1. Explain different modifiers that can be specified for C# method

The modifiers specify keywords that decide the nature of accessibility and the mode ofapplication of the method.

A method can take one or more of the modifiers listed in Table

Modifier	Description
Public	The method can be accessed from anywhere, including outside the class
Protected	The method is available both within theclass and within the derived class
Internal	The method is available only within the file
Private	The method is available only within theclass
Static	To declare as class method
Abstract	A abstract method which defines the signature of the method, but doesn't provide an implementation
Virtual	The method can be overridden by aderived class
Override	The method overrides an inherited virtualor abstract method
New	The method hides an inherited method withthe same signature
Sealed	The method overrides an inherited virtual method, but cannot be overriddenby any classes which inherit from this class.

2. Explain different types of method parameters in c#

Method parameters :

The invocation involves not only passing the values into the method but also getting backthe results from the method.

For managing the process of passing values and getting back the results, C# employs fourkinds of parameters.

- Value parameters

• Reference parameters

• Output parameters

• Parameter arrays

Value parameters are used for passing parameters into methods by value. On the other hand reference parameters are used to pass parameters into methods by reference. Outputparameters, as the name implies, are used to pass results back from a method. Parameter arrays are used in a method definition to enable it to receive variable number of arguments when called.

Pass by value: By default, method parameters are passed by value. That is, a parameter declared with no modifier is passed by value and is called a value parameter. When a method is invoked, the values of actual parameters are assigned to the corresponding formal parameters. The values of the value parameters can be changed within the method.The value of the actual parameter that is passed by value to a method is not changed by any changes made to the corresponding formal parameter within the body of the method. This is because the methods refer to only copies of those variables when they are passed by value.

Pass by reference :Default behaviour of methods in C# is pass by value. We can, however, force the value parameters to be passed by reference. To do this, we use the refkeyword. A parameter declared with the ref modifier is a reference parameter.

Example:

```
void Modify ( ref int x )
```

Here, x is declared as a reference parameter.

Unlike a value parameter, a reference parameter does not create a new storage location. Instead, it presents the same storage location as the actual parameter used in themethod invocation.

Example:

```
void Modify ( ref int x )
    { x+=10;
      } int m = 5;
      Modify ( ref m ); // pass by reference
```

Reference parameters are used in situations where we would

like tochange the values of variables in the calling method.

When we pass arguments by reference, the 'formal' arguments in the called methodbecome aliases to the 'actual" arguments in the calling method.

This means that when the method is working with its own arguments, it is actuallyworking with the original data.

The output parameters: Output parameters are used to pass results back to the calling method. This is achieved by declaring the parameters with an out keyword. Similar to a reference parameter, an output parameter does not create a new storage location. Instead,it becomes an alias to the parameter in the calling method. When a formal parameter is declared as out, the corresponding actual parameter in the calling method must also be declared as out.

Example:

```
void Output ( out int x )
        { x = 100; } int m; //m is
 uninitializedOutput ( out m ); //value of
 m is set
```

Note that the actual parameter m is not assigned any values before it is passed as outputparameter. Since the parameters x and m refer to the same storage location, m takes the value that is assigned to x.

Variable argument lists : In C#, we can define methods that can handle variable number of arguments using what are known as parameter arrays. Parameter arrays aredeclared using the keyword params.

Example:

```
void Functionl (Params int [] x )
 {
        }
```

Here, x has been declared as a parameter array. Note that parameter arrays must be onedimensional arrays. A parameter may be a part of a formal parameter list and in suchcases, it must be the last parameter. The method Functionl defined above can be invoked in two ways: • Using int type an-ay as a value parameter. Example: Functionl(a); Here, a is an array of type int • Using zero or more int type arguments for the parameter array. Example: Function (10, 20); The second invocation creates an int type array with two elements 10 and 20 and passes the newly created array as the actualargument to the method.

3. Explain how to define class and objects in C# with example of stack operations

A class is a user-defined data type with a template that serves to define its properties.

Once the class type has been defined, we can create 'variables' of that type using declarationsthat are similar to the basic type declarations.

In C#, these variables are termed as instances of classes, which the actual

objects.The basic form of a class definition is:

Categories of Class members Adding variables Data is encapsulated in a class by placingdata fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. We candeclare the instance variables exactly the same way as we declare local variables.

Example: Adding methods A class with only data fields and without methods that operate onthat data has no life. The objects created by such a class cannot respond to any messages. Wemust therefore add methods that are necessary for manipulating the data contained in the class.

Methods are declared inside the body of the class, usually after the declaration of instancevariables.

The general form of a method declaration is Member class modifiers One of the goals of object-oriented programming is 'data hiding'. That is, a class may be designed to hide its members from outside accessibility. C# provides a set of 'access modifiers' that can be used with the members of a class to control their visibility to outside users. Creating Objects An object in C# is essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred to as instantiating an object. Object in C# are created using the new operator. The new operator creates an object of the specified class andreturns a reference to that object.

Example fot stack:

```
using System;

namespace

Stack_Implement{public

class Stack
 {
      public int[] stack;

     public readonly int

   Capacity;public int Top=-1;
     public  Stack()
      {
         Console.WriteLine("Enter the capacity for
         stack");
         Capacity=Convert.ToInt32(Console.ReadLine()
         ); stack = new int[Capacity];
      }
      public void Push()
```

```
{

    if (IsOverflow())
    {
        Console.WriteLine("Stack Overflow!!");
```

```
    }
    els
    e
    {          Console.WriteLine("Enter the element to be
               pushed");int
               a=Convert.ToInt32(Console.ReadLine());
       Top++;
       stack[Top] =
       a;
       Console.WriteLine("Element pushed: " +stack[Top]);


    }
 }
 public void Pop()
 {
    if (IsUnderflow())
    {
       Console.WriteLine("Stack Underflow!! Cannot Remove!!");

    }
    else
    {
       Console.WriteLine("Element popped: "
       +stack[Top]);Top--;
    }

 }
 public void Peep()
 {
    if (IsUnderflow())
    {
       Console.WriteLine("No element in the stack to Peep!! ");


    }
    else
       Console.WriteLine("Top element in the stack: "+stack[Top]);

 }
 public bool IsUnderflow()
 {
    if (Top<=-1)
       return
       true;
    else
       return false;
 }
public  bool IsOverflow()
```

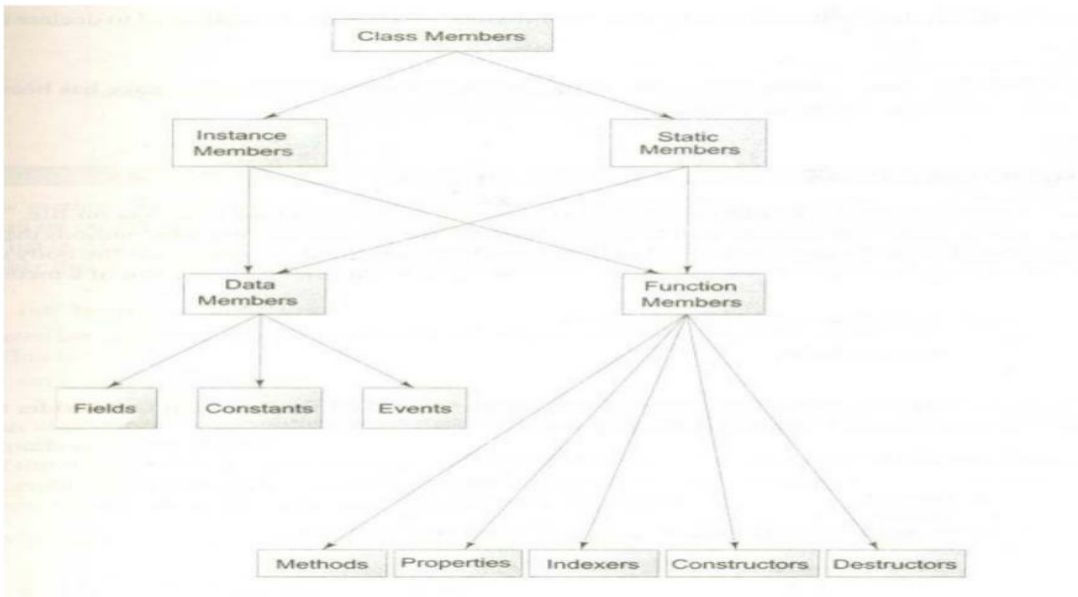
```
{
   if (Top >= Capacity-1)
      return true;
```

```
        else
           return false;

      }
public void Print()
{
 if (IsUnderflow())
        {
           Console.WriteLine("No elements in the stack to print!! ");


        }
         els
         e
         {        Console.WriteLine("The stack elements
                  are");for(int i=Top;i>=0;i-
                  -){ Console.WriteLine(stack[i]);

         }
}

}

  }
 public class CLient_Stack
  {
      public static void Main(string[] args)
     {
        Stack s = new
Stack();while(true){
Console.WriteLine("MENU OPTIONS");
Console.WriteLine("1.PUSH");
Console.WriteLine("2.POP");
Console.WriteLine("3.PEEP");
Console.WriteLine("4.DISPLAY");
Console.WriteLine("5.EXIT");
Console.WriteLine("Enter your
choice");
int
ch=Convert.ToInt32(Console.ReadLine());
switch(ch)
{
case 1:
s.Push();break;
case 2:
s.Pop();break;
case 3:
s.Peep();break;
case 4:
s.Print();break;
```

```
case 5:
System.Environment.Exit(0);
break;
}
```

```
}
    }
  }
}
```

4. Explain classification of C# class members



Instance members:

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of theclass is instantiated. We can declare the instance variables exactly the same way as we declare local variables.

Data members:

A class with only data fields and without methods that operate on that data has no life. The objects created by such a class cannot respond to any messages. We must therefore add methodsthat are necessary for manipulating the data contained in the class. Methods are declared inside the body of the class, usually after the declaration of instance variables.

Events:

An event is an action or occurrence, such as clicks, key presses, mouse movements, or system generated notifications. Applications can respond to events when they occur. An example of a notification is interrupts. Events are messages sent by the object to

indicate the occurrence of theevent. Events are an effective mean of inter-process communication.

Consider an example of an event and the response to the event. A clock is an object that shows 6AM time and generates an event in the form of an alarm. You accept the alarm event and act accordingly.

Static members:

static int count ;

static int max(int x, int y);

The members that are declared static as shown above are called static members. Since thesemembers are associated with the class itself rather than with individual objects, the static variables and static methods are often referred to as class variables and class methods.

Properties:

Properties are class members, consists of two accessor methods get and set to modify the field value. The advantage of using a property is it allows a programmer to validate the client's request for any change in the value of a class field and may allow or reject such a request. Using
a property, a programmer can get access to data members as though they are public fields.

Indexers:

An indexer is a class member, which allows us to access member of a class as if it were an array.'this' keyword refers to the object instance. The return type determines what will be returned.
The parameter inside the square brackets is used as the index.

Constructors:

C# supports a special type of method called a constructor that enables an object to initialize itselfwhen it is created. Constructors have the same name as the class itself. Secondly, they do not specify a return type, not even void. This is because they do not return any value. Constructors are usually public because they are provided to create objects. However, they can also be declared as private or protected. In such cases, the objects of that class cannot be created and alsothe class cannot be used as a base class for inheritance.

Destructors:

A destructor is opposite to a constructor. It is a method called when an object is no more required. The name of the destructor is the same as the class name and is preceded by a tilde (~).Like constructors, a destructor has no return type.

5. Explain different types of constructors supported in C#

C# supports a special type of method called a constructor that enables an object to initializeitself when it is created.

Constructors have the same name as the class itself.

