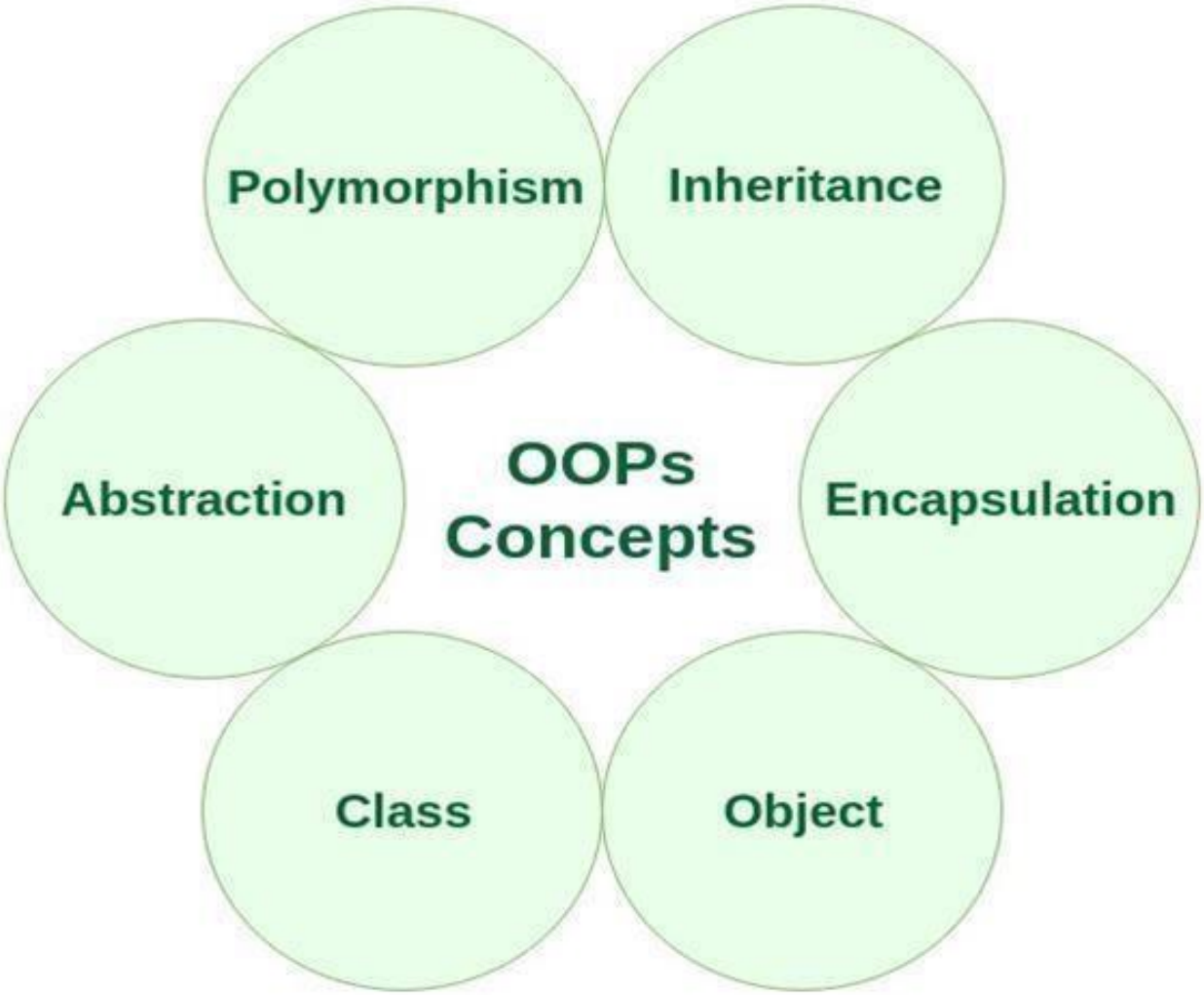
OOP


Polymorphism
Inheritance
Abstraction
OOPs Concepts
Encapsulation
Class
Object

Access Specifier

Class Name

Field

Property

Method

```
public class Employee
{
    private int experience;

    public int Experience
    {
        get
        {
            return experience;
        }
        set
        {
            experience = value;
        }
    }

    public void CalculateSalary()
    {
        int salary = Experience * 300000;

        Console.WriteLine("salary:{0} ", salary);
    }
}
```

Object

```
public class Company
{
    public static void Main()
    {
        Employee obj = new Employee();

        obj.Experience = 3;

        obj.CalculateSalary();
    }
}
```

Encapsulation

```
public class Student
{
    private string StudentName;

    public string Name
    {
        get { return studentName; }

        set { studentName = value; }
    }
}
```

