

^{TOP}
500

.NET

INTERVIEW QUESTIONS

PART - I

HAPPY RAWAT

PREFACE

ABOUT THE BOOK

Part I - This book contains 250 very important .NET interview questions.

Part II - Next part of this book contains 250 more interview questions.



ABOUT THE AUTHOR

Happy Rawat has around 15 years of experience in software development. He helps candidates in clearing technical interview in tech companies.



Chapters



Chapter 1 : OOPS & C# - Basics

Q1. What is C#? What Is the difference between C# and .NET?

Q2. What is OOPS? What are the main concepts of OOPS? V Imp

Q3. What are the advantages of OOPS? V Imp

Q4. What are the limitations of OOPS?

Q5. What are Classes and Objects?

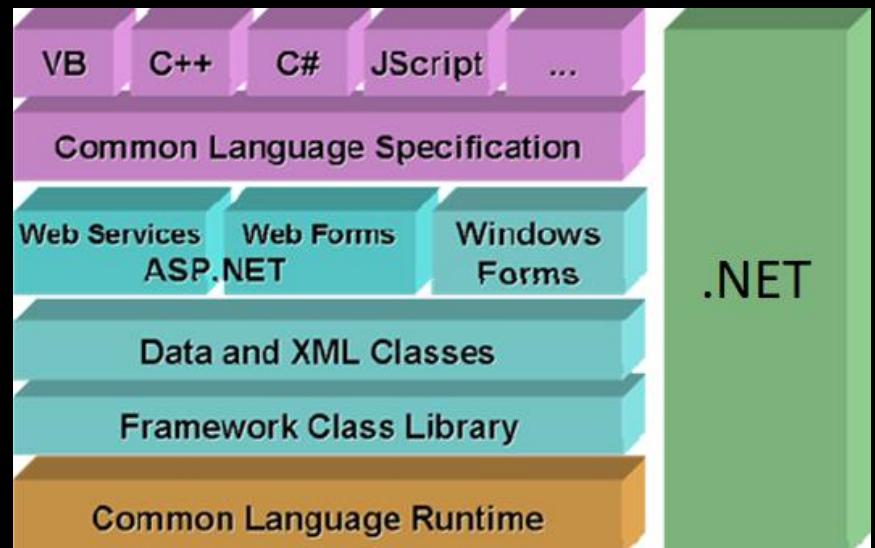
Q6. What are the types of classes in C#?

Q7. Is it possible to prevent object creation of a class in C#?

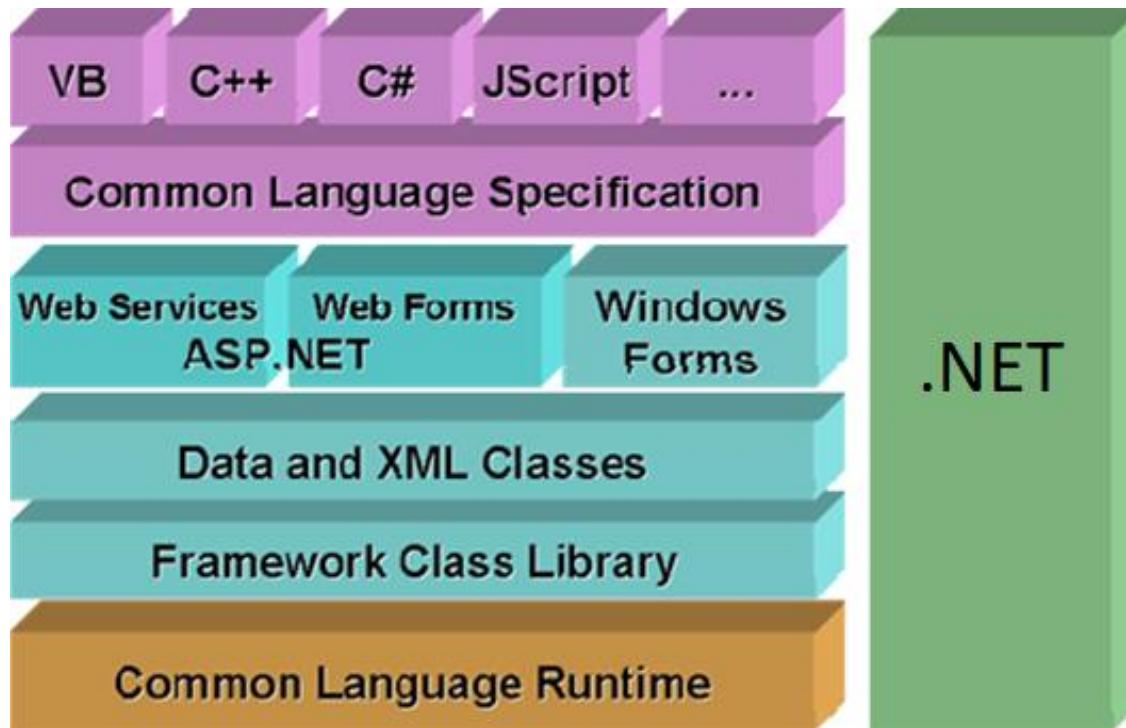
Q8. What is Property?

Q9. What is the difference between Property and Function?

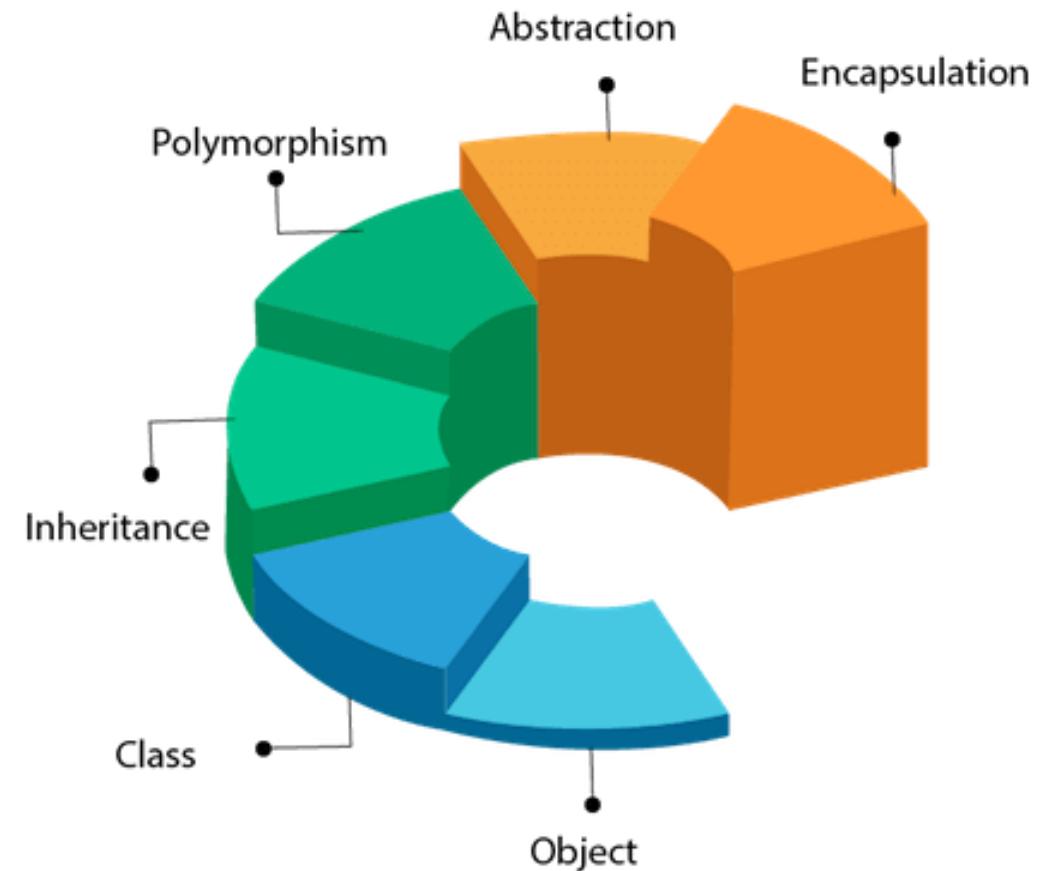
Q10. What are Namespaces?



- ❖ C# is an object-oriented **programming language** which runs on the .NET framework.
- ❖ .NET is a **framework** for building and running software's and applications.

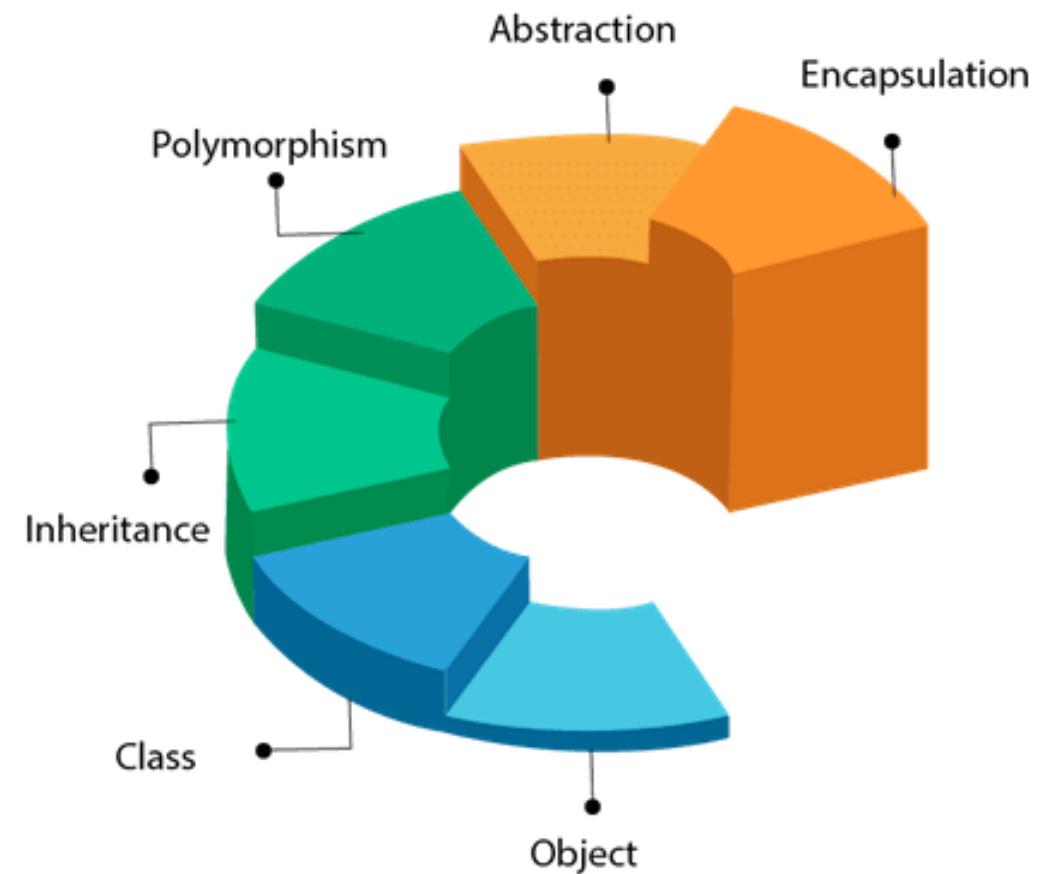


- ❖ OOP stands for **Object-Oriented Programming**, which means it is a way to create software around **objects**.
- ❖ OOPs provide a clear structure for the software's and web applications.



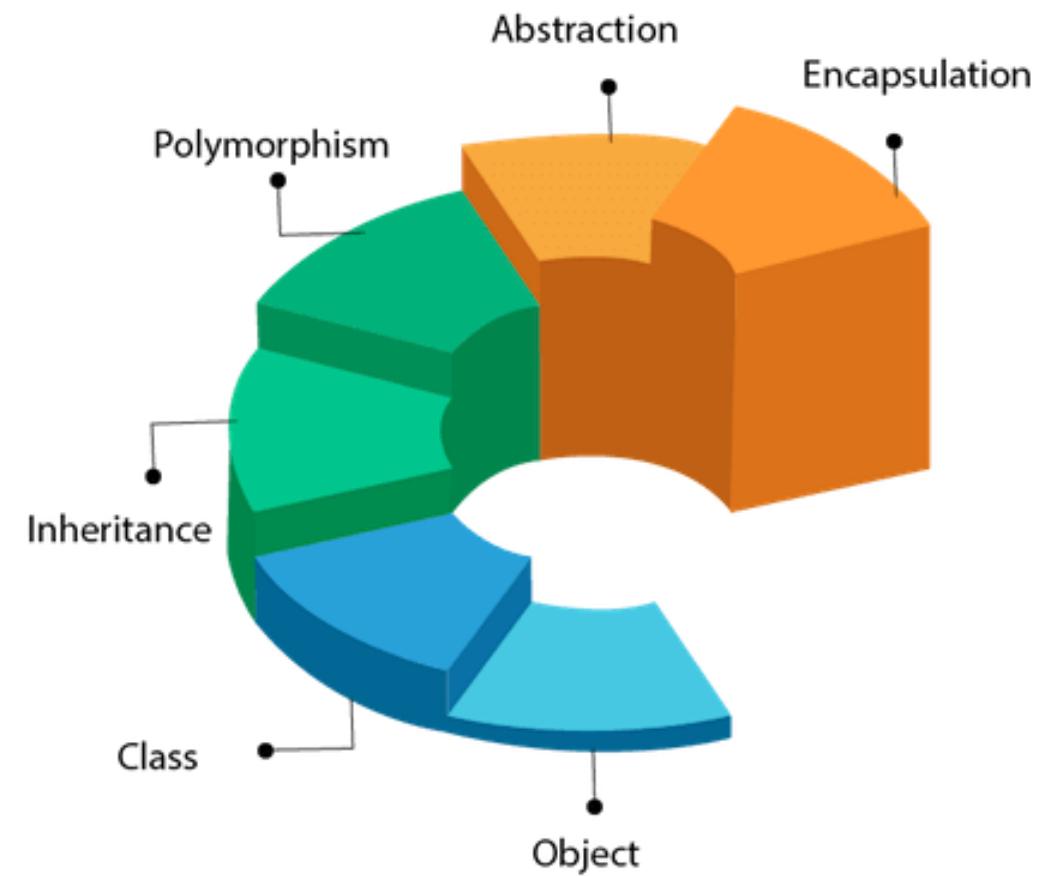
❖ Advantages of OOPS:

1. **Reuse** of code using inheritance.
2. **Flexibility** of code using polymorphism.
3. **Secure** application by using Encapsulation.
4. **Easily scalable** from small to large applications.
5. **Easier troubleshooting** of code because of modularity.



❖ Disadvantage of OOPS:

1. It is not suitable for **small** applications.



- ❖ A class is a **LOGICAL UNIT** or **BLUEPRINT**. It contains fields, methods and properties.

- ❖ Class members are:

1. A **constructor** is a method in the class which gets executed when a class object is created.
2. A **field** is a variable of any type. It is basically the data.
3. A **property** is a member that provides helps in read and write of private field.
4. A **method** is a code block that contains a series of statements.

```
public class Employee
{
    public Employee()
    {
        //code
    }

    private int experience;

    public int Experience
    {
        get { return experience; }

        set { experience = value; }
    }

    //public int Experience { get; set; }

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

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

Constructor

Field

Property

Method

- ❖ **Object** - An object is an **INSTANCE** of a class.