Secondly, they do not specify a return type, not even void. This is because they do not returnany value.

Constructors are usually public because they are provided to create

objects.However, they can also be declared as private or protected.

In such cases, the objects of that class cannot be created and also the class cannot be used asa base class for inheritance.

Overloaded constructors: It is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. Methodoverloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, C# matches up the method name firstand then the number and type of parameters to decide which one of the definitions to execute. This process is known as polymorphism. We can extend the concept of method overloading to provide more than one constructor to a class. To create an overloaded constructor method, all we have to do is to provide several different constructor definitions with different parameter lists. The difference may be in either the number or type of arguments. That is, each parameter list should be unique. Here is an example of creating an overloaded constructor

```
class Room
{
      public double length ;
      public double breadth ;

      public Room(double x, double y)              // constructor1
      {
            length = x ;
            breadth = y ;
      }

      public Room(double x)                        // constructor2
      {
            length = breadth = x ;
      }

      public int Area( )
      {
            return (length * breadth) ;
      }
}
```

Here we are overloading the constructor method Room( ).

An object representing a rectangular room will be created as Room rooml = newRoom(25.0,15.0); //using constructor!

On the other hand, if the room is square, then we may create the corresponding object asRoom room2 = new Room(20.0); // using constructor2

A static constructor is declared by prefixing a static keyword to the constructor definition.It cannot have any parameters.

Example:

```
class Abc
{
      static Abc ( ) //No parameters
      {
            . . . .   //set values for static members here
      }
      . . . .
}
```

Note that there is no access modifier on static constructors. It cannot take any. A class can have only one static constructor.

Private constructors: C# does not have global variables or constants. All declarations mustbe contained in a class. In many situations, we may wish to define some utility classes that contain only static members. Such classes are never required to instantiate objects. Creatingobjects using such classes may be prevented by adding a private constructor to the class.

Copy constructors :A copy constructor creates an object by copying variables from another object. For example, we may wish to pass an Item object to the Item constructor sothat the new Item object has the same values as the old one. Since C# does not provide a copy constructor. we must provide it ourselves if we wish to add this feature to the class. Acopy constructor is defined as follows:

```
public Item (Item item)
{
    code = item.code;
    price = item.price;
}
```

The copy constructor is invoked by instantiating an object of type **Item** and passing it the object to be copied. Example:

```
Item item2 = new Item (item1);
```

Now, **item2** is a copy of **item1**.

Static members A class contains two sections. One declares variables and the other declares methods. These variables and methods are called instance variables and instance methods.This is because everytime the class is instantiated, a new copy of each is created. They areaccessed using the objects (with dot operator).

Let us assume that we want to define a member that is common to all the objects and accessed without using a particular object. That is, the member belongs to the class as awhole rather than the objects created from the class. Such members can be defined as follows:

```
static int count ;
```

6. With example explain static members of the class

```
static int max(int x, int y);
```

The members that are declared static as shown above are called static members. Since these members are associated with the class itself rather than with individual objects, the static variables and static methods are often referred to as class variables and class methods. Staticvariables are used when we want to have a variable common to all instances of a class. Likestatic variables, static methods can be called without using the objects. They are also available for use by other classes.

Methods that are of general utility but do not directly affect an instance of that class are usually declared as class methods. C# class libraries contain a large number of class methods.

For example, the Math class of C# System namespace defines many static methods to

perform math operations that can be used in any

program.double x = Math.Sqrt(25.0);

The method Sqrt is a class method (or static method) defined in Math class.

Note that the static methods are called using class

names.In fact, no objects have been created for

use.

Static methods have several restrictions:

• They can only call other static methods.

• They can only access static data.

• They cannot refer to this

```
using System;
class Mathoperation
{
    public static float mul(float x, float y);
    {
        return x*y;
    }
    public static float divide(float x, float y)
    {
        return x/y ;
    }
}

class MathApplication
{
    public void static Main( )
    {
        float a = MathOperation.mul(4.0F,5.0F) ;
        float b = MathOperation.divide(a,2.0F) ;
        Console.WriteLine("b = "+ b) ;
    }
}
```

7. Explain properties with example in C#

Properties:

One of the design goals of object-oriented systems is not to permit any direct access todata members because of the implications of integrity.

It is normal practice to provide special methods known as accessor methods to haveaccess to data members.

We must use only these methods to set or retrieve the values of these members.

The drawback with this type of accessor method is users have to remember that theyhave to use accessor methods to work with data members.

To overcome this problem, C# provides a mechanism known as properties that has thesame capabilities as accessor methods but simple to use.

Using a property, a programmer can get access to data members as though they arepublic fields.

Properties are class members, consists of two accessor methods get and set to modify thefield value.

The advantage of using a property is it allows a programmer to validate the client's request for any change in the value of a class field and may allow or reject such a request.

Example:

Accessing private data using accessor methods

```
using System;
class Number
{
    private int number;
    public void SetNumber( int x )    //accessor method
    {
        number = x;                   //private number accessible
    }
    public int GetNumber( )           //accessor method
    {
        return number;
    }
}

class NumberTest
{
    public static void Main ( )
    {
        Number n = new Number ( );
        n.SetNumber (100);   // set value
        Console.WriteLine("Number = " + n.GerNumber( ));    // get value

        // n.number; //Error! Cannot access private data
    }
```

8. Explain indexers with example in C#

Indexers:

An indexer is a class member, which allows us to access member of a class as if it were anarray.

'this' keyword refers to the object instance.

The return type determines what will be returned.

The parameter inside the square brackets is used as

the index.Example:

Implementation of an indexer

```
using System;
using System.Collections;
class List
{
   ArrayList array = new ArrayList ( );
   public object this[ int index ]
   {
    get
    {
          if(index < 0  index >= array.Count)
          {
               return null;
          }
          else
          {
               return (array [index] );
          }
    }
    set
    {
          array[index] = value;
    }
   }
}

class IndexerTest
{
   public static void Main ( )
   {
    List list = new List ( );
    list [0] = "123";
    list [1] = "abc";
    list [2] = "xyz";
    for (int i = 0, i < list.Count; i++)
          Console.WriteLine( list[i]);
   }
}
```

UNIT-5

Inheritance :

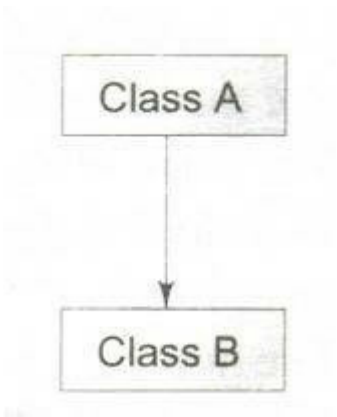
The mechanism of designing or constructing one class from another is calledinheritance.

This may be achieved in two different forms.

- Classical form
- Containment form

Classical form of inheritance:

Inheritance represents a kind of relationship between two classes.Let us consider two classes A and B. We can create a class hierarchy such that B is derived from A as shown in figure.



Class A, the initial class that is used as the basis for the derivedclass is referred to as the base class, parent class or super class.

Class B, the derived class, is referred to as derived class, child class or subclass.

A derived class is a completely new class that incorporates all thedata and methods of its base class. It can also have its own data and method members that are unique to itself. That is, it can enhance the content and behaviour of the base class. We can nowcreate objects of classes A and B independently.

Example:

A a; //

a such cases, we say that the object b is a type of a. Such relationship between a and b is referred to as 'is-a'

i relationship.s Examples of is-a relationship are:

• Dog is-a type of animal

o • Manager is-a type of employee

b • Ford is-a type of car

j The is-a relationship is illustrated in Fig.

e

c

t

o

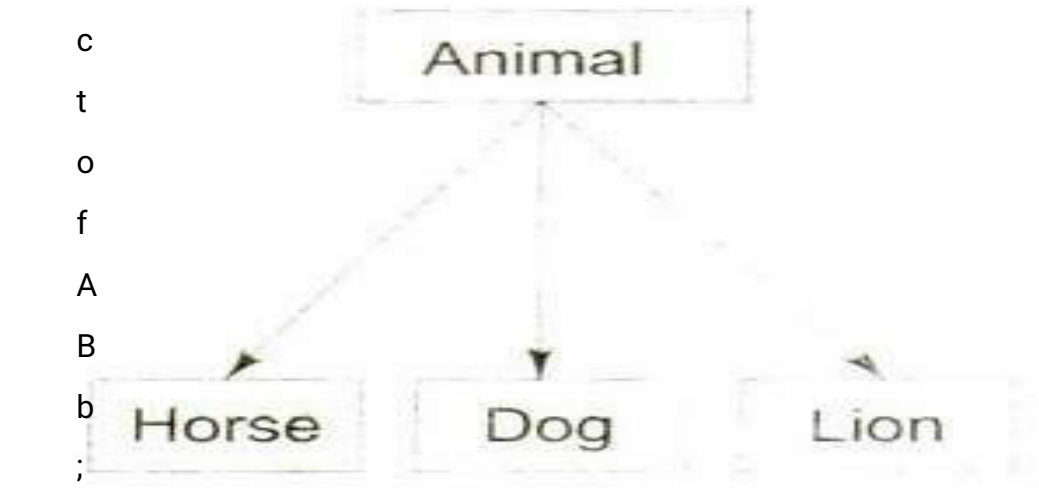
f

A

B

b

;



/

/ Different types of relationship

b C# does not directly implement multiple inheritance. However, this concept is implemented using secondary inheritance path in the

i form ofinterfaces.

s Containment Inheritance: We can also define another form of inheritance relationship known as containership between

o c1assAand B.

b Example:

j

```
class A
{
    . . . .
}
class B
{
    . . . .
    A a; // a is contained in b
}
B b;
. . . .
```

B

I In such cases, we say that the object a is contained in the

object b. This relationship between a and b is referred to as 'has-

n

a'

r
class A ise
l

er class A is termed the 'parent' class and the contained termed a 'child' class.

Examples are:

- Car has-a radio
- House has-a store room
- City has-a road

The has-a relationship is illustrated in Fig.13.4.

Defining a subclass: The definition is very similar to a normal class definition except for the use of colon :and baseclass-name. The colon signifies that the properties of the baseclass are extended to the subclass- name. When implemented the subclass will contain its own members as well those of the baseclass. This kind of situation occurs when we want to add more properties to an existing class without actually modifying it. For example: Visibility Control Class visibility is used to decide which parts of the system can create class objects. A C# class can have one of the two visibility modifiers: public or internal. If we do not explicitly mark the visibility modifier of a class, it is implicitly set to 'internal'; that is, by default allclasses are internal. Internal classes are accessible within the same program assembly and are not accessible from outside the assembly. Classes marked public are accessible everywhere, both within and outside the program assembly. Although classes are normally marked either public or internal, a class mayalso be marked private when it is declared as a member of another class. In such cases, the class behaves like a member of the enclosing class and therefore the modifiers applicable to class members are also applicable to it. It is important to remember that the accessibility domain of a member is never larger than that of the class containing it. Following table gives the visibility domain of members under different combinations of class access specifiers. Defining subclass constructors A subclass constructor is used to construct the instance variables of both the subclass and the superclass. The subclass constructor uses the keyword base to invoke the constructor method of the superclass. Here .the derived constructor takes threearguments, the first two to provide values to the base constructor and the third one to provide value to its own class member. Note that base(x,y) behaves like a method call and therefore arguments are specified without types. The type and order of these arguments must match the type and order of base constructor arguments. When the compiler encounters base(x,y), it passes the values x and y to the base class constructor which takes two int arguments. The base class constructor is called before the derived class constructor is executed. That is, the data members length and breadth are assigned values before the member height is assigned its value. Multilevel Inheritance The class A serves as a base class for the derived class B which in turn serves as a base class for the derived class C. The chain ABC is known as inheritance path. A derived class with multilevel base classes is declared as follows: Hierarchical Inheritance Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below the level. Following figure shows a hierarchical c1assification of accounts in a commercial bank. This is possible because all the accounts possess certain common features.

