UNIT II OBJECT ORIENTED ASPECTS OF C#

Classes-Adding Variables – Adding Methods - Objects – Accessing class Members – Constructors – Abstract classes and Methods - Inheritance - Polymorphism – Interfaces - Operator Overloading - Delegates and Events -Errors and Exceptions.

A sample program in c#

```
using System;

class Program
{
   static void Main()
   {
      Console.WriteLine("Welcome to c#");
   }
}
```

The above program prints Welcome to c# in the console

Every C# program will have a Main method which is the entry point of a program.

Main method will always be static and returns void.It will be defined within a class.

C# is case sensitive.

If multiple mains are vailable only one main method will be invoked first.

Console.writeline is used to print the values. The above program prints Welcome to c# in the console

System is the namespace. If there are any calls to Console.WriteLine, the System namespace is required. Many other common functions require System as well.The syntax is

```
using System;
```

CLASSES- ADDING VARIABLES – ADDING METHODS

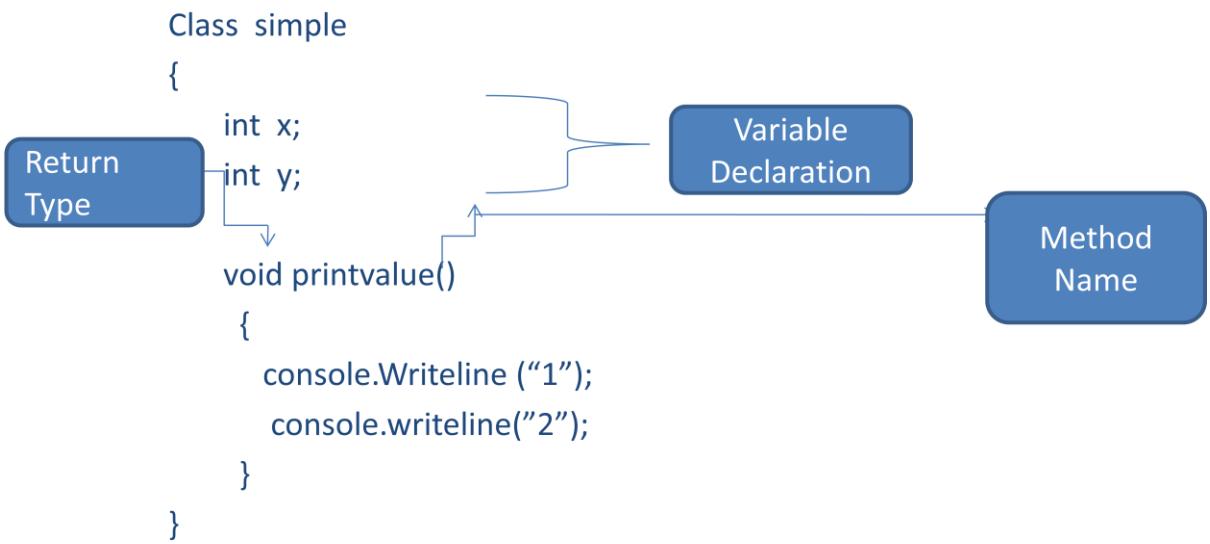
A class is a construct that enables you to create your own custom types by grouping together variables of other types, methods and events. A class is like a blueprint. It defines the data and behavior of a type

A class contains either variables or methods or it can be both.Following is the syntax for a class

SYNTAX:

```
class   classname
    {
        Variable Declaration
        Returntype Methodname (parameter list)
          {
            Statements
          }
 }
```

Based on the above syntax following is an example of class



CREATING OBJECTS

An object is a block of memory that contains space to store all the instance variables.Once a class is declared you can create an instance of that class .

SYNTAX :

Classname objectname;

Objectname = new classname();

Combining the above two statements can be given as

Classnmae objectname = new classname();

Creating objects exapmple

```
Class first
{

}
Class sample
{
  static void Main()
                {
                 first  fr ;
                 fr = new first()
                }
}
```

In the above example sample class contains Main method from where the program starts.first is the name of the class whose objects we have created as fr.Like this we can create any number of instances.In the above program the statement

```
first  fr ;
fr = new first()
```

can also be given as

```
first fr = new first();
```

ACCESSING CLASS MEMBERS

After creating objects,we must now learn to utilise those objects by accessing the class members through them.To access a public instance member outside the class you must use the variable name and member name separated by a dot

SYNTAX :

```
Objectname.variablename
Objectname.methodname();
```

Example 1: Accessing class memebers

```
Class first
{
                int x,y
}
Class sample
{
  static void Main()
                {
                 first  fr = new first()
                 fr.x = 10;
                 fr.y  = 20;
                }
}
```

In the above example first we have created an object fr and with that we access the class members as fr.x and fr.y.