```
static void Main(string[] args)
{
    Employee objEmployee = new Employee();

    objEmployee.Experience = 3;

    objEmployee.CalculateSalary();

    Console.ReadLine();
}
```

Object

```
public class Employee
{
    public Employee()
    {
        //code
    }

    private int experience;

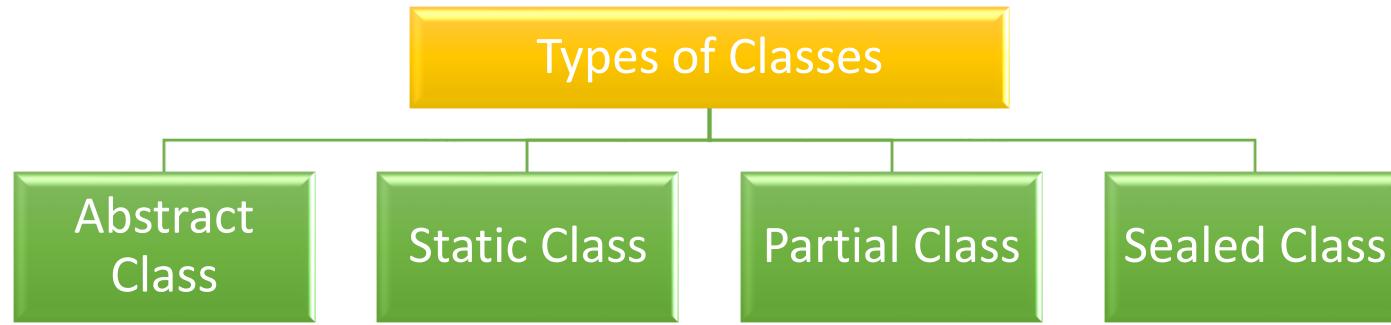
    public int Experience
    {
        get { return experience; }

        set { experience = value; }
    }

    //public int Experience { get; set; }

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

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



```
public abstract class Class1  
{  
}  
  
public static class Class2  
{  
}  
  
public partial class Class3  
{  
}  
  
public sealed class Class4  
{  
}
```

- ❖ Object creation of a class can be prevented by:
 1. Abstract Class
 2. Private Class
 3. Static Class

```
public abstract class Maths
{
}

private class Science
{
}

public static class Social
{
}

public class Student
{
    Maths objMaths = new Maths();

    Science objScience = new Science();

    Social objSocial = new Social();
}
```

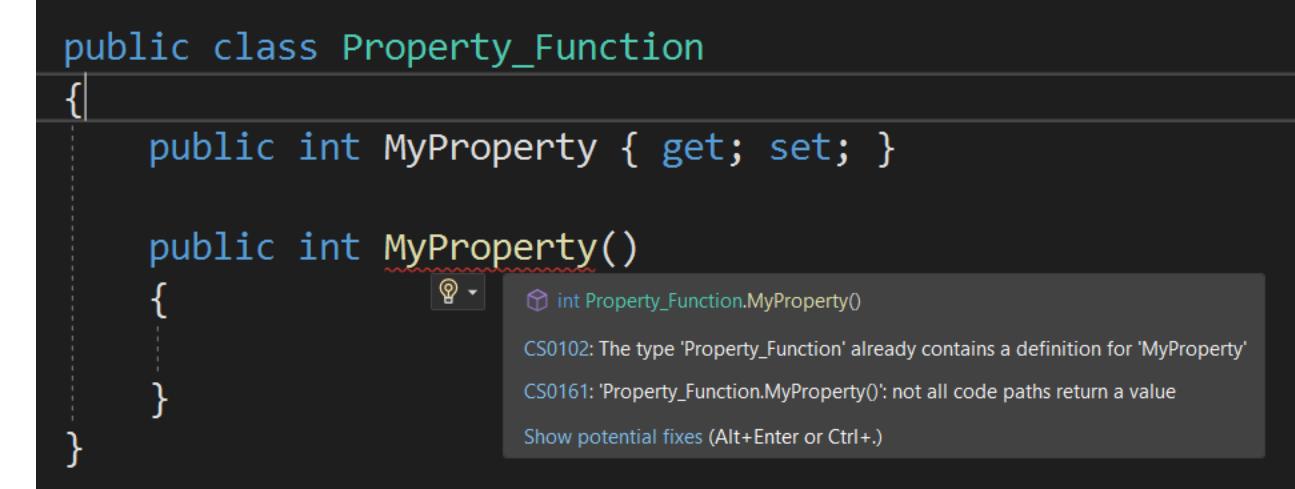
- ❖ A property is a class member that provides a flexible mechanism to **read** and **write** private field.

```
public class MyClass
{
    private int myProperty;

    public int MyProperty
    {
        get { return myProperty; }
        set { myProperty = value; }
    }.....

    public string MyProperty1 { get; set; }
}
```

- ❖ Property is a **specialized function** only.
- ❖ Specialized means properties are used only to get and set field values.



```
public class Property_Function
{
    public int MyProperty { get; set; }

    public int MyProperty()
    {
    }
}
```

The screenshot shows a code editor with the following code:

```
public class Property_Function
{
    public int MyProperty { get; set; }

    public int MyProperty()
    {
    }
}
```

A tooltip is displayed over the second `MyProperty` declaration, showing:

- int Property_Function.MyProperty()
- CS0102: The type 'Property_Function' already contains a definition for 'MyProperty'
- CS0161: 'Property_Function.MyProperty()': not all code paths return a value
- Show potential fixes (Alt+Enter or Ctrl+.)

- ❖ A namespace is a container for a set of related classes and other types.

```
using System.Text;

namespace myNamespace
{
    public class myClass
    {
        StringBuilder sb = new StringBuilder();
    }
}
```

```
namespace MyNamespace
{
    public class MyClass
    {
        // class implementation
    }
}
```

```
using MyNamespace;
MyClass myInstance = new MyClass();
```

Chapter 2 : OOPS & C# - Inheritance, Abstraction, Encapsulation & Polymorphism

Q11. What is Inheritance? When to use Inheritance?

Q12. What are the different types of Inheritance? V Imp

Q13. Does C# support Multiple Inheritance?

Q14. How to prevent a class from being Inherited?

Q15. Are private class members inherited to the derived class?

Q16. What is Abstraction? How to implement abstraction? V Imp

Q17. What is Encapsulation? How to implement encapsulation? V Imp

Q18. What is the difference between Abstraction and Encapsulation?

Q19. What is Polymorphism and what are its types? When to use polymorphism?

Q20. What is Method Overloading? In how many ways a method can be overloaded?

Q21. When should you use method overloading in real applications?

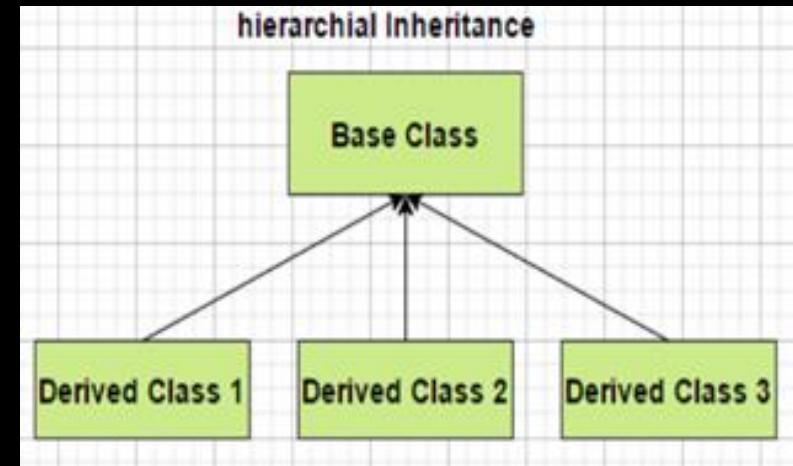
Q22. If two methods are same except return type, then methods are overloaded or what will happen?

Q23. What is the difference between Overloading and Overriding? V Imp

Q24. What is the use of Overriding?

Q25. If a method is marked as virtual, do we must have to "override" it from the child class?

Q26. What is the difference between Method Overriding and Method Hiding?



- 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
}

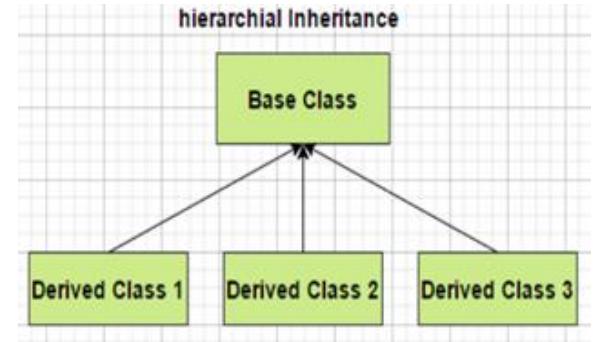
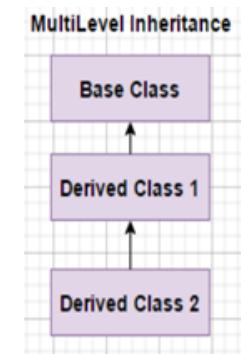
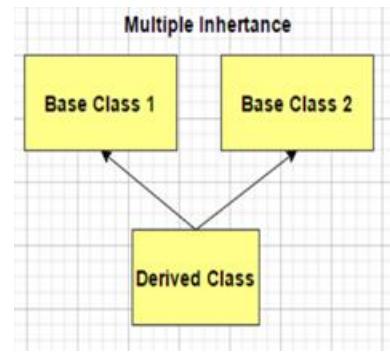
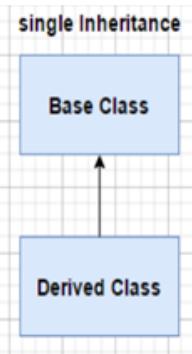
static void Main(string[] args)
{
    PermanentEmployee pEmployee = new PermanentEmployee();
    pEmployee.Experience = 5;
    pEmployee.CalculateSalary();

    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 its parent



```

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("Cat");
    }
}
  
```

- ❖ NO
- ❖ C# support inheritance by using **multiple Interfaces**. This is an alternative way of multiple inheritance.

- ❖ By using **SEALED** keyword in class

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

CS0509: '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
```

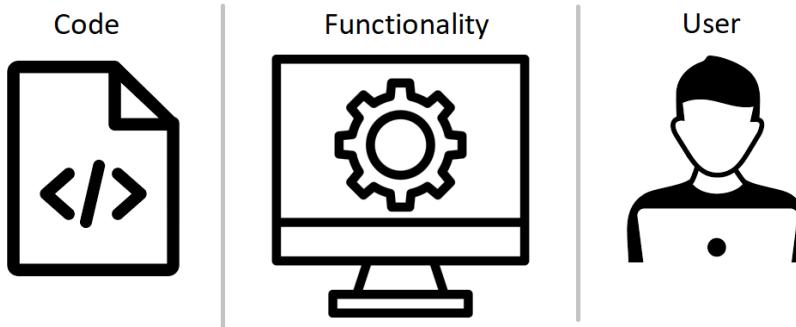
CS0709: 'PermanentEmployee': cannot derive from static class 'Employee'

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



- ❖ No, the private members cannot be inherited to derived class.
- ❖ Only public and protected class members can be inherited.

- ❖ Abstraction means showing only required things and hide the **BACKGROUND** details.



```
Employee employee = new Employee();

int sal = employee.CalculateSalary();

Console.WriteLine( sal );

//Output: 3000000
```

```
String name = "InterviewHappy";

string firstname = name.Substring(8);

Console.WriteLine(firstname);

//Output: Interview
```

Background/Hidden method.

```
public abstract class EmployeeSalary
{
    public int CalculateSalary()
    {
        return 10 * 300000;
    }
}
```

```
public class Employee : EmployeeSalary
{
}
```

- ❖ Mostly we use **abstract classes** and **interfaces** for implementing abstraction.

- ❖ Encapsulation means **WRAPPING** of data and methods into a single unit.

```
public class Employee
{
    public int empExperience;
}

static void Main(string[] args)
{
    Employee objEmployee = new Employee();
    objEmployee.empExperience = 3;
}
```

Field/Data is accessed without property or function.
It will work but it violating encapsulation

```
OR:
public class Employee
{
    //Make field private
    private int empExperience;

    public int EmpExperience
    {
        get { return empExperience; }
        set { empExperience = value; }
    }

    //Shortcut Property
    //public int EmpExperience { get; set; }
}
```

This field cannot be accessed from outside without the property

```
static void Main(string[] args)
{
    Employee objEmployee = new Employee();
    objEmployee.EmpExperience = 3;
}
```

- ❖ Abstraction means showing only required things and hide the **BACKGROUND** details.
- ❖ Encapsulation means **WRAPPING** of data and methods into a single unit.