2. Explain sealed class with an example

Sealed Classes: Preventing Inheritance Sometimes, we may like to prevent a class being furthersubc1assed for security reasons. A class that cannot be subclassed is called a sealed class.
This is achieved in C# using the modifier sealed as follows: A sealed class cannot also be an

abstract class. Sealed Methods When an instance method declaration includes the sealed modifier, the method is said to be a sealed method. It means a derived class cannot overridethis method.

```
class A
{
    public virtual void Fun( )
    {
    . . . .
    }
}
class B : A
{
    public sealed override void Fun ( )
    {
    . . . .
    }
}
```

3. Explain visibility of class members in C# inheritance

Visibility Control Class visibility is used to decide which parts of the system can create classobjects.

A C# class can have one of the two visibility modifiers: public or internal. If we do not explicitlymark the visibility modifier of a class, it is implicitly set to 'internal'; that is, by default all classes are internal. Internal classes are accessible within the same program assembly and arenot accessible from outside the assembly. Classes marked public are accessible everywhere, both within and outside the program assembly. Although classes are normally marked either public or internal, a class may also be marked private when it is declared as a member of another class. In such cases, the class behaves like a member of the enclosing class and therefore the modifiers applicable to class members are also applicable to it.

Class Members Visibility

As mentioned in the previous chapter, a class member can have any one of the five visibility modifiers:

- public
- protected
- private
- internal
- protected internal

Table 13.1 Visibility of class members

Keyword	Visibility			
	Containing classes	*Derived classes*	*Containing program*	*Anywhere outside the containing program*
Private	✓			
protected	✓	✓		
Internal	✓		✓	
protected internal	✓	✓	✓	
Public	✓	✓	✓	✓

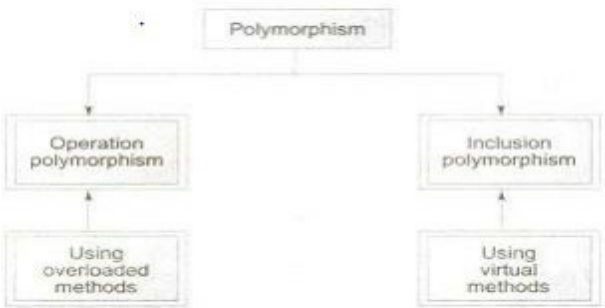
It is important to remember that the accessibility domain of a member is never larger than that ofthe class containing it. Following table gives the visibility domain of members under different combinations of class access specifiers.

Table 13.2 Accessibility domain of class members

Member modifier	*Modifier of the containing class*		
	public	*internal*	*private*
public	Everywhere	only program	only class
internal	only program	only program	only class
private	only class	only class	only class

4. Explain with example how to implement polymorphism in C#

Polymorphism means 'one name, many forms'. Essentially, polymorphism is the capability of oneobject to behave in multiple ways. Polymorphism can be achieved in two ways as shown in Figure. C# supports both of them.



Operation Polymorphism Operation polymorphism is implemented using overloaded methods and operators. The overloaded methods are 'selected' for invoking by matching arguments, in terms of number, type and order. This information is known to the compiler at the time of compilation and, therefore, the compiler is able to select and bind the appropriate method to the object for a particular call at compile time itself. This process is called early binding, or static binding, or static linking. It is also known as compile time polymorphism.

Operation polymorphism

```
using System;
class Dog
{
}
class Cat
{
}
class Operation
{

    static void Call (Dog d)
    {

        Console.WriteLine ("Dog is called");
    }
    static void Call (Cat c)
    {
        Console.WriteLine (" Cat is called ");
    }

    public static void Main( )
    {
        Dog dog = new Dog( );
        Cat cat = new Cat ( );
        Call(dog); //invoking Call( )
        Call(cat); //again invoking Call( )
    }
}
```

Inclusion Polymorphism Inclusion polymorphism is achieved through the use of virtual functions. Assume that the classA implements a virtual method M and classes B and C that arederived from A override the virtual melhod M. When B is cast to A, a call to the method M from A is dispatched to B. Similarly, when C is cast to A a call to M is dispatched to C. The decision on exactly which method to call is delayed until runtime and therefore, it is also known as runtime polymorphism. Since the method is linked with a particular class much later after compilation, this process is termed late binding. It is also known as dynamic binding because the selection of the appropriate method is done dynamically at runtime

Inclusion polymorphism

```
using System;
class Maruthi
{
     public virtual void Display ( ) //virtual method
     {
          Console.WritLine("Maruthi car");
     }
}
class Esteem : Maruthi
{
     public override void Display( )
     {
          Console.WriteLine("Maruthi Esteem");
     }
}
class Zen : Maruthi
{
     public override void Display ( )
     {
          Console.WriteLine("Maruthi Zen");
     }
}
class Inclusion
{
     public static void Main( )
     {
          Maruthi m = new Maruthi ( );
          m = new Esteem ( );  //upcasting
          m.Display ( );
          m = new Zen ( );     //upcasting
          m.Display ( )
     }
}
```

5. Explain exception handling in C#

Exceptions An exception is a condition that is caused by a run-time error in the program. When the C# compiler encounters an error such as dividing an integer by zero, it creates an exception object and throws it to inform us that an error has occurred. If the exception objectis not caught and handled properly, the compiler will display an error message and will terminate the program. If we want the program to continue with the execution of the

remaining code then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task isknown as exception handling. Steps in exception handling :

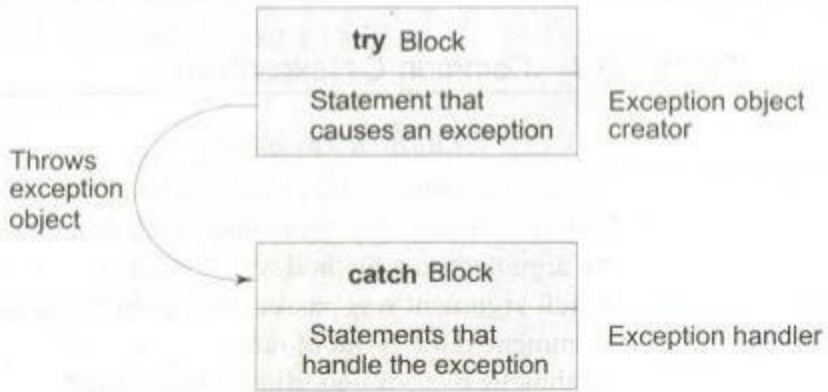
• Find the problem (Hit the exception)

• Inform that an error has occurred (Throw the exception)

• Receive the error information (Catch the exception)

• Take corrective actions (Handle the exception) Syntax of exception handling C# uses a keyword try to preface a block of code that is likely to cause an error condition and 'throw' an exception. A catch block defined by the keyword catch 'catches' the exception 'thrown' bythe try block and handles it appropriately. The catch block is added immediately after the tryblock. The following example illustrates the use of simple try and catch statements: The exception hierarchy All C# exceptions are derived from the class Exception. When an exception occurs, the proper catch handler is determined by matching the type of exception to the name of the exception mentioned. If we are going to catch exceptions at different levels in the hierarchy, we need to put them in the right order.
The rule is that we must always put the handlers for the most derived exception class first. Consider the following code : This code will generate a compiler error, because the exceptionis caught by the first catch (which is a more general one) and the second catch is therefore unreachable. In C#, having unreachable code is always an error. The code must be rewritten as follows: General catch handler A catch block which will catch any exception is called a general catch handler. A general catch handler does not specify any parameter and can be written as: Note that catch (Exception e) can handle all the exceptions thrown by the C# codeand therefore can be used as a general catch handler. However, if the program uses libraries written in other languages, then there may be an exception that is not derived from the class Exception. Such exceptions can be handled by the parameter-less catch statement. This handler is always placed at the end. Since there is no parameter, it does not catch any information about the exception and therefore we do not know what went wrong.
Refer study material for more example :

```
.....
.....
try

  {
      statement;      // generates an exception
  }
  catch (Exception e)
  {
      statement;      // processes the exception
  }
  .....
  .....
```



6. With example explain how to implement user defined exception in C#

Ans: User-defined exception classes are derived from the Exception class. Custom exceptions arewhat we call user-defined exceptions.

In the below example, the exception created is not a built-in exception; it is a customexception –

```
using System;

class Test
{
      public static void Main()
      {
            Console.WriteLine("enter the
            string");string
            s=Console.ReadLine();
            string str=s.ToLower();

            tr
            y
            {     if(str.CompareTo("india")!=0)
                  {
                  //if(s!="INDIA")
```

```
throw (new
MatchNotFoundE
xception("match
not found"));
```

```
                }


            }
            catch(MatchNotFoundException e)
            {
                Console.WriteLine("not found", e.Message);
            }
            Console.WriteLine("found..!\ncontinue");
        }
}

class MatchNotFoundException : Exception
{
    //string message;
    public MatchNotFoundException(string message): base(message)
    {
        //this.message = message;
    }


}
```

7. What is interface? Explain how to define, extend and implement interface in C# with example

C# does not support multiple inheritance. That is, classes in C# cannot have more than onesuperclass.

For instance is not permitted in C#. However, the designers of C# could not overlook theimportance of multiple inheritance.

A large number of real-life applications require the use of multiple inheritance wherebywe inherit methods and properties from several distinct classes.

C# provides an alternate approach known as interface to support the concept of multipleinheritance.

Although a C# class cannot be a subclass of more than one superclass, it can implementmore than one interface, thereby enabling us to create classes that build upon other classes without the problems created by multiple inheritance.

An interface in C# is a reference type. It is basically a kind of class with somedifferences.

Major differences include:

All the members of an interface are implicitly public and abstract

. • An interface cannot contain constant fields, constructors and destructors.

• Its members cannot be declared static.

• Since the methods in an interface are abstract, they do not includeimplementation code.

• An interface can inherit multiple interfaces

Defining an Interface :An interface can contain one or more methods, properties,indexers and events but none of them are implemented in the interface itself. It is the responsibility of the class that implements the interface to define the code for implementation of these members.

The syntax for defining an interface is very similar to that used for defining aclass.

The general form of an interface definition is:

```
interface InterfaceName
{
    Member declarations;
}
```

Here, interface is the keyword and InterfaceName is a valid C# identifier.Remember declarations will contain only a list of members without implementation code.

Given below is a simple interface that defines a single method:

```
interface Show
{
    void Display ( );    // Note semicolon here
}
```

In addition to methods, interfaces can declare properties, indexers and events. Example:

```
interface Example
{
    int Aproperty
    {
        get ;
    }
    event someEvent Changed;
    void Display ( );
}
```

Extending an interface Like classes, interfaces can also be extended. That is, aninterface can be subinterfaced from other interfaces.

The new subinterface will inherit all the members of the superinterface in themanner similar to subclasses.

This is achieved as follows: For example, we can put all members of particular behaviour category in one interface and the members of another category in theother.