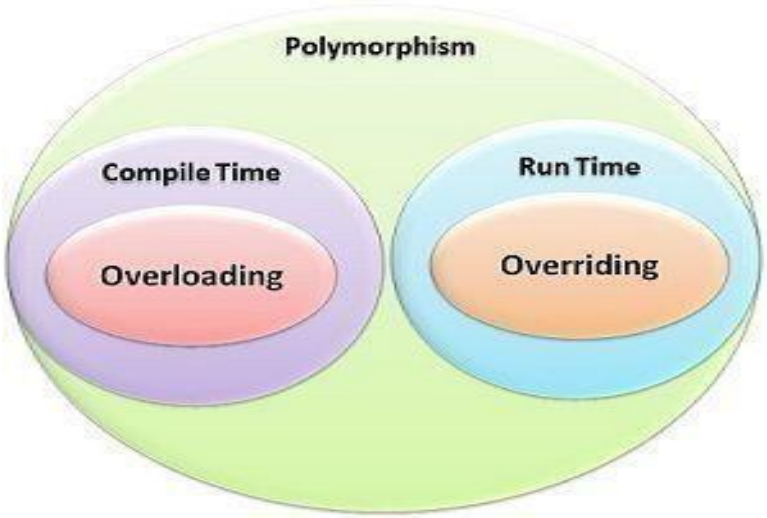
This field cannot be accessed from outside without the property

```
class DemoEncapsulation {
    // Main Method
    static public void Main()
    {
        // creating object
        Student obj = new Student();
        obj.Name = "Ankita";
        // Displaying values of the variables
        Console.WriteLine("Name: " + obj.Name);
    }
}
```

Polymorphism

```
public class TestData
{
    public int Add(int a, int b, int c)
    {
        return a + b + c;
    }
    public int Add(int a, int b)
    {
        return a + b;
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        TestData dataClass = new TestData();

        int add2 = dataClass.Add(10, 20, 30);

        Console.WriteLine(add2);

        int add1 = dataClass.Add(10, 20);

        Console.WriteLine(add1);
    }
}
//Output: 60 30
```



Overriding & Overloading

```
public class Methodoveloading
  {
    public int add(int a, int b)
    {
        return a + b;
    }

    public int add(int a, int b, int c)
    {
        return a + b + c;
    }

    public float add(float a, float b, int c)
    {
        return a + b + c;
    }

    public float add(float a, int c, float b)
    {
        return a + b + c;
    }
  }
```

1. Number of parameters are different

2. Type of parameters are different

2. Order of parameters are different

تفاوت بين overriding & overloading

```
class baseClass
{
    public virtual void Greetings()
    {
        Console.WriteLine("baseClass Saying Hello!");
    }
}
class subClass : baseClass
{
    public override void Greetings()
    {
        Console.WriteLine("subClass Saying Hello!");
    }
}
class Program
{
    static void Main(string[] args)
    {
        baseClass obj1 = new subClass();

        obj1.Greetings();
    }
}
//Output: subClass Saying Hello!
```

Same method name but one is in base class and another is in derived class

This will call subclass Greetings() method because of overriding.

تفاوت بين overriding & method hiding

```
public class BaseClass
{
    public virtual void Print()
    {
        Console.WriteLine("Base Class Print Method");
    }
}
public class DerivedClass : BaseClass
{
    public override void Print()
    {
        Console.WriteLine("Child Class Print Method");
    }
}
public class Program
{
    public static void Main()
    {
        BaseClass B = new DerivedClass();
        B.Print();
    }
}
```

```
public class BaseClass
{
    public virtual void Print()
    {
        Console.WriteLine("Base Class Print Method");
    }
}
public class DerivedClass : BaseClass
{
    public new void Print()
    {
        Console.WriteLine("Child Class Print Method");
    }
}
public class Program
{
    public static void Main()
    {
        BaseClass B = new DerivedClass();
        B.Print();
    }
}
```

مزایا و معایب OOP

- استفاده مجدد از کدها
- انعطاف پذیری
- امنیت برنامه و دیتا
- توسعه آسان
- عیب یابی آسان

تفاوت بین Abstract class و Interface

```
// abstract class 'Car'
public abstract class Car {
    // abstract method
    public abstract void Engine();

    public void Dashboard()
    {
        Console.WriteLine("Dashboard");
    }
}
```

Method Declared

Method Defined

سازی

```
interface Car {
    // methods having only
    //declaration not definition
    void Engine();

    void Dashboard();
}
```

Only method Declaration is allowed

ک ن نن د

```
//Interfaces use - all methods declared
interface IWorldTaxSystem
{
    //Just declared because it is different
    //for an individual and for a company
    //will be defined in derived class
    public abstract int CalculateTax();

    //Currency will be as per country
    //therefore it declared only
    public string TaxCurrency();
}
```

```
//Abstract class use - Some mehods decla
public abstract class USATaxSystem
{
    //Just declared because it is differ
    //for an individual and for a compan
    //will be defined in derived class
    public abstract int CalculateTax();

    //Always will be in USD(Dollar) ther
    public string TaxCurrency()
    {
        return USD;
    }
}
```

چرا Interface ؟ !!!

آیا Interface میتونه

Constructor داشته باشه ؟آیا

و

Abstrac

t Class

میشه ا ز

نمونه

سازی

کرد ؟

Interfa

ce

¿out - ref - params

Params

```
class InterviewHappy
{
    static void Main(string[] args)
    {
        // Calling function by passing 5
        // arguments as follows
        int y = Add(12,13,10,15,56);

        // Displaying result
        Console.WriteLine(y);
    }
    // function containing params parameters
    public static int Add(params int[] ListNumbers)
    {
        int total = 0;
        // foreach loop
        foreach(int i in ListNumbers)
        {
            total += i;
        }
        return total;
    }
}
```

You can pass any number of parameters here.

