Introduction to Dot Net Framework for Application Development.

17CS564

1. a) What are generics? What are the advantages of Generics?

Ans:

- Arrays are strong type-:
  - no need of boxing and unboxing.(Advantage)

  - Fixed Length(Disadvantage)

- Array list and hash tables:
  -Resizable(Advantage)

  -they take only objects, hence boxing and un boxing required (Disadvantage)

- Hence, we need strong typed and also flexible.
- Thus require Generics
- Generics helps us to create flexible strong typed collection.
- Generics separate logic from the datatypes.
- This increases reusability and better maintenance of code

Advantages:

- **Reusability:** You can use a single generic type definition for multiple purposes in the same code without any alterations. For example, you can create a generic method to add two numbers. This method can be used to add two integers as well as two floats without any modification in the code.
- **Type Safety:** Generic data types provide better type safety, especially in the case of collections. When using generics you need to define the type of objects to be passed to a collection. This helps the compiler to ensure that only those object types that are defined in the definition can be passed to the collection.
- **Performance:** Generic types provide better performance as compared to normal system types because they reduce the need for boxing, unboxing, and typecasting of variables or objects.

1. b) List the functions of Enqueue, Dequeue, peek and Capacity of Queue class.

- **Enqueue** adds an element to the end of the Queue.
- **Dequeue** removes the oldest element from the start of the Queue.
- **Peek** returns the oldest element that is at the start of the Queue but does not remove it from the Queue.
- The **capacity** of a Queue is the number of elements the Queue can hold.

- As elements are added to a Queue, the capacity is automatically increased as required by reallocating the internal array.
- Queue accepts **null** as a valid value for reference types and allows duplicate elements.

2. a) Implement Push, Pop and Display operations on Stack using collections.

```
using System;

using System.Collections;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;


namespace DemoApplication

{

 class Program

 {

  static void Main(string[] args)

  {

   Stack st = new Stack();

   st.Push(1);

   st.Push(2);

   st.Push(3);


   st.Pop();


   foreach (Object obj in st)

   {

    Console.WriteLine(obj);

   }
```

```
    Console.ReadKey();

  }

 }

}
```

Output:

2

1

2. b) What is the difference between Array and ArrayList and SortedList in C#?

	Array	ArrayList
1	Array is strongly typed. This means that an array can store only specific type of items\elements	ArrayList can store any type of items\elements.
2	In arrays we can store only one datatype either int, string, char etc…	In arraylist we can store all the datatype values
3	Array cant accept null	ArrayList collection accepts null
4	Arrays belong to System.Array namespace using System;	Arraylist belongs to System.Collection namespaces using System.Collections;

5	Example -
	int[] intArray=new int[]{2};
	intArray[0] = 1;
	intArray[2] = 2;
		Example -
		ArrayList Arrlst = new ArrayList();
		Arrlst.Add("Sagar");
		Arrlst.Add(1);
		Arrlst.Add(null);

SortedList - In C#, SortedList is a collection of key/value pairs which are sorted according to keys. By default, this collection sort the key/value pairs in ascending order. It is of both generic and non-generic type of collection.

3.a) Explain with a program how we can add Integers, Strings and Boolean values to the ArrayList collection.

ArrayList.add(element)
Below are some examples of how the "add" method can be used. The add method can be used to add various data types to the Array List collection.

Below you can see examples of how we can add Integer's Strings and even Boolean values to the Array List collection.

a1.add(1) – This will add an Integer value to the collection
a1.add("Example") – This will add a String value to the collection
a1.add(true) – This will add a Boolean value to the collection
Now let's see this working at a code level. All of the below-mentioned code will be written to our Console application. The code will be written to our Program.cs file.

In the program below, we will write the code to create a new array list. We will also show to add elements and to display the elements of the Array list.

C# Collections

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


namespace DemoApplication
{
 class Program
 {
  static void Main(string[] args)
  {
   ArrayList a1 = new ArrayList();
   a1.Add(1);
   a1.Add("Example");
   a1.Add(true);

   Console.WriteLine(a1[0]);
   Console.WriteLine(a1[1]);
   Console.WriteLine(a1[2]);
   Console.ReadKey();
  }
 }
```

}

4. a) Write an example program with results to implement InsertWith, UnionWith and ExceptWith methods in HashSet

- HashSet<T> class is optimized for performing set operations.
- Like, Union and intersection of sets.
- Add and Remove methods are used for insert and delete items.

Speciality of HashSet<T> is following methods:

- IntersectWith
- UnionWith
- ExceptWith

- HashSet<string> employees = new HashSet<string>(new string[] {“a1”, “a2”, “a3”, “a4”, “b2”});

- HashSet<string> customers = new HashSet<string>(new string[] {“a3”, “b2”, “b3”, “a1”});

- customers.UnionWith(employees);
  \\ Ans: a1,a2,a3,a4,b2,b3

- customers.ExceptWith(employees);
  \\ Ans: b2, b3

  // to find customers who are also employees

- customers.IntersectWith(employees);
  \\ Ans: a3, a1, b2

4. b) List the methods and its function used in Linked List Collections

- AddFirst method: insert an element at the start of the List.
- AddLast method: insert element at the end of the List.
- AddBefore method: to add before a specific item
- AddAfter method: to add after a specific item

5. b) What is a Delegate? Explain with syntax how to do Declaration, Instantiation and Invocation of Delegates .

- A delegate is a reference to a method.
- Assign a reference to a method to a delegate
- Delegate is a reference type variable that can hold a reference to the methods.

- For example, if you click an *Button* on a form , the program would call a specific method. In simple words, it is a type that represents references to methods with a particular parameter list and return type and then calls the method in a program for execution when it is needed.

- Delegate type can be declared using the **delegate** keyword.
- Once a delegate is declared, delegate instance will refer and call those methods whose return type and parameter-list matches with the delegate declaration.
- Syntax:

[modifier] delegate [return_type] [delegate_name] ([parameter_list]);

Instantiation & Invocation of Delegates

- After declaring a delegate, a delegate object is created with the help of **new** keyword. – Instantiation of Delegate
- Once a delegate is instantiated, a method call made to the delegate is pass by the delegate to that method. The parameters passed to the delegate by the caller are passed to the method.
- Return value, if any, from the method, is returned to the caller by the delegate. This is known as invoking the delegate.
- Syntax

[delegate_name] [instance_name] = new [delegate_name](calling_method_name);

6. a) Write a note on Anonymous Delegate

- Binding an unnamed code block to a delegate is called Anonymous method.

