C# Features and OOP Summary

What is the difference between "Readonly" and "Constant" variables (atleast 3)? V. IMP.

- Constant

```
class Example
{
    public const int myConst = 10;

    public const int myConst1;

    public Example(int b)
    {
        myConst = 20;
    }
}
```

- Readonly

```
class Example
{
    public readonly int myReadonly1 = 100;

    public readonly int myReadonly2;

    public Example(int b)
    {
        myReadonly2 = b * 100;
    }
}
```

1. Using constant, we must assign values with declaration itself. But readonly fields can be assigned in declaration as well as in the constructor part.
2. Constant field value cannot be changed, but Readonly field value can be changed.
3. "const" keyword for Constant and "readonly" keyword is used for Readonly.
4. Constant is a COMPILE time constant, and ReadOnly is a RUNTIME constant.

What is the difference between "var" and "dynamic" in C#?

- VAR - The type of the variable is decided by the compiler at **compile time**.

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

    a = "Interview";

    Console.WriteLine(a);
}
```

- DYNAMIC - The type of the variable is decided at **run time**.

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

    dynamic b = 10;

    b = "Happy";

    Console.WriteLine(b);
}
```

Method Declaration

Method declaration means the way to construct method including its naming.

Syntax:

<Access_Modifier><return_type><method_name>([<param_list>])

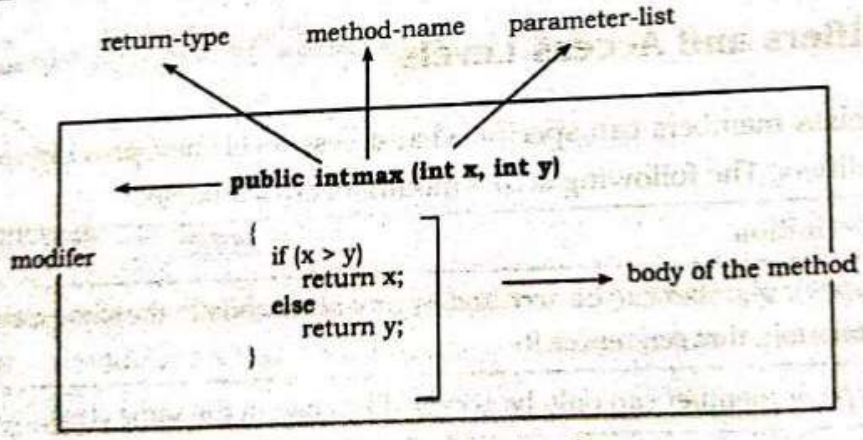


Figure 9: Method Declaration

Q What are Extension Methods in C#? When to use extension methods in real applications? **V. IMP.**

- Extension method allows you to add new methods in the **existing class without modifying** the source code of the original class.
- Extension method must be static because this will be directly called from the class name, not by the object creation.
- **this** keyword is used for binding this method with the main class.
- USE - Use them when you want to add a method in a class which code you don't have.

```
static void Main(string[] args)
{
    string test = "HelloWorld";

    string left = test.Substring(0, 5);

    Console.WriteLine(left);

    string right = test.RightSubstring(5);

    Console.WriteLine(right);

    //Output: Hello World
}
```

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

Q What are Delegates in C#? When to use delegates in real applications? V. IMP.

- A Delegate is a variable that holds the reference to a method or Pointer to a method.
- A delegate can refer to more than one methods of same return type and parameters.
- When to use delegate?

When we need to pass a method as a parameter.

```
delegate void Calculator(int x, int y);

class Program
{
    public static void Add(int a, int b)
    {
        Console.WriteLine(a + b);
    }
    public static void Mul(int a, int b)
    {
        Console.WriteLine(a * b);
    }

    static void Main(string[] args)
    {
        //Instantiating Delegate
        Calculator calc = new Calculator(Add);

        //Calling method using delegate
        calc(20, 30);
    }
```

Q What are Multicast Delegates?

- A Multicast Delegate in C# is a delegate that holds the references of more than one function.

```
delegate void Calculator(int x, int y);

class Program
{
    public static void Add(int a, int b)
    {
        Console.WriteLine(a + b);
    }
    public static void Mul(int a, int b)
    {
        Console.WriteLine(a * b);
    }
    static void Main(string[] args)
    {
        Calculator calc = new Calculator(Add);

        calc += Mul;

        calc(20, 30);
    }
    //Output: 50  600
}
```

What are Anonymous Delegates in C#?