```
public abstract class EmployeeSalary
{
    public int CalculateSalary()
    {
        return 10 * 300000;
    }
}

public class Employee : EmployeeSalary
{}
```

```
Employee employee = new Employee();

int sal = employee.CalculateSalary();

Console.WriteLine( sal );

//Output: 3000000
```

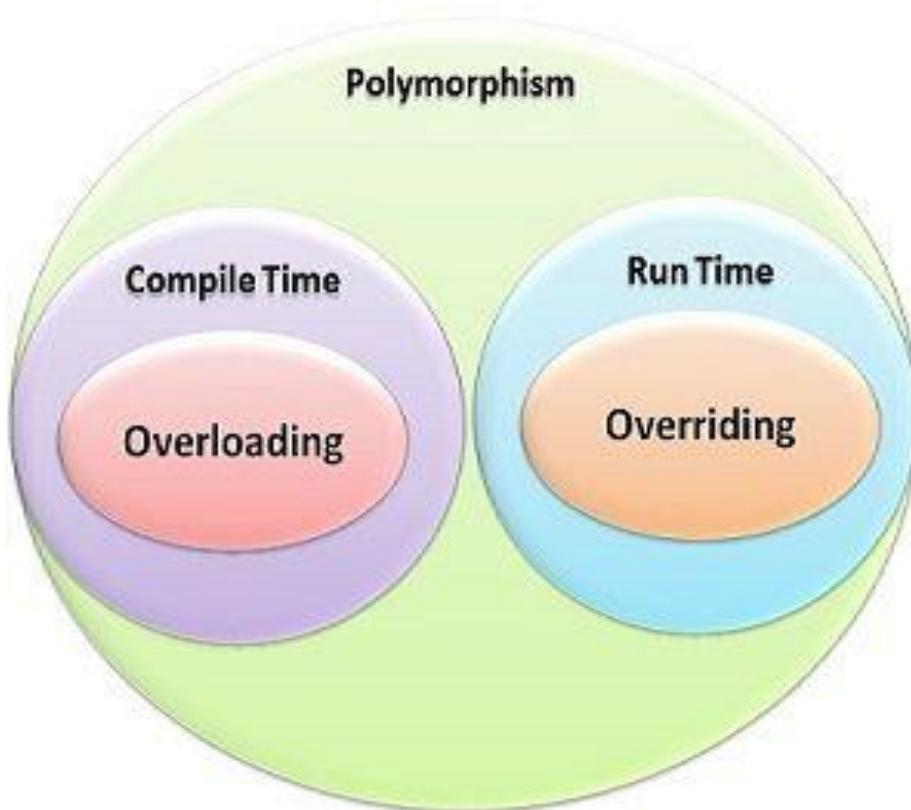
```
public class Employee
{
    //Make field private
    private int empExperience;

    public int EmpExperience
    {
        get { return empExperience; }

        set { empExperience = value; }
    }

    //Shortcut Property
    //public int EmpExperience { get; set; }
}
```

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



o use polymorphism?

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

Function with same name but have multiple forms

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

```
Polymorphism obj = new Polymorphism();  
  
int i = obj.Add(50, 60);  
  
string str = obj.Add("Interview", "Happy");  
  
Console.WriteLine(i + " - " + str);  
  
//Output: 110 - InterviewHappy
```

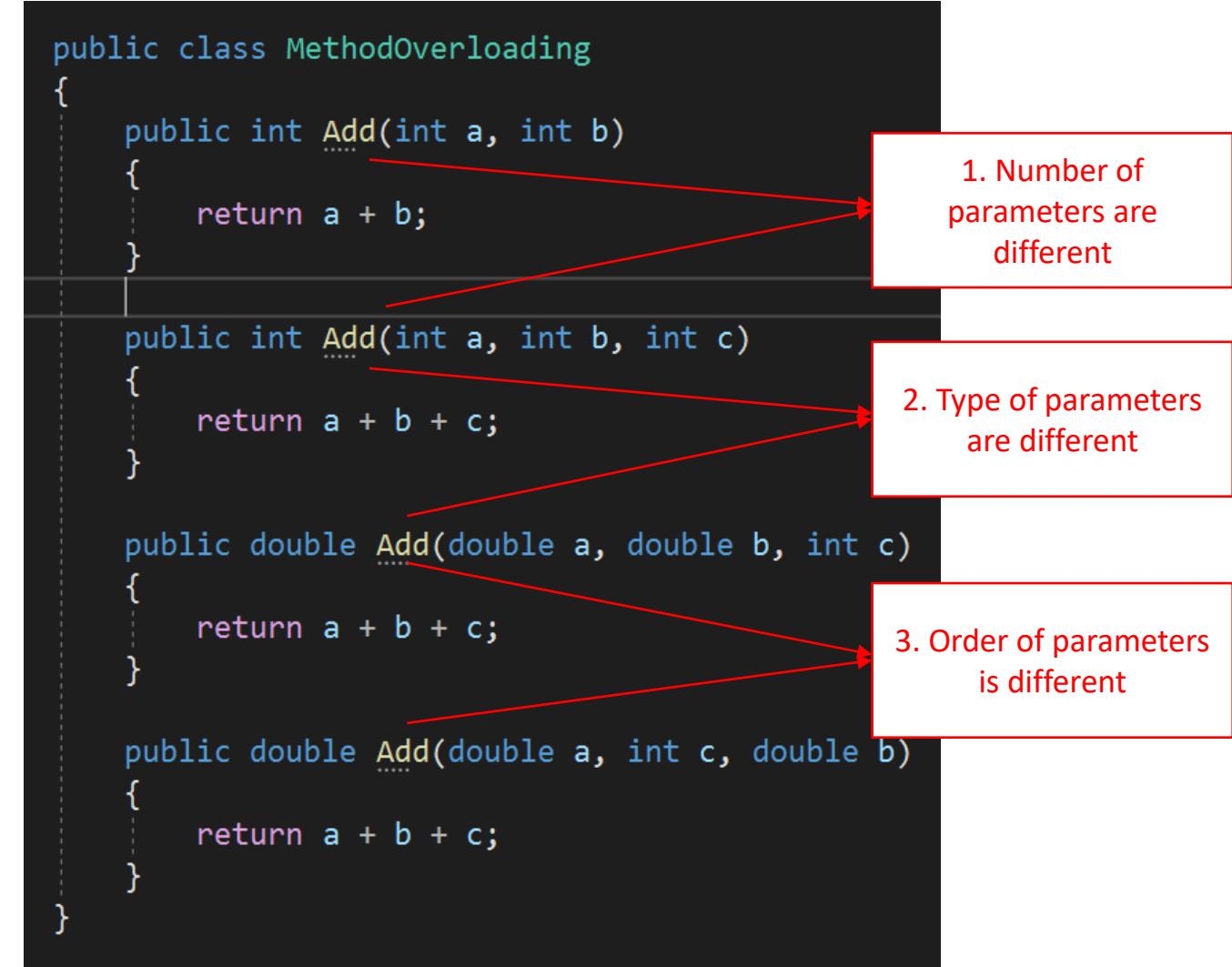
- Method overloading is a type of polymorphism in which we can create multiple methods of the **same name in the same class**, and all methods work in different ways.

```
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;
    }
}
```



The diagram illustrates four overloaded methods named `Add` within a class. Red arrows point from each method signature to a corresponding callout box:

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

```
name.Substring(b.ToString().Length);
```

▲ 1 of 2 ▼ string string.Substring(int startIndex)

Retrieves a substring from this instance. The substring starts at a specified **startIndex**: The zero-based starting character position of a substring in this

```
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;
    }
}
```

- ❖ No, this will show compile time error.

```
public class Employee
{
    public int GetSalary(int designation)
    {
        return 100000;
    }

    public string GetSalary(int designation)
    {
        string Employee.GetSalary(int designation)
        return "100000";
    }
}
```

CS0111: Type 'Program.Employee' already defines a member called 'GetSalary' with the same parameter types
CA1822: Member 'GetSalary' does not access instance data and can be marked as static
Show potential fixes (Alt+Enter or Ctrl+.)

❖ 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();
}

//Output: DerivedClass Hello
```

❖ 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.

- ❖ Overriding is used to modify and provide a new implementation of the method inherited from a base class.

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

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

//Output: Testing English
```

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

```
public class Java : Technology
{
}

public class DotNet : Technology
{
}
```

- ❖ NO. Overriding virtual method is optional.

```
FROM THE CHILD CLASS:  
public class Technology  
{  
    public virtual void TechnicalSkill()  
    {  
        Console.WriteLine("Coding");  
    }  
}
```

```
public class Java : Technology  
{  
}  
  
static void Main(string[] args)  
{  
    Java java = new Java();  
    java.TechanicalSkill();  
}  
  
//Output: Coding
```

- ❖ 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();
}

//Output: 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();
}

//Output: BaseClass Hello
```

Chapter 3 : OOPS & C# - Abstract Class & Interface

Q27. What is the difference between **Abstract** class and an **Interface**? V Imp

Q28. When to **use Interface** and when Abstract class in real applications? V Imp

Q29. Why to **create Interfaces** in real applications?

Q30. Can we define body of Interfaces methods? When to define methods in Interfaces?

Q31. Can you create an instance of an Abstract class or an Interface?

Q32. Do Interface can have a **Constructor**?

Q33. Do abstract class have Constructors in C#?

Q34. What is the difference between abstraction and abstract class? V Imp

Q35. Can Abstract class be Sealed or Static in C#?

Q36. Can you declare **abstract methods** as private in C#?

Q37. Does Abstract class support multiple Inheritance?



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

```
public abstract class Employee
{
    public abstract void Project();
    public void Role()
    {
        Console.WriteLine("Engineer");
    }
}
```

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();
}
```

2. Abstract class keyword: **ABSTRACT**

2. Interface keyword: **INTERFACE**

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");
    }
}
```

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();
}
```

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.

❖ 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.

```
IS:
```

```
public class PermanentEmployee
{
}
public class ContractualEmployee
{
}
```

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

❖ 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");  
    }  
}
```



```
public class PermanentEmployee  
{  
}  
  
public class ContractualEmployee  
{  
}
```

❖ Benefits of Interfaces:

1. Help in defining the **contract** of the system.
2. **Unit testing** is easy in application having interfaces.
3. Used for implementing **dependency injection**.

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



- ❖ Yes. From C# 8.0 it is possible to put method bodies in Interfaces.

```
public interface IEmployee
{
    public void Role()
    {
        Console.WriteLine("Developer");
    }
}
```

- ❖ No. Abstract class and interface purpose is to act as base class via inheritance. There object creation is not possible.

```
public abstract class Employee
{
    public abstract int Salary();

    public string Role()
    {
        return "Developer";
    }
}
```

```
public interface IEmployee
{
    public int Role();
}
```

```
static void Main(string[] args)
{
    Employee employee = new Employee();

    IEmployee employee2 = new IEmployee();
}
```

The code shows a main method with two variable declarations. The first variable, 'employee', is of type 'Employee' (an abstract class), and the second variable, 'employee2', is of type 'IEmployee' (an interface). A tooltip appears over the 'new IEmployee()' line, containing the following information:

- interface Can_you_create_object_of_an_abstract_class.Program.IEmployee
- CS0144: Cannot create an instance of the abstract type or interface 'Program.IEmployee'
- Show potential fixes (Alt+Enter or Ctrl+.)

- ❖ NO. Interface can only be used for inheritance, so constructor is not required.

```
interface IEmployee1
{
    public IEmployee1()
    {
        IEmployee1.IEmployee10
    }
    public void Project1();
}
```

CS0526: Interfaces cannot contain instance constructors

- ❖ YES, Abstract class can have constructors.
- ❖ The reason is, when the abstract class is inherited in the derived class, and whenever the object of the derived class is created then FIRST the constructor of the abstract/ base class will be called and then only the constructor of derived class will be called.

```
public abstract class Employee
{
    public Employee()
    {
        Console.WriteLine("Employee Constructor");
    }
}

public class PermanentEmployee : Employee
{
}
```

```
static void Main(string[] args)
{
    //Employee employee = new Employee();
    PermanentEmployee employee = new PermanentEmployee();
}

//Output: Employee Constructor
```

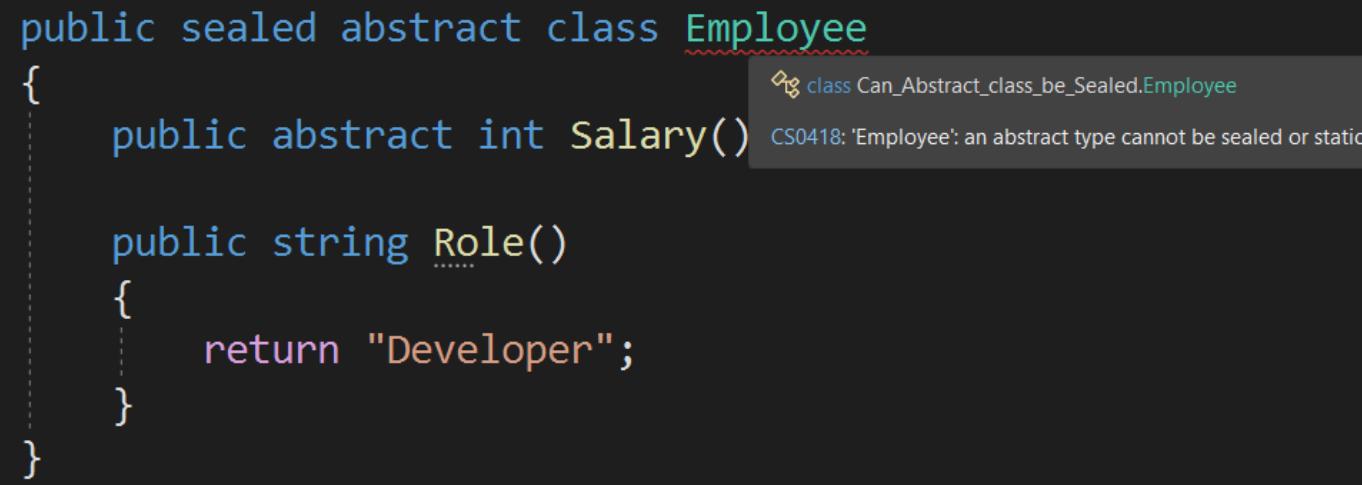
- ❖ Abstraction means showing only required things and hide the **BACKGROUND** details.

- ❖ Abstract class is one of the ways to implement abstraction.

- ❖ NO. Abstract class are used for inheritance, but sealed and static both will not allow the class to be inherited.

```
public sealed abstract class Employee
{
    public abstract int Salary();

    public string Role()
    {
        return "Developer";
    }
}
```



The screenshot shows a portion of a C# code editor. A tooltip is displayed over the word 'Employee' in the first line of code. The tooltip contains the text 'CS0418: 'Employee': an abstract type cannot be sealed or static'. The code itself defines an abstract class named 'Employee' with two methods: 'Salary()' and 'Role()'. The 'Role()' method returns the string 'Developer'.

- ❖ NO. Abstract methods are only declaration, and they must be implemented in the derive class, therefore they can not be private.

```
public abstract class Employee
{
    private abstract int Salary();
    public string Role()
    {
        return "Developer";
    }
}
```

CS0621: 'Employee.Salary()': virtual or abstract members cannot be private

- ❖ NO. Only at maximum one abstract class can be used for inheritance.
- ❖ But multiple interfaces can be used.

```
public abstract class Technology
{
    public abstract void DotNet();
}

public abstract class Role
{
    public abstract void Lead();
}

public class Employee : Technology, Role
```

class Does_Abstract_class_support_multiple_Inheritance.Role

CS1721: Class 'Employee' cannot have multiple base classes: 'Technology' and 'Role'

Chapter 4 : OOPS & C# - Access Specifiers, Boxing, Unboxing, String & StringBuilder

Q38. What are Access Specifiers? V Imp

Q39. What is internal access modifier? Show example?

Q40. What is the default access modifier in a class?

Q41. What is Boxing and Unboxing? V Imp

Q42. Which one is explicit Boxing or Unboxing?

Q43. Is Boxing and Unboxing good for performance?

Q44. What are the basic string operations in C#?

Q45. What is the difference between “String” and “StringBuilder”? V Imp

Q46. When to use String and when StringBuilder in real applications?