```
using System;

public class Program

{

        public delegate void Print(int value);

        public static void Main()

        {

                Print objdelegate = delegate(int val)

                {

                 Console.WriteLine("Inside Anonymous method. Value: {0}", val);
```

```
            };

        objdelegate(100);

    }

}
```

b) What is the function of Events. With an example, explain how to use Events

- Although we can invoke any number of methods using Delegates, but we have to invoke it explicitly.
- In may cases, it is useful to have a delegate that run automatically.
- Example: need to invoke stopmachinery Delegate and halt machine when machine gets heated up.
- .Net framework provides events for such applications that arrange for the delegate to be called to handle the situations.

- We declare an event in a class intended to act as an event source.
- Event source is a class that monitors its environment and raises an event when something significant happens.
- Ex:

✓ In class that monitors the temperature of each equipments, event is declared.
✓ This class raises "Machine overheating" event if it detects the machine is getting heated up.
✓ An event maintains a list of methods to call when it is raised.
✓ This list are called as subscribers.
✓ These methods will handle the event and take necessary corrective actions like, shut down the machines.
✓ Syntax to declare an event:
event delegateTypeName eventName

```
Public event StopMachineryDelegate MachineOverheating;
```

Ex:

```
Class temperaturemonitor
{

    Public delegate void StopMachineryDelegate();

    Public event StopMachineryDelegate MachineOverheating;
```

}

- We can subscribe to an event by using += operator.

```
Class temperaturemonitor
{

    Public delegate void StopMachineryDelegate();

    Public event StopMachineryDelegate MachineOverheating;

}

.....

temperaturemonitor temp = new temperaturemonitor();

temp.MachineOverheating +=welder.Finishwelding;

temp.MachineOverheating +=painter.Finishpainting;
```

7 a) Mention the need of Language Integrated Query (LINQ) in C#.

- Using LINQ, we can write query on wide variety of Data sources like, Array, collections, Database Tables etc.
- **Familiar language:** Developers don't have to learn a new query language for each type of data source or data format.
- **Less coding:** It reduces the amount of code to be written as compared with a more traditional approach.
- **Readable code:** LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Standardized way of querying multiple data sources:** The same LINQ syntax can be used to query multiple data sources.
- **Compile time safety of queries:** It provides type checking of objects at compile time.
- **IntelliSense Support:** LINQ provides IntelliSense for generic collections.

7. b) Explain LINQ to select and order integer data in descending order with an example.

```
from <alias> in <coll or Array> [<clauses>] select <alias>
```

Note: clauses are optional

```
var brr = from i in arr select i;
```

Note: var is a keyword, implicitly typed local variable.

```
var brr =  from i in arr  where i>40 select i;

\\ data greater than 40 is selected


var brr =  from i in arr  where i>40  orderby  i select i;

\\  arrange in ascending order


var brr =  from i in arr  where i>40  orderby  i descending select i;

\\ arrange in descending order
```

8. a) Explain Operator Overloading and their constraints.

- Operator Overloading is an approach defining multiple behavior of an operator.
- Behavior varies based on the type of operands between which operator is used. i. e + will add integers and concatenate with strings.
- Operator methods are written in language library.
- public static int operator +(int a, int b)
- public static string operator +(string a, string b)

- We can write an operator method using following Syntax.

```
[<modifier>] static <return type> operator <opt>(operand type)

{

        Logic

}
```

Constraints:

- Cannot change precedence or Associatively.
- We cannot change the multiplicity of an operand. Ex. + is a binary operator.
- We cannot invent new operator symbols
- Some operators symbols cannot be overloaded. Ex. Dot(.)

8. b) Implement Addition of Matrix using Operator Overloading.

```
  using System;
namespace demo


{
 class Matrix
{
        int a,b,c,d;
        public Matrix(int a, int b, int c, int d)
                {
                 this.a=a; this.b=b; this.c = c; this.d=d;
                }
        public  static Matrix operator +(Matrix obj1, Matrix obj2)
        {
        Matrix obj = new Matrix(obj1.a+obj2.a, obj1.b+obj2.b,
        obj1.c+obj2.c, obj1.d+obj2.d);
                return obj;
        }
}
}
```

```
class maxadd
    {
    public static void Main()
    {
    Matrix m1 = new Matrix(1,2,3,4);
    Matrix m2 = new Matrix(1,2,3,4);
    Matrix m3 = m1 + m2;
    }
    }
```