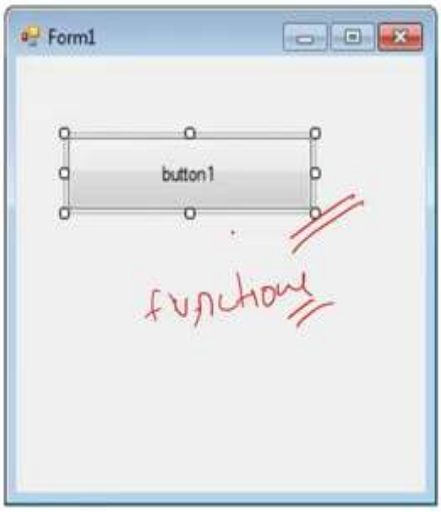
- Delegates pointing methods without name are called anonymous delegates.

```
public delegate void Calculator(int x, int y);

class Program
{
    static void Main(string[] args)
    {
        Calculator calcAdd = delegate(int a, int b)
        {
            //Inline content of the method;
            Console.WriteLine(a + b);
        };

        calcAdd(20, 30);
    }
}

//Ouput: 50
```

What are the differences between Events and Delegates? V. IMP.

- The event is a notification mechanism that depends on delegates.

- An event is dependent on a delegate and cannot be created without delegates.
- Event is like a wrapper over the delegate to improve its security.



How to implement Exception Handling in C#? V. IMP.

- Exception handling in Object-Oriented Programming is used to **MANAGE ERRORS.**

1. **TRY** – A try block is a block of code inside which any error can occur.

1. **CATCH** – When any error occur in TRY block then it is passed to catch block to handle it.

2. **FINALLY** – The finally block is used to execute a given set of statements, whether an exception occur or not.

```
try
{
    int i = 0;
    int j = 0;

    int k = i / j;
}
catch (Exception ex)
{
    //LogError(ex.Message)
    Console.WriteLine(ex.Message);

}
finally
{
    //object.Dispose()
    Console.WriteLine("Finally");
}
//Output: Attempted to divide by zero.
//Finally
```

Can we execute multiple Catch blocks?

- **NO**

*We can write multiple catch blocks but when we will run the application and if any error occur, **only one** out of them will execute based on the type of error.*

```
try
{
    int i = 0;
    int j = 0;

    int k = i / j;
}
catch (ArithmeticException ex)
{
    Console.WriteLine("Alert");
    Console.WriteLine(ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine(ex.Message);
}
```

When to use Finally in real applications?

- Finally block is mostly used to dispose the unwanted objects when they are no more required. This is good for performance, otherwise you have to wait for garbage collector to dispose them.

```
SqlConnection con = new SqlConnection("conString");
try
{
    con.Open();

    //Some logic

    // Error occurred

    con.Close();
}
catch(Exception ex)
{
    // error handled
}
finally
{
    // Connection closed
    con.Close();
}
```

Q What is the difference between Finally and Finalize?

- Finally, is used in exception handling.
- Finalize is a method which is automatically called by the **garbage collector** to dispose the no longer needed objects.

Q Explain Generics in C#? When and why to use them in real applications? V. IMP.

- Generics allows us to make classes and methods - **type independent or type safe**.

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

```
static void Main(string[] args)
{
    bool equal = Calculator.AreEqual(4, 4);
bool strEqual = Calculator.AreEqual("Interview", "Happy");
}
```

- The problem is, it involves Boxing from converting string (value) to object (reference) type. This will impact the performance.

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

Explain Generics in C#? When and why to use them in real applications? V. IMP.

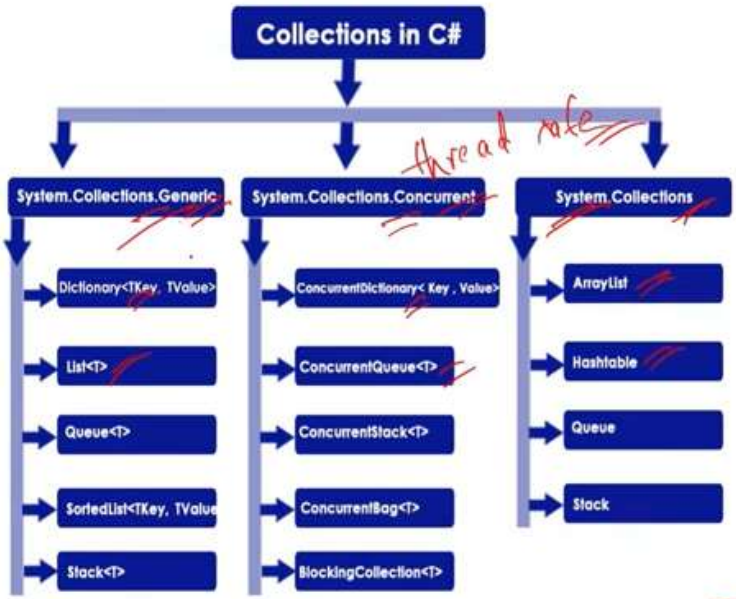
- Generic Method

```
public class Calculator
{
    public static bool AreEqual<T>(T value1, T value2)
    {
        return value1.Equals(value2);
    }
}
```

```
static void Main(string[] args)
{
    //bool equal = Calculator.AreEqual(4, 4);
    //bool strEqual = Calculator.AreEqual("Interview", "Happy");

    bool equal = Calculator.AreEqual<int>(4, 4);

    bool strEqual = Calculator.AreEqual<string>("Interview", "Happy");
}
```