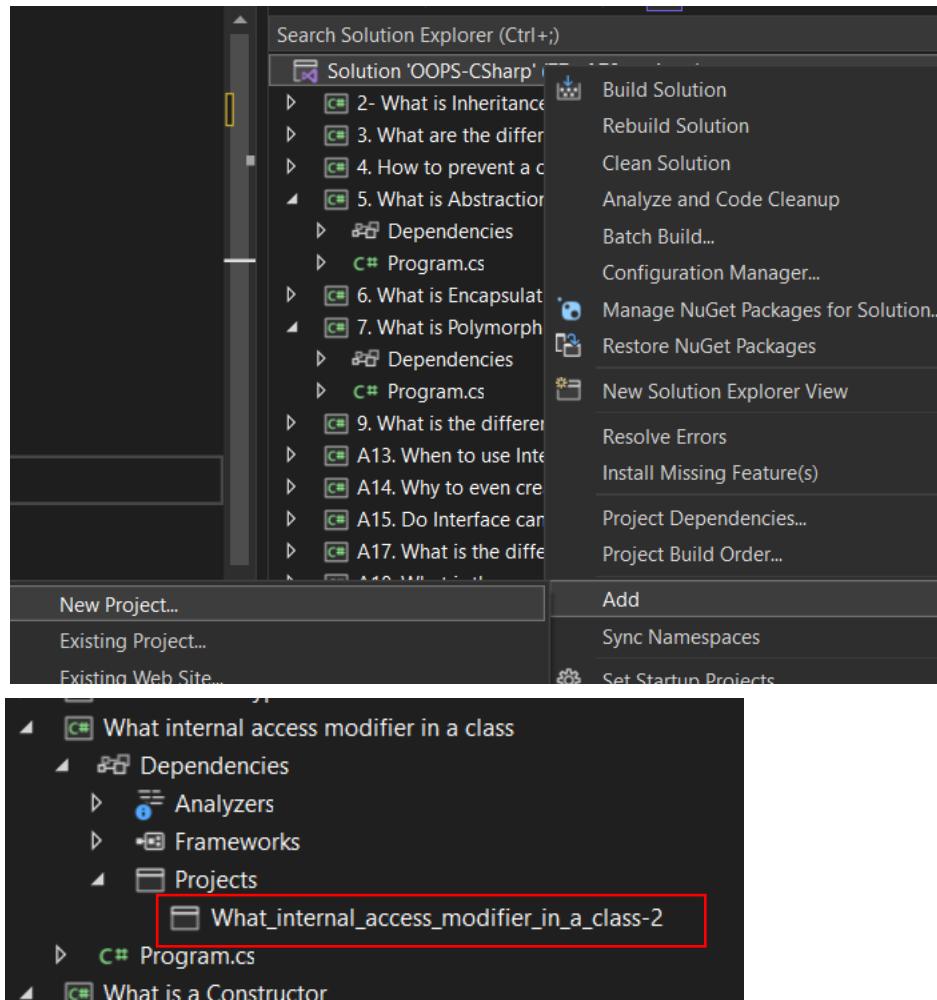
Q47. What is String Interpolation in C#?

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	✓	✓	✓	✓	✗

- ❖ Access specifiers are keywords to specify the **accessibility** of a class, method, property, field.
- ❖ The keywords are – Public, Private, Protected, Internal, Protected Internal.

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 modifier is used to tell that access is limited to the current assembly.



```
namespace What_internal_access_modifier_in_a_class_2
{
    internal class Employee
    {
        public int GetSalary()
        {
            return 100000;
        }
    }
}
```

```
using What_internal_access_modifier_in_a_class_2;

namespace What_internal_access_modifier_in_a_class
{
    public class Program
    {
        static void Main(string[] args)
        {
            Employee employee = new Employee();
        }
    }
}
```

CS0122: 'Employee' is inaccessible due to its protection level
Show potential fixes (Alt+Enter or Ctrl+.)

Internal is the default access modifier of a class.

```
??? class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
    }
}
```

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

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

    object obj = num;      //Boxing

    int i.... = (int)obj;  //Unboxing

}
```

- ❖ We internally use boxing when item is added to ArrayList. And we use unboxing when item is extracted from ArrayList.

```
|     int num = 100;  
|  
ArrayList arrayList = new ArrayList();  
  
arrayList.Add(i);      //Boxing  
  
int k = (int)arrayList[0]; //Unboxing
```

- ❖ Unboxing is explicit conversion process.

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

    object obj = num;      //Boxing

    int i... = (int)obj;   //Unboxing

}
```

- ❖ No, it is recommended to use boxing and unboxing when it is necessary only.
- ❖ Converting from one type to another type is not good from performance point of view.

```
ArrayList arrayList = new ArrayList();  
  
arrayList.Add(i); //Boxing  
  
int k = (int)arrayList[0]; //Unboxing
```

❖ **Concatenate:**

Two strings can be concatenated either by using a System.String.Concat or by using + operator.

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

//Output: This is one This is two
```

❖ **Replace:**

Replace(a,b) is used to replace a string with another string.

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

//Output: This is two
```

❖ **Trim:**

Trim() is used to trim the white spaces in a string at the end.

```
string str1 = "This is one  ";
str1.Trim();
//Output: "This is one"
```

❖ **Contains:**

Check if a string contains a pattern of substring or not.

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

//Output: true

- ❖ String is **IMMUTABLE** in C#.

It means if you defined one string then you couldn't modify it. Every time you will assign some value to it, it will create a new string.

```
String str1 = "Interview";  
str1 = str1 + "Happy";  
Console.WriteLine(str1);
```

Both these strings
are different and
occupy different
memory in
process

- ❖ StringBuilder is **MUTABLE** in C#.

This means if any manipulation will be done on string, then it will not create a new instance every time.

```
StringBuilder str2 = new StringBuilder();  
str2.Append("Interview");  
str2.Append("Happy");
```

Only one string in
memory.

- ❖ If you want to change a string **multiple times**, then **StringBuilder** is a better option from **performance** point of view because it will not require new memory every time.

```
String str1 = "Interview";  
  
str1 = ..... + "Happy";  
  
Console.WriteLine(str1);
```

```
StringBuilder str2 = new .....;  
  
str2.Append("Interview");  
  
str2.Append("Happy");
```

- ❖ String interpolation is a technique that enables you to insert expression values into strings.

```
static void Main(string[] args)
{
    string name = "Happy";

    Console.WriteLine("My name is {0}", name);

    Console.WriteLine("My name is " + name);

    //String Interpolation
    Console.WriteLine($"My name is {name}");

    //Output: My name is Happy
}
```

Chapter 5 : OOPS & C# - Loops, Conditions, Exception Handling

Q48. What are the **Loop types** in C#? When to use what in real applications?

Q49. What is the difference between “**continue**” and “**break**” statement? V Imp

Q50. What are the alternative ways of writing if-else conditions? When to use what?

Q51. How to implement **Exception Handling** in C#? V Imp

Q52. Can we execute multiple Catch blocks?

Q53. When to use Finally in real applications?

Q54. Can we have only “**Try**” block without “**Catch**” block in real applications?

Q55. What is the difference between **Finally** and **Finalize**?

Q56. What is the difference between “**throw ex**” and “**throw**”? Which one to use in real applications? V Imp

```
//While Loop  
  
int i = 0; -----> Initialize  
while(i < 5) -----> Condition  
{  
    Console.WriteLine(i);  
    i++; -----> Increment  
}  
  
//Output: 0 1 2 3 4
```

```
//Do While  
  
int j = 100;  
do  
{  
    Console.WriteLine(j);  
    j++;  
}  
while(j < 10);  
  
//Output: 100
```

Whether condition true or false, this statement will run at least first time.

```
//For Loop  
  
for(int k = 0 ; k < 5; k++)  
{  
    Console.WriteLine(k);  
}  
  
//Output: 0 1 2 3 4
```

```
//ForEach Loop  
  
int[] arr = new int[] { 1,2,3,4};  
  
foreach (int items in arr)  
{  
    Console.WriteLine(items);  
}  
  
//Output: 1 2 3 4
```

- ❖ **Continue** statement end the loop iteration and transfer the control to the beginning of the loop.
- ❖ **Break** statement end the loop iteration and exit the loop.

```
for (int i = 0; i < 5; i++)  
{  
    if (i == 3)  
    {  
        continue;  
    }  
    Console.WriteLine("Print: " + i);  
}
```

//Output:
Print: 0
Print: 1
Print: 2
Print: 4

```
for (int i = 0; i < 5; i++)  
{  
    if (i == 3)  
    {  
        break;  
    }  
    Console.WriteLine("Print: " + i);  
}
```

//Output:
Print: 0
Print: 1
Print: 2

❖ Switch statement

```
switch (level)
{
    case "SE":
        salary = 70000;
        break;
    case "SSE":
        salary = 100000;
        break;
    default:
        salary = 150000;
        break;
}

Console.WriteLine(salary);
```

❖ Ternary Operator ?:

```
salary = level == "SE" ? 70000 : level == "SSE" ? 100000 : 150000;

Console.WriteLine(salary);
```

❖ If-else if-else statement

```
int salary;

string level = "SSE";

if (level == "SE")
{
    salary = 70000;
}
else if (level == "SSE")
{
    salary = 100000;
}
else
{
    salary = 150000;

}

Console.WriteLine(salary);
```

- ❖ 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
```

❖ 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 (ArithmetcException ex)
{
    Console.WriteLine("Alert");
    Console.WriteLine(ex.Message);
}
catch (ArgumentOutOfRangeException ex)
{
    Console.WriteLine(ex.Message);
}
```

- ❖ 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();
}
```

- ❖ YES - We can have only try block without catch block, but then we must have **finally** block with try block.

```
using System;
using System.Data.SqlClient;

class Application
{
    static void Main(string[] args)
    {
        SqlConnection con = new SqlConnection("conString");
        try
        {
            con.Open();
            Random rnd = new Random();
            int num = rnd.Next();

            if(num == 5)
            {
                return;
                //After this nothing will execute
            }
            //con.Close();
        }
        finally
        {
            con.Close();
        }
    }
}
```

- ❖ 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.

- ❖ Throw ex will change the stack trace, where as throw will preserve the whole **stack trace**.

Error:

```
at Throw_ex_and_throw.Program.DivideZeroByZero() in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 32
  at Throw_ex_and_throw.Program.Main(String[] args) in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 15
```

```
11     static void Main(string[] args)
12     {
13         try
14         {
15             DivideZeroByZero();
16         }
17         catch (Exception ex)
18         {
19             Console.WriteLine(ex.StackTrace);
20             Console.ReadLine();
21         }
22     }
23     public static void DivideZeroByZero()
24     {
25         try
26         {
27             int i = 0, j = 0;
28             int k = i / j;
29         }
30         catch (Exception ex)
31         {
32             throw ex;
33         }
34     }
}
```

- ❖ Throw ex will change the stack trace, where as throw will preserve the whole stack trace.

Error:

```
at Throw_ex_and_throw.Program.DivideZeroByZero() in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 28
  at Throw_ex_and_throw.Program.Main(String[] args) in
D:\InterviewHappy\InterviewHappy-Hindi\31-Oct-
2022\OOP_CSharp_Code\Throw_ex_Throw\Program.cs:line 15
```

- ❖ Its a best practice to use *throw* as it preserve the whole **stack trace**.

```
11  static void Main(string[] args)
12  {
13      try
14      {
15          DivideZeroByZero();
16      }
17      catch (Exception ex)
18      {
19          Console.WriteLine(ex.StackTrace);
20          Console.ReadLine();
21      }
22  }
23  public static void DivideZeroByZero()
24  {
25      try
26      {
27          int i = 0, j = 0;
28          int k = i / j;
29      }
30      catch (Exception ex)
31      {
32          throw;
33      }
34 }
```

Chapter 6 : OOPS & C# - Generics & Collections

Q57. Explain **Generics** in C#? When and why to use? V Imp

Q58. What are **Collections** in C# and what are their types?

Q59. What is the difference between **Array** and **ArrayList** (atleast 2)? V Imp

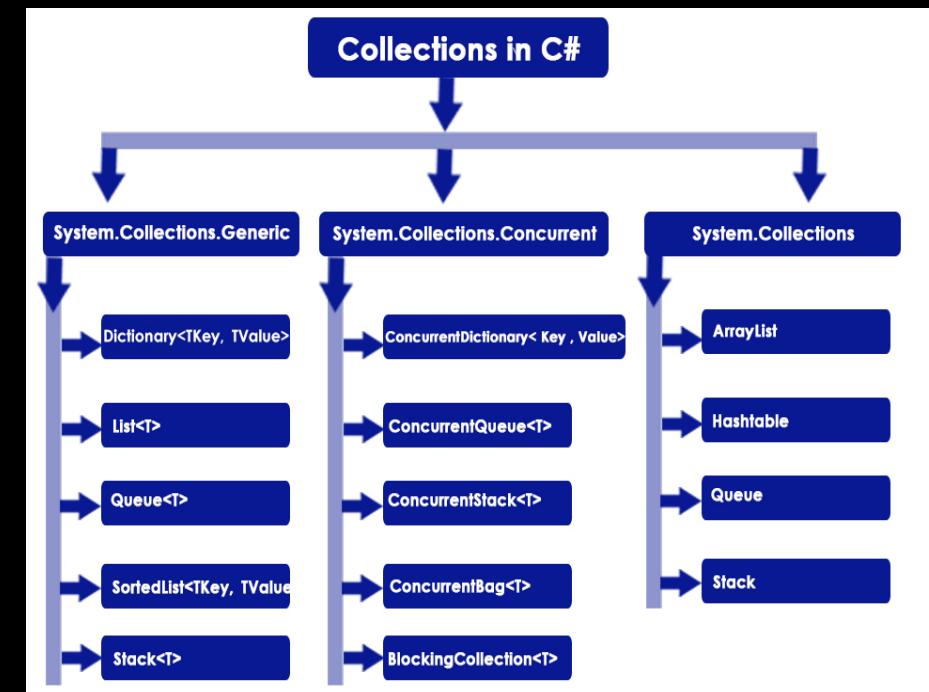
Q60. What is the difference between **ArrayList** and **Hashtable**?

Q61. What is the difference between **List** and **Dictionary** Collections? V Imp

Q62. What is **IEnumerable** in C#?

Q63. What is the difference between **IEnumerable** and **IEnumerator** in C#?

Q64. What is the difference between **IEnumerable** and **IQueryable** in C#? 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");
}
```