Consider the code below:

We can also combine several interfaces together into a single interface. Followingdeclarations are valid: Implementing Interfaces Interfaces are used as 'superclasses' whose properties are inherited by classes. It is therefore necessary tocreate a class that inherits the given interface. This is done as follows: Here the class classname 'implements' the interface interfacename. A more general form ofimplementation may look like this: Abstract class and interface Like any other class, an abstract class can use an interface in the base class list. However, the interface methods are implemented as abstract methods. Example: Note that the class B does not implement the interface method; it simply redeclares as a public abstract method. It is the duty of the class that derives from B to override

and implement the method. Note that interfaces are similar to abstract classes. In fact,

we can convert an interface into an abstract class. Consider the following interface: Now, a class can inherit from B instead of implementing the interface
A. However, the class that inherits B cannot inherit any other class directly. If it isan interface, then the class can not only implement the interface but also use another class as base class thus implementing in effect multiple inheritance

Example :

```
//area
using System;
using
System.Collections.Generic;
using System.Linq;
using System.Text;
namespace
Polymorphism
{
    class Program
    {
        public static void Main(string[] args)
        {
            double a;
            int
            choice=1;
            do
            {
Console.WriteLine("_____");
            Console.WriteLine("Select your choice:\n1.Area of the Circle\n2.Area of the Square\n3.Area
            of
Rectangle\n4.Exit")
;               choice=Int32.Parse(Console.ReadLine());
                Console.WriteLine("___you have entered : "+choice+"____");

                switch(choice)
                {
                    case 1:Circle c=new
                        Circle();a=c.area();
                        Console.WriteLine("Area of the Circle is:"
                        +a);break;
                    case 2:Square s=new
                        Square();a=s.area();
                        Console.WriteLine("Area of the Square is:"
                        +a);break;
                    case 3:Rectangle r=new
                        Rectangle();a=r.area();
                        Console.WriteLine("Area of the Rectangle is:" +a);

                        break;
                    case 4:break;
                }
            }
            while(choice!=4);
        }
    }
    interface Shape
    {
        double area();
    }
    class Circle:Shape
    {
        double pi=3.14;
        public double
        area()
        {
```

```
        int r;
        Console.WriteLine("enter Radius of a
        circle:");r=Int32.Parse(Console.ReadLine());
        return pi*r*r;
}
```

```
        }
        class Square:Shape
        {
                public double area()
                {
                        int side;
                        Console.WriteLine("Enter the length of the
                        side:");side=Int32.Parse(Console.ReadLine());
                        return side*side;
                }
        }
        class Rectangle:Shape
        {
                public double area()
                {
                        int len,brdth;
                        Console.WriteLine("Enter the length of the
                        rectangle:"); len=Int32.Parse(Console.ReadLine());
                        Console.WriteLine("Enter the breadth of the
                        rectangle:");brdth=Int32.Parse(Console.ReadLine());
                        return len*brdth;
                }
        }
}
```

8. What is an event? Explain with example how to define, subscribe and notify an event in C#

An event is an action or occurrence, such as clicks, key presses, mouse movements, or systemgenerated notifications.

Applications can respond to events when they

occur.An example of a notification is interrupts.

Events are messages sent by the object to indicate the occurrence of

the event.Events are an effective mean of inter-process

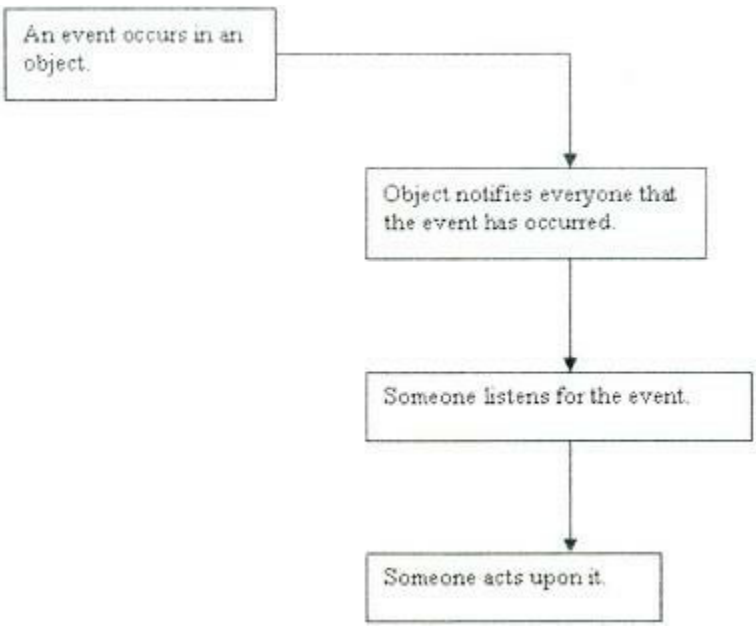
communication.

Consider an example of an event and the response to the event.

A clock is an object that shows 6 AM time and generates an event in the form of an alarm.You accept the alarm event and act accordingly.

The following figure shows the alarm event and handling of the event

v The following figure is the generalized representation that explains events andevent handling.



In C#, delegates are used with events to implement event handling.

Defining an Event The definition of the event in a publisher class includes the declaration of thedelegate as well as the declaration of the event based on the delegate.

The following code defines a delegate named TimeToRise and an event named RingAlarm,which invokes the TimeToRise delegate when it is raised:

```
using System;

namespace Example_4
{
  /// <summary>
  /// The program demonstrates the use of Events.
  /// </summary>

  // First class
  class ClassA
  {
    public void DispMethod()
    {
      Console.WriteLine("Class A has been notified of NotifyEveryOne Event!");
    }
  }
  // Second class
  class ClassB
  {
    public void DispMethod()
    {
      Console.WriteLine("Class B has been notified of NotifyEveryOne Event!");
    }
  }

  class TestEvents
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
      // Object of delegate
      Delegate objDelegate = new Delegate();
      // Object of ClassA
      ClassA objClassA = new ClassA();
      // Object of ClassB
      ClassB objClassB = new ClassB();

      // Subscribing to the event
      objDelegate.NotifyEveryOne += new Delegate.MeDelegate(objClassA.
DispMethod);
      objDelegate.NotifyEveryOne += new Delegate.MeDelegate(objClassB.
DispMethod);

      // Invoking method that contains code to raise the event
      objDelegate.Notify();
```

```
    }
}

class Delegate
{
    // Defining a delegate
    public delegate void MeDelegate();
    // Defining an event
    public event MeDelegate NotifyEveryOne;

    public void Notify()
    {
        // If the event is not null
        if(NotifyEveryOne != null)
        {
            Console.WriteLine("Raise Event : ");
            // Raising the event
            NotifyEveryOne();
        }
    }
}
}
```

Subscribing to an Event The event of the publisher class needs to be associated with its handler. The event handler method is associated with the event using the delegate. When thepublisher object raises the event, the subscribing object associates the method, which needs to be called.

Consider a class named Student which contains a method named WakeUp (). The requirement states that the method WakeUp() should be called at 6 AM. The requirement could be implemented using events. The following code shows how the Student class subscribes to the event named TimeToRi se: Notifying Subscribing objects To notify all the objects that have subscribed to an event, the event needs to be raised. Raising an event is likecalling a method. if(condition is true) RaiseEvent(); When even is raised it will invoke all thedelegates of the objects that are subscribed to that particular event.

9. What is delegate? With example explain how to declare, instantiate and use delegates in C#

A delegate is a userdefined reference type contains a reference to the methods in a class.This reference can be changed dynamically at run time as desired.

Using delegates the method to be called at runtime can be identified.

Executing: a delegate will in turn execute the method that it references.

Consider an example of a coffee vending machine, which dispenses different flavors of coffee, such as cappuccino and black coffee. On selecting the desired flavor of coffee, the vending machine decides to dispense the ingredients, such as milk powder, coffee powder,hot water, cappuccino coffee powder. All the materials are placed in different containers inside the vending machine. The required material is dispensed when you select a flavor. Suppose, you select black coffee, the vending machine will call methods to dispense hot water and coffee powder only. The reference to these methods is made dynamically, when you press the required button to dispense black coffee.

To implement delegates in your application you need to declare delegates, instantiatedelegates and use delegates.

Declaring Delegates :The methods that can be referenced by a delegate are determined by the delegate declaration. The delegate can refer to the methods, which have the same signature asthat of the delegate.

Instantiating Delegates: Instantiating delegates means making it point (or refer) to somemethod. We can instantiate delegate by creating delegate object of the delegate type and assign the address of the required method to the delegate object.

Using delegate: Using delegate means invoking a method using delegates. A

delegate iscalled in a manner similar to calling a method.

Types of Delegates: There are two types of delegates,

Single-cast delegate and Multicast delegate.

A Single-cast delegate can call only one method at a time, whereas a Multicast delegate cancall multiple methods at the same time. The following code shows how to use a multicast delegate

```
77 Program to wrctG the duta                    console and file
namespace Chapterl2_Ex2

        ¿ubi .o C c55 FrantTDf•ez•ia*

                atps.lr. SorepmWriter SWriter;
                //Defining a Delegate
                ;›uDliodelegatevoidrttntCatalsLrlngs):

                //Merh‹>d tn print a z"ring to rhe  console
                p u hi i c s cani z ve i d Qtr a ':e C o n sof e (S lr i n q s cr l

                /ZHotñcd to print g g:ring tc a ftle
                pubJic otatc void WritefiJe lstrinq<

                        //Initializing stream objects
                        FStream = new FileStream("c:\\StoreData.txt",
                        FileMode.Append, FileAccess.Write);
                        SHriter = new StreamBriter(FStream),
                        s= s +' File';
                        Z/Writing a strlng to che f:.eSFricer.WriteLine(sl,
                        //remeving the cQntent from the buffer
                        SRritPz.FluZh();
                        SWriter.Close():
                        FZtreom.close();

                //Method to send the4tring data to rcsp‹:ctive mczhodspublic static void
                DisplayDatzlPzint0at6 PNethodl

                        PHe hodl'thin sñould go to khe"l,

                publzc static void Main(l

                        //InStta11zing the Delegate object
                        PritDData lBr i teri l e), Disp3ayDatalMlDelegatt];
                        Ploplegdte -= new PrintData(Sri eFile);
```

10. With an example explain how to work with files in C#.

The following example shows how to use a delegate:

```
// This code is to print data to the output device, which is either a
file or a screen
using System;
using System.IO;

// Program to write the data to the console and file
namespace Chapter12_Ex1
{
public class PrintToDevice
  {
```

```
        //Creating the variables of Stream classes
        static FileStream FStream;
        static StreamWriter SWriter;
        //Defining a Delegate
        public delegate void PrintData(String s);

        //Method to print a string to the console
        public static void WriteConsole (string str)
        {
        Console.WriteLine("{0} Console",str);
        }

        //Method to print a string to a file
        public static void WriteFile (string s)
    {
                //Initializing stream objects
                FStream  = new FileStream("c:\\StoreData.txt",
                FileMode.Append, FileAccess.Write);
                SWriter = new StreamWriter(FStream);
                s= s + " File";
                //Writing a string to the file
                SWriter.WriteLine(s);
                //removing the content from the buffer
                SWriter.Flush();
                SWriter.Close();
                FStream.Close();
    }
         //Method to send the string data to respective methods
        public static void DisplayData(PrintData PMethod)
    {
        PMethod("This should go to the");
    }
        public static void Main()
        {
                //Initializing the Delegate object

                PrintData Cn = new PrintData (WriteConsole);
                PrintData Fl = new PrintData (WriteFile);
                //Invoking the DisplayData method with the Delegate
                object as the argument
                //Using Delegate
                DisplayData (Cn);
                DisplayData (Fl);
                Console.ReadLine();
        }
    }
}
```

1. Explain different modifiers that can be specified for C# method

The modifiers specify keywords that decide the nature of accessibility and the mode ofapplication of the method.