What are Collections in C# and what are their types?

- C# collection are used to **store, manage and manipulate** data.
- *For example ArrayList, Dictionary, List, Hashtable etc.*



Q What is the difference between Array and ArrayList (atleast 2)? V. IMP.

1. Array is STRONGLY typed.
This means that an array can store only specific type of items/ elements.

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

    array = new int[10];

    array[0] = 1;

    array[1] = "Happy";

}
```

2. Array can contain FIXED number of items.

1. ArrayList can store ANY type of items\elements.

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

    ArrayList arrayList;
    arrayList = new ArrayList();

    arrayList.Add(1);
    arrayList.Add("Happy");

}
```

2. ArrayList can store ANY number of items.

Q What is the difference between List and Dictionary Collections? V. IMP.

- List is a collection of items.
- It is the generic version of Arraylist.

```
List<string> employees = new List<string>();
employees.Add("Happy");
employees.Add("Rana");
employees.Add("Roy");

foreach (var employee in employees)
{
    Console.WriteLine(employee);
}
//Output: Happy Rana Roy
```

- Dictionary is a collection of key value pair.
- It is the generic version of Hashtable.

```
Dictionary<int, string> employeesD = new Dictionary<int, string>();
employeesD.Add(123, "HappyD");
employeesD.Add(124, "RanaD");
employeesD.Add(125, "RoyD");

foreach (KeyValuePair<int, string> emp in employeesD)
{
    Console.WriteLine($"{emp.Key} { emp.Value}");
}

//Output: 123 Happy 124 Rana 125 Roy
```

Q What is Inheritance? When to use Inheritance?

- Inheritance is creating a **PARENT-CHILD** relationship between two classes, where child class will **automatically** get the properties and methods of the parent.
- Inheritance is good for: **REUSABILITY** and **ABSTRACTION** of code

```
public class ContractEmployee : Employee
{
    //No method or Property here
}
```

```
public class Employee
{
    public int Experience { get; set; }

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

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

public class PermanentEmployee : Employee
{
    //No method or Property here
}

static void Main(string[] args)
{
    PermanentEmployee pEmployee = new PermanentEmployee();
    pEmployee.Experience = 5;
    pEmployee.CalculateSalary();

    Console.ReadLine();
}
```

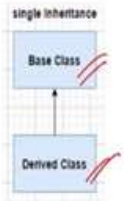
Base/ Parent/ Super class

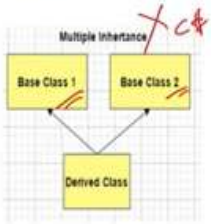
Derived/ Child/ Sub class

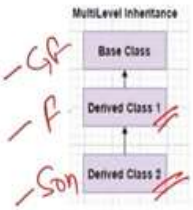
CalculateSalary() method is not present in PermanentEmployee class, but it will get it automatically from it's parent

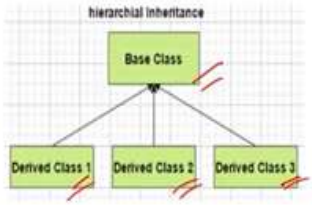
Q What are the different types of Inheritance? V. IMP.

OOP










```
class BaseClass1
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
class DerivedClass1 : BaseClass1
{
    public void Dog()
    {
        Console.WriteLine("Dog");
    }
}
```

```
class BaseClass2
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
interface I2
{
    void Fly();
}
class DerivedClass2 : BaseClass2, I2
{
    public void Eagle()
    {
        Console.WriteLine("Eagle");
    }
    public void Fly()
    {
        Console.WriteLine("Fly");
    }
}
```

```
class BaseClass3
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}
class DerivedClass3 : BaseClass2
{
    public void Dog()
    {
        Console.WriteLine("Dog");
    }
}
class DerivedClass4 : DerivedClass3
{
    public void Labrador()
    {
        Console.WriteLine("Labrador");
    }
}
```

```
class BaseClass4
{
    public void Animal()
    {
        Console.WriteLine("Animal");
    }
}

class DerivedClass5 : BaseClass4
{
    public void Dog()
    {
        Console.WriteLine("Dog");
    }
}
class DerivedClass6 : BaseClass4
{
    public void Cat()
    {
        Console.WriteLine("Dog");
    }
}
```

Q How to prevent a class from being inherited?