Example 2: Accessing class member and method

```
class program
{
   public int x, y;
   public int add()
   {
      return (x + y);
   }
}
class sample
{
   static void Main()
   {
      program prg1 = new program();
      prg1.x = 85;
      prg1.y = 36;
      Console.WriteLine(prg1.add());

   }
}
```

- In the above program we have two classes. class sample has Main() method.
- Class program has two variables x,y
- Class program has one method add(). It has return type as int.
- Both variables and method are declared as public. We will discuss about public in the next topic under access modifier.
- We create an instance or object of program class in the Main class ,
- so that we can access the variables and methods of program class in main class.
- Dot operator acts as an interface between object and class member.

- Object and instance are same.

We can also create multiple instances. Each instance can have different values with same class variables.

Example_3: Accessing class memebers

```
class program
{
   public int x, y;
   public int add()
   {
      return (x + y);
   }
}
class sample
{
   static void Main()
   {
      program prg1 = new program();
      prg1.x = 85;
      prg1.y = 36;
      Console.WriteLine(prg1.add());
      program prg2 = new program();
      prg2.x = 851;
      prg2.y = 361;
      Console.WriteLine(prg2.add());
   }
}
```

We are using the same variable x and y for storing two different sets of value. we are creating two instances prg1 and prg2 here.

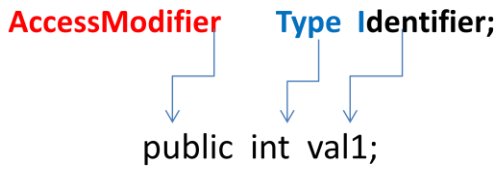
ACCESS MODIFIERS

The access modifier is placed before the declaration
It tells the access level of variables and methods.

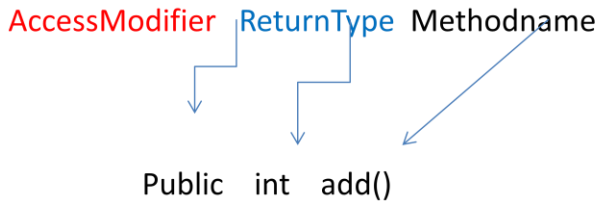
Fields SYNTAX:
Acccessmodifier type identifier;
Method SYNTAX:
Accessmodifier type methodname()

Access

Fields



Methods



Access modifier can be classified as

- Private
- Public
- Protected
- Internal
- Protected Internal

Private

- ❑ Private Members are only accessible within the class they have declared.
- ❑ If it is declared within a method it can be accessed within a method.
- ❑ If a member is declared without an access modifier by default it is private.
- ❑ There is no difference between declaring variables explicitly and implicitly.

Public

- ❑ Public access is the most permissive access level.
- ❑ There are no restrictions on accessing public members.
- ❑ Can be accessed from anywhere.

Example :Public access

```
using System;

class MyClass1
{
   public int x;
```

```
    public int y;
}
class MyClass2
{
    public static void Main()
    {
        MyClass1 mC = new MyClass1();
        // Direct access to public members:
        mC.x = 10;
        mC.y = 15;
        Console.WriteLine("x = {0}, y = {1}", mC.x, mC.y);
    }
}
```

Protected

- A protected member is accessible from within the class in which it is declared, and from within any class derived from the class that declared this member.
- This is mainly used in inheritance where we can access the base class members in derived calss

Internal

- Internal members are accessible only within files in the same assembly.

Declared accessibility	Meaning
Public	Access is not restricted.
Protected	Access is limited to the containing class or types derived from the containing class.
Internal	Access is limited to the current assembly.
protected internal	Access is limited to the current assembly or types derived from the containing class.
Private	Access is limited to the containing type.

CONSTRUCTOR

- ✓ Constructor is a special method that is executed whenever a new instance of the class is executed.
- ✓ Constructors has the same name as the class itself.
- ✓ Constructors do not have return type not even void.
- ✓ A class can have multiple constructors i.e. it supports overloading.
- ✓ Constructor can have parameters. The syntax is same as like methods.

SYNTAX:

```
Class classname
{
    classname()
   {

    }
}
```

There are different types of constructor

1. Default Constructor
2. Parameterized Constructor
3. Copy constructor
4. Static Constructor
5. Private Constructor

Default constructor

- ✓ Also called as Instance constructor
- ✓ A constructor without any parameters is called as default constructor.