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

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

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

❖ 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");
}
```

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

❖ Generic Class

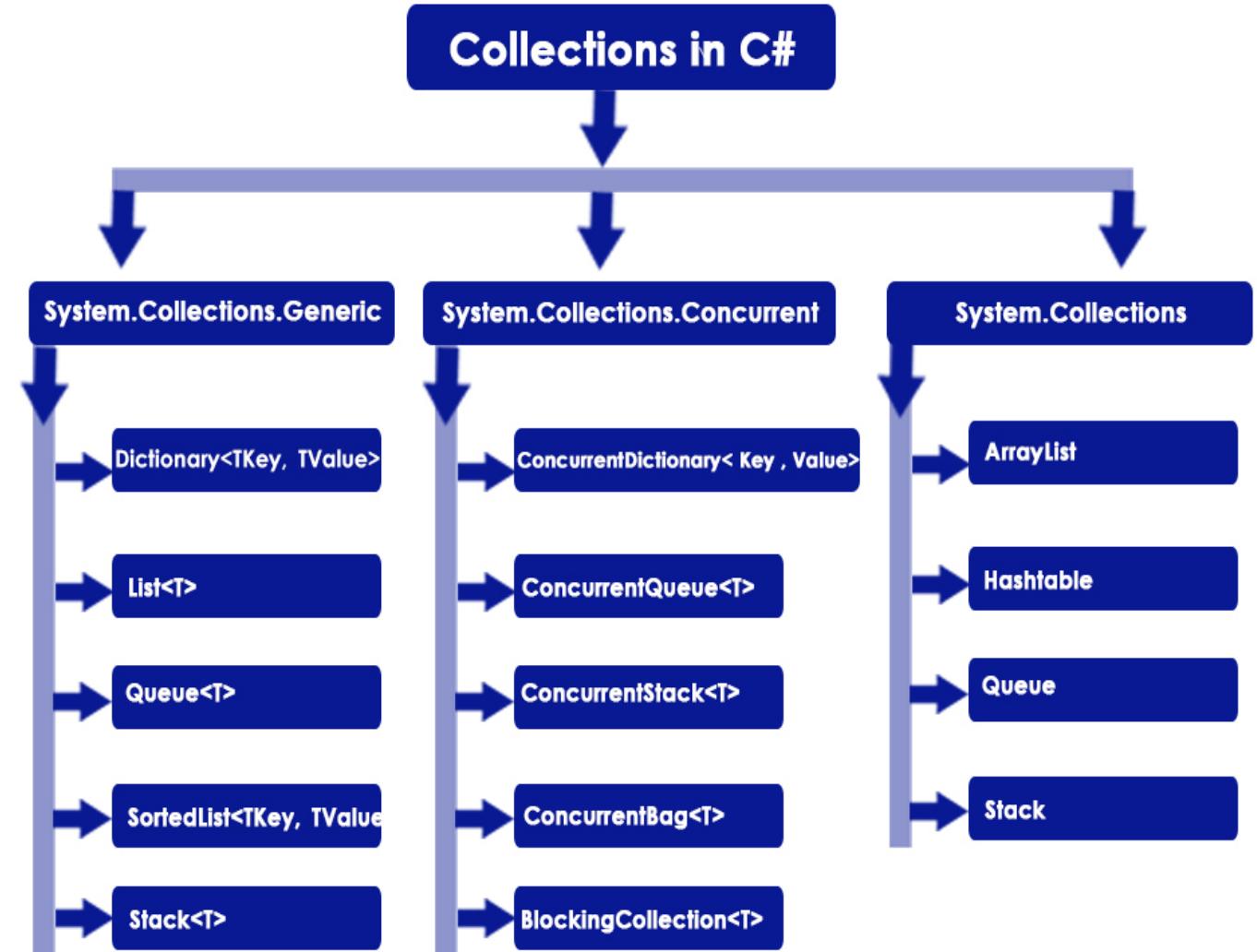
```
public class Calculator<T>
{
    public static bool AreEqual(T value1, T value2)
    {
        return value1.Equals(value2);
    }
}
```

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

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

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

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



❖ Array

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.

❖ ArrayList

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.

- ❖ In ArrayList we can only add Items/ Values to the list.

```
ArrayList arrList = new ArrayList();  
  
arrList.Add(7896);  
  
arrList.Add("Happy");
```

Value

- ❖ In Hashtable we can add Items/Values with the Keys.

```
Hashtable hashTable = new Hashtable();  
  
hashTable.Add("Number", 1);  
  
hashTable.Add("Car", "Ferrari");
```

Key Value

- ❖ 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
```

- ❖ IEnumerable interface is used when we want to ITERATE among our collection classes using a **FOREACH** loop.

```
static void Main(string[] args)
{
    var employees = new List<Employee>() {
        new Employee(){ Id = 1, Name="Bill" },
        new Employee(){ Id = 2, Name="Steve" }
    };

    foreach (var employee in employees)
    {
        Console.WriteLine(employee.Id + ", " + employee.Name);
    }

    Console.ReadLine();
}
```

```
public class Employee
{
    public int Id { get; set; }
    public string Name { get; set; }
}
```

```
namespace System.Collections.Generic
{
    ...
    public class List<T> : ICollection<T>, IEnumerable<T>, IEnumerable, IList<T>,
    {
        ...
        public List();
        ...
        public List(IEnumerable<T> collection);
        ...
        public List(int capacity);
```

- ❖ IEnumerable internally uses IEnumerator only to iterate the collection via foreach loop.
- ❖ IEnumerable simplifies the use of IEnumerator.

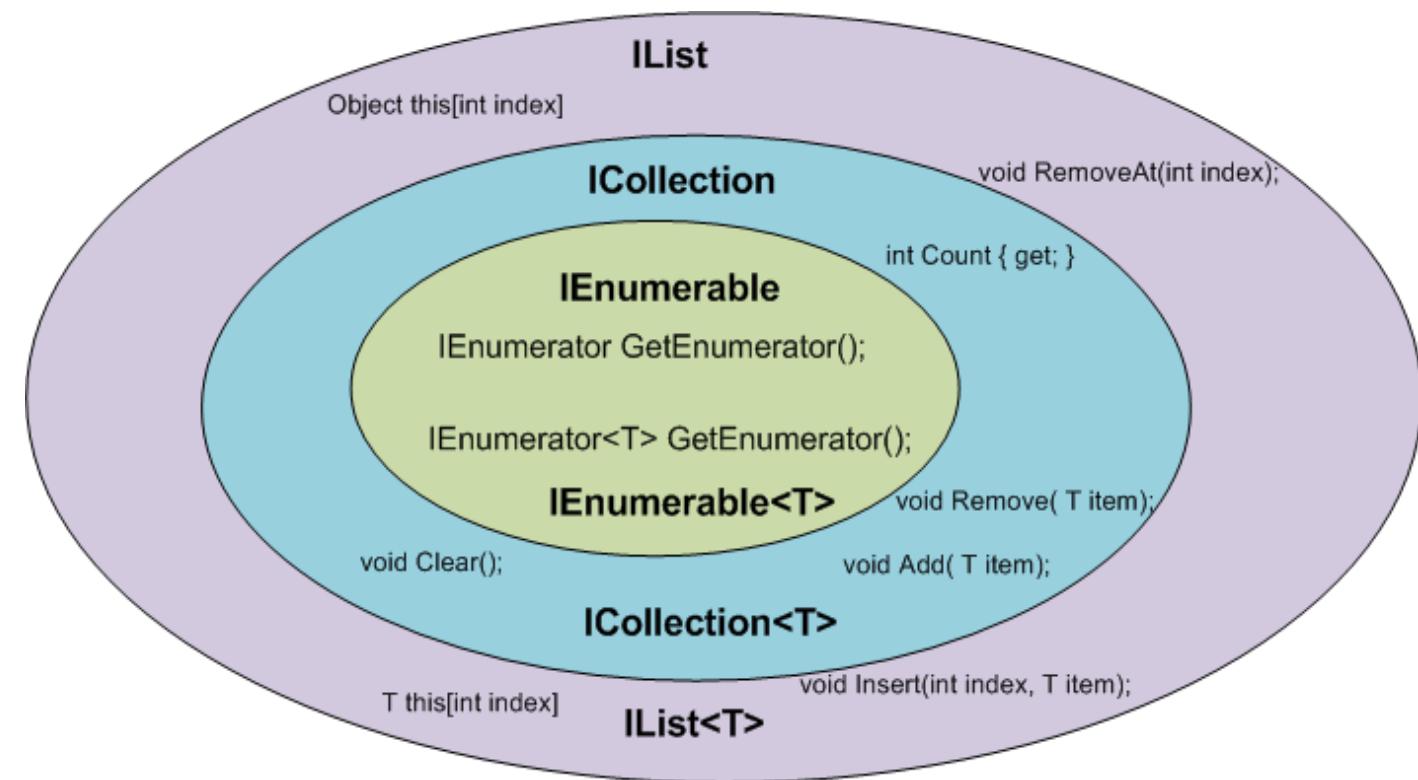
```
//IEnumerable Example
List<string> employees = new List<string>();
employees.Add("Happy");
employees.Add("Joe");
employees.Add("John");

IEnumerable<string> iEnumerableEmployees = employees;

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

```
//IEnumerator Example
IEnumerator<string> iEnumeratorEmployees = employees.GetEnumerator();

while (iEnumeratorEmployees.MoveNext())
{
    Console.WriteLine(iEnumeratorEmployees.Current);
}
//Output: Happy Joe John
```



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

- ❖ 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
```

```
//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
```

- ❖ IEnumerable is used with in-memory collection.

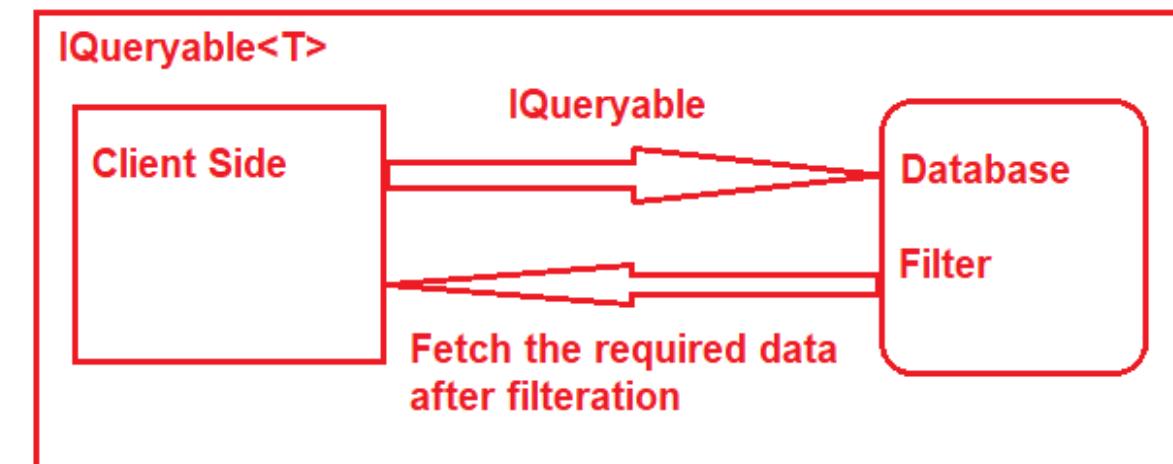
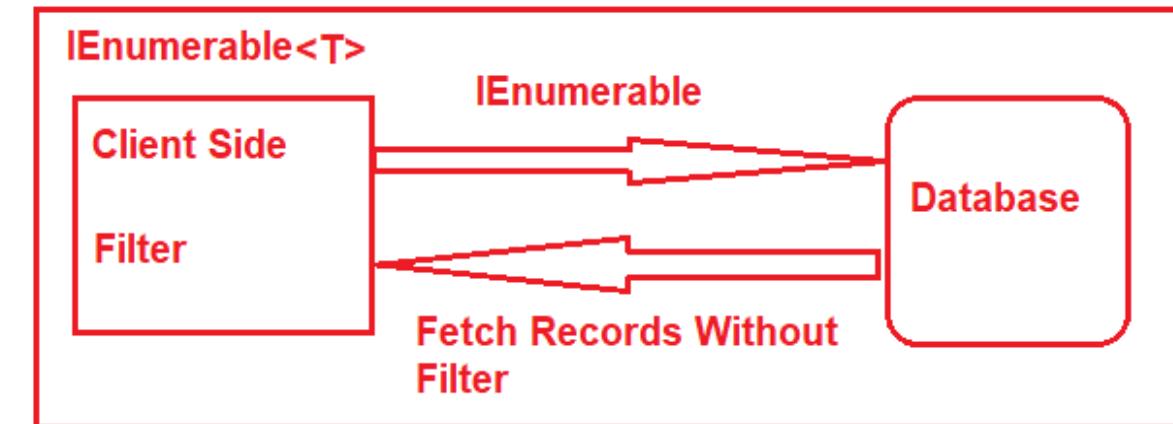
- ❖ IQueryable is better in getting result from database.

```
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"));
```

- ❖ 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.
IEnumerable is under System.Collections namespace.



Chapter 7 : OOPS & C# - Constructors

Q65. What is a **Constructor**? When to use constructor in real applications?

Q66. What are the types of constructor? V Imp

Q67. What is **Default** constructor?

Q68. What is **Parameterized** constructor?

Q69. What is **Static** constructor? What is the use in real applications?

Q70. Can we have parameters or access modifier in static constructor?

Q71. What is **Copy** constructor?

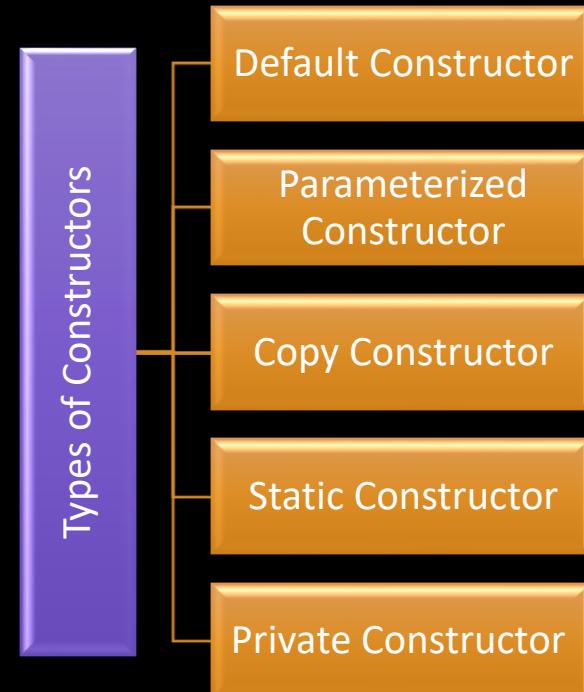
Q72. What is **Private** constructor? What is the use?

Q73. What is **Constructor overloading**?

Q74. What is **Destructor**?

Q75. Can you create object of class with private constructor in C#?

Q76. If base class and child class both have constructor which one will be called first, when derived class object is created?

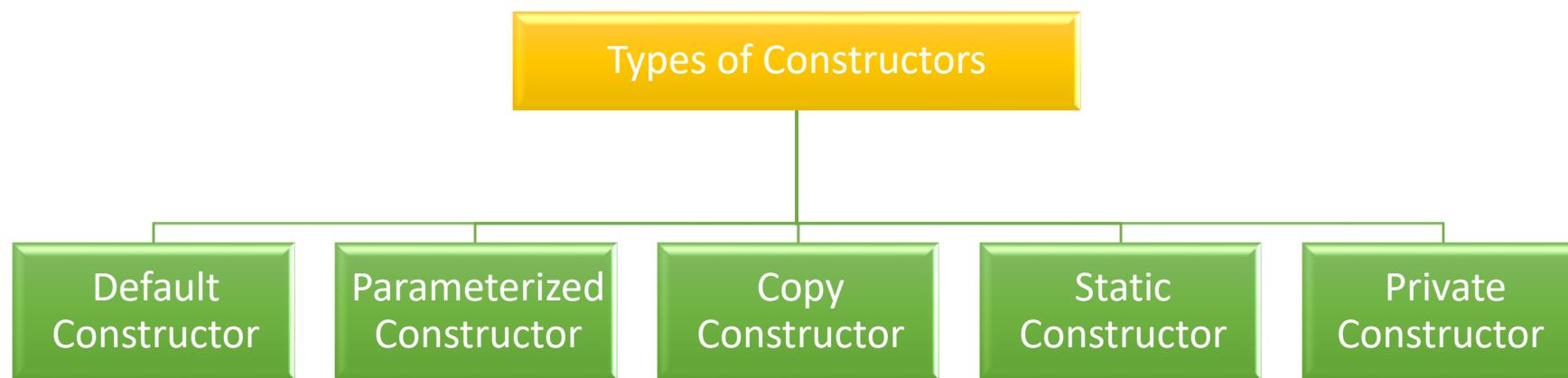


- ❖ A constructor is a **specialized method** in the class which gets executed when a class object is created.
- ❖ Constructor name will same as of Class name.
- ❖ A constructor is used **to set default values** for the class.

```
public class Employee
{
    public Employee()
    {
        Console.WriteLine("CompanyABC");
    }
}
```

```
public class Program
{
    static void Main(string[] args)
    {
        Employee emp = new Employee();

        //Output: CompanyABC
    }
}
```



- ❖ A default constructor is a constructor that is automatically created by the compiler, if no other constructors are defined in the class.

```
public class MyClass
{
    public MyClass()
    {
    }
}
```

```
class Program
{
    static void Main(string[] args)
    {
        MyClass myClass = new MyClass();
    }
}
```

- ❖ A constructor with at least one parameter is a parameterized constructor.

```
public class MyClass
{
    public MyClass(int x)
    {
        Console.WriteLine(x);
    }
}
```

```
public class Program
{
    static void Main(string[] args)
    {
        MyClass myClass = new MyClass(100);
    }
}

//Output: 100
```

- ❖ Static constructor is used to be called before any static member of a class is called.