- By using **SEALED** keyword in class

```
public sealed class Employee
{
    public void GetSalary()
    {
        Console.WriteLine("100000");
    }
}
public class PermanentEmployee : Employee
{
}
```

'PermanentEmployee': cannot derive from sealed type 'Employee'

- By using **STATIC** keyword in base class

```
public static class Employee
{
    public static void GetSalary()
    {
        Console.WriteLine("100000");
    }
}
public class PermanentEmployee : Employee
{
}
```

'PermanentEmployee': cannot derive from static class 'Employee'

- Difference between sealed & static is, you can create the object of sealed class, but you cannot create the object of static class.

SUBSCRIBE

What is the purpose of base keyword in C#?

- The base keyword in C# is used to access members of a base class from within a derived class.

```
public class BaseClass
{
    public virtual void Print()
    {
        Console.WriteLine("Base");
    }
}
```

```
public class DerivedClass : BaseClass
{
    public override void Print()
    {
        base.Print();
        Console.WriteLine("Derived");
    }
}
```

Q What is the difference between an Abstract class and an Interface (atleast 4)? V. IMP.

1. Abstract class contains both **DECLARATION & DEFINITION** of methods.

```
public abstract class Employee
{
    public abstract void Project();

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

Method Declared

Method Defined

2. Abstract class keyword: **ABSTRACT**

1. Interface should contain **DECLARATION** of methods.

- *With C# 8.0, you can now have default implementations/ definition of methods in an interface. But that is recommended in special case*.*

```
interface IEmployee
{
    public void Project1();

    public void Manager1();
}
```

Only method Declaration is allowed

2. Interface keyword: **INTERFACE**

\

What is the difference between an Abstract class and an Interface (atleast 4)? V. IMP.

3. Abstract class does not support **multiple inheritance**

```
public abstract class Employee
{
    public abstract void Project();

    public void Role()
    {
        Console.WriteLine("Engineer");
    }
}

public abstract class Employee1
{
    public abstract void Project1();

    public void Role1()
    {
        Console.WriteLine("Engineer1");
    }
}

public class PermanentEmployee: Employee, Employee1
{

}
```

3. Interface supports **multiple inheritance.**

```
interface IEmployee1
{
    public void Project1();
}

interface IEmployee2
{
    public void Project2();
}


public class NewEmployee : IEmployee1, IEmployee2
{
    public void Project1()
    {
        Console.WriteLine("Print 1");
    }

    public void Project2()
    {
        Console.WriteLine("Print 2");
    }
}
```

What is the difference between an Abstract class and an Interface (atleast 4)? V. IMP.

4. Abstract class can have **constructors**.

```
public abstract class Employee1
{
    public Employee1()
    {

    }

    public abstract void Project1();

    public void Role1()
    {
        Console.WriteLine("Engineer1");
    }
}
```

4. Interface do not have constructors.

```
interface IEmployee1
{
    public IEmployee1()
    {

    }
    public void Project1();
}
```

What is the difference between an Abstract class and an Interface (atleast 4)? V. IMP.

Abstract Class	Interface
1. Abstract class contains both **DECLARATION & DEFINITION** of methods.	Mostly Interfaces contain **DECLARATION** of methods. From C# 8.0 definition is also possible.
2. Abstract class keyword: **ABSTRACT**	2. Interface keyword: **INTERFACE**
3. Abstract class does not support **multiple inheritance**	3. Interface supports **multiple inheritance.**
4. Abstract class can have **constructors**.	4. Interface do not have constructors.

Q When to use Interface and when Abstract class in real applications? V. IMP.

❖ When to use Interface?

An interface is a good choice when you know a method has to be there, but it can be implemented **DIFFERENTLY** by independent derived classes.

```
public class PermanentEmployee
{

}

public class ContractualEmployee
{

}
```

```
interface IEmployee
{
    public void AssignEmail();

    public void AssignManager();
}
```

When to use Interface and when Abstract class in real applications? V. IMP.

- When to use Abstract class?

Abstract class is a good choice when you are sure some methods are concrete/defined and must be implemented in the **SAME WAY** in all derived classes.

```
public class PermanentEmployee
{

}

public class ContractualEmployee
{

}
```

```
public abstract class EmployeeDress
{
    public abstract void DressCode();

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

When to use Interface and when Abstract class in real applications? V. IMP.