Example : Default constructor

```
public class Taxi
{
  public bool isInitialized;
  public Taxi()
  {
    isInitialized = true;
  }
}
class TestTaxi
{
  static void Main()
  {
```

```
        Taxi t = new Taxi();
        Console.WriteLine( t.isInitialized);
    }
}
```

- In the above example Taxi is a constructor where the method name and class name are same
- Since it is a constructor the method Taxi doesn't have any return value.
- The method Taxi need not be called separately in the main method.It will be called when we instantiate itself Taxi t = new Taxi();

Comparison with Constructor

Constructor

```
public class Taxi
{
   public bool isInitialized;
   public Taxi()
   {
      isInitialized = true;
   }
}
class TestTaxi
{
   static void Main()
   {
      Taxi t = new Taxi();
      Console.WriteLine( t.isInitialized);
   }
}
```

Normal

```
using System;

public class Taxi
{
   public bool isInitialized;
   public void Taxivalue()
   {
      isInitialized = true;
   }
}
class TestTaxi
{
   static void Main()
   {
     Taxi t = new Taxi();
      t.Taxivalue ();
     Console.WriteLine( t.isInitialized);
   }
}
```

Parameterised constructor

- ✓ A constructor with at least one parameter is called as parameterized constructor
- ✓ Constructor can have multiple parameters also
- ✓ Supports overloading
- ✓

```
class Class1
{
   int id;
```

```
   string Name;
   public Class1()
   {
      id = 28;
      Name = "Bill Gates";
   }
   public Class1(int val)
   {
      id = val;
      Name = "Steve Jobs";
   }
   public Class1(String name)
   {
      id = 88;
      Name = name;
   }
   public void display()
   { Console.WriteLine("name {0},Id {1}", Name, id); }
}
class Program
{
    static void Main()
          {
            Class1 a = new Class1();
            Class1 b = new Class1(62);
            Class1 c = new Class1("Steve Balmer");
              a.display();
                    b.display();
                    c.display();
    }
     }
```

- In the above example Class1 is the constructor. There are 3 Class1 methods. Each differs by the parameters.
- Class1 a = new Class1(); will call the method class1 without any parameters and will assign the name billgates and id as 28.
- Class1 b = new Class1(62); will pass 62 as parameter and will call class1 which has integer as parameter.

Copy constructor

- ✓ A parameterized constructor that contains a parameter of same class type is called as copy constructor
- ✓ Main purpose of copy constructor is to initialize new instance to the values of an existing instance

Copy constructor: Example .

class Test2

```
{
   int A, B;
   public Test2(int X, int Y)
   {  A = X;
      B = Y;
   }
   //Copy Constructor
   public Test2(Test2 T)
   {  A = T.A;
      B = T.B;
   }
    public void Print()
   {  Console.WriteLine("A = {0}\tB = {1}", A, B);
   }
}
class CopyConstructor
{
   static void Main()
   {
      Test2 T2 = new Test2(80, 90);

      //Invoking copy constructor
      Test2 T3 = new Test2(T2);

      T2.Print();
      T3.Print();
   }
}
```

In the above example the T2 object passed as parameter for the constructor as follows

```
Test2 T3 = new Test2(T2);
```

Static constructor

- ✓ When a constructor is created as static, it will be invoked only once for any number of instances of the class .
- ✓ it is during the creation of first instance of the class or the first reference to a static member in the class the constructor method is invoked.
- ✓ Static constructor is used to initialize static fields of the class and to write the code that needs to be executed only once.
- ✓ A static constructor can not be a parameterized constructor.
- ✓ Within a class you can create only one static constructor.

```
class Test3
{
   public Test3()
   {
```

```
        Console.WriteLine("Instance  Constructor");
    }

    static Test3()
    {
        Console.WriteLine("Static  Constructor");
    }
}
class StaticConstructor
{
    static void Main()
    {
        //Static Constructor and instance constructor, both are invoked for first
instance.
        Test3 T1 = new Test3();

        //Only instance constructor is invoked.
        Test3 T2 = new Test3();

        Console.Read();
    }
}
```

- In the above example Test3 is a constructor
- There are two test3() methods .One is declared as public and another as static
- When we invoke this first time

  Test3 T1 = new Test3();

  Then static Constructor and instance constructor, both are invoked for first instance.

- When we invoke the second time

  Test3 T2 = new Test3();

  Then only instance constructor is invoked.

Private constructor

- ✓ When a class contains at least one private constructor, then it is not possible to create an instance for the class.
- ✓ It is generally used in classes that contain static members only.
- ✓ Private constructor is used to restrict the class from being instantiated when it contains every member as static.

Static

- ✓ Variables and method can be declared as static
- ✓ These Static members retain their value throughout the program
- ✓ No need to create instance.
- ✓ Can be accessed directly by class name instead of creating an object.
- ✓ Static functions can access only static data.
- ✓ Static functions cannot call instance functions.
- ✓ Instance functions can call static functions and access static data.