Out و ref

به وسیله کلمه کلیدی های ref و out میتوانیم به وسیله رفرنس ها پارام ت دهیم

```
public static void Main(string[] args)
{
    int a = 10;
    int b = 5;

    OutRefExample p = new OutRefExample();

    int c = p.Update( a, b );

    Console.WriteLine(c);
}

public class OutRefExample
{
    public int Update( int c, int d )
    {
        return c + d;
    }
}
```

ارسال پارام ت از طریق value

```
public static void Main(string[] args)
{
    int a;

    int b = 5;

    OutRefExample p = new OutRefExample();

    p.Update( out a, ref b );

    Console.WriteLine("out value: {0}", a);

    Console.WriteLine("ref value: {0}", b);
}

public class OutRefExample
{
    public void Update( out int a, ref int b )
    {
        a = 10;

        b = 20;
    }
}
```

1. No need to initialize out parameter before passing it.

1. Must initialize ref parameter else error.

2. Out parameter must be initialized before returning.

2. Initialize not necessary, you can comment this line still fine.

Extension ها یچ هس نت و چه زما
Methods

ن

ی باید ازشون استفاده بشه ؟

```
public class Program
{
    public static void Main(string[] args)
    {
        string test  = "HellowWorld";

        Console.WriteLine( test.Substring(0, 5) );

        Console.WriteLine(test.Right(5));
    }
}
```

Right() method is no
present in String clas
But we can add it by
using extension
method

```
public static class StringExtensions
{
   public static string Right(this string s, int count)
   {
        return s.Substring(s.Length - count, count);
   }
}
```

سطح دس تش کلاس ها (ACCESS SPECIFIERS) ؟

Access Specifier	Inside Same Assembly where member is declared			Other Assembly where containing Assembly is referenced	
	Inside Same Class	Inside Derived Class	Other Code	Inside Derived Class	Other Code
Public	✓	✓	✓	✓	✓
Private	✓	✗	✗	✗	✗
Internal	✓	✓	✓	✗	✗
Protected	✓	✓	✗	✓	✗
ProtectedInternal	✓	✓	✓	✓	✗

Internal is the default access modifier of a class.

```
??? class Program
 {
     public static void Main(string[] args)
     {
         string test  = "HellowWorld";

         Console.WriteLine(test);
     }
 }
 //Output: HelloWorld
```

Constructor چیه و چند نوع دارد ؟

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

```
public class Program
{
    public Program()
    {
        //Logic written here is automatically
        //called whenever you will
        //create object of this Program class...

        Console.WriteLine ("Hello Constructor");
    }
    public static void Main(string[] args)
    {
        Program p = new Program();
    }
}
```

Constructor

```
class addition
{
    int a, b;
    public addition()   //default contructor
    {
        a = 100;
        b = 175;
    }

    public static void Main()
    {
        //an object is created , constructor is called
        addition obj = new addition();
        Console.WriteLine(obj.a);
        Console.WriteLine(obj.b);
        Console.Read();
    }
}
```