- When to use Abstract class?

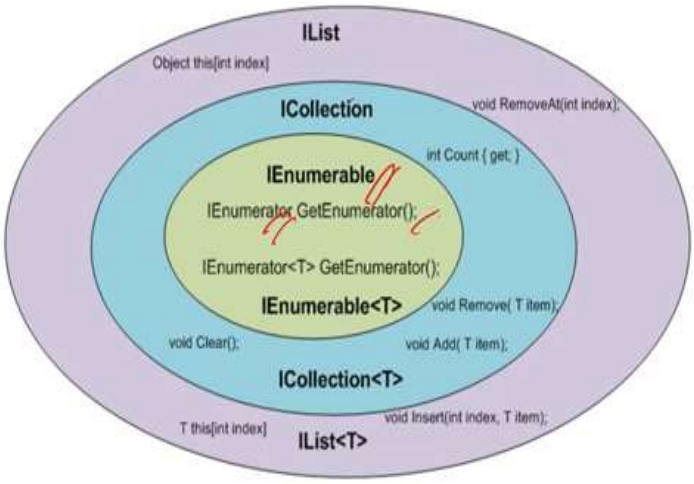
Abstract class is a good choice when you are sure some methods are concrete/defined and must be implemented in the **SAME WAY** in all derived classes.

- Normally we prefer Interface because it gives us the flexibility to modify the behavior at later stage.

```
public class PermanentEmployee
{

}

public class ContractualEmployee
{

}
```

```
public abstract class EmployeeDress
{
    public abstract void DressCode();

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

Q What is the difference between IEnumerable and IEnumerator in C#?




```
public interface IEnumerable
{
    //
    // Summary:
    //     Returns an enumerator that iterate
    //
    // Returns:
    //     An System.Collections.IEnumerator
    //     the collection.
    IEnumerator GetEnumerator();
}
```

What is the difference between IEnumerable and IQueryable in C#? Why to use IQueryable for sql queries?

- IQueryable inherited from IEnumerable interface only, so anything you can do with a IEnumerable, you can also do with an IQueryable also.
- For example, iterating the collection can be done by both IEnumerable and IQueryable.

```
public interface IQueryable<out T> : IEnumerable<T>, IEnumerable, IQueryable
{
}
```

What is the difference between IEnumerable and IQueryable in C#? Why to use IQueryable for sql queries?

```
//IEnumerable Example
List<string> employees = new List<string>();
employees.Add("Happy");
employees.Add("Joe");

IEnumerable<string> iEnumerableEmployees = employees;

foreach (string employee in iEnumerableEmployees)
{
    Console.WriteLine(employee);
}
//Output: Happy Joe John
```

```
//IQueryable Example
IQueryable<string> iQueryableEmployees = (IQueryable<string>)employees;

foreach (string employee in iQueryableEmployees)
{
    Console.WriteLine(employee);
}
//Output: Happy Joe John
```

What is the difference between IEnumerable and IQueryable in C#? Why to use IQueryable for sql queries?

- IEnumerable is used with in-memory collection.
- IQueryable is better in getting result from database.

```
EmployeeDbContext dc = new EmployeeDbContext();

//IEnumerable Example - Better with in-memory collection
IEnumerable<Employee> listE = dc.Employees.Where(p => p.Name.StartsWith("H"));
```

```
//IQueryable Example - Better with database interaction
IQueryable<Employee> listQ = dc.Employees.Where(p => p.Name.StartsWith("H"));
```

What is the difference between IEnumerable and IQueryable in C#? Why to use IQueryable for sql queries?

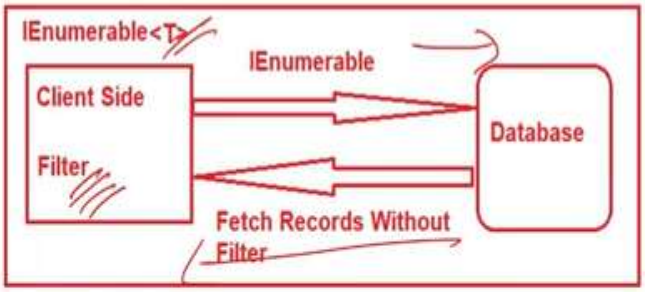
- IEnumerable is used with in-memory collection.
- IQueryable is better in getting result from database.

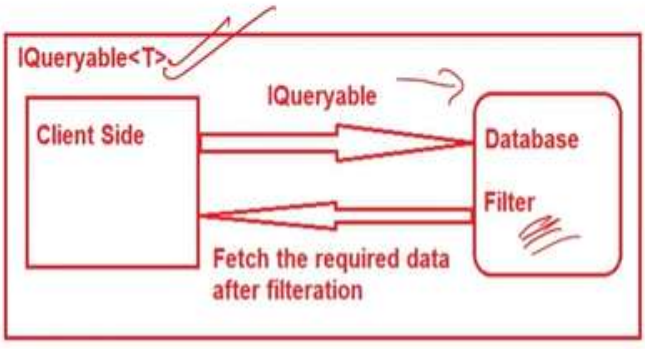
```
EmployeeDbContext dc = new EmployeeDbContext();

