**Use of oops**

To make more complex with easier development

**Advantage**: Reuse of code

**(OOP) concepts in C#:** Abstraction, Encapsulation, Inheritance and Polymorphism.

**Class**

* It is a collection of objects
* That contains variables for storing data and functions to perform operations on the data.
* A class will not occupy any memory space
* Hence, it is only a logical representation of data.
* To create a class, you simply use the keyword "class" followed by the class name:
* Class Employee   
  {   
  }

**Object**

* It is a real time entity
* They may represent a person, a place or any item
* An object is an instance of a class
* When an object is created using the new operator, memory is allocated for the class in the heap, the object is called an instance and its starting address will be stored in the object in stack memory.
* When an object is created without the new operator, memory will not be allocated in the heap, in other words an instance will not be created and the object in the stack contains the value null.
* class Employee  
  {   
  }

Syntax to create an object of class Employee:  
   
Employee objEmp = new Employee();

**Abstraction:**

* Abstraction is a process of hiding the implementation details and displaying the necessary features or providing relevant information
* abstraction is implemented using interface and abstract class
* The advantage of abstraction is that every user will get his own view of the data according to his requirements and will not get confused with unnecessary data
* Define common definition of class that multiple derived class can share

class Program

{

           static void Main(string[] args)

           {

                Hyundai hyn = new Hyundai();

                String descp = hyn.Describe();

                Console.WriteLine(descp);

                Console.ReadKey();

            }

 }

abstract class Car

{

              public virtual string Describe()

              {

                      return "Description of the car";

               }

   }

class Hyundai : Car

{

         public override string Describe()

         {

                 return "Description of the car is now Hyundai";

          }

 }

**Encapsulation**

* Encapsulation is used for hide the code and data in a single unit to protect the data from the outside the world.
* Encapsulation is implemented using access modifier.
* Class is the best example of encapsulation
* Encapsulation is a technique used to protect the information in an object from another object.
* What are the access modifier

**Access modifier**

There are 4 access modifiers (public, protected, internal, private) which defines the 6 accessibility levels as follows:

* public
* protected
* internal
* protected internal
* private
* private protected

|  |  |
| --- | --- |
| Modifier | Description |
| Public | There are no restrictions on accessing public members. |
| Private | Access is limited to within the class definition. This is the default access modifier type if none is formally specified |
| protected | Access is limited to within the class definition and any class that inherits from the class |
| Internal | Access is limited exclusively to classes defined within the current project assembly |
| protected internal | Access is limited to the current assembly and types derived from the containing class. All members in current project and all members in derived class can access the variables. |
| private protected | Access is limited to the containing class or types derived from the containing class within the current assembly. |

**Inheritance**

* Create the new class from existing class or base class..
* so it can access all properties of base class
* purpose--- code reusability
* When a class includes a property of another class it is known as inheritance.
* Types of inheritance
* For example, a child includes  the properties of its parents

public class ParentClass

    {

        public ParentClass()

        {

            Console.WriteLine("Parent Constructor.");

        }

        public void print()

        {

            Console.WriteLine("I'm a Parent Class.");

        }

    }

    public class ChildClass : ParentClass

    {

        public ChildClass()

        {

            Console.WriteLine("Child Constructor.");

        }

        public static void Main()

        {

            ChildClass child = new ChildClass();

            child.print();

        }

    }

**Polymorphism**

* one object behaving diff form at diff situations
* its implemented using overload & override
* overload--->in a class having 2 or more methods with name & diff parameter
* override---> in derived class v will create method which is already

Implemented in base class with same signature.

* Polymorphism has ability to provide different implementation of methods that are implemented with same name

**Virtual method** - this will have implementation but can also be overridden in the derived class to provide additional logic or completely replace its logic

**Pure virtual function** is must to be overridden in derived class

When its declared as 0 its called pure virtual function

Class base

{

Virtual void rint(); //virtual method

Virtual void print1()=0; // pure virtual method

}

**Difference b/w new and override keyword**

* New – hides the base class function
* Override - override the base class function
* Both the key words used in derived class to implement the virtual method

**Private constructor**

* If u declare constructor as private , it does not allow to create object for derived class
* It loose inherits properties
* We can use private cons.. in singleton pattern
* It throws compile error in derived class

**Constructor**

* Used for initialising the members of class
* Whenever an object is created , default value for initialization
* If we not creating any const.. CLR automatically created constructor. That is called as default constructor.
* Do not have return type
* Can’t be overloaded
* Any no of cons..with diff parameters

**Diff B/w server. Transfer and response .redirect**

Response. Redirect –

res.redir(“google.com”)

url will change

Server. Transfer-

server.redirect(“login.aspx”)

url never change

**Can u override the private virtual method?**

No u cannot inherit

**Static methods can use non-static members.**

No

**What is the difference between a class and a structure?**

**Class**:

* A class is a reference type.
* While instantiating a class, CLR allocates memory for its instance in heap.
* Classes support inheritance.
* Variables of a class can be assigned as null.
* Class can contain constructor/destructor.
* **Structure**:
* A structure is a value type.
* In structure, memory is allocated on stack.
* Structures do not support inheritance.
* Structure members cannot have null values.
* Structure does not require constructor/destructor and members can be initialiazed automatically

**Define enumeration**

Enumeration is defined as a value type that consists of a set of named values. These values are constants and are called enumerators. An enumeration type is declared using the enum keyword. Each enumerator in an enumeration is associated with an underlying type that is set, by default, on the enumerator. The following is an example that creates an enumeration to store different varieties of fruits:  
  
enum Fruits {Mango, Apple, orange, Guava};   
  
In the preceding example, an enumeration Fruits is created, where number 0 is associated with Mango, number 1 with Apple, number 2 with Orange, and number 3 with Guava. You can access the enumerators of an enumeration by these values

Enums are value types and are created on the stack and not on the heap.