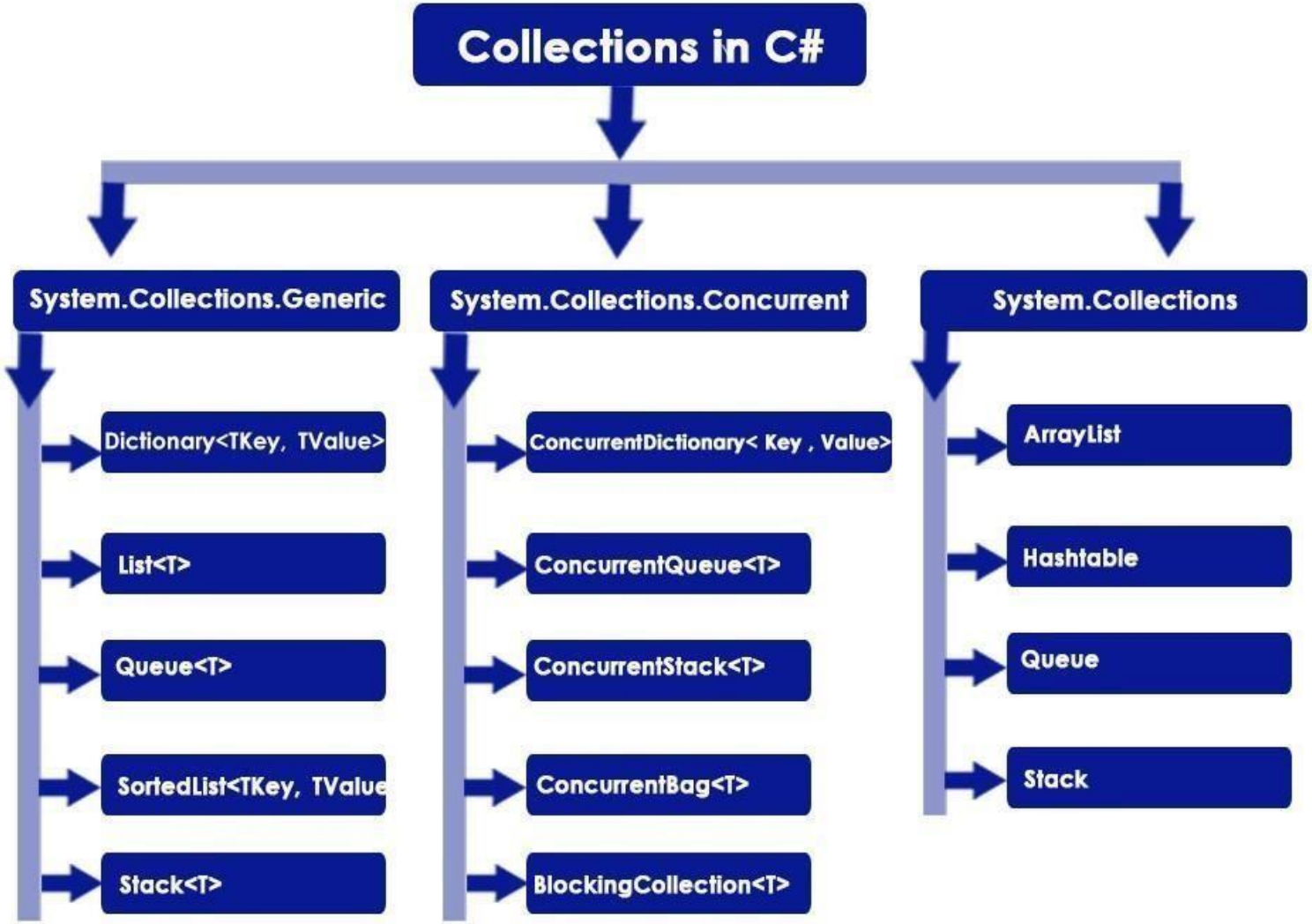
```
class paraconstrctor
{
  public  int a, b;
  // decalaring Paremetrized Constructor with ing x,y parameter
  public paraconstrctor(int x, int y)
    {
        a = x;
        b = y;
    }
}
class MainClass
{
    static void Main()
    {
        paraconstrctor v = new paraconstrctor(100, 175);   // Creating
            object of Parameterized Constructor and ing values
        Console.WriteLine("-----------parameterized constructor example by
            vithal wadje---------------");
        Console.WriteLine("\t");
        Console.WriteLine("value of a=" + v.a );
        Console.WriteLine("value of b=" + v.b);
        Console.Read();
    }
}
```

```
class Test1
{
    //Static constructor
    static Test1()
    {
        Console.WriteLine("Static Constructor Called");
    }

    public static void print()
    {
        Console.WriteLine("Print Method Called");
    }

    public static void Main(string[] args)
    {
        Test1.print();
    }
}

//OUTPUT:
//Static Constructor Called
//Print Method Called
```

```
public class Counter
{
    private Counter()   //private constrctor declaration
    {
    }
    public static int currentview;
    public static int visitedCount()
    {
        return ++ currentview;
    }
}
class viewCountedetails
{
    static void Main()
    {
        // Counter aCounter = new Counter();   // Error
        Console.WriteLine("-------Private constructor ---");
        Console.WriteLine();
        Counter.currentview = 500;
        Counter.visitedCount();
        Console.WriteLine("view count is: {0}", Counter.currentview);
        Console.ReadLine();
    }
}
```

```
class employee
{
    private string name;
    private int age;
    public employee(employee emp)   // declaring Copy constructor.
    {
        name = emp.name;
        age = emp.age;
    }
    public employee(string name, int age)  // Instance constructor.
    {
        this.name = name;
        this.age = age;
    }
    public string Details     // Get deatils of employee
    {
        get
        {
            return  " The age of " + name +" is "+ age.ToString();
        }
    }
}
class empdetail
{
    static void Main()
    {
        employee emp1 = new employee("Vithal", 23);  // Create a new employee object.
        employee emp2 = new employee(emp1);          // here is emp1 details is copied to emp2.
        Console.WriteLine(emp2.Details);
        Console.ReadLine();
    }
}
```

سوالات مرتبط به Collections ، IEnumerable و IQueryable


Collections in C#
System.Collections.Generic
Dictionary<TKey, TValue>
List<T>
Queue<T>
SortedList<TKey, TValue
Stack<T>
System.Collections.Concurrent
ConcurrentDictionary< Key , Value>
ConcurrentQueue<T>
ConcurrentStack<T>
ConcurrentBag<T>
BlockingCollection<T>
System.Collections
ArrayList
Hashtable
Queue
Stack