//IEnumerable Example - Better with in-memory collection
IEnumerable<Employee> listE = dc.Employees.Where(p => p.Name.StartsWith("H"));
```

```
//IQueryable Example - Better with database interaction
IQueryable<Employee> listQ = dc.Employees.Where(p => p.Name.StartsWith("H"));
```

What is the difference between IEnumerable and IQueryable in C#? Why to use IQueryable for sql queries?

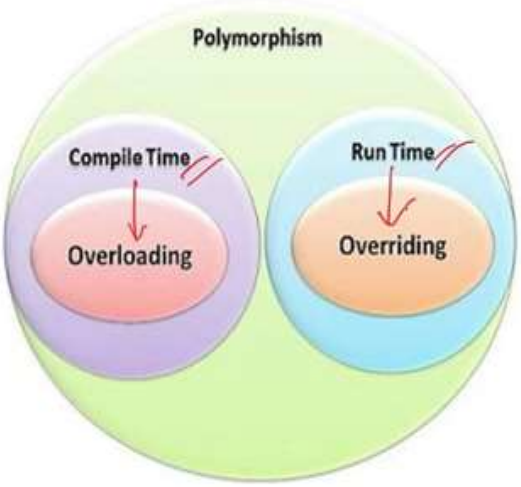
- IQueryable inherited from IEnumerable interface only, so anything you can do with a IEnumerable, you can also do with an IQueryable also.
- IEnumerable bring all result from database and then filter it at code side, which is a network load and performance issue.
- IQueryable filter the result at database only and then get only filtered result, therefore less network load and better performance.
- IQueryable is under SYSTEM.LINQ namespace. IEnuberable is under System.Collections namespace.





Q What is Polymorphism and what are its types? When to use polymorphism?

- Polymorphism is the ability of a variable, object, or function to take on **MULTIPLE FORMS.**




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

    public string Add(string str1, string str2)
    {
        return str1 + str2;
    }
}
```

Function with same name but have multiple forms

```
Polymorphism obj = new Polymorphism();

int i = obj.Add(50, 60);

string str = obj.Add("Interview", "Happy");

Console.WriteLine(i + " - " + str);

//Output: 110 - InterviewHappy
```

Q What is Method Overloading? In how many ways a method can be overloaded? V. IMP.

- Method overloading is a type of polymorphism in which we can create multiple methods of the **same name in the same class**, and all methods work in different ways.

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

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

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

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

1. Number of parameters are different
2. Type of parameters are different
3. Order of parameters is different

If two methods are same except return type, then methods are overloaded or what will happen?

- No, this will show compile time error.

What is the difference between Method Overriding and Method Hiding?

- In Method Hiding, you can completely hide the implementation of the methods of a base class from the derived class using the **new keyword**.

```
public class BaseClass
{
    public virtual void Greetings()
    {
        Console.WriteLine("BaseClass Hello!");
    }
}

public class DerivedClass : BaseClass
{
    public override void Greetings()
    {
        Console.WriteLine("DerivedClass Hello!");
    }
}
```

```
static void Main(string[] args)
{
    BaseClass objDerived = new DerivedClass();

    objDerived.Greetings();
}
//Ouptut: DerivedClass Hello
```

```
public class BaseClass
{
    public void Greetings()
    {
        Console.WriteLine("BaseClass Hello!");
    }
}

public class DerivedClass : BaseClass
{
    public new void Greetings()
    {
        Console.WriteLine("DerivedClass Hello!");
    }
}
```

```
static void Main(string[] args)
{
    BaseClass objDerived = new DerivedClass();

    objDerived.Greetings();
    Console.ReadLine();
}

//Ouptut: BaseClass Hello
```

What is the difference between Overloading and Overriding? V. IMP.