Example :Static variable and method

```
class X
{

        static public int A;
   static public void Printval()
            {
              Console.WriteLine("Value of A:{0}",A);
             }
}
class program
{
   static void Main()
         {
           X.A = 10;
           X.Printval();
        }
}
```

- Note that printval method and variable A are accessed directly.No instance is created.

Accessing through instance

```
Class X
    {
      static int length;
       static void volume()
         {
         int vol = length*length*length;
          Console.WriteLine("Volume : {0}" , vol);
        }
    }
Class program
{
    Static       void Main()
        {

            x .length=10;
             x.volume()
         }
}
```

Accessing through static

```
Class X
    {
       int length;
       void volume()
         {
         int vol = length*length*length;
          Console.WriteLine("Volume : {0}" , vol);
        }
    }
Class program
{
      Static       void Main()
          {
              X prg = new X();
              prg.length=10;
               prg.volume()
           }
}
```

TYPE CONVERSION

Microsoft .NET provides three ways of type conversion:

1. Parsing
2. Convert Class
3. Explicit Cast Operator ()

Parsing : Parsing is used to convert string type data to primitive value type. For this we use parse methods with value types.

```
class Program
{
  static void Main(string[] args)
  {
    int number;
    float weight;

    Console.Write("Enter any number : ");
    number = int.Parse(Console.ReadLine());

    Console.Write("Enter your weight : ");
    weight = float.Parse(Console.ReadLine());

    Console.WriteLine("You have entered : " + number);
    Console.WriteLine("You weight is : " + weight);
  }
}
```

In the above example int.parse is used to convert to integer.

In the above example float.parse is used to convert to float.

Convert Class : One primitive type to another primitive type. This class contains different static methods like ToInt32(), ToInt16(), ToString(), ToDateTime() etc used in type conversion.

```
class Program
{
  static void Main()
  {
    string num = "23";
    int number = Convert.ToInt32(num);

    int age = 24;
    string vote = Convert.ToString(age);

    Console.WriteLine("Your number is : " + number);
    Console.WriteLine("Your voting age is : " + age);
  }
}
```

Convert.tostring is used to convert to string.

Explicit Cast Operator : In general this operator is used with non-primitive types to up level or down level casting. But it can also used with any type having type compatibility and data type compatibility.

```
class Program
{
  static void Main(string[] args)
  {
    int num1, num2;
    float avg;
    num1 = 10;
    num2 = 21;
    avg = (float)(num1 + num2) / 2;

    Console.WriteLine("average is : " + avg);
  }
}
```

Float is typecasted to convert it to float.

INHERITANCE

- ✓ Inheritance allows a class to extend its functionality by making its features to be used in the derived class
- ✓ The class which extends its functionality for reuse by other classes is known as Base class.
- ✓ Class which inherits the functionality of the base class along with its unique features is called Derived class
- ✓ The class member and method should be declared either public or protected in inheritance to be accessed in derived class.

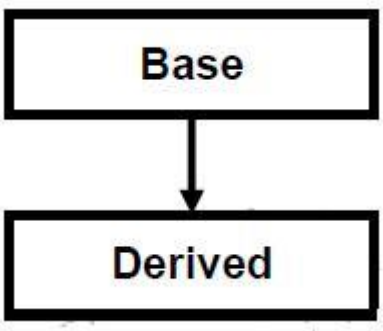
Following are the types of inheritance

1. Single Inheritance
2. Multi Level Inheritance
3. Hierarchical Inheritance
4. Multiple Inheritance

Single Inheritance

When a single derived class is created from a single base class then the inheritance is called as single inheritance.

The inheritance is represented by :basecalss in the derived class



Example : Single Inheritance

```
using System;

  class Base
  {
    int A, B;
    public void GetAB()
    {
```

```
        Console.Write("Enter Value For A : ");
        A = int.Parse(Console.ReadLine());
        Console.Write("Enter Value For B : ");
        B = int.Parse(Console.ReadLine());
    }

    public void PrintAB()
    {
        Console.Write("A = {0}\tB = {1}\t", A, B);
    }
}
class Derived : Base
{
    int C;
    public void Get()
    {
        //Accessing base class method in derived class
        GetAB();

        Console.Write("Enter Value For C : ");
        C = int.Parse(Console.ReadLine());
    }

    public void Print()
    {

        //Accessing base class method in derived class
        PrintAB();

        Console.WriteLine("C = {0}", C);

    }
}

class MainClass
{

    static void Main(string[] args)
    {

        Derived obj = new Derived();
        obj.Get();
        obj.Print();

        //Accessing base class method with derived class instance
        obj.PrintAB();

        Console.Read();
    }
```

```
}
```