A method can take one or more of the modifiers listed in Table

Modifier	Description
Public	The method can be accessed from anywhere, including outside the class
Protected	The method is available both within theclass and within the derived class
Internal	The method is available only within the file
Private	The method is available only within theclass
Static	To declare as class method
Abstract	A abstract method which defines the signature of the method, but doesn't provide an implementation
Virtual	The method can be overridden by aderived class
Override	The method overrides an inherited virtualor abstract method
New	The method hides an inherited method withthe same signature
Sealed	The method overrides an inherited virtual method, but cannot be overriddenby any classes which inherit from this class.

2. Explain different types of method parameters in c#

Method parameters :

The invocation involves not only passing the values into the method but also getting backthe results from the method.

For managing the process of passing values and getting back the results, C# employs fourkinds of parameters.

- Value parameters

• Reference parameters

• Output parameters

• Parameter arrays

Value parameters are used for passing parameters into methods by value. On the other hand reference parameters are used to pass parameters into methods by reference. Outputparameters, as the name implies, are used to pass results back from a method. Parameter arrays are used in a method definition to enable it to receive variable number of arguments when called.

Pass by value: By default, method parameters are passed by value. That is, a parameter declared with no modifier is passed by value and is called a value parameter. When a method is invoked, the values of actual parameters are assigned to the corresponding formal parameters. The values of the value parameters can be changed within the method.The value of the actual parameter that is passed by value to a method is not changed by any changes made to the corresponding formal parameter within the body of the method. This is because the methods refer to only copies of those variables when they are passed by value.

Pass by reference :Default behaviour of methods in C# is pass by value. We can, however, force the value parameters to be passed by reference. To do this, we use the refkeyword. A parameter declared with the ref modifier is a reference parameter.

Example:

```
void Modify ( ref int x )
```

Here, x is declared as a reference parameter.

Unlike a value parameter, a reference parameter does not create a new storage location. Instead, it presents the same storage location as the actual parameter used in themethod invocation.

Example:

```
void Modify ( ref int x )
    { x+=10;
      } int m = 5;
      Modify ( ref m ); // pass by reference
```

Reference parameters are used in situations where we would

like tochange the values of variables in the calling method.

When we pass arguments by reference, the 'formal' arguments in the called methodbecome aliases to the 'actual" arguments in the calling method.

This means that when the method is working with its own arguments, it is actuallyworking with the original data.

The output parameters: Output parameters are used to pass results back to the calling method. This is achieved by declaring the parameters with an out keyword. Similar to a reference parameter, an output parameter does not create a new storage location. Instead,it becomes an alias to the parameter in the calling method. When a formal parameter is declared as out, the corresponding actual parameter in the calling method must also be declared as out.

Example:

```
void Output ( out int x )
        { x = 100; } int m; //m is
 uninitializedOutput ( out m ); //value of
 m is set
```

Note that the actual parameter m is not assigned any values before it is passed as outputparameter. Since the parameters x and m refer to the same storage location, m takes the value that is assigned to x.

Variable argument lists : In C#, we can define methods that can handle variable number of arguments using what are known as parameter arrays. Parameter arrays aredeclared using the keyword params.

Example:

```
void Functionl (Params int [] x )
 {
        }
```

Here, x has been declared as a parameter array. Note that parameter arrays must be onedimensional arrays. A parameter may be a part of a formal parameter list and in suchcases, it must be the last parameter. The method Functionl defined above can be invoked in two ways: • Using int type an-ay as a value parameter. Example: Functionl(a); Here, a is an array of type int • Using zero or more int type arguments for the parameter array. Example: Function (10, 20); The second invocation creates an int type array with two elements 10 and 20 and passes the newly created array as the actualargument to the method.

3. Explain how to define class and objects in C# with example of stack operations

A class is a user-defined data type with a template that serves to define its properties.

Once the class type has been defined, we can create 'variables' of that type using declarationsthat are similar to the basic type declarations.

In C#, these variables are termed as instances of classes, which the actual

objects.The basic form of a class definition is:

Categories of Class members Adding variables Data is encapsulated in a class by placingdata fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of the class is instantiated. We candeclare the instance variables exactly the same way as we declare local variables.

Example: Adding methods A class with only data fields and without methods that operate onthat data has no life. The objects created by such a class cannot respond to any messages. Wemust therefore add methods that are necessary for manipulating the data contained in the class.

Methods are declared inside the body of the class, usually after the declaration of instancevariables.

The general form of a method declaration is Member class modifiers One of the goals of object-oriented programming is 'data hiding'. That is, a class may be designed to hide its members from outside accessibility. C# provides a set of 'access modifiers' that can be used with the members of a class to control their visibility to outside users. Creating Objects An object in C# is essentially a block of memory that contains space to store all the instance variables. Creating an object is also referred to as instantiating an object. Object in C# are created using the new operator. The new operator creates an object of the specified class andreturns a reference to that object.

Example fot stack:

```
using System;

namespace

Stack_Implement{public

class Stack
 {
      public int[] stack;

     public readonly int

   Capacity;public int Top=-1;
     public  Stack()
      {
         Console.WriteLine("Enter the capacity for
         stack");
         Capacity=Convert.ToInt32(Console.ReadLine()
         ); stack = new int[Capacity];
      }
      public void Push()
```

```
{

    if (IsOverflow())
    {
        Console.WriteLine("Stack Overflow!!");
```

```
    }
    els
    e
    {          Console.WriteLine("Enter the element to be
               pushed");int
               a=Convert.ToInt32(Console.ReadLine());
       Top++;
       stack[Top] =
       a;
       Console.WriteLine("Element pushed: " +stack[Top]);


    }
 }
 public void Pop()
 {
    if (IsUnderflow())
    {
       Console.WriteLine("Stack Underflow!! Cannot Remove!!");

    }
    else
    {
       Console.WriteLine("Element popped: "
       +stack[Top]);Top--;
    }

 }
 public void Peep()
 {
    if (IsUnderflow())
    {
       Console.WriteLine("No element in the stack to Peep!! ");


    }
    else
       Console.WriteLine("Top element in the stack: "+stack[Top]);

 }
 public bool IsUnderflow()
 {
    if (Top<=-1)
       return
       true;
    else
       return false;
 }
public  bool IsOverflow()
```

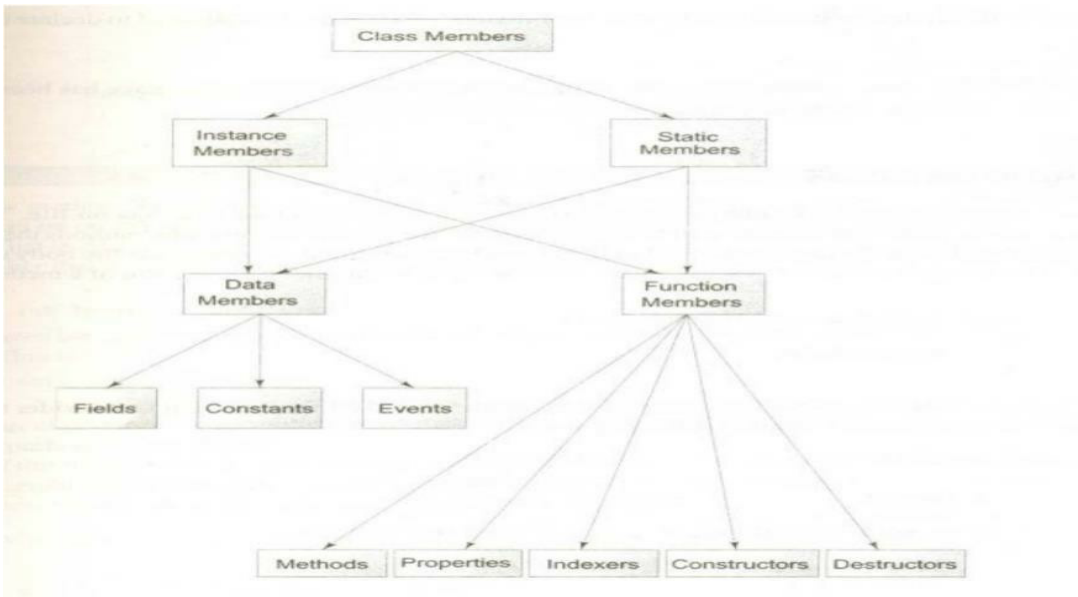
```
{
   if (Top >= Capacity-1)
      return true;
```

```
        else
           return false;

      }
public void Print()
{
 if (IsUnderflow())
        {
           Console.WriteLine("No elements in the stack to print!! ");


        }
         els
         e
         {        Console.WriteLine("The stack elements
                  are");for(int i=Top;i>=0;i-
                  -){ Console.WriteLine(stack[i]);

         }
}

}

  }
 public class CLient_Stack
  {
      public static void Main(string[] args)
     {
        Stack s = new
Stack();while(true){
Console.WriteLine("MENU OPTIONS");
Console.WriteLine("1.PUSH");
Console.WriteLine("2.POP");
Console.WriteLine("3.PEEP");
Console.WriteLine("4.DISPLAY");
Console.WriteLine("5.EXIT");
Console.WriteLine("Enter your
choice");
int
ch=Convert.ToInt32(Console.ReadLine());
switch(ch)
{
case 1:
s.Push();break;
case 2:
s.Pop();break;
case 3:
s.Peep();break;
case 4:
s.Print();break;
```

```
case 5:
System.Environment.Exit(0);
break;
}
```

```
}
    }
  }
}
```

4. Explain classification of C# class members



Instance members:

Data is encapsulated in a class by placing data fields inside the body of the class definition. These variables are called instance variables because they are created whenever an object of theclass is instantiated. We can declare the instance variables exactly the same way as we declare local variables.

Data members:

A class with only data fields and without methods that operate on that data has no life. The objects created by such a class cannot respond to any messages. We must therefore add methodsthat are necessary for manipulating the data contained in the class. Methods are declared inside the body of the class, usually after the declaration of instance variables.

Events:

An event is an action or occurrence, such as clicks, key presses, mouse movements, or system generated notifications. Applications can respond to events when they occur. An example of a notification is interrupts. Events are messages sent by the object to

indicate the occurrence of theevent. Events are an effective mean of inter-process communication.

Consider an example of an event and the response to the event. A clock is an object that shows 6AM time and generates an event in the form of an alarm. You accept the alarm event and act accordingly.

Static members:

static int count ;

static int max(int x, int y);

The members that are declared static as shown above are called static members. Since thesemembers are associated with the class itself rather than with individual objects, the static variables and static methods are often referred to as class variables and class methods.

Properties:

Properties are class members, consists of two accessor methods get and set to modify the field value. The advantage of using a property is it allows a programmer to validate the client's request for any change in the value of a class field and may allow or reject such a request. Using
a property, a programmer can get access to data members as though they are public fields.

Indexers:

An indexer is a class member, which allows us to access member of a class as if it were an array.'this' keyword refers to the object instance. The return type determines what will be returned.
The parameter inside the square brackets is used as the index.

Constructors:

C# supports a special type of method called a constructor that enables an object to initialize itselfwhen it is created. Constructors have the same name as the class itself. Secondly, they do not specify a return type, not even void. This is because they do not return any value. Constructors are usually public because they are provided to create objects. However, they can also be declared as private or protected. In such cases, the objects of that class cannot be created and alsothe class cannot be used as a base class for inheritance.

Destructors:

A destructor is opposite to a constructor. It is a method called when an object is no more required. The name of the destructor is the same as the class name and is preceded by a tilde (~).Like constructors, a destructor has no return type.

5. Explain different types of constructors supported in C#

C# supports a special type of method called a constructor that enables an object to initializeitself when it is created.

Constructors have the same name as the class itself.

Secondly, they do not specify a return type, not even void. This is because they do not returnany value.