- Method Overriding

1. Multiple methods of same name are in **different class**.
2. **Inheritance is used**, as it is in different class.
3. Both methods have **same signature**.
4. It's a **run time** polymorphism.
5. **Virtual** & **override** keywords.

```
public class BaseClass
{
    public virtual void Greetings()
    {
        Console.WriteLine("BaseClass Hello!");
    }
}

public class DerivedClass : BaseClass
{
    public override void Greetings()
    {
        Console.WriteLine("DerivedClass Hello!");
    }
}
```

```
static void Main(string[] args)
{
    DerivedClass objDerived = new DerivedClass();
    objDerived.Greetings();
    Console.ReadLine();
}

//Ouptut: DerivedClass Hello
```

Q What is the difference between Overloading and Overriding? V. IMP.

Method Overloading

1. Multiple methods of same name in **single class**.
2. No need of **inheritance**, as it is in single class.
3. All methods have **different signature.**
4. It's a **compile time** polymorphism.
5. No special keyword used.

Method Overriding

1. Multiple methods of same name in **different class.**
2. **Inheritance is used**, as it is in different class.
3. All methods have **same signature.**
4. It's a **run time** polymorphism.
5. **Virtual** & **override** keywords.

What is the use of Overriding? When should I override the method in real applications?

```
public class Technology
{
    public virtual void TechnicalSkill()
    {
        Console.WriteLine("Coding");
    }
    public virtual void CommunicationSkill()
    {
        Console.WriteLine("English");
    }
}
```

```
public class Testing : Technology
{
    public override void TechnicalSkill()
    {
        Console.WriteLine("Testing");
    }
}
```

```
static void Main(string[] args)
{
    Testing test = new Testing();
    test.TechnicalSkill();
    test.CommunicationSkill();
}

//Output: Testing English
```

```
public class Java : Technology
{

}

public class DotNet : Technology
{

}
```

What is "Static" class? When to use static class in real application?

- A static class is a class which object can not be created, and which can not be inherited.

- Use of static class:

Static classes are used as **containers** for static members like methods, constructors and others.

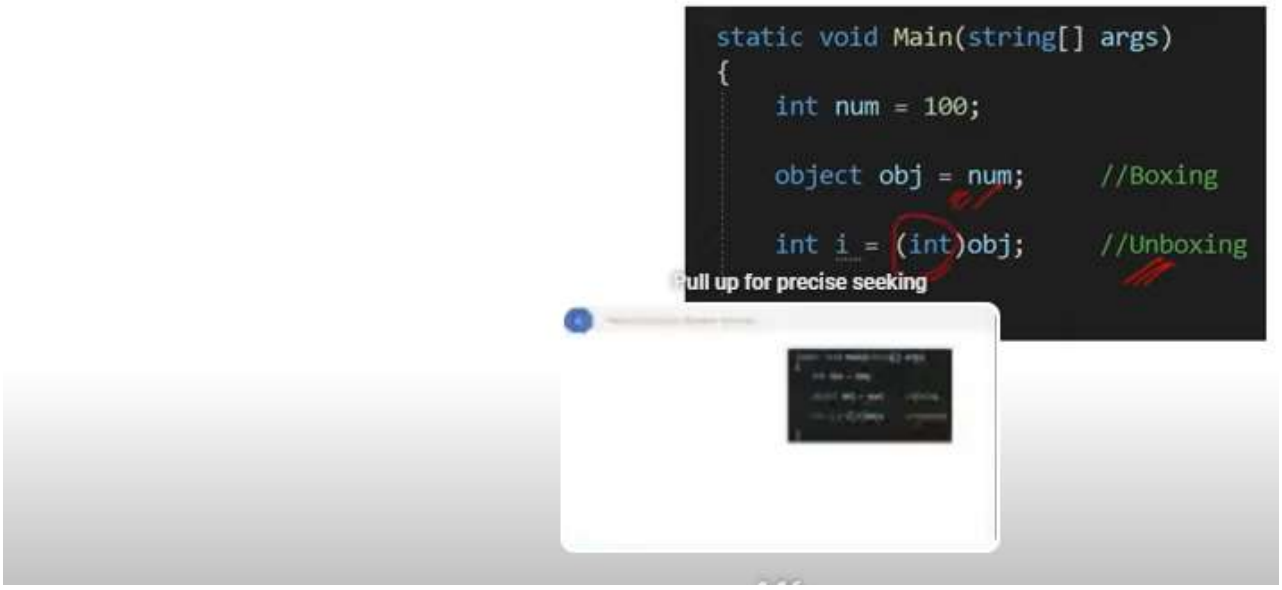
```
public static class MyCollege
{
    //static fields
    public static string collegeName;
    public static string address;

    //static constructor
    static MyCollege()
    {
        collegeName = "ABC College";
    }

    // static method
    public static void CollegeBranch()
    {
        Console.WriteLine("Computers");
    }

}
```

What is Boxing and Unboxing? V. IMP.