- Baseclass is the the class base and derived class is class derived.
- The base class is inherited in the derived class by giving
  class Derived : Base
- GetAB(),PrintAB() are the two methods in the base class.Since it is inherited these methods are used directly in the derived class.
- Whenever an instance is created for the derived class,it can access the baseclass objects also.
- With the same object obj we can access the baseclass methods and variables also

Output

Enter Value For A : 77

Enter Value For B : 88

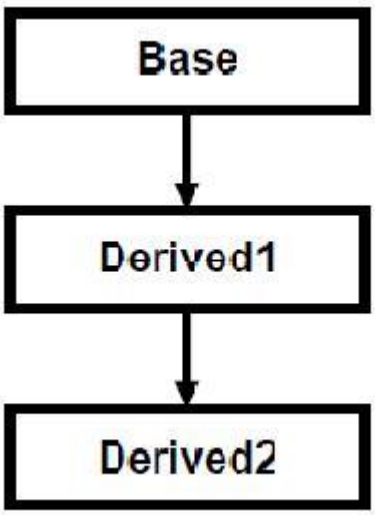
Enter Value For C : 99

A = 77 B = 88 C = 99

A = 77 B = 88

Multi Level Inheritance

- ✓ When a derived class is created from another derived class , then that inheritance is called as multi level inheritance.
- ✓ The fig shows two level , but can be up to any level .



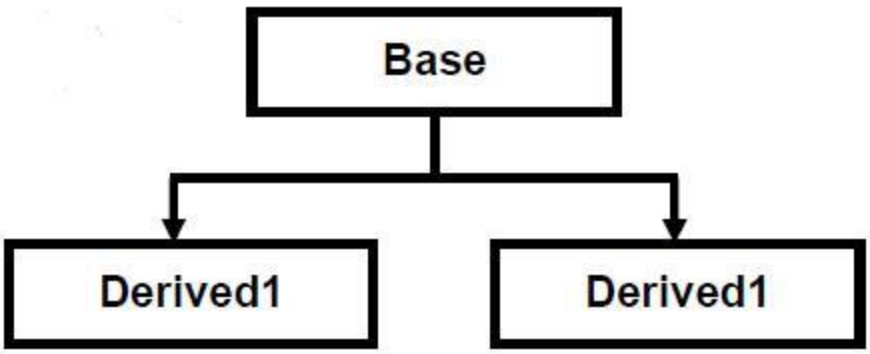
Example : Multilevel Inheritance

```
class car
{
        public void display()
        {
                Console.WriteLine("This is the car class and Display method");
        }
}
class maruti:car
{
        public void print()
        {
                Console.WriteLine("This is the maruti class and print method");
        }
}
class esteem:maruti
{
        public void view()
        {
                Console.WriteLine("This is the esteem class and view method");
                }
}

class program
{
  static void Main()
  {

    maruti mrt = new maruti();
    esteem etm = new esteem();
  }
}
```

Through mrt we can access
        mrt.print()
        mrt.display()
Through etm we can access
        etm.display()
        etm.print()
        etm.view()

Hierarchical Inheritance

- ✓ When more than one derived class are created from a single base class, then that inheritance is called as hierarchical inheritance.
- ✓ There can be second level from deriverd1 where derived1 acts as base class



Example : Hierarchical Inheritance

```
class car
{
        public void display()
        {
                Console.WriteLine("Car --> Base Class");
        }
}
class Santro:car  //Extending class car
{
        public void print()
        {
                Console.WriteLine("Santro --> Derived class of car class");
        }
}
class Maruti : car   //Extending class car
{
  public void view()
  {
    Console.WriteLine("Maruti --> Derived class of car class");
  }
}
  class program
  {
    static void Main()
    {

      Santro snt = new Santro();
      Maruti mrt = new Maruti();
    }
  }
```

Through snt we can access

snt.display()

snt.print()

Through mrt we can access

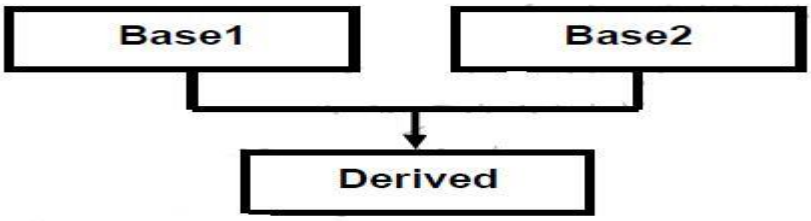
mrt.display()

mrt.view()

Multiple Inheritance

When a derived class is created from more than one base class then that inheritance is called as multiple inheritance.