Constructors are usually public because they are provided to create

objects.However, they can also be declared as private or protected.

In such cases, the objects of that class cannot be created and also the class cannot be used asa base class for inheritance.

Overloaded constructors: It is possible to create methods that have the same name, but different parameter lists and different definitions. This is called method overloading. Methodoverloading is used when objects are required to perform similar tasks but using different input parameters. When we call a method in an object, C# matches up the method name firstand then the number and type of parameters to decide which one of the definitions to execute. This process is known as polymorphism. We can extend the concept of method overloading to provide more than one constructor to a class. To create an overloaded constructor method, all we have to do is to provide several different constructor definitions with different parameter lists. The difference may be in either the number or type of arguments. That is, each parameter list should be unique. Here is an example of creating an overloaded constructor

```
class Room
{
      public double length ;
      public double breadth ;

      public Room(double x, double y)              // constructor1
      {
            length = x ;
            breadth = y ;
      }

      public Room(double x)                        // constructor2
      {
            length = breadth = x ;
      }

      public int Area( )
      {
            return (length * breadth) ;
      }
}
```

Here we are overloading the constructor method Room( ).

An object representing a rectangular room will be created as Room rooml = newRoom(25.0,15.0); //using constructor!

On the other hand, if the room is square, then we may create the corresponding object asRoom room2 = new Room(20.0); // using constructor2

A static constructor is declared by prefixing a static keyword to the constructor definition.It cannot have any parameters.

Example:

```
class Abc
{
      static Abc ( ) //No parameters
      {
            . . . .   //set values for static members here
      }
      . . . .
}
```

Note that there is no access modifier on static constructors. It cannot take any. A class can have only one static constructor.

Private constructors: C# does not have global variables or constants. All declarations mustbe contained in a class. In many situations, we may wish to define some utility classes that contain only static members. Such classes are never required to instantiate objects. Creatingobjects using such classes may be prevented by adding a private constructor to the class.

Copy constructors :A copy constructor creates an object by copying variables from another object. For example, we may wish to pass an Item object to the Item constructor sothat the new Item object has the same values as the old one. Since C# does not provide a copy constructor. we must provide it ourselves if we wish to add this feature to the class. Acopy constructor is defined as follows:

```
public Item (Item item)
{
    code = item.code;
    price = item.price;
}
```

The copy constructor is invoked by instantiating an object of type **Item** and passing it the object to be copied. Example:

```
Item item2 = new Item (item1);
```

Now, **item2** is a copy of **item1**.

Static members A class contains two sections. One declares variables and the other declares methods. These variables and methods are called instance variables and instance methods.This is because everytime the class is instantiated, a new copy of each is created. They areaccessed using the objects (with dot operator).

Let us assume that we want to define a member that is common to all the objects and accessed without using a particular object. That is, the member belongs to the class as awhole rather than the objects created from the class. Such members can be defined as follows:

```
static int count ;
```

6. With example explain static members of the class

```
static int max(int x, int y);
```

The members that are declared static as shown above are called static members. Since these members are associated with the class itself rather than with individual objects, the static variables and static methods are often referred to as class variables and class methods. Staticvariables are used when we want to have a variable common to all instances of a class. Likestatic variables, static methods can be called without using the objects. They are also available for use by other classes.

Methods that are of general utility but do not directly affect an instance of that class are usually declared as class methods. C# class libraries contain a large number of class methods.

For example, the Math class of C# System namespace defines many static methods to

perform math operations that can be used in any

program.double x = Math.Sqrt(25.0);

The method Sqrt is a class method (or static method) defined in Math class.

Note that the static methods are called using class

names.In fact, no objects have been created for

use.

Static methods have several restrictions:

• They can only call other static methods.

• They can only access static data.

• They cannot refer to this

```
using System;
class Mathoperation
{
    public static float mul(float x, float y);
    {
        return x*y;
    }
    public static float divide(float x, float y)
    {
        return x/y ;
    }
}

class MathApplication
{
    public void static Main( )
    {
        float a = MathOperation.mul(4.0F,5.0F) ;
        float b = MathOperation.divide(a,2.0F) ;
        Console.WriteLine("b = "+ b) ;
    }
}
```

7. Explain properties with example in C#

Properties:

One of the design goals of object-oriented systems is not to permit any direct access todata members because of the implications of integrity.

It is normal practice to provide special methods known as accessor methods to haveaccess to data members.

We must use only these methods to set or retrieve the values of these members.

The drawback with this type of accessor method is users have to remember that theyhave to use accessor methods to work with data members.

To overcome this problem, C# provides a mechanism known as properties that has thesame capabilities as accessor methods but simple to use.

Using a property, a programmer can get access to data members as though they arepublic fields.

Properties are class members, consists of two accessor methods get and set to modify thefield value.

The advantage of using a property is it allows a programmer to validate the client's request for any change in the value of a class field and may allow or reject such a request.

Example:

Accessing private data using accessor methods

```
using System;
class Number
{
    private int number;
    public void SetNumber( int x )    //accessor method
    {
        number = x;                   //private number accessible
    }
    public int GetNumber( )           //accessor method
    {
        return number;
    }
}

class NumberTest
{
    public static void Main ( )
    {
        Number n = new Number ( );
        n.SetNumber (100);   // set value
        Console.WriteLine("Number = " + n.GerNumber( ));    // get value

        // n.number; //Error! Cannot access private data
    }
```

8. Explain indexers with example in C#

Indexers:

An indexer is a class member, which allows us to access member of a class as if it were anarray.

'this' keyword refers to the object instance.

The return type determines what will be returned.

The parameter inside the square brackets is used as

the index.Example:

Implementation of an indexer

```
using System;
using System.Collections;
class List
{
   ArrayList array = new ArrayList ( );
   public object this[ int index ]
   {
    get
    {
          if(index < 0  index >= array.Count)
          {
               return null;
          }
          else
          {
               return (array [index] );
          }
    }
    set
    {
          array[index] = value;
    }
   }
}

class IndexerTest
{
   public static void Main ( )
   {
    List list = new List ( );
    list [0] = "123";
    list [1] = "abc";
    list [2] = "xyz";
    for (int i = 0, i < list.Count; i++)
          Console.WriteLine( list[i]);
   }
}
```

UNIT-5

Inheritance :

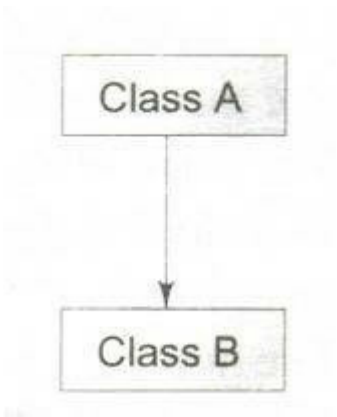
The mechanism of designing or constructing one class from another is calledinheritance.

This may be achieved in two different forms.

- Classical form
- Containment form

Classical form of inheritance:

Inheritance represents a kind of relationship between two classes.Let us consider two classes A and B. We can create a class hierarchy such that B is derived from A as shown in figure.



Class A, the initial class that is used as the basis for the derivedclass is referred to as the base class, parent class or super class.

Class B, the derived class, is referred to as derived class, child class or subclass.

A derived class is a completely new class that incorporates all thedata and methods of its base class. It can also have its own data and method members that are unique to itself. That is, it can enhance the content and behaviour of the base class. We can nowcreate objects of classes A and B independently.

Example:

A a; //

a such cases, we say that the object b is a type of a. Such relationship between a and b is referred to as 'is-a'

i relationship.s Examples of is-a relationship are:

• Dog is-a type of animal

o • Manager is-a type of employee

b • Ford is-a type of car

j The is-a relationship is illustrated in Fig.

e

c

t

o

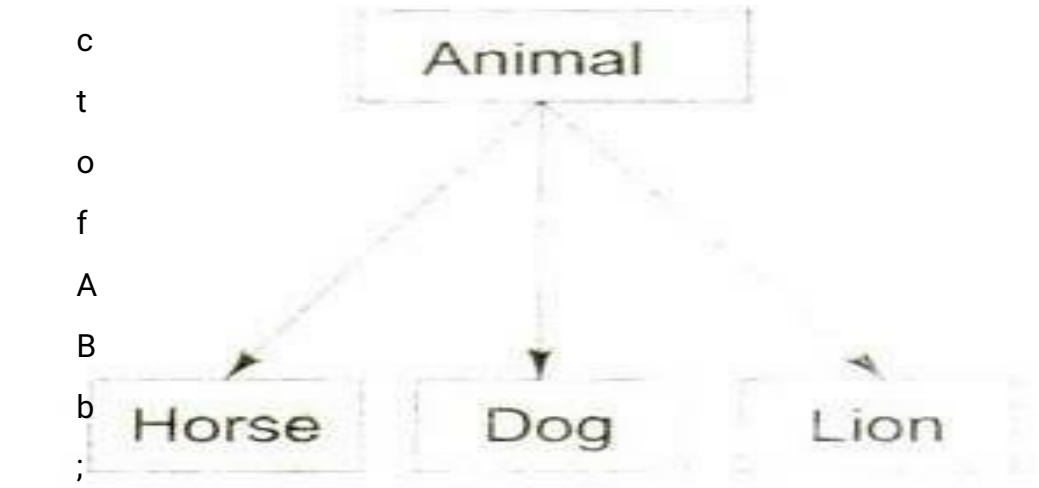
f

A

B

b

;



/

/ Different types of relationship

b C# does not directly implement multiple inheritance. However, this concept is implemented using secondary inheritance path in the

i form ofinterfaces.

s Containment Inheritance: We can also define another form of inheritance relationship known as containership between

o c1assAand B.

b Example:

j

```
class A
{
    . . . .
}
class B
{
    . . . .
    A a; // a is contained in b
}
B b;
. . . .
```

B

I In such cases, we say that the object a is contained in the

object b. This relationship between a and b is referred to as 'has-

n

a'

r
class A ise
l

er class A is termed the 'parent' class and the contained termed a 'child' class.

Examples are:

- Car has-a radio
- House has-a store room
- City has-a road

The has-a relationship is illustrated in Fig.13.4.

Defining a subclass: The definition is very similar to a normal class definition except for the use of colon :and baseclass-name. The colon signifies that the properties of the baseclass are extended to the subclass- name. When implemented the subclass will contain its own members as well those of the baseclass. This kind of situation occurs when we want to add more properties to an existing class without actually modifying it. For example: Visibility Control Class visibility is used to decide which parts of the system can create class objects. A C# class can have one of the two visibility modifiers: public or internal. If we do not explicitly mark the visibility modifier of a class, it is implicitly set to 'internal'; that is, by default allclasses are internal. Internal classes are accessible within the same program assembly and are not accessible from outside the assembly. Classes marked public are accessible everywhere, both within and outside the program assembly. Although classes are normally marked either public or internal, a class mayalso be marked private when it is declared as a member of another class. In such cases, the class behaves like a member of the enclosing class and therefore the modifiers applicable to class members are also applicable to it. It is important to remember that the accessibility domain of a member is never larger than that of the class containing it. Following table gives the visibility domain of members under different combinations of class access specifiers. Defining subclass constructors A subclass constructor is used to construct the instance variables of both the subclass and the superclass. The subclass constructor uses the keyword base to invoke the constructor method of the superclass. Here .the derived constructor takes threearguments, the first two to provide values to the base constructor and the third one to provide value to its own class member. Note that base(x,y) behaves like a method call and therefore arguments are specified without types. The type and order of these arguments must match the type and order of base constructor arguments. When the compiler encounters base(x,y), it passes the values x and y to the base class constructor which takes two int arguments. The base class constructor is called before the derived class constructor is executed. That is, the data members length and breadth are assigned values before the member height is assigned its value. Multilevel Inheritance The class A serves as a base class for the derived class B which in turn serves as a base class for the derived class C. The chain ABC is known as inheritance path. A derived class with multilevel base classes is declared as follows: Hierarchical Inheritance Many programming problems can be cast into a hierarchy where certain features of one level are shared by many others below the level. Following figure shows a hierarchical c1assification of accounts in a commercial bank. This is possible because all the accounts possess certain common features.