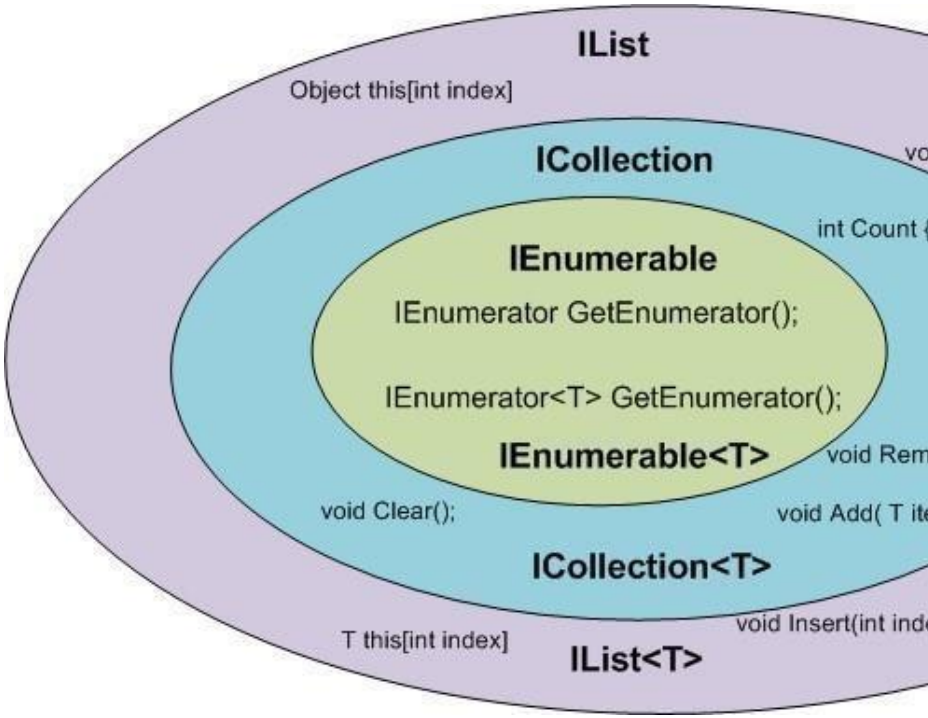
IEnumerable

```
public class Program
{
    public static void Main()
    {
        var students = new List<Student>() {
                new Student(){ Id = 1, Name="Bill" },
                new Student(){ Id = 2, Name="Steve" }
        };

        foreach(var student in students)
        {
            Console.WriteLine(student.Id + ", " +student.Name);
        }
    }
}
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

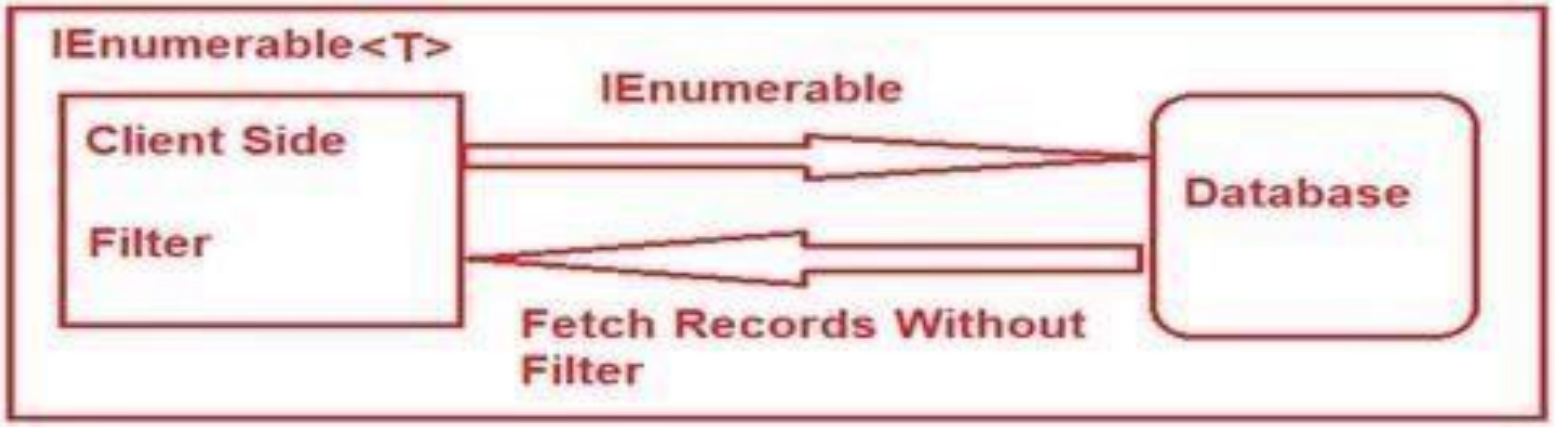
```
public class List<T> : System.Collections.Generic.IC
System.Collections.Generic.IEnumerable<T>, System.Co
System.Collections.Generic.IReadOnlyCollection<T>,
System.Collections.Generic.IReadOnlyList<T>, System.
```

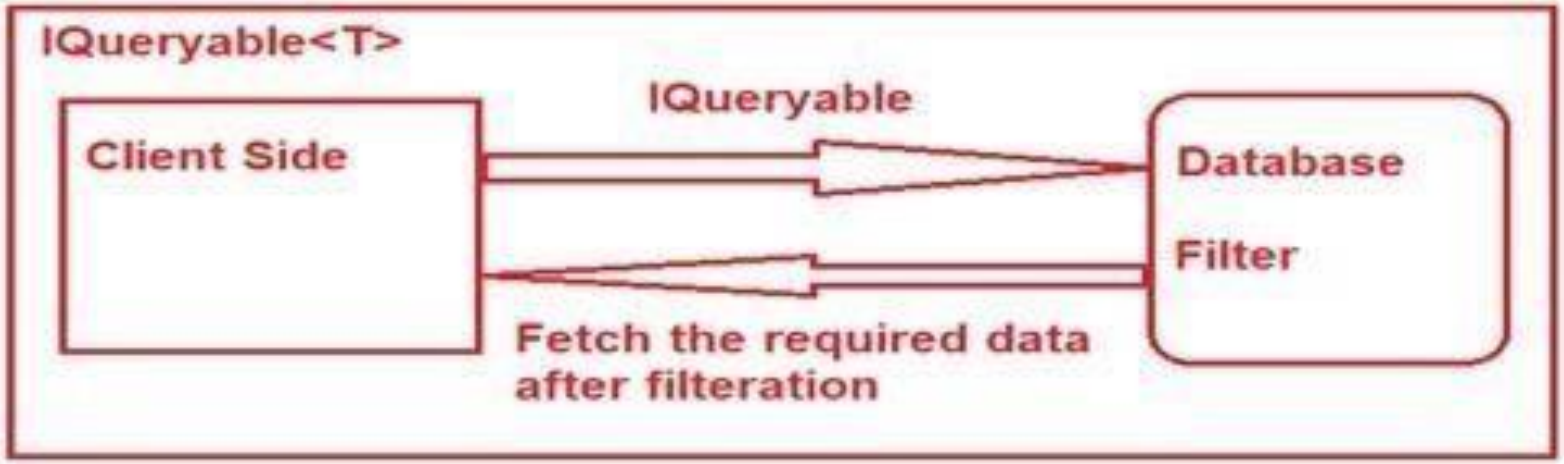
IEnumerable و IEnumeratorتفاوت