But multiple inheritance is not supported by .net using classes and can be done using interfaces



ABSTRACT CLASS AND METHODS

- ✓ An abstract class is declared using the abstract modifier
- ✓ Abstract methods have no implementations.
- ✓ The implementation logic is provided by classes that derive from them.
- ✓ In the implementation class use the Override keyword
- ✓ This is mainly used when a common base class is shared by multiple derived classes
- ✓ You cannot create instances of abstract class. Only derived class can be Instantiated
- ✓ An abstract class can contain abstract members and normal members
- ✓ Any class derived from abstract class must implement all the abstract members of the class using override keyword

Abstract : SYNTAX

```
Public abstract class classname
{
  classmembers;
  public abstract void methodname (parameterlist);

}
```

Abstract implementation : SYNTAX

```
Class classname
{
   public override void methodname(parameterlist);
                              {
                statements
                              }
}
```

Example1 : abstract

```
abstract class Test
{
   public int temp = 10;
   public abstract void A();
}

class Example1 : Test
{
   public override void A()
   {
      Console.WriteLine("Classname : Example1.A ");
      Console.WriteLine("Value of temp is " + temp);
   }
}

class Example2 : Test
{
   public override void A()
   {
      Console.WriteLine("Example2.A");
      Console.WriteLine("ValueType of temp is " + temp);
   }
}

class Program
{
   static void Main()
   {
      // Reference Example1 through Test type.
      Example1 test1 = new Example1();
      test1.A();

      // Reference Example2 through Test type.
      Example2 test2 = new Example2();
      test2.temp = 20;
      test2.A();
   }
}
```

- Abstract test class has the signature of the abstract class test.
- The implementation of test is done in class Example1 by using override keyword.
- In the above program there are two implementation for the abstract method A()

Example2 : abstract

```
abstract class abclass
{
   public void identifyBase()
   {
      Console.WriteLine("Normal Method");
   }

   abstract public void identifyderived();
}
class derivedclass : abclass
{

   override public void identifyderived()
   {
      Console.WriteLine("Iam a derived class");
   }
}

class example
{
   static void Main()
   {
      derivedclass b = new derivedclass();
      b.identifyBase();
      b.identifyderived();
   }
}
```

- The abstract class abclass has two methods.One method is normal method identifyBase and the other method is abstract method identifyderived.
- The implementation of identifyderived is done in derived class.

POLYMORHISM OVERLOADING AND OVERRIDING

Polymorhism means "one thing in many different forms"
Overloading and overriding are used to implement polymorphism.

Polymorphism is classified into

1. Compile time polymorphism or early binding or static binding
2. Runtime polymorphism or late binding or dynamic binding

Compile time polymorphism

- ✓ The polymorphism in which compiler identifies which polymorphic form it has to execute at compile time itself is called as compile time polymorphism or early binding.
- ✓ Advantage of early binding is execution will be fast. Because every thing about the method is known to compiler during compilation it self
- ✓ Examples of early binding are overloaded methods, overloaded operators and overridden methods that are called directly by using derived objects.

Example1 : Compiletime polymorphism(overloading)

```
class program
{
   public void add(int x)
   {
      Console.WriteLine("two integer parameters " + (x + 1));
   }
   public void add(int x, float y)
   {
      Console.WriteLine("one integer  and one float parameters " + (x + y));
   }
   public void add(float x, int y)
   {
      Console.WriteLine("one float  and one integer parameters " + (x + y));
   }
}
class sample
{
   static void Main()
   {
      program prg = new program();
      prg.add(7);
      prg.add(20, 5f);
      prg.add(10.5f, 10);
   }

}
```

- add method is overloaded with integer parameter
- add method is overloaded with integer and float parameter
- add method is overloaded with float and integer paramter(order changed)

Run time polymorphism

✓ The polymorphism in which compiler identifies which polymorphic form to execute at runtime but not at compile time is called as runtime polymorphism or late binding.

✓ Advantage of late binding is flexibility and disadvantage is execution will be slow as compiler has to get the information about the method to execute at runtime.

✓ Example of late binding is overridden methods that are called using base class object.

Method Overriding

✓ Creating a method in derived class with same signature as a method in base class is called as method overriding.
✓ Same signature means methods must have same name, same number of arguments and sametype of arguments.
✓ Method overriding is possible only in derived classes, but not within the same class.
✓ When derived class needs a method with same signature as in base class, but wants to execute different code than provided by base class then method overriding will be used.
✓ To allow the derived class to override a method of the base class, C# provides two options,**virtual** methods and **abstract** methods.