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

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

```
static void Main(string[] args)
{
    MyClass.Print();
}

//Output
//Constructor
//Method
```

[back to chapter index](#)

- ❖ No, static constructor can not have parameters or access modifier.

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

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

- ❖ The constructor which creates an object by copying variables from another object is called a copy constructor.

```
static void Main(string[] args)
{
    MyClass myClass = new MyClass("Happy");

    MyClass myClassCopy = new MyClass(myClass);

    Console.WriteLine(myClassCopy.name);
}

//Output: Happy
```

```
public class MyClass
{
    public string name;

    //Parametrized Constructor
    public MyClass(string name)
    {
        this.name = name;
    }

    //Copy Constructor
    public MyClass(MyClass copy)
    {
        name = copy.name;
    }
}
```

- ❖ When a constructor is created with a private specifier, it is not possible for other classes to derive from this class, neither is it possible to create an instance of this class.

```
public class MyClass
{
    private MyClass()
    {
        Console.WriteLine("Private");
    }
}
```

```
static void Main(string[] args)
{
    MyClass myClass = new MyClass();
}
```

- ❖ Constructor Overloading is a technique to define multiple constructors within a class with different sets of parameters.

```
public class Employee
{
    public Employee()
    {
        Console.WriteLine("Blank");
    }
    public Employee(int id)
    {
        Console.WriteLine(id);
    }
    public Employee(int id, string name)
    {
        Console.WriteLine(id + " " + name);
    }
}
```

```
static void Main(string[] args)
{
    Employee employee = new Employee();

    Employee employee1 = new Employee(30);

    Employee employee2 = new Employee(50, "Happy");
}
//Output: Blank
//30
//50 Happy
```

- ❖ Constructors in C# are methods inside the class used to destroy instances of that class when they are no longer needed.
- ❖ The Destructor is called implicitly by the .NET Framework's Garbage collector

```
public class Employee
{
    public Employee()
    {
    }

    ~Employee()
    {
    }
}
```

❖ No

```
public class PrivateConstructor
{
    private PrivateConstructor()
    {
        Console.WriteLine("Private Constructor Called");
    }
}
```

```
static void Main(string[] args)
{
    PrivateConstructor privateConstructor = new PrivateConstructor();
}
```



A screenshot of a C# IDE showing a tooltip for a private constructor call. The tooltip contains the following information:

- Lightbulb icon with a dropdown arrow.
- Text: "class Private_Constructor.PrivateConstructor (+ 1 overload)"
- Text: "CS0122: 'PrivateConstructor.PrivateConstructor()' is inaccessible due to its protection level"
- Text: "Show potential fixes (Alt+Enter or Ctrl+.)"

❖ Base class

```
static void Main(string[] args)
{
    DerivedClass obj = new DerivedClass();
    Console.ReadLine();
}
//Output
//Base
//Derived
```

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

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

Chapter 8 : OOPS & C# - Method Parameters, Delegates & Events

Q77. What is a **Method** in C#?

Q78. What is the difference between **Pass by Value** and **Pass by Reference** Parameters?

Q79. How to return more than one value from a method in C#? V Imp

Q80. What is the difference between “**out**” and “**ref**” parameters? V Imp

Q81. What is “**params**” keyword? When to use params keyword in real applications?

Q82. What are **optional parameters** in a method?

Q83. What are **named parameters** in a method?

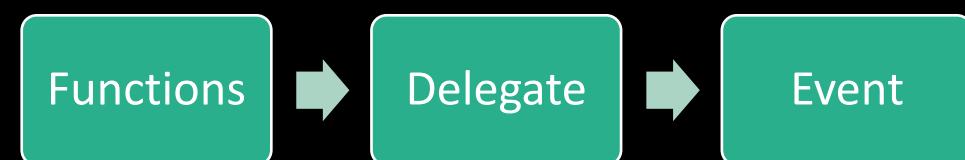
Q84. What are **Extension Methods** in C#? When to use extension methods? V Imp

Q85. What are **Delegates** in C#? When to use delegates in real applications? V Imp

Q86. What are **Multicast Delegates**?

Q87. What are **Anonymous Delegates** in C#?

Q88. What are the differences between **Events** and **Delegates**? V Imp



- ❖ A **method** is a code block that contains a series of statements.

```
public class Employee
{
    public Employee()
    {
        //code
    }

    private int experience;

    public int Experience
    {
        get { return experience; }

        set { experience = value; }
    }

    //public int Experience { get; set; }

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

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



Method

```
//Passing by Value

static void Main(string[] args)
{
    int x = 5;
    int y = 10;

    Console.WriteLine(x + " " + y);
    PassByValue(x, y);
    Console.WriteLine(x + " " + y);

    //Output: 5 10
    //Output: 5 10
}

static void PassByValue(int x, int y)
{
    x = 100;
    y = 200;
}
```

```
//Passing by Reference

static void Main(string[] args)
{
    int x = 5;
    int y = 10;

    Console.WriteLine(x + " " + y);
    PassByReference(ref x, ref y);
    Console.WriteLine(x + " " + y);

    //Output: 5 10
    //Output: 100 200
}

static void PassByReference(ref int x, ref int y)
{
    x = 100;
    y = 200;
}
```

- ❖ By using ref and out keywords.
- ❖ By using a List, ArrayList(any collection) in the return type.

```
//Passing by Reference

static void Main(string[] args)
{
    int x = 5;
    int y = 10;

    Console.WriteLine(x +" "+ y);
    PassByReference(ref x, ref y);
    Console.WriteLine(x + " " + y);

    //Output: 5 10
    //Output: 100 200
}

static void PassByReference(ref int x, ref int y)
{
    x = 100;
    y = 200;
}
```

- ❖ When to use out and when to use ref?

Use out parameter to return a new and fresh value and use ref to modify an existing value.

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. For Ref parameter, Initialization is not necessary before returning.

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

    WithRefOut obj = new WithRefOut();

    int x = obj.Update(out a, ref b);

    Console.WriteLine(x);
    Console.ReadLine();
}

public class WithRefOut
{
    public int Update(out int c, ref int d)
    {
        c = 100;
        return c + d;
    }
}
```

- ❖ Params keyword is used as a parameter which can take the **VARIABLE** number of parameters.
- ❖ It is useful when programmer don't have any prior knowledge about the number of parameters to be used.

```
static void Main(string[] args)
{
    int sum = Add(5, 10, 15, 20, 30, 40);

    Console.WriteLine(sum);
    Console.ReadLine();
}

public static int Add(params int[] numbers)
{
    int total = 0;

    foreach (int i in numbers)
    {
        total += i;
    }
    return total;
}

//Output: 120
```

You can pass any number of parameters here.

- ❖ Optional parameters allow some arguments which are not mandatory to pass, and their default value is set.

```
static void Main(string[] args)
{
    AddOptional(10, 20);

    AddOptional(10, 20, 30);
}
```

```
public static void AddOptional(int i, int j, int k = 50)
{
    Console.WriteLine(i + j + k);
}
```

- ❖ Named parameters are used to specify an argument based on the name of the argument and not the position

```
static void Main(string[] args)
{
    //Normal way
    Add(5, 10, 8, 2, 3, 6);

    //Named Parameters
    Add(a: 2.0, b: 3, c: 6.0, x: 5, y: 10, z: 8);
}
```

```
public static void Add(int x, float y, int z, double a, decimal b, double c)
{
}
```

- ❖ 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);
    }
}
```

- ❖ 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);  
    }  
}
```

- ❖ 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
}
```

- ❖ Delegates pointing methods without name are called anonymous delegates.

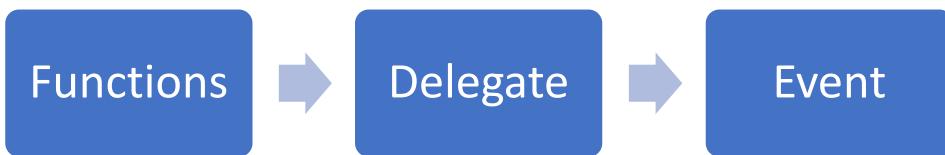
```
public delegate void Calculator(int x, int y);

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

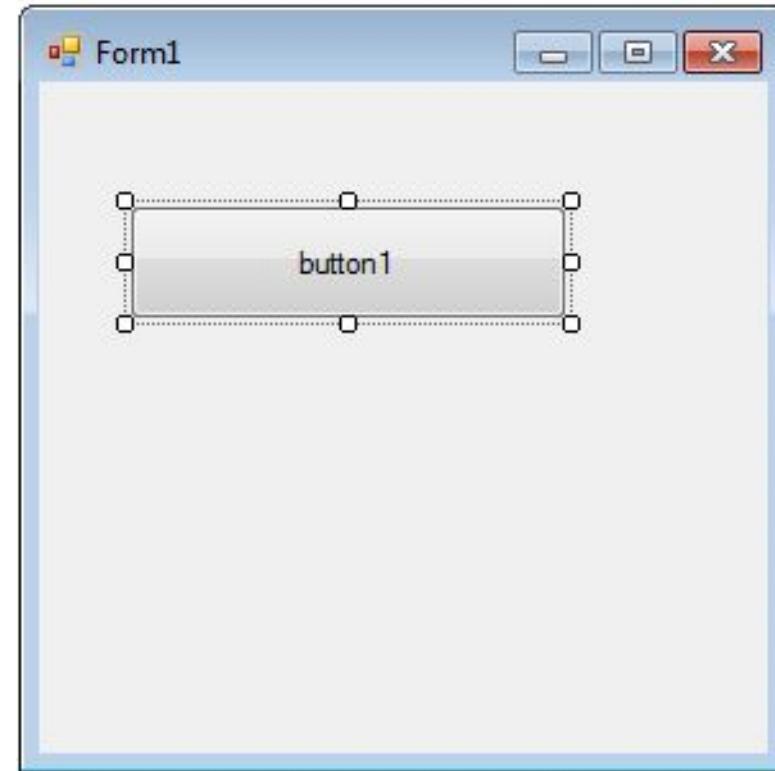
        calcAdd(20, 30);
    }
}

//Output: 50
```

- ❖ 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.



Chapter 9: OOPS & C# - Important Keywords

Q89. What is ‘this’ keyword in C#? When to use it in real applications?

Q90. What is the purpose of “using” keyword in C#? V Imp

Q91. Can we use Using keyword with other classes apart from DBConnection?

Q92. What is the difference between “is” and “as” operators?

Q93. What is the difference between “ Readonly” and “Constant” variables? V Imp

Q94. What is “Static” class? When to use static class in real application?

Q95. What is the difference between “var” and “dynamic” in C#?

Q96. What is **Enum** keyword used for?

Q97. Is it possible to inherit Enum in C#?

Q98. What is the use of **Yield** keyword in C#?

abstract	bool	break	byte	case	catch
char	class	const	continue	default	double
enum	else	false	finally	float	for
foreach	goto	if	int	interface	long
namespace	new	public	private	protected	return
sbyte	short	static	string	struct	switch
throw	true	try	ushort	void	while

- ❖ *this* keyword is used to refer to the CURRENT INSTANCE of the class.

```
class Program
{
    static void Main(string[] args)
    {
        Student std1 = new Student(001, "Jack");
        std1.GetStudent();
    }
}
```

- ❖ *this* keyword avoids the name confusion between class fields and constructor parameters.

```
class Student
{
    public int id;
    public string name;
    public Student(int id, string name)
    {
        this.id = id;
        this.name = name;
    }
    public void GetStudent()
    {
        Console.WriteLine(id + " : " + name);
    }
}
```

- ❖ There are two purpose of using keyword in C#:

1. USING DIRECTIVE

```
using System;
using System.Data.SqlClient;
```

2. USING STATEMENT

- The using statement ensures that DISPOSE() method of the class object is called even if an exception occurs.

```
static void Main(string[] args)
{
    using(var connection = new SqlConnection("ConnectionString"))
    {
        var query = "UPDATE YourTable SET Property = Value";
        var command = new SqlCommand(query, connection);

        connection.Open();
        command.ExecuteNonQuery();

        //connection.Dispose();
    }
}
```

```
... public sealed class SqlConnection : DbConnection, ICloneable
{
```

```
... public abstract class DbConnection : Component, IDbConnection, IDisposable, IAsyncDisposable
{
```

[back to chapter index](#)

- ❖ Yes, using keyword can be used with any class which is inherited from **IDisposable** class. For example, with **StreamReader** class.

- ❖ The **IS** operator is **USED TO CHECK** the type of an object.
- ❖ The **AS** operator is used to **PERFORM CONVERSION** between compatible reference type.

```
static void Main(string[] args)
{
    int i = 5;

    bool check = i is int;

    Console.WriteLine(check);

    //Output: true
}
```

```
static void Main(string[] args)
{
    object obj = "Hello";

    string str1 = obj as string;

    Console.WriteLine(str1);

    //Output: Hello
}
```

- ❖ The **IS** operator will return boolean type.
- ❖ The **AS** operator is not of boolean type.

❖ Constant

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

    public const int myConst1;

    public Example(int b)
    {
        myConst1 = 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.

- ❖ 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");
    }
}
```

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

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

    a = "Interview";

    Console.WriteLine(a);
}
```

```
static void Main(string[] args)
{
    dynamic b = 10;

    b = "Happy";

    Console.WriteLine(b);
}

//Output: Happy
```

- ❖ An **enum** is a special "class" that represents a group of constants.

```
class LevelConstants
{
    public const int Low = 0;

    public const int Medium = 1;

    public const int High = 2;
}
```

```
enum Level
{
    Low,
    Medium,
    High
}
```

```
enum Weekdays
{
    Sunday,
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
}
```

```
static void Main(string[] args)
{
    Level myLevel = Level.Medium;

    Console.WriteLine(myLevel);
}
```

- ❖ Enum is a **sealed** class type, so it cannot be inherited.