2. Explain sealed class with an example

Sealed Classes: Preventing Inheritance Sometimes, we may like to prevent a class being furthersubc1assed for security reasons. A class that cannot be subclassed is called a sealed class.
This is achieved in C# using the modifier sealed as follows: A sealed class cannot also be an

abstract class. Sealed Methods When an instance method declaration includes the sealed modifier, the method is said to be a sealed method. It means a derived class cannot overridethis method.

```
class A
{
    public virtual void Fun( )
    {
    . . . .
    }
}
class B : A
{
    public sealed override void Fun ( )
    {
    . . . .
    }
}
```

3. Explain visibility of class members in C# inheritance

Visibility Control Class visibility is used to decide which parts of the system can create classobjects.

A C# class can have one of the two visibility modifiers: public or internal. If we do not explicitlymark the visibility modifier of a class, it is implicitly set to 'internal'; that is, by default all classes are internal. Internal classes are accessible within the same program assembly and arenot accessible from outside the assembly. Classes marked public are accessible everywhere, both within and outside the program assembly. Although classes are normally marked either public or internal, a class may also be marked private when it is declared as a member of another class. In such cases, the class behaves like a member of the enclosing class and therefore the modifiers applicable to class members are also applicable to it.

Class Members Visibility

As mentioned in the previous chapter, a class member can have any one of the five visibility modifiers:

- public
- protected
- private
- internal
- protected internal

Table 13.1 Visibility of class members

Keyword	Visibility			
	Containing classes	*Derived classes*	*Containing program*	*Anywhere outside the containing program*
Private	✓			
protected	✓	✓		
Internal	✓		✓	
protected internal	✓	✓	✓	
Public	✓	✓	✓	✓

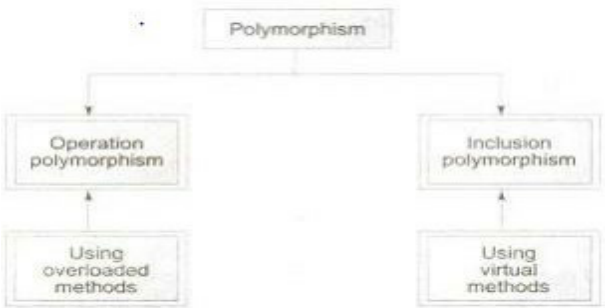
It is important to remember that the accessibility domain of a member is never larger than that ofthe class containing it. Following table gives the visibility domain of members under different combinations of class access specifiers.

Table 13.2 Accessibility domain of class members

Member modifier	*Modifier of the containing class*		
	public	*internal*	*private*
public	Everywhere	only program	only class
internal	only program	only program	only class
private	only class	only class	only class

4. Explain with example how to implement polymorphism in C#

Polymorphism means 'one name, many forms'. Essentially, polymorphism is the capability of oneobject to behave in multiple ways. Polymorphism can be achieved in two ways as shown in Figure. C# supports both of them.



Operation Polymorphism Operation polymorphism is implemented using overloaded methods and operators. The overloaded methods are 'selected' for invoking by matching arguments, in terms of number, type and order. This information is known to the compiler at the time of compilation and, therefore, the compiler is able to select and bind the appropriate method to the object for a particular call at compile time itself. This process is called early binding, or static binding, or static linking. It is also known as compile time polymorphism.

Operation polymorphism

```
using System;
class Dog
{
}
class Cat
{
}
class Operation
{

    static void Call (Dog d)
    {

        Console.WriteLine ("Dog is called");
    }
    static void Call (Cat c)
    {
        Console.WriteLine (" Cat is called ");
    }

    public static void Main( )
    {
        Dog dog = new Dog( );
        Cat cat = new Cat ( );
        Call(dog); //invoking Call( )
        Call(cat); //again invoking Call( )
    }
}
```

Inclusion Polymorphism Inclusion polymorphism is achieved through the use of virtual functions. Assume that the classA implements a virtual method M and classes B and C that arederived from A override the virtual melhod M. When B is cast to A, a call to the method M from A is dispatched to B. Similarly, when C is cast to A a call to M is dispatched to C. The decision on exactly which method to call is delayed until runtime and therefore, it is also known as runtime polymorphism. Since the method is linked with a particular class much later after compilation, this process is termed late binding. It is also known as dynamic binding because the selection of the appropriate method is done dynamically at runtime

Inclusion polymorphism

```
using System;
class Maruthi
{
     public virtual void Display ( ) //virtual method
     {
          Console.WritLine("Maruthi car");
     }
}
class Esteem : Maruthi
{
     public override void Display( )
     {
          Console.WriteLine("Maruthi Esteem");
     }
}
class Zen : Maruthi
{
     public override void Display ( )
     {
          Console.WriteLine("Maruthi Zen");
     }
}
class Inclusion
{
     public static void Main( )
     {
          Maruthi m = new Maruthi ( );
          m = new Esteem ( );  //upcasting
          m.Display ( );
          m = new Zen ( );     //upcasting
          m.Display ( )
     }
}
```

5. Explain exception handling in C#

Exceptions An exception is a condition that is caused by a run-time error in the program. When the C# compiler encounters an error such as dividing an integer by zero, it creates an exception object and throws it to inform us that an error has occurred. If the exception objectis not caught and handled properly, the compiler will display an error message and will terminate the program. If we want the program to continue with the execution of the

remaining code then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task isknown as exception handling. Steps in exception handling :

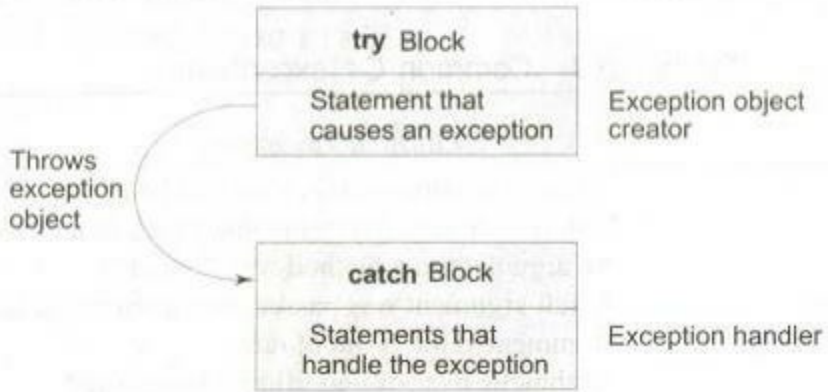
• Find the problem (Hit the exception)

• Inform that an error has occurred (Throw the exception)

• Receive the error information (Catch the exception)

• Take corrective actions (Handle the exception) Syntax of exception handling C# uses a keyword try to preface a block of code that is likely to cause an error condition and 'throw' an exception. A catch block defined by the keyword catch 'catches' the exception 'thrown' bythe try block and handles it appropriately. The catch block is added immediately after the tryblock. The following example illustrates the use of simple try and catch statements: The exception hierarchy All C# exceptions are derived from the class Exception. When an exception occurs, the proper catch handler is determined by matching the type of exception to the name of the exception mentioned. If we are going to catch exceptions at different levels in the hierarchy, we need to put them in the right order.
The rule is that we must always put the handlers for the most derived exception class first. Consider the following code : This code will generate a compiler error, because the exceptionis caught by the first catch (which is a more general one) and the second catch is therefore unreachable. In C#, having unreachable code is always an error. The code must be rewritten as follows: General catch handler A catch block which will catch any exception is called a general catch handler. A general catch handler does not specify any parameter and can be written as: Note that catch (Exception e) can handle all the exceptions thrown by the C# codeand therefore can be used as a general catch handler. However, if the program uses libraries written in other languages, then there may be an exception that is not derived from the class Exception. Such exceptions can be handled by the parameter-less catch statement. This handler is always placed at the end. Since there is no parameter, it does not catch any information about the exception and therefore we do not know what went wrong.
Refer study material for more example :

```
.....
.....
try

  {
      statement;      // generates an exception
  }
  catch (Exception e)
  {
      statement;      // processes the exception
  }
  .....
  .....
```



6. With example explain how to implement user defined exception in C#

Ans: User-defined exception classes are derived from the Exception class. Custom exceptions arewhat we call user-defined exceptions.

In the below example, the exception created is not a built-in exception; it is a customexception –

```
using System;

class Test
{
      public static void Main()
      {
            Console.WriteLine("enter the
            string");string
            s=Console.ReadLine();
            string str=s.ToLower();

            tr
            y
            {     if(str.CompareTo("india")!=0)
                  {
                  //if(s!="INDIA")
```

```
throw (new
MatchNotFoundE
xception("match
not found"));
```

```
                }


            }
            catch(MatchNotFoundException e)
            {
                Console.WriteLine("not found", e.Message);
            }
            Console.WriteLine("found..!\ncontinue");
        }
}

class MatchNotFoundException : Exception
{
    //string message;
    public MatchNotFoundException(string message): base(message)
    {
        //this.message = message;
    }


}
```

7. What is interface? Explain how to define, extend and implement interface in C# with example

C# does not support multiple inheritance. That is, classes in C# cannot have more than onesuperclass.

For instance is not permitted in C#. However, the designers of C# could not overlook theimportance of multiple inheritance.

A large number of real-life applications require the use of multiple inheritance wherebywe inherit methods and properties from several distinct classes.

C# provides an alternate approach known as interface to support the concept of multipleinheritance.

Although a C# class cannot be a subclass of more than one superclass, it can implementmore than one interface, thereby enabling us to create classes that build upon other classes without the problems created by multiple inheritance.

An interface in C# is a reference type. It is basically a kind of class with somedifferences.

Major differences include:

All the members of an interface are implicitly public and abstract

. • An interface cannot contain constant fields, constructors and destructors.

• Its members cannot be declared static.

• Since the methods in an interface are abstract, they do not includeimplementation code.

• An interface can inherit multiple interfaces

Defining an Interface :An interface can contain one or more methods, properties,indexers and events but none of them are implemented in the interface itself. It is the responsibility of the class that implements the interface to define the code for implementation of these members.

The syntax for defining an interface is very similar to that used for defining aclass.

The general form of an interface definition is:

```
interface InterfaceName
{
    Member declarations;
}
```

Here, interface is the keyword and InterfaceName is a valid C# identifier.Remember declarations will contain only a list of members without implementation code.

Given below is a simple interface that defines a single method:

```
interface Show
{
    void Display ( );    // Note semicolon here
}
```

In addition to methods, interfaces can declare properties, indexers and events. Example:

```
interface Example
{
    int Aproperty
    {
        get ;
    }
    event someEvent Changed;
    void Display ( );
}
```

Extending an interface Like classes, interfaces can also be extended. That is, aninterface can be subinterfaced from other interfaces.

The new subinterface will inherit all the members of the superinterface in themanner similar to subclasses.

This is achieved as follows: For example, we can put all members of particular behaviour category in one interface and the members of another category in theother.