- **Boxing -** Boxing is the process of converting from value type to reference type.

```
static void Main(string[] args)
{
    int num = 100;

    object obj = num;       //Boxing

    int i = (int)obj;       //Unboxing

}
```

Which one is explicit Boxing or Unboxing?



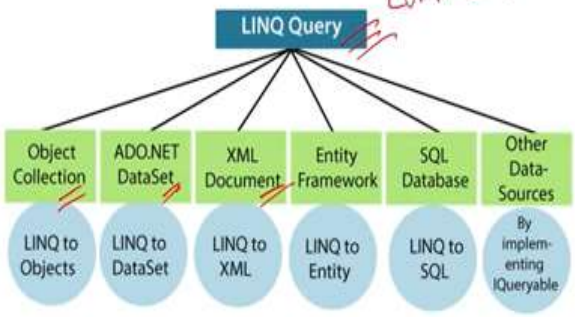
Is Boxing and Unboxing good for performance?

No, it is recommended to use boxing and unboxing when it is necessary only.

```
ArrayList arrayList = new ArrayList();

arrayList.Add(i);      //Boxing

int k = (int)arrayList[0]; //Unboxing
```

Q What is LINQ? When to use LINQ in real applications?

- LINQ (Language Integrated Query) is uniform query syntax in C# to retrieve data from different sources.




```
List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

List<int> filteredNumbers = new List<int>();

foreach (int n in numbers)
{
    if (n > 2)
    {
        filteredNumbers.Add(n);
    }
}
```

```
//using System.Linq
IEnumerable<int> filteredNumbers = from n in numbers
                                   where n > 2
                                   select n;
```

What are the advantages & disadvantages of LINQ?

Advantages of LINQ

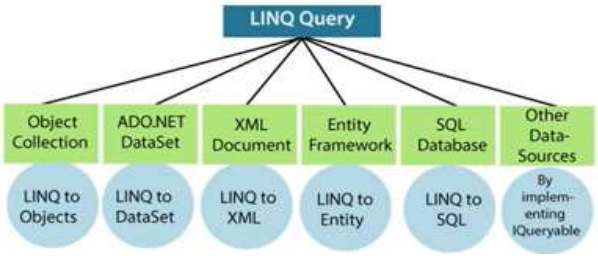
1. Easy and simple syntax to Learn
2. Improved code readability
3. Improved performance
4. Type safety

```
//using System.Linq
IEnumerable<int> filteredNumbers = from n in numbers
                                   where n > 2
                                   select n;
```

```
//using System.Linq
IEnumerable<string> filteredNumbers = from n in numbers
                                      where n > 2
                                      select n;
```

Disadvantages of LINQ

1. Limited support for some data sources
2. Difficult to maintain and debug



What is Lambda Expressions? What is the use in real applications?

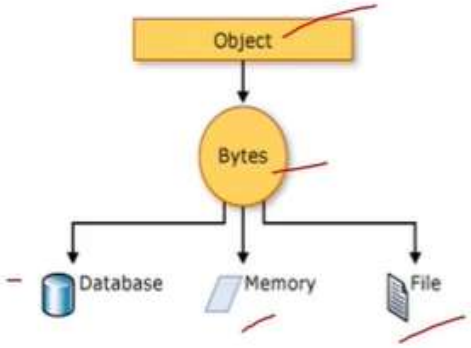
A lambda expression is used to **simplify** the syntax of anonymous methods.

```
static void Main(string[] args)
{
    List<int> numbers = new List<int> { 1, 2, 3, 4, 5 };

    List<int> evenNumbers = GetEvenNumbers(numbers);
}

static List<int> GetEvenNumbers(List<int> numbers)
{
    List<int> evenNumbers = new List<int>();

    foreach (int n in numbers)
    {
        if (n % 2 == 0)
        {
            evenNumbers.Add(n);
        }
    }
    return evenNumbers;
}
```

```
//List method and Lambda expression

List<int> evenNumbers = numbers.FindAll(x => x % 2 == 0);
```

What is Serialization? V. IMP.

Serialization is a process of converting object to its BINARY FORMAT (BYTES)



Once it is converted to bytes, it can be easily stored and written to a disk or any such storage devices.

When to use it.

It is mostly used in Web API to convert class objects into JSON string.

```
private void JSONSerilaize()
{
    // Serializaion
    Employee empObj = new Employee();
    empObj.ID = 1;
    empObj.Name = "Manas";
    empObj.Address = "India";

    // Convert Employee object to JOSN string format
    string jsonData = JsonConvert.SerializeObject(empObj);
                      jsonData = "{\"ID\":1,\"Name\":\"Manas\",\"Address\":\"India\"}"
    Response.Write(jsonData);
}
```