```
enum Level
{
    Low,
    Medium,
    High
}

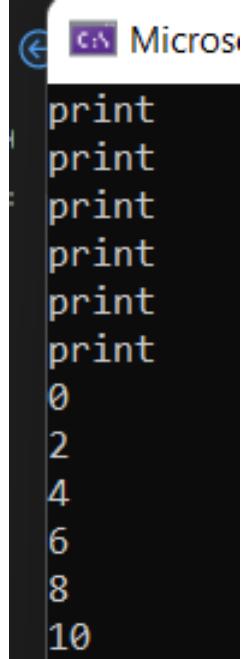
class Inherit : Level
{
}
```

enum What_is_Enum_keyword_used_for.Level
CS0509: 'Inherit': cannot derive from sealed type 'Level'

```
//Input: 10  
  
//Output: 0 2 4 6 8
```

```
static IEnumerable<int> GetEvenNumbers(int upto)  
{  
    List<int> numbers = new List<int>();  
  
    for (int i = 0; i <= upto; i += 2)  
    {  
        numbers.Add(i);  
        Console.WriteLine("print");  
    }  
    return numbers;  
}
```

```
static void Main(string[] args)  
{  
    IEnumerable<int> getEventNumbers = GetEvenNumbers(10);  
  
    foreach(int eventNumber in getEventNumbers)  
    {  
        Console.WriteLine(eventNumber);  
    }  
}
```



- The yield keyword will act as an iterator blocker and generate or return values.

```
//Input: 10  
  
//Output: 0 2 4 6 8
```

```
print  
print  
print  
print  
print  
print  
print  
0  
2  
4  
6  
8  
10
```

```
0  
print  
2  
print  
4  
print  
6  
print  
8  
print  
10  
print
```

```
static IEnumerable<int> GetEvenNumbers(int upto)  
{  
    //List<int> numbers = new List<int>();  
  
    for (int i = 0; i <= upto; i += 2)  
    {  
        //numbers.Add(i);  
        yield return i;  
        Console.WriteLine("print");  
    }  
    //return numbers;  
}
```

```
static void Main(string[] args)  
{  
    IEnumerable<int> getEventNumbers = GetEvenNumbers(10);  
  
    foreach(int eventNumber in getEventNumbers)  
    {  
        Console.WriteLine(eventNumber);  
    }  
}
```

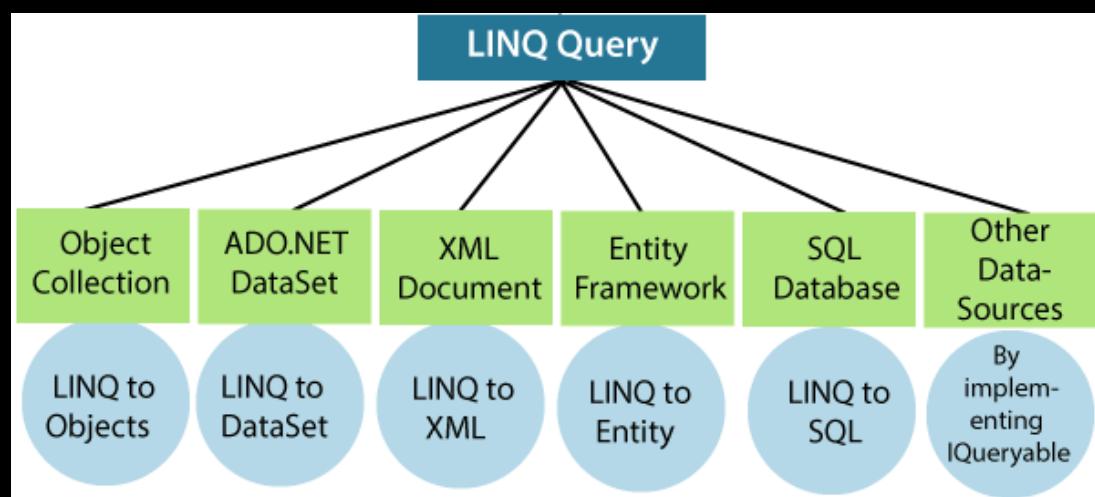
Chapter 10 : OOPS & C# - LINQ

Q99. What is **LINQ**? When to use LINQ in real applications?

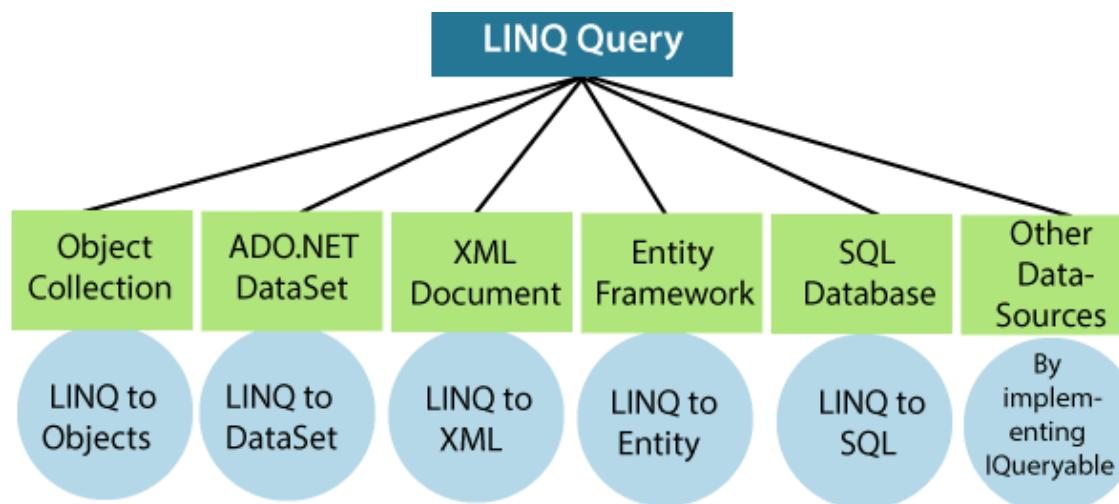
Q100. What are the advantages & disadvantages of **LINQ**?

Q101. What is **Lambda Expressions**? What is the use in real applications?

Q102. What is the difference between **First** and **FirstOrDefault** methods in **LINQ**? V Imp



- ❖ 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
IQueryable<int> filteredNumbers = from n in numbers
where n > 2
select n;
```

Advantages of LINQ

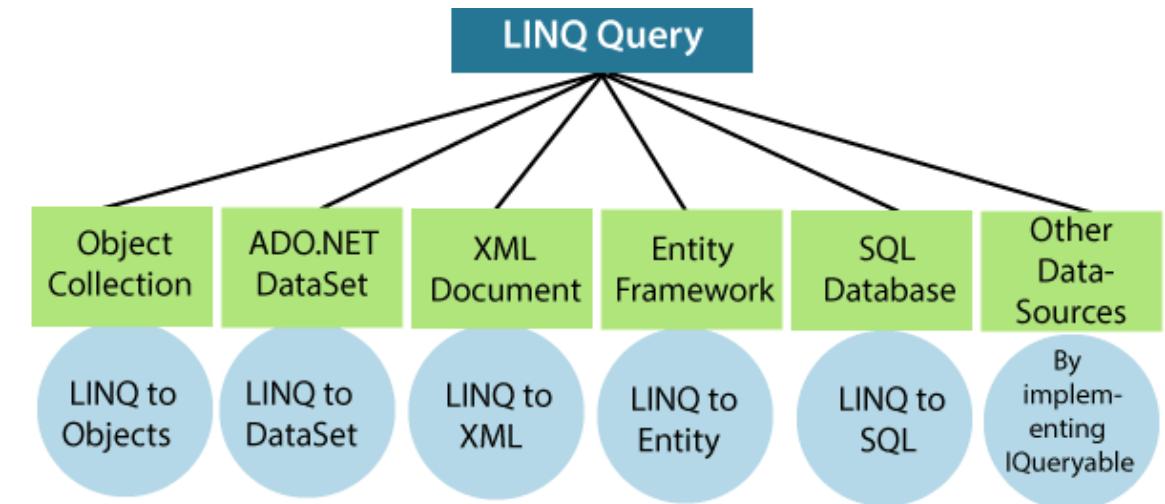
1. Easy and simple syntax to Learn
2. Improved code readability
3. Improved performance
4. Type safety

```
//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



- ❖ 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
```

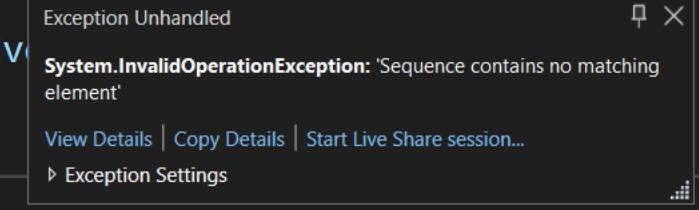
```
List<int> evenNumbers = numbers.FindAll(x => x % 2 == 0);
```

- ❖ First method will return the first value, but it is not able to handle null values.
- ❖ FirstOrDefault will return the first value, and it is able to handle null values also.

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

    int firstEvenNumber = numbers.First(x => x % 2 == 0); ✖

    Console.WriteLine(firstEvenNumber);
}
```



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

    int firstEvenNumber = numbers.FirstOrDefault(x => x % 2 == 0);

    Console.WriteLine(firstEvenNumber);

    //Output: 0
}
```

Chapter 11 : .NET Framework - Basics

Q103. What are the important components of .NET framework? V Imp

Q104. What is an Assembly? What are the different types of assembly?

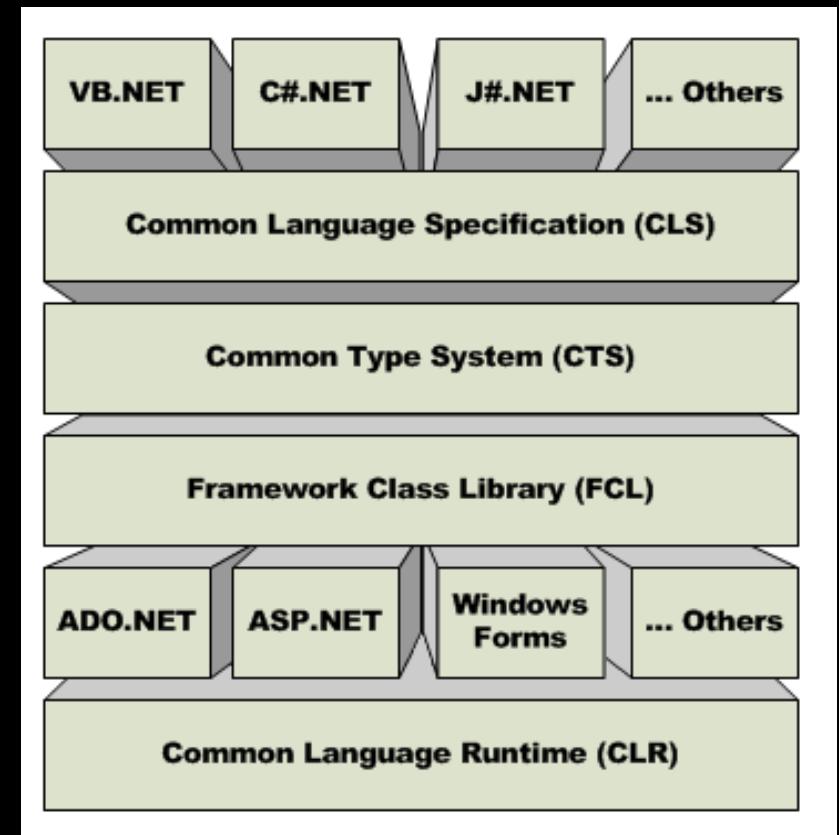
Q105. What is GAC?

Q106. What is Reflection?

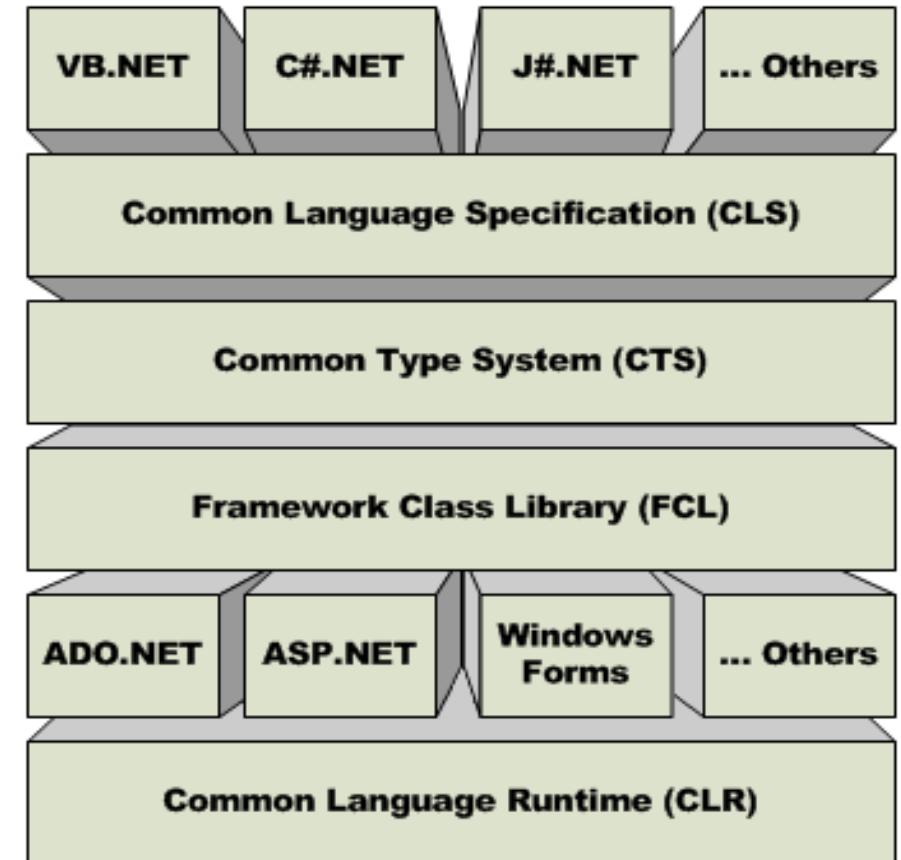
Q107. What are Serialization and Deserialization? V Imp

Q108. What is meant by Globalization and Localization?

Q109. What are Window Services?



- ❖ CLR (Common Language Runtime)
 - ❖ CLR **manages** the execution of programs for example operations like memory management, security checks etc.
 - ❖ CLR provides execution of **managed code** only (Code written in any .NET-supported language such as C#, Visual Basic .NET, or F#).
- ❖ CTS(Common Type System)
 - ❖ CTS is a set of rules that define how **types** are declared, used, and managed in the .NET Framework. Types like int, string, double.
- ❖ CLS(Common Language Specification)
 - ❖ CLS is a **subset** of CTS. It defines a set of **rules** that every language must follow which runs under .NET framework.
 - ❖ The benefit is, if you create same program logic in different languages of .NET, then the compiler will generate same dll.
- ❖ FCL(Framework Class Library)
 - ❖ FCL is the collection of classes, namespaces, interfaces and value types that are used for .NET applications.
 - ❖ For example, String, ArrayList, List classes are provided by FCL only.