**Differentiate between an abstract class and an interface.**

Abstract Class:

* Can have some of methods have implementations
* Can have fields
* Can inherit other abstract class .and inherit other interface class
* Class can’t inherit multilple abstract class
* Have access modiiers

Interface

* Can have no implementations
* No fields
* Can inherit other interface only .can’t inherit abstract class
* Class can inherit multilple inteface
* No ..default its public

| **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'n contains Cunstructors | Abstract class contains Cunstructors |
| 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 |

**When do you really need to create an abstract class**

We define abstract classes when we define a template that needs to be followed by all the derived classes.

**Delegates**

It holds the address or reference of single or more method

Signature of delegate should match with method signature(void)

Suppose if you have multiple methods with same signature (return type & number of parameters) and want to call all the methods with single object then we can go for delegates.

Delegates are two types

      -   Single Cast Delegates

      -  Multi Cast Delegates

**Single Cast Delegates**

Single cast delegate means which hold address of single method like as explained in above example.

**Multicast Delegates**

Multi cast delegate is used to hold address of multiple methods in single delegate. To hold multiple addresses with delegate we will use overloaded += operator and if you want remove addresses from delegate we need to use overloaded operator -=

public delegate void MultiDelegate(int a,int b);

public class Sampleclass

{

public static void Add(int x, int y)

{

Console.WriteLine("Addition Value: "+(x + y));

}

public static void Sub(int x, int y)

{

Console.WriteLine("Subtraction Value: " + (x - y));

}

public static void Mul(int x, int y)

{

Console.WriteLine("Multiply Value: " + (x \* y));

}

}

class Program

{

static void Main(string[] args)

{

Sampleclass sc=new Sampleclass();

MultiDelegate del = Sampleclass.Add;

del += Sampleclass.Sub;

del += Sampleclass.Mul;

del(10, 5);

Console.ReadLine();

}

}

**Output**

Whenever we run above code we will get output like as shown below

|  |
| --- |
| **Addition Value : 15**  **Subtraction Value : 5**  **Multiply Value : 50** |

**Call by value and call by reference**

**Method 1**:   
If you want to keep the original image as well as the edited image. What you'll do is make a copy of the image, modify it and save it.  
This is call by value method - You make a copy of the actual parameters, perform operations on those copies but do not modify the actual parameters.  
  
**Method 2**:   
If you want to modify the original image permanently. What you will do is access the image from its location (reference), modify it and save it.  
This is call by reference - you use references to access the actual parameters, perform operations on those parameters using those references and hence end up modifying the actual parameters.

**Types of Constructors**

Basically constructors are 5 types those are

      1.    Default Constructor

      2.    Parameterized Constructor

      3.    Copy Constructor

      4.    Static Constructor

      5.    Private Constructor

**Extension Method**

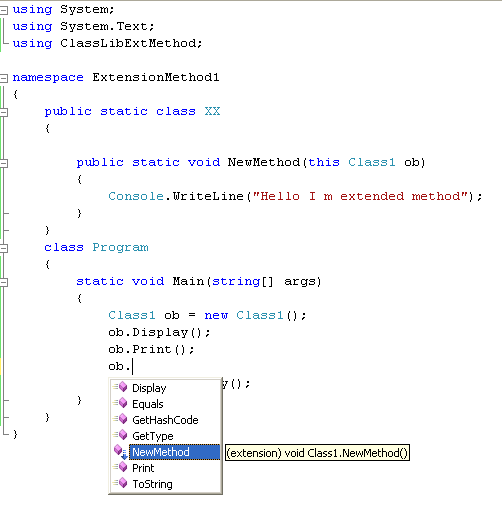
<https://www.c-sharpcorner.com/uploadfile/puranindia/extension-methods-in-C-Sharp-3-0/>

extension method concept allows you to add new methods in the existing class or in the structure without modifying the source code of the original type and you do not require any kind of special permission from the original type and there is no need to re-compile the original type

Let us discuss this concept with the help of an example. Suppose you have a class or a structure which contains three methods and you want to add two new methods in this class or structure, you did not have the source code of the class/structure, or do not have permissions from the class/structure, or the class is a sealed class, but you still want to add new methods in it, then you can use the concept extension method to add the new method in the existing class/structure.

An extension method is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended

1. using System;
2. using System.Text;
4. namespace ClassLibExtMethod
5. {
6. public class Class1
7. {
8. public string Display()
9. {
10. return ("I m in Display");
11. }
13. public string Print()
14. {
15. return ("I m in Print");
16. }
17. }
18. }



* Extension methods allow existing classes to be extended without relying on inheritance or having to change the class's source code.
* If the class is sealed than there in no concept of extending its functionality. For this a new concept is introduced, in other words extension methods

**SOLID principles**

<https://www.c-sharpcorner.com/UploadFile/damubetha/solid-principles-in-C-Sharp/>

**Collections**

<https://www.c-sharpcorner.com/UploadFile/736bf5/collection-in-C-Sharp/>

Data manipulation includes adding, removing, finding, and inserting data in the collection. Collection types implement the following common functionality:

* Adding and inserting items to a collection
* Removing items from a collection
* Finding, sorting, searching items
* Replacing items
* Non-generic                          Generic
* ArrayList     ------------->          List
* HashTable  ------------->          Dictionary
* SortedList   ------------->          SortedList
* Stack           ------------->          Stack
* Queue         ------------->          Queue