```
public class Program
{
    public static void Main()
    {
        var students = new List<Student>() {
                new Student(){ Id = 1, Name="Bill" },
                new Student(){ Id = 2, Name="Steve" }
        };

        foreach(var student in students)
        {
            Console.WriteLine(student.Id + ", " +student.Name);
        }
    }
}
public class Student
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```



IEnumerable و Iqueryableتفاوت





سوالات مرتبط به Delegate

```
delegate int Calculator(int x, int y);//declaring delegate
```

```
public class DelegateExample
{
    public static int add(int a, int b)
    {
        return a + b;
    }
    public static int mul(int a, int b)
    {
       return a * b;
    }
```

```
    public static void Main(string[] args)
    {
       Calculator c1 = new Calculator(add); //Instantiating delegate

       int result = c1(20, 30); //calling method using delegate

       Console.WriteLine(result);
    }
    //Output: 50
}
```

```
public static void Main(string[] args)
{
    int result;

    if(operation = "add")
    {
      result = DelegateExample.add(20, 30);
    }
    else if(operation = "mul")
    {
      result = DelegateExample.mul(20, 30);
    }

    Console.WriteLine(result);
}
```

Multicast Delegate چیه ؟

```
namespace MulticastDelegateDemo
{
    public class Rectangle
    {
        public void GetArea(double Width, double Height)
        {
            Console.WriteLine(@"Area is {0}", (Width * Height));
        }
        public void GetPerimeter(double Width, double Height)
        {
            Console.WriteLine(@"Perimeter is {0}", (2 * (Width + Height)))
        }
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle();

            rect.GetArea(10, 20);

            rect.GetPerimeter(10, 20);
        }
    }
    //output
    //Area is 200
    //Perimeter is 60
}
```

```
namespace MulticastDelegateDemo
{
    public delegate void RectangleDelegate(double W
    public class Rectangle
    {
        public void GetArea(double Width, double He
        {
            Console.WriteLine(@"Area is {0}", (Widt
        }
        public void GetPerimeter(double Width, doub
        {
            Console.WriteLine(@"Perimeter is {0}",
        }
        static void Main(string[] args)
        {
            Rectangle rect = new Rectangle();

            RectangleDelegate rectDel = new Rectang

            //Chaining of delegates
            rectDel += rect.GetPerimeter;

            rectDel.Invoke(20, 30); //output: Areas
        }
    }
}
```