Example1 : Runtime polymorphism(overriding)

```
class BaseClass
{
   public virtual  string YourCity()
   {
      return "New York";
   }
}
class DerivedClass : BaseClass
{
   public override string YourCity()
   {
      return "London";
   }
}
class Program
{

   static void Main(string[] args)
```

```
    {
        DerivedClass obj = new DerivedClass();
        string city = obj.YourCity();
        Console.WriteLine(city);
        Console.Read();
    }
}
```

INTERFACE

- An interface is a reference type that represent set of function members but does not implement them.
- It contains only the signature of the methods
- The implementation of the method is done in the class that implements the interface
- It is declared using interface keyword
- The class implementing an interface must define all the methods contained inside the interface. Otherwise compiler raises error messages

Example1 : Interface

```
class implementationclass : isampleinterface1, isampleinterface2
{
    public void sample()
    {
        //method implementation
    }

    void isampleinterface1.samplemethod()
    {
        //method implementation
    }
    void isampleinterface2.samplemethod()
    {
        //method implementation
    }
    static void Main()
    {
        implementationclass impcls = new implementationclass();
        impcls.sample();

        isampleinterface1 obj1 = (isampleinterface1)impcls;
        isampleinterface2 obj2 = (isampleinterface2)impcls;


        obj1.samplemethod();
```

```
        obj2.samplemethod();
    }
}
```

Example2 : Interface

```
interface vehicle

{
        void display();
        void print();
}

interface car
{
        void print();
}

class esteem :vehicle,car
{
        public void display()
        {
        Console.WriteLine("This is a implementation of interface vehicle");
        }

        void vehicle.print()
        {
        Console.WriteLine("This is a explicit implementation of interface vehicle");

        }
        void car.print()
        {
        Console.WriteLine("This is a explicit implementation of interface car");

        }
}

class program
{
   static void Main()
   {
      esteem etm = new esteem();
      etm.display();

      vehicle vhl = (vehicle)etm;
      car cr = (car)etm;
      vhl.print();
      cr.print();
```

```
    }
}
```

Difference between interface and Abstract.

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'nt contains Constructors	Abstract class contains Constructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member
An interface cannot have access modifiers by default everything is assumed as public	An abstract class can contain access modifiers for the subs, functions, properties
Member of interface can not be Static	Only Complete Member of abstract class can be Static

DELEGATES AND EVENTS

In C#, delegate is a *reference to the method*. It works like *function pointer* in C and C++. But it is objected-oriented, secured and type-safe than function pointer.

For static method, delegate encapsulates method only. But for instance method, it encapsulates method and instance both.

The best use of delegate is to use as event.

Internally a delegate declaration defines a class which is the derived class of **System.Delegate**.

C# Delegate Example

```
using System;
delegate int Calculator(int n);//declaring delegate

public class DelegateExample
{
  static int number = 100;
```

```
    public static int add(int n)
    {
      number = number + n;
      return number;
    }
    public static int mul(int n)
    {
      number = number * n;
      return number;
    }
    public static int getNumber()
    {
      return number;
    }
    public static void Main(string[] args)
    {
      Calculator c1 = new Calculator(add);//instantiating delegate
      Calculator c2 = new Calculator(mul);
      c1(20);//calling method using delegate
      Console.WriteLine("After c1 delegate, Number is: " + getNumber());
      c2(3);
      Console.WriteLine("After c2 delegate, Number is: " + getNumber());

    }
  }
```

ERRORS AND EXCEPTIONS.

Exception Handling in C# is *a process to handle runtime errors*. We perform exception handling so that normal flow of the application can be maintained even after runtime errors.

In C#, exception is an event or object which is thrown at runtime. All exceptions the derived from *System.Exception* class. It is a runtime error which can be handled. If we don't handle the exception, it prints exception message and terminates the program.

Advantage

It *maintains the normal flow* of the application. In such case, rest of the code is executed event after exception

C# Exception Classes

All the exception classes in C# are derived from **System.Exception** class. Let's see the list of C# common exception classes.

Exception	Description
System.DivideByZeroException	handles the error generated by dividing a number with zero.
System.NullReferenceException	handles the error generated by referencing the null object.
System.InvalidCastException	handles the error generated by invalid typecasting.
System.IO.IOException	handles the Input Output errors.
System.FieldAccessException	handles the error generated by invalid private or protected field access.

C# Exception Handling Keywords

In C#, we use 4 keywords to perform exception handling:

try
catch
finally, and
throw

C# try/catch

In C# programming, exception handling is performed by try/catch statement. The **try block** in C# is used to place the code that may throw exception. The **catch block** is used to handled the exception. The catch block must be preceded by try block.

```
using System;
public class ExExample
{
  public static void Main(string[] args)
  {
    int a = 10;
    int b = 0;
    int x = a/b;
    Console.WriteLine("Rest of the code");
  }
}
```

C# try/catch example

```
using System;
public class ExExample
{
  public static void Main(string[] args)
```

```
  {
    try
    {
      int a = 10;
      int b = 0;
      int x = a / b;
    }
    catch (Exception e) { Console.WriteLine(e); }

    Console.WriteLine("Rest of the code");
  }
}
```

C# finally

C# finally block is used to execute important code which is to be executed whether exception is handled or not. It must be preceded by catch or try block.