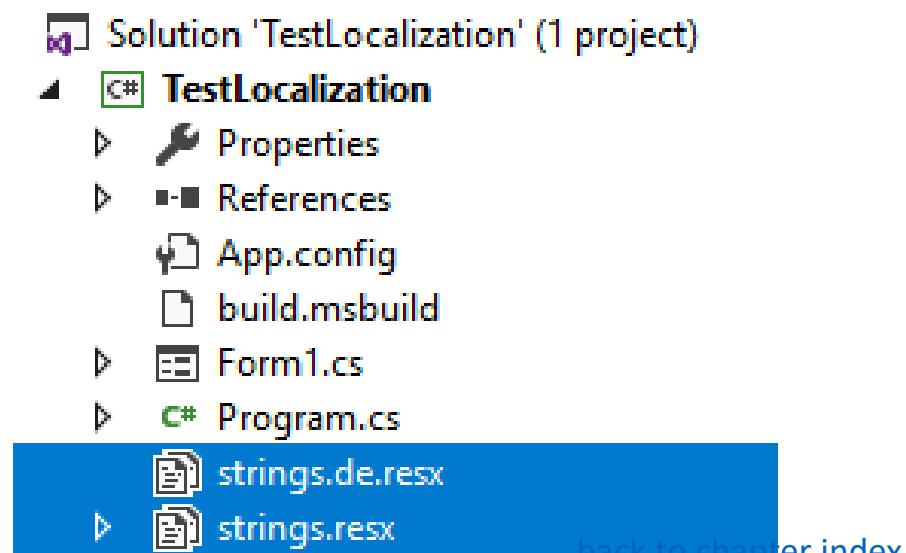
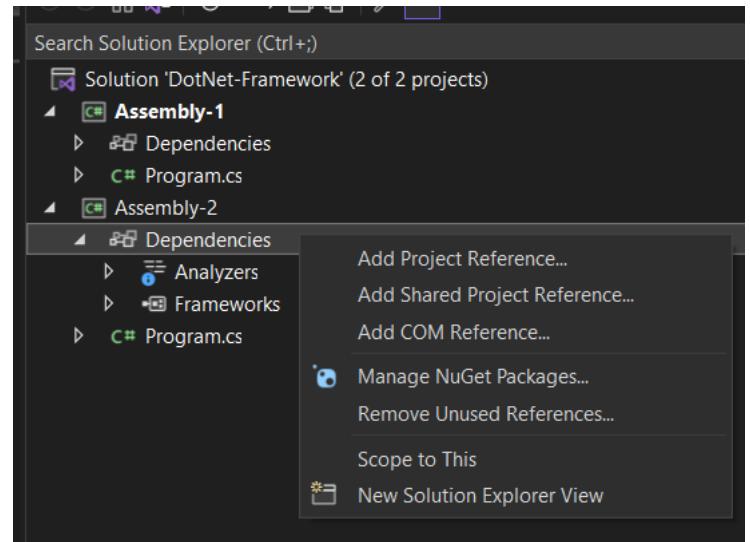


- ❖ **Assembly** is unit of deployment like EXE or a DLL.

When you create a code and build the solution, then the .NET Framework convert your code into Intermediate Language and that is placed inside the assembly(dll), which you can find inside bin folder.

- ❖ There are 3 types of assemblies:

1. **Private assembly** - A private assembly can be used by a single application only. It is not accessible outside. So, all the projects you create will by default create private assembly only.
2. **Public/ shared assembly** - Shared assemblies are usually libraries of code, which **multiple** applications can use. It is registered in the **global assembly cache(GAC)**.
3. **Satellite assembly** - A satellite Assembly is defined as an assembly with resources only, no executable code.



- ❖ GAC stands for **Global Assembly Cache**.
- ❖ GAC is the place where **public assemblies** are stored.
- ❖ Private assembly can be converted into public assembly by adding them in GAC using **gacutil** tool.

- ❖ Reflection is the ability of a code to access the metadata of the assembly during runtime.
- ❖ Metadata is information about data.



❖ Suppose you have to check the version of the assembly at runtime, then you can use reflection.

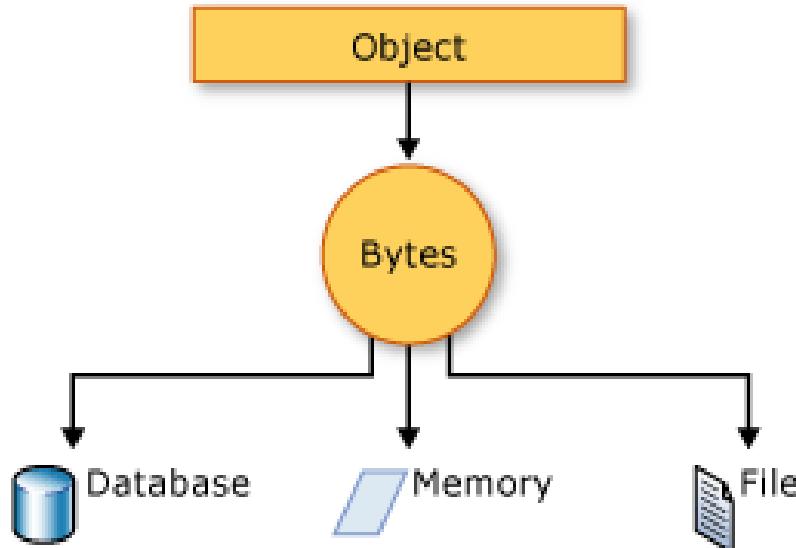
```
// Get the version number of the current assembly
Assembly assembly = Assembly.GetExecutingAssembly();
AssemblyName assemblyName = assembly.GetName();
Version version = assemblyName.Version;
string versionString = version.ToString();

// Print the version number to the console
Console.WriteLine("Assembly version: " + versionString);

Console.Read();
```

```
// Get a reference to the method using reflection
Type t = obj.GetType();
MethodInfo method = t.GetMethod(methodName);
```

- ❖ Serialization is the process of **converting an object** into a format that can be stored, transmitted, or reconstructed later.



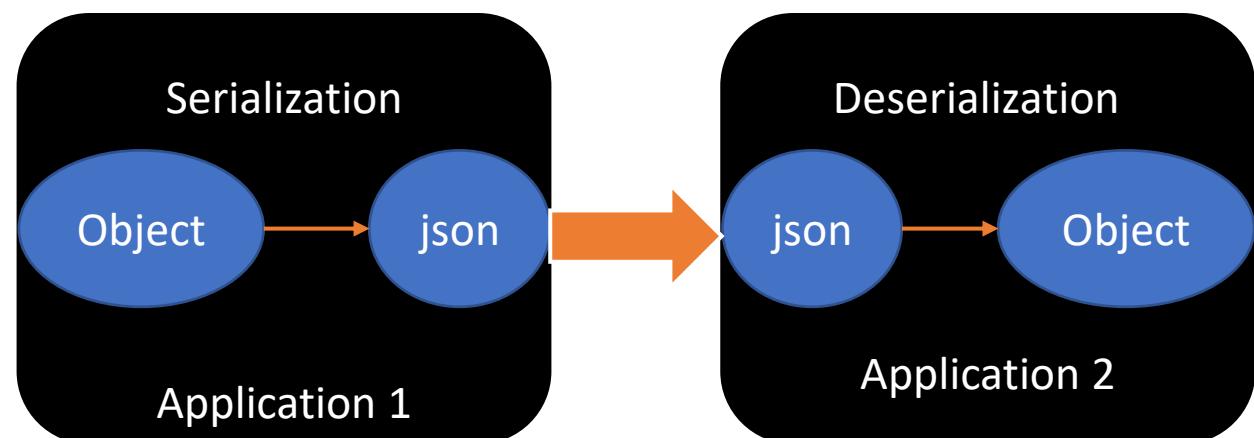
Types of Serialization

Binary
Serialization

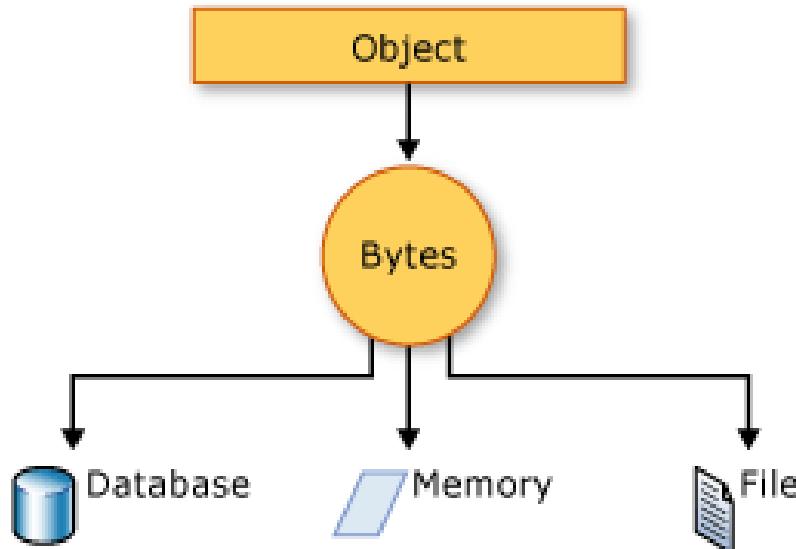
XML
Serialization

JSON
Serialization

- ❖ Deserialization is the process of converting serialized data, such as binary/ XML/ json data, back into an object.



- ❖ Serialization is the process of converting an object into a format that can be stored, transmitted, or reconstructed later.



- ❖ When to use serialization??

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

```
Employee employee = new Employee();  
  
employee.Id = 100;  
employee.Name = "Happy";  
  
//Convert object to json  
string json = JsonConvert.SerializeObject(employee);  
  
Console.WriteLine(json);  
Console.ReadLine();  
  
//Output: {"Id":100,"Name":"Happy"}
```

- ❖ Deserialization is the process of converting serialized data, such as binary/ XML/ json data, back into an object.

- ❖ Globalization is the process of designing and developing a software product that can support different cultures and regions of the world.

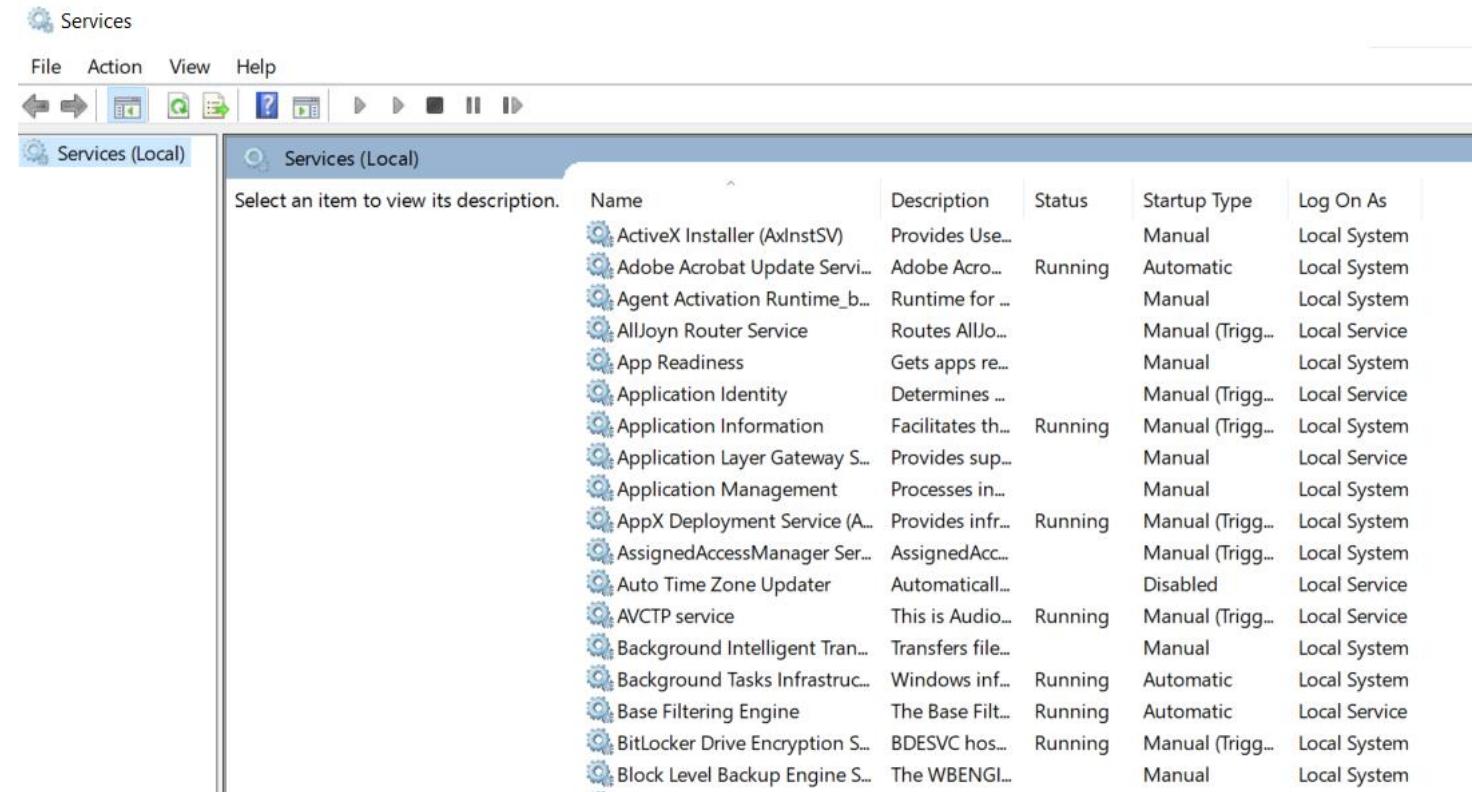
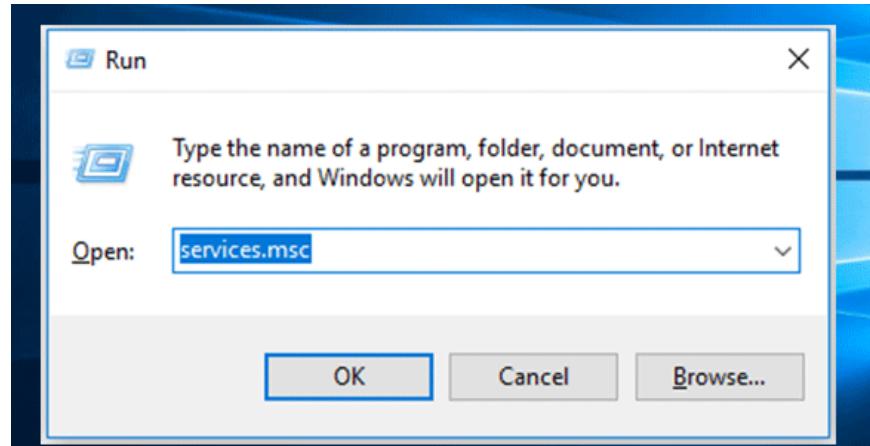


- ❖ Localization is the process of adapting a globalized application, to a particular culture/locale.

- English
- French
- German
- Spanish
- Italian
- Portuguese

-->

- ❖ Windows service is a computer program that runs in the **BACKGROUND** to execute some tasks.



The image shows the Windows Services (Local) console window. The title bar says "Services" and the sub-title bar says "Services (Local)". The menu bar includes File, Action, View, and Help. The toolbar has