Consider the code below:

We can also combine several interfaces together into a single interface. Followingdeclarations are valid: Implementing Interfaces Interfaces are used as 'superclasses' whose properties are inherited by classes. It is therefore necessary tocreate a class that inherits the given interface. This is done as follows: Here the class classname 'implements' the interface interfacename. A more general form ofimplementation may look like this: Abstract class and interface Like any other class, an abstract class can use an interface in the base class list. However, the interface methods are implemented as abstract methods. Example: Note that the class B does not implement the interface method; it simply redeclares as a public abstract method. It is the duty of the class that derives from B to override

and implement the method. Note that interfaces are similar to abstract classes. In fact,

we can convert an interface into an abstract class. Consider the following interface: Now, a class can inherit from B instead of implementing the interface
A. However, the class that inherits B cannot inherit any other class directly. If it isan interface, then the class can not only implement the interface but also use another class as base class thus implementing in effect multiple inheritance

Example :

```
//area
using System;
using
System.Collections.Generic;
using System.Linq;
using System.Text;
namespace
Polymorphism
{
    class Program
    {
        public static void Main(string[] args)
        {
            double a;
            int
            choice=1;
            do
            {
Console.WriteLine("_____");
            Console.WriteLine("Select your choice:\n1.Area of the Circle\n2.Area of the Square\n3.Area
            of
Rectangle\n4.Exit")
;               choice=Int32.Parse(Console.ReadLine());
                Console.WriteLine("___you have entered : "+choice+"____");

                switch(choice)
                {
                    case 1:Circle c=new
                        Circle();a=c.area();
                        Console.WriteLine("Area of the Circle is:"
                        +a);break;
                    case 2:Square s=new
                        Square();a=s.area();
                        Console.WriteLine("Area of the Square is:"
                        +a);break;
                    case 3:Rectangle r=new
                        Rectangle();a=r.area();
                        Console.WriteLine("Area of the Rectangle is:" +a);

                        break;
                    case 4:break;
                }
            }
            while(choice!=4);
        }
    }
    interface Shape
    {
        double area();
    }
    class Circle:Shape
    {
        double pi=3.14;
        public double
        area()
        {
```

```
        int r;
        Console.WriteLine("enter Radius of a
        circle:");r=Int32.Parse(Console.ReadLine());
        return pi*r*r;
}
```

```
        }
        class Square:Shape
        {
                public double area()
                {
                        int side;
                        Console.WriteLine("Enter the length of the
                        side:");side=Int32.Parse(Console.ReadLine());
                        return side*side;
                }
        }
        class Rectangle:Shape
        {
                public double area()
                {
                        int len,brdth;
                        Console.WriteLine("Enter the length of the
                        rectangle:"); len=Int32.Parse(Console.ReadLine());
                        Console.WriteLine("Enter the breadth of the
                        rectangle:");brdth=Int32.Parse(Console.ReadLine());
                        return len*brdth;
                }
        }
}
```

8. What is an event? Explain with example how to define, subscribe and notify an event in C#

An event is an action or occurrence, such as clicks, key presses, mouse movements, or systemgenerated notifications.

Applications can respond to events when they

occur.An example of a notification is interrupts.

Events are messages sent by the object to indicate the occurrence of

the event.Events are an effective mean of inter-process

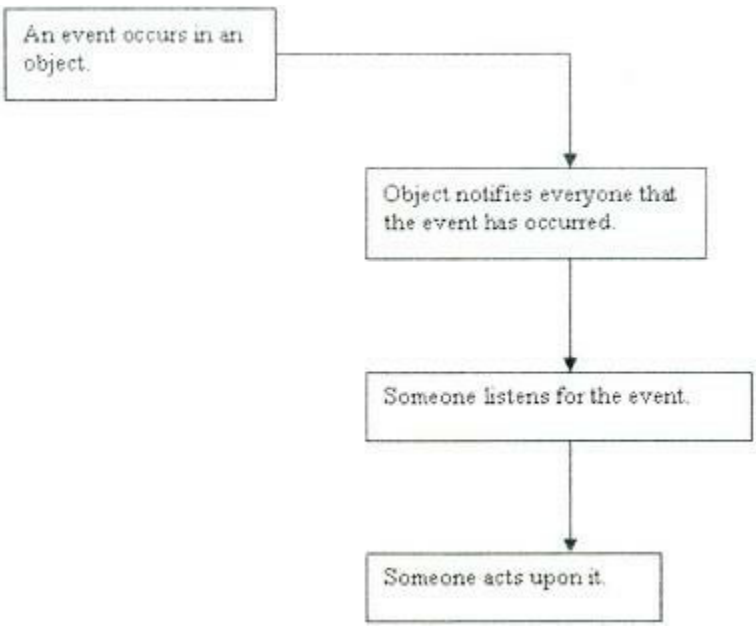
communication.

Consider an example of an event and the response to the event.

A clock is an object that shows 6 AM time and generates an event in the form of an alarm.You accept the alarm event and act accordingly.

The following figure shows the alarm event and handling of the event

v The following figure is the generalized representation that explains events andevent handling.



In C#, delegates are used with events to implement event handling.

Defining an Event The definition of the event in a publisher class includes the declaration of thedelegate as well as the declaration of the event based on the delegate.

The following code defines a delegate named TimeToRise and an event named RingAlarm,which invokes the TimeToRise delegate when it is raised:

```
using System;

namespace Example_4
{
  /// <summary>
  /// The program demonstrates the use of Events.
  /// </summary>

  // First class
  class ClassA
  {
    public void DispMethod()
    {
      Console.WriteLine("Class A has been notified of NotifyEveryOne Event!");
    }
  }
  // Second class
  class ClassB
  {
    public void DispMethod()
    {
      Console.WriteLine("Class B has been notified of NotifyEveryOne Event!");
    }
  }

  class TestEvents
  {
    /// <summary>
    /// The main entry point for the application.
    /// </summary>
    [STAThread]
    static void Main(string[] args)
    {
      // Object of delegate
      Delegate objDelegate = new Delegate();
      // Object of ClassA
      ClassA objClassA = new ClassA();
      // Object of ClassB
      ClassB objClassB = new ClassB();

      // Subscribing to the event
      objDelegate.NotifyEveryOne += new Delegate.MeDelegate(objClassA.
DispMethod);
      objDelegate.NotifyEveryOne += new Delegate.MeDelegate(objClassB.
DispMethod);

      // Invoking method that contains code to raise the event
      objDelegate.Notify();
```

```
    }
}

class Delegate
{
    // Defining a delegate
    public delegate void MeDelegate();
    // Defining an event
    public event MeDelegate NotifyEveryOne;

    public void Notify()
    {
        // If the event is not null
        if(NotifyEveryOne != null)
        {
            Console.WriteLine("Raise Event : ");
            // Raising the event
            NotifyEveryOne();
        }
    }
}
}
```

Subscribing to an Event The event of the publisher class needs to be associated with its handler. The event handler method is associated with the event using the delegate. When thepublisher object raises the event, the subscribing object associates the method, which needs to be called.

Consider a class named Student which contains a method named WakeUp (). The requirement states that the method WakeUp() should be called at 6 AM. The requirement could be implemented using events. The following code shows how the Student class subscribes to the event named TimeToRi se: Notifying Subscribing objects To notify all the objects that have subscribed to an event, the event needs to be raised. Raising an event is likecalling a method. if(condition is true) RaiseEvent(); When even is raised it will invoke all thedelegates of the objects that are subscribed to that particular event.

9. What is delegate? With example explain how to declare, instantiate and use delegates in C#

A delegate is a userdefined reference type contains a reference to the methods in a class.This reference can be changed dynamically at run time as desired.

Using delegates the method to be called at runtime can be identified.

Executing: a delegate will in turn execute the method that it references.

Consider an example of a coffee vending machine, which dispenses different flavors of coffee, such as cappuccino and black coffee. On selecting the desired flavor of coffee, the vending machine decides to dispense the ingredients, such as milk powder, coffee powder,hot water, cappuccino coffee powder. All the materials are placed in different containers inside the vending machine. The required material is dispensed when you select a flavor. Suppose, you select black coffee, the vending machine will call methods to dispense hot water and coffee powder only. The reference to these methods is made dynamically, when you press the required button to dispense black coffee.

To implement delegates in your application you need to declare delegates, instantiatedelegates and use delegates.

Declaring Delegates :The methods that can be referenced by a delegate are determined by the delegate declaration. The delegate can refer to the methods, which have the same signature asthat of the delegate.

Instantiating Delegates: Instantiating delegates means making it point (or refer) to somemethod. We can instantiate delegate by creating delegate object of the delegate type and assign the address of the required method to the delegate object.

Using delegate: Using delegate means invoking a method using delegates. A

delegate iscalled in a manner similar to calling a method.

Types of Delegates: There are two types of delegates,

Single-cast delegate and Multicast delegate.

A Single-cast delegate can call only one method at a time, whereas a Multicast delegate cancall multiple methods at the same time. The following code shows how to use a multicast delegate

```
77 Program to wrctG the duta                    console and file
namespace Chapterl2_Ex2

        ¿ubi .o C c55 FrantTDf•ez•ia*

                atps.lr. SorepmWriter SWriter;
                //Defining a Delegate
                ;›uDliodelegatevoidrttntCatalsLrlngs):

                //Merh‹>d tn print a z"ring to rhe  console
                p u hi i c s cani z ve i d Qtr a ':e C o n sof e (S lr i n q s cr l

                /ZHotñcd to print g g:ring tc a ftle
                pubJic otatc void WritefiJe lstrinq<

                        //Initializing stream objects
                        FStream = new FileStream("c:\\StoreData.txt",
                        FileMode.Append, FileAccess.Write);
                        SHriter = new StreamBriter(FStream),
                        s= s +' File';
                        Z/Writing a strlng to che f:.eSFricer.WriteLine(sl,
                        //remeving the cQntent from the buffer
                        SRritPz.FluZh();
                        SWriter.Close():
                        FZtreom.close();

                //Method to send the4tring data to rcsp‹:ctive mczhodspublic static void
                DisplayDatzlPzint0at6 PNethodl

                        PHe hodl'thin sñould go to khe"l,

                publzc static void Main(l

                        //InStta11zing the Delegate object
                        PritDData lBr i teri l e), Disp3ayDatalMlDelegatt];
                        Ploplegdte -= new PrintData(Sri eFile);
```

10. With an example explain how to work with files in C#.

The following example shows how to use a delegate:

```
// This code is to print data to the output device, which is either a
file or a screen
using System;
using System.IO;

// Program to write the data to the console and file
namespace Chapter12_Ex1
{
public class PrintToDevice
  {
```

```
        //Creating the variables of Stream classes
        static FileStream FStream;
        static StreamWriter SWriter;
        //Defining a Delegate
        public delegate void PrintData(String s);

        //Method to print a string to the console
        public static void WriteConsole (string str)
        {
        Console.WriteLine("{0} Console",str);
        }

        //Method to print a string to a file
        public static void WriteFile (string s)
    {
                //Initializing stream objects
                FStream  = new FileStream("c:\\StoreData.txt",
                FileMode.Append, FileAccess.Write);
                SWriter = new StreamWriter(FStream);
                s= s + " File";
                //Writing a string to the file
                SWriter.WriteLine(s);
                //removing the content from the buffer
                SWriter.Flush();
                SWriter.Close();
                FStream.Close();
    }
         //Method to send the string data to respective methods
        public static void DisplayData(PrintData PMethod)
    {
        PMethod("This should go to the");
    }
        public static void Main()
        {
                //Initializing the Delegate object

                PrintData Cn = new PrintData (WriteConsole);
                PrintData Fl = new PrintData (WriteFile);
                //Invoking the DisplayData method with the Delegate
                object as the argument
                //Using Delegate
                DisplayData (Cn);
                DisplayData (Fl);
                Console.ReadLine();
        }
    }
}
```