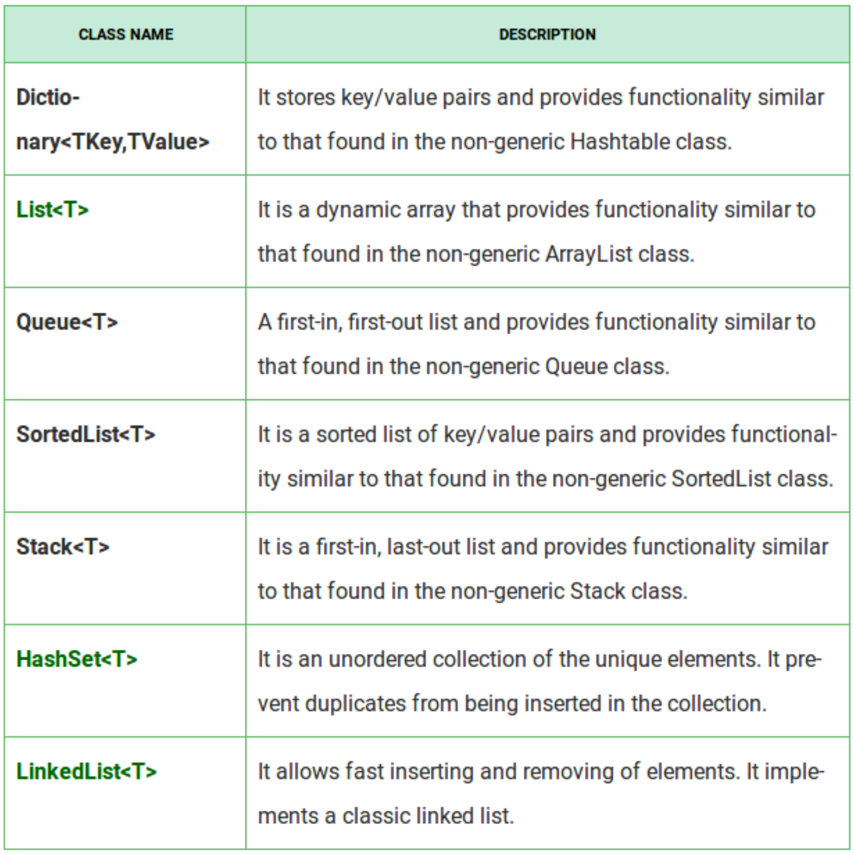
1. Single inheritance
2. Multilevel inheritance
3. Hierarchical inheritance
4. Multiple inheritance
5. Hybrid inheritance

7 (a) **What are collection classes? Explain in detail about collection classes.** [10]

Answer:

- Collections standardize the way of which the objects are handled by your program.
- It contains a set of classes to contain elements in a generalized manner.
- With the help of collections, the user can perform several operations on objects like the store, update, delete, retrieve, search, sort etc.





CLASS NAME	DESCRIPTION
Dictio-nary<TKey,TValue>	It stores key/value pairs and provides functionality similar to that found in the non-generic Hashtable class.
List<T>	It is a dynamic array that provides functionality similar to that found in the non-generic ArrayList class.
Queue<T>	A first-in, first-out list and provides functionality similar to that found in the non-generic Queue class.
SortedList<T>	It is a sorted list of key/value pairs and provides functional-ity similar to that found in the non-generic SortedList class.
Stack<T>	It is a first-in, last-out list and provides functionality similar to that found in the non-generic Stack class.
HashSet<T>	It is an unordered collection of the unique elements. It pre-vent duplicates from being inserted in the collection.
LinkedList<T>	It allows fast inserting and removing of elements. It imple-ments a classic linked list.