Introduction to Dot Net Framework for Application Development.

IAT-2- Solution

1. a) Define inheritance in C#? Explain Single Level and Multilevel Inheritance with example. 8M

Ans:.

C# Inheritance

In C#, inheritance is a process in which one object acquires all the properties and behaviors of its parent object automatically. In such way, you can reuse, extend or modify the attributes and behaviors which is defined in other class.

In C#, the class which inherits the members of another class is called **derived class** and the class whose members are inherited is called **base** class. The derived class is the specialized class for the base class.

Advantage of C# Inheritance

Code reusability: Now you can reuse the members of your parent class. So, there is no need to define the member again. So less code is required in the class.

C# Single Level Inheritance Example:

When one class inherits another class, it is known as single level inheritance. Let's see the example of single level inheritance which inherits the fields only.

```
using System;
public class Employee
{
   public float salary = 40000;
}
public class Programmer: Employee
{
   public float bonus = 10000;
}
class TestInheritance{
   public static void Main(string[] args)
   {
      Programmer p1 = new Programmer();
```

```
12.
13.             Console.WriteLine("Salary: " + p1.salary);
14.             Console.WriteLine("Bonus: " + p1.bonus);
15.
16.         }
17.     }
```

In the above example, Employee is the **base** class and Programmer is the **derived** class.

C# Multi Level Inheritance Example

When one class inherits another class which is further inherited by another class, it is known as multi level inheritance in C#. Inheritance is transitive so the last derived class acquires all the members of all its base classes.

Let's see the example of multi level inheritance in C#.

```
1.  using System;
2.    public class Animal
3.     {
4.       public void eat() { Console.WriteLine("Eating..."); }
5.    }
6.    public class Dog: Animal
7.    {
8.       public void bark() { Console.WriteLine("Barking..."); }
9.    }
10.         public class BabyDog : Dog
11.         {
12.            public void weep() { Console.WriteLine("Weeping..."); }
13.         }
14.         class TestInheritance2{
15.            public static void Main(string[] args)
16.             {
17.                BabyDog d1 = new BabyDog();
```

```
18.             d1.eat();
19.             d1.bark();
20.             d1.weep();
21.           }
22.       }
```

1 b). What are Virtual Methods? 2M

A virtual method is a declared class method that allows overriding by a method with the same derived class signature. Virtual methods are tools used to implement the polymorphism feature of an object-oriented language, such as C#. When a virtual object instance method is invoked, the method to be called is determined based on the object's runtime type, which is usually that of the most derived class.

2 a).Explain the need of Virtual Methods. List the conditions required while using Virtual Methods. 6M

When a method is declared as a virtual method in a base class then that method can be defined in a base class and it is optional for the derived class to override that method. The overriding method also provides more than one form for a method. Hence it is also an example for polymorphism.

When a method is declared as a virtual method in a base class and that method has the same definition in a derived class then there is no need to override it in the derived class. But when a virtual method has a different definition in the base class and the derived class then there is a need to override it in the derived class.

When a virtual method is invoked, the run-time type of the object is checked for an overriding member. The overriding member in the most derived class is called, which might be the original member, if no derived class has overridden the member.

Virtual Method in C#

1. By default, methods are non-virtual. We can't override a non-virtual method.
2. We can't use the virtual modifier with the static, abstract, private or override modifiers.

conditions required while using Virtual Methods.

- Virtual method cannot be private.
- Override methods cannot be private, but can be of protected access.
- Override only virtual method.
- If no override keyword is used, then it will be hiding.

2 b) Mention the functions of the following Keywords. 4M

1. Override: It shows the reader of the code that "this is a virtual method, that is overriding a virtual method of the base class."

2. Sealed: Sealed classes are used to restrict the users from inheriting the class. A class can be sealed by using the ***sealed*** keyword. The keyword tells the compiler that the class is sealed, and therefore, cannot be extended.

3. this: The “this” keyword in C# is used to refer to the current instance of the class. It is also used to differentiate between the method parameters and class fields if they both have the same name.