C# finally example if exception is handled

```
using System;
public class ExExample
{
  public static void Main(string[] args)
  {
    try
    {
      int a = 10;
      int b = 0;
      int x = a / b;
    }
    catch (Exception e) { Console.WriteLine(e); }
    finally { Console.WriteLine("Finally block is executed"); }
    Console.WriteLine("Rest of the code");
  }
}
```

C# User-Defined Exceptions

C# allows us to create user-defined or custom exception. It is used to make the meaningful exception. To do this, we need to inherit Exception class.

C# user-defined exception example

```
using System;
public class InvalidAgeException : Exception
{
```

```
  public InvalidAgeException(String message)
    : base(message)
  {

  }
}
public class TestUserDefinedException
{
  static void validate(int age)
  {
    if (age < 18)
    {
      throw new InvalidAgeException("Sorry, Age must be greater than 18");
    }
  }
  public static void Main(string[] args)
  {
    try
    {
      validate(12);
    }
    catch (InvalidAgeException e) { Console.WriteLine(e); }
    Console.WriteLine("Rest of the code");
  }
}
```

WRITE A PROGRAM TO FIND LARGEST ARRAY ELEMENT AND AVERAGE OF ARRAY ELEMENTS VIA METHODS

```
using System;
class ArrayFunction
{
public static void Main()
{
long Largest;
double Average;
int c;
int num;
int[] array1;
```

```
Console.Write("Enter the number of Elements in an Array : ");
c=int.Parse(Console.ReadLine());
array1=new int[c];
for (int i=0 ; i>c ;i++)
{
Console.WriteLine("Enter the element " + i);
num=int.Parse(Console.ReadLine());
array1[i]=num;
}
foreach (int i in array1)
{
Console.Write(" " + i);
}
Console.WriteLine ();
Largest = Large(array1);
Average = Avg(array1);
Console.WriteLine ("\n The largest element in the array is " +
Largest);
Console.WriteLine ("The Average of elements in the array is " +
Average);
Console.ReadLine();
}
// Determining the largest array element
static int Large (params int [] arr)
{
int temp=0;
for ( int i = 0; i < arr.Length; i++)
{
if (temp <= arr[i])
{
temp = arr[i];
}
}
return(temp);
}
// Determining the average of array elements
static double Avg (params int [] arr)
{
double sum=0;
for ( int i = 0; i < arr.Length; i++)
```

```
{
sum = sum + arr[i];
}
sum = sum/arr.Length;
return(sum);
}
}
```

Output:

```
Enter the number of Elements in an Array : 5
Enter the element 1 : 5
Enter the element 2 : 7
Enter the element 3 : 3
Enter the element 4 : 1
Enter the element 5 : 8
largest element in the array is 8
The Average of elements in the array is 4.8
```

2 MARKS

1. What is a class?

   A class is a construct that enables you to create your own custom types by grouping together variables of other types, methods and events. A class is like a blueprint. It defines the data and behavior of a type

2. Define Delegate.

   A delegate is a type that represents references to methods with a particular parameter list and return type. Delegates are used to pass methods as arguments to other methods.

3. What is meant by protected in access modifier?

A protected member is accessible from within the class in which it is declared, and from within any class derived from the class that declared this member.

4. How to parse a string to integer?

   - int.Parse
   - Convert.ToInt32

5. Name the types of polymorphism.

- Compile time polymorphism or early binding or static binding
- Runtime polymorphism or late binding or dynamic binding

6. State any two difference between abstract and Interface

Interface	Abstract class
Interface support multiple inheritance	Abstract class does not support multiple inheritance
Interface does'n Contains Data Member	Abstract class contains Data Member
Interface does'nt contains Constructors	Abstract class contains Constructors
An interface Contains only incomplete member (signature of member)	An abstract class Contains both incomplete (abstract) and complete member

7. List out the different types of constructor

- Default Constructor
- Parameterized Constructor
- Copy constructor
- Static Constructor
- Private Constructor

8. What is the use of break statement ?

When the break statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop. It can be used to terminate a case in the switch statement.

9. List atleast 4 Exceptions that occur commonly in C# Program

Exception	Description
System.DivideByZeroException	handles the error generated by dividing a number with zero.
System.NullReferenceException	handles the error generated by referencing the null object.

System.InvalidCastException	handles the error generated by invalid typecasting.
System.IO.IOException	handles the Input Output errors.
System.FieldAccessException	handles the error generated by invalid private or protected field access.

10. Mention any four compile time error and runtime error

Compile time error

- Syntax errors
- Typechecking errors
- Compiler crashes

Runtime error

- Division by zero
- Dereferencing a null pointer
- Running out of memory