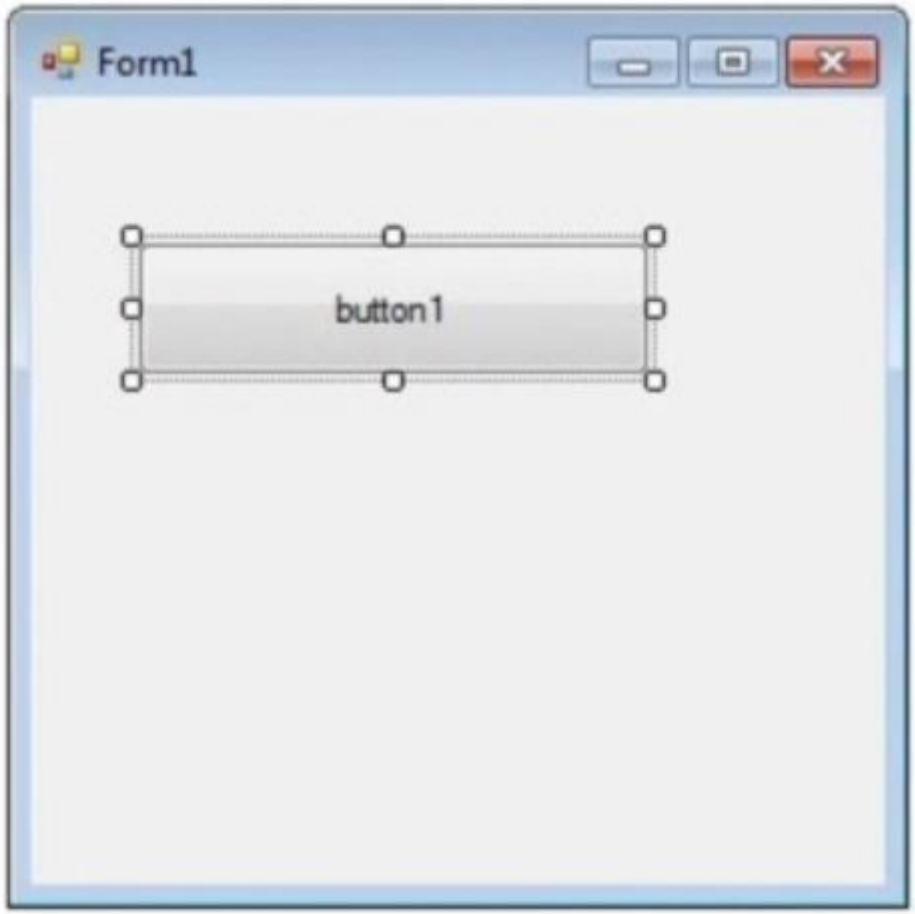
چیه ؟

Anonymous Delegate

```
public delegate void AnonymousDelegate();

public class Program
{
    static void Main()
    {
        AnonymousDelegate delObject = delegate()
        {
            //Inline content of the method;
            Console.WriteLine("Anonymous Delegate method");
        };

        delObject();
    }
}
//Output: Anonymous Delegate method
```

تفاوت Delegate و Event ؟

Functions → Delegate → Event



سوالات مرتبط به this، using ، is و as

کلمه کلیدی this

```
Student {
blic int id;
blic String name;

blic Student(int id, String name) {
 id = id;
 name = name;


blic void showInfo() {
 Console.WriteLine(id + " " + name);
```

```
public Student(int id, String na
    this.id = id;
    this.name = name;
}
```

مفهومکلمه کلیدی using

```
using System.IO; using
System.Text;
```

```
using(var connection = new SqlConnection("{your-connection-string}"))
{
    var query = "UPDATE YourTable SET Property = Value WHERE Foo = @foo";
    using(var command = new SqlCommand(query,connection))
    {
        connection.Open();
        // Perform your update
        command.ExecuteNonQuery();
    }
}
```

تفاوت ب نت is و as

```
P o1 = new P();
P1 o2 = new P1();
Console.WriteLine(o1 is P);

//output: true
```

```
object[] o = new object[1];
o[0] = "Hello";
string str1 = o[0] as string;
Console.Write(str1);

//output: Hello
```

سوالات مرتبط ب هvar - dynamic - const - readonly

const و readonlyتفاوت ب نت

```
class InterviewHappy {
    // Constant fields
    public const int myvar = 10;

    static public void Main()
    {
        Console.WriteLine(myvar);
    }
}
//Output: 10
```

```
class Example {
    // readonly variables
    public readonly int myvar1;
    public readonly int myvar2 = 200;

    public Example(int b)
    {
        myvar1 = b;
        Console.WriteLine(myvar1);
        Console.WriteLine(myvar2);
    }

    static public void Main()
    {
        Example obj1 = new Example(100);
    }
}

//Output: 100 200
```

1. Using readonly fields, we can assign values in DECLARATION as well as in the CONSTRUCTOR PART.

تفاوت ب ن ت var و dynamic

```
public static void Main(string
{
    dynamic a = 10;

    a = "Happy";

    Console.WriteLine(a);
}
//Output: Happy
```

```
public class HelloWorld
{
    public static void Main(string[] args)
    {
        var a = 10;

        a = "Happy"; //Build error,
        //error CS0029: Cannot implicitly convert type 'string' to 'int'

        dynamic b = 10;

        b = "Happy"; //No Build error
    }
}
```

سوالات مرتبط با stirng

string های مختلف Operation

Concatenate:

```
string str1 = "This is one"; string str2 = "This is two"; string str2 = str1
+ str2;
```

Modify:

```
string str1 = "This is one"; string str2 = str1.Replace("one", "two");
```

Contains:

```
string str = "This is test"; if (str.Contains("test"))

Console.WriteLine("The 'test' was found.");
```

Trim:

Space های اضافه را از آخر string پاک میکند

stringBuilder و string تفاوت ب ن ن

```
public class HelloWorld
{
    public static void Main(string[] args)
    {
        String str1 = "Interview";

        String str2 = "Happy";

        str1 = str1 + str2;

        Console.WriteLine(str1);
    }
}
```

Both these strings are different and occupy different memory in process

```
public static void Main(string[] args)
{
    StringBuilder str1 = new StringBui

    String str2 = "Happy";

    str1.Append(str2);

    Console.WriteLine(str1);
}
```

Generics

Generics

```
namespace InterviewHappy
{
    public class GenericExample
    {
        public static void Main(string[] args)
        {
            bool Equal = Calculator.AreEqual(4, 4);
            bool strEqual = Calculator.AreEqual("Interview", "Happy");
            Console.WriteLine(Equal);
            Console.WriteLine(strEqual);
        }
    }
    public class Calculator
    {
        public static bool AreEqual(int value1, int value2)
        {
            return value1.Equals(value2);
        }
    }
}
```

Will work

Will not work

```
public class Calculator
{
    public static bool AreEqual(object v
    {
        return value1.Equals(value2);
    }
}
```

```
namespace InterviewHappy
{
    public class GenericExample
    {
        public static void Main(string[] args)
        {
            bool Equal = Calculator.AreEqual<int>(4, 4);
            bool strEqual = Calculator.AreEqual<string>("Interview", "Happy");
            Console.WriteLine(Equal);
            Console.WriteLine(strEqual);
        }
    }
    public class Calculator
    {
        public static bool AreEqual<T>(T value1, T value2)
        {
            return value1.Equals(value2);
        }
    }
}
```

Will work

Will work

```
namespace InterviewHappy
{
    public class GenericExample
    {
        public static void Main(string[] args)
        {
            bool Equal = Calculator<int>.AreEqual(4, 4);
            bool strEqual = Calculator<string>.AreEqual("Interview", "Happy");
            Console.WriteLine(Equal);
            Console.WriteLine(strEqual);
        }
    }
    public class Calculator<T>
    {
        public static bool AreEqual(T value1, T value2)
        {
            return value1.Equals(value2);
        }
    }
}
```

Will work

Will work

نوع null

```
public class InterviewHappy
{
    public static void Main(string[] args)
    {
        // this will give compile time error
        int j = null;

        // Valid declaration
        Nullable<int> j = null;

        // Valid declaration
        int? j = null;
    }
}
```

اصول SOLID

The Single Responsibility

```
d InsertData()

og("Log successfully");



d WriteLog(string message)

Line("Logged Time:" + DateTime.Now.ToLongTimeString() + message);
```

Principle

یک کلاس باید فقط یک دلیل برای بوجود آمدنداشته باشد و فقط یک وظیفه به آن داده شو د.

The Open Closed Principle

شما باید بتوانید عملکرد کلاس را بدون تغی ی آن گس ریش دهید

```
public abstract class Shape
{
    public abstract double Area();
}

public class Rectangle : Shape
{
    public double Width { get; set; }
    public double Height { get; set; }
    public override double Area()
    {
        return Width * Height;
    }
}

public class Circle : Shape
{
    public double Radius { get; set; }
    public override double Area()
    {
        return Radius * Radius * Math.PI;
    }
}
```

یا به عبار ی کلاس نسبت به تغی یات باید بسته باشد و نسبت بهعملیات جدید باز باش د .

The Liskov Substitution Principle

شامل دارای کلاس مشتق شده از والد

```
public class User
{
    public string Name { set; get; }
}

public class Customer : User
{
    public int CustomerType { set; get; }
}

public class Admin : User
{
    public int Role { set; get; }
}
```

بایداطمینان حاصل کنید که کلاس مشتق شده تاث ییدر عملکرد کلاس والد ندارد و بتوانید کلاس هایمشتق شده از والد را در جای والد تعویض کنی د.

The Interface Segregation Principle

ر ر

ی

```
class ToyPlane implements IToy, IMovable, IFlyable {

    double price;
    string color;

    public void setPrice(int price)
    {
        this.price = price;
    }

    public void setColor(String color)
    {
        this.color = color;
    }

    public void move()
    {
        //code related to moving plane
    }

    public void fly()
    {
        // code related to flying plane}
    }

}
```

اینی اخاسزا و دانسودخ ازنیارا ک چوکیاه سیینل

کلاسهای مشتق شده را مجبور به پیادهسازی کنید که به آن نیاز ندار د . بجای یک
کلا
س
بزر
گ
چن
دکلا
س
کو
چ
ک
بس
ازی
د .

Dependency Inversion Principle

کلاس های سطح بالاتر نباید به کلاس های سطح پایین وابسته باشن د . هر دو باید به انتزاعات وابسته باشند و انتزاعاتنباید به جزییات دیگر

```
class Customer
{
    private Ilogger obj;
    public Customer(ILogger i)
    {
        obj = i;
    }
    public virtual void Add()
    {
        try
        {
            // Database code goes here
        }
        catch (Exception ex)
        {
            obj.Handle(ex.ToString());
        }
    }
}
```

وابسته باشند آنها هم باید به ان$^{ر ن}$یاعاتوابسته باشن د.