4. base: The `base` keyword is used to access members of the base class from within a derived class:

3. a) Define Interface? Explain how interfaces are created with an example. 6M

Ans:

- An interface can only contain declarations but not implementations.
- When we implement an interface, we must ensure that each method matches its corresponding interface method.
- Return type should match
- Any parameters passed should match exactly.
- In C#, you cannot use any access modifier for any member of an interface.
-

Example:

```
interface  landbound

{
        int numberoflegs();
}
class horse : landbound
```

```
{
        public int numberoflegs()
                {
                        return  4;
                }
}
```

3. b) With an example, explain how to explicitly implement an Interface. 4M

If a class implements two interfaces that contain a member with the same signature that leads to disambiguity. To solve this problem , we can implement interface explicitly.

Example:

```
interface  landbound
        {
        int numberoflegs();
        }
interface  newintr
        {
        int numberoflegs();
        }

class  horse : landbound, newintr
         {
         int   landbound.numberoflegs()
          {
        return 4;
          }

     int newintr.numberoflegs()
          {
          return 2;
         }

class GFG {


   static void Main(string[] args)

   {

        landbound   I1 = new horse();

        int ans = I1.numberoflegs();
```

```
        Console.WriteLine(ans);

        newintr I2 = new horse();

        int abc= I2.numberoflegs();

        Console.WriteLine(abc);


}
```

4 a). Write a Note on abstract Class. 5M

In C#, abstract class is a class which is declared abstract. It can have abstract and non-abstract methods. It cannot be instantiated. Its implementation must be provided by derived classes. Here, derived class is forced to provide the implementation of all the abstract methods.

- Generally, we use abstract class at the time of *inheritance*.
- A user must use the *override* keyword before the method which is declared as abstract in child class, the abstract class is used to inherit in the child class.
- It can contains constructors or destructors.
- It can implement functions with non-Abstract methods.
- It cannot support multiple inheritance.
- It can't be static.

4 b) Mention the advantages of Sealed classes and Sealed Methods. 5M

Sealed Classes:

- Sealed class cannot be a base class.
- Cannot declare any virtual class.
- Abstract class cannot be sealed.

Sealed Methods:

- methods in an unsealed class can be sealed.

- Derived class cannot override this method
- Can seal only method declared as override

5 a) What is Garbage Collection? What is the need of Garbage collection? 5M

Ans:

When a program starts, the system allocates some memory for the program to get executed.

When a C# program instantiates a class, it creates an object.

The program manipulates the object, and at some point the object may no longer be needed.
When the object is no longer accessible to the program and becomes a candidate for garbage collection.

There are two places in memory where the CLR stores items while your code executes.

1. stack 2. heap
The stack keeps track of what's executing in your code (like your local variables), and the heap keeps track of your objects.

Value types can be stored on both the stack and the heap.

For an object on the heap, there is always a reference on the stack that points to it.

The garbage collector starts cleaning up only when there is not enough room on the heap to construct a new object
The stack is automatically cleared at the end of a method. The CLR takes care of this and you don't have to worry about it.

The heap is managed by the garbage collector.

In unmanaged environments without a garbage collector, you have to keep track of which objects were allocated on the heap and you need to free them explicitly. In the .NET Framework, this is done by the garbage collector.

5 b). Compare Finalize and Dispose in Garbage Collection 5M

Difference between Dispose & Finalize Method

Dispose	Finalize
It is used to free unmanaged resources like files, database connections etc. at any time.	It can be used to free unmanaged resources (when you implement it) like files, database connections etc. held by an object before that object is destroyed.
Explicitly, it is called by user code and the class which is implementing dispose method, must has to implement IDisposable interface.	Internally, it is called by Garbage Collector and cannot be called by user code.
It belongs to IDisposable interface.	It belongs to Object class.
It's implemented by implementing IDisposable interface Dispose() method.	It's implemented with the help of destructor in C++ & C#.
There is no performance costs associated with Dispose method.	There is performance costs associated with Finalize method since it doesn't clean the memory immediately and called by GC automatically.

6 a) Briefly explain the properties in C#. 4M

Ans:

- A property is a member 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.*
- This enables data to be accessed easily and still helps promote the safety and flexibility of methods.
- C# allows encapsulation of data through the use of
  -accessors (to get data)
  -mutators (to modify data)
  which help in manipulating private data indirectly without making it public.
- In computer programming, an **accessor** method is a method that fetches private data that is stored within an object.
- While a **mutator**, in the context of C#, is a method, with a public level of accessibility, used to modify value of a private member variable of a class.
-

6 b) Explain read write, read only, write only properties with example. 6M

Ans:
Properties can be

- *read-write* (they have both a get and a set accessor),
- *read-only* (they have a get accessor but no set accessor),
- *write-only* (they have a set accessor, but no get accessor).

Read Only Property:

```
public int Age
   {

      get
      {
         return  studentAge;
      }


   }
```

Write Only Property:

```
public int Age
   {
      set
      {
         studentAge = value;
```

```
    }

  }
```

Read Write Properties:

```
public int Age
  {

    get
    {
      return  studentAge;
    }

    set
    {
      studentAge = value;
    }

  }
```

7 a) Write a program to demonstrate access of two private fields using properties. 8M

```
using System;


public class DemoEncap {


  private String  studentName;


  private int  studentAge;


  public String Name

  {


    get

    {

      return  studentName;
```

```
        }

        set
        {
            studentName = value;
        }

    }

    public int Age
    {

        get
        {
            return  studentAge;
        }

        set
        {
            studentAge = value;
        }

    }

}
```

```
class xyz {


   // Main Method

   static public void Main()

   {

        DemoEncap obj = new DemoEncap();

           obj.Name = "Ankita";

        obj.Age = 21;

      Console.WriteLine("Name: " + obj.Name);

      Console.WriteLine("Age: " + obj.Age);

   }

}
```

7 b) List the Property Restrictions 2M

- We can assign a value through a property only after class has been initialized.
- Property can contain at most one get and one set accessors.
- Property cannot contain other methods, fields or properties.
- get and set accessors cannot take any parameters

8 a) What are Indexers? Write a program to implement Indexers, where both string and integer are used as index. 8M

- Indexer can be thought of as a smart Array.
- Property is a Smart Field.
- Property encapsulates a single value in a class.

- Indexer Encapsulates a set of values.
- Introduce indexer with this keyword.
- Specify type of value returned by indexer.
- Specify the type of value to use as index into the indexer between Square brackets.

```
using System;


namespace IndexerApplication

{

   class IndexedNames

{

      private string[] namelist = new string[size];

      static public int size = 10;


      public IndexedNames() {

         for (int i = 0; i < size; i++) {

            namelist[i] = "N. A.";

         }

      }

      public string this[int index] {

         get {

            string tmp;


            if( index >= 0 && index <= size-1 )

              {

                tmp = namelist[index];

              }
```

```
            else
            {
               tmp = "";
            }

            return ( tmp );
              }
        set
            {
            if( index >= 0 && index <= size-1 ) {
               namelist[index] = value;
            }
        }
    }

    public int this[string name] {
        get
            {
           int index = 0;

           while(index < size) {
              if (namelist[index] == name) {
               return index;
              }
              index++;
           }
```

```
            return index;
         }
      }

      static void Main(string[] args) {
         IndexedNames names = new IndexedNames();
         names[0] = "Zara";
         names[1] = "Riz";
         names[2] = "Nuha";
         names[3] = "Asif";
         names[4] = "Davinder";
         names[5] = "Sunil";
         names[6] = "Rubic";

         //using the first indexer with int parameter
         for (int i = 0; i < IndexedNames.size; i++) {
            Console.WriteLine(names[i]);
         }

         //using the second indexer with the string parameter
         Console.WriteLine(names["Nuha"]);

      }
   }
}
```

8. b) Differentiate arrays and Indexers. 2M

- Indexers can use nonnumeric subscripts whereas Arrays use only integer subscripts.
- Indexers can be overloaded.
- Indexers cannot be used as ref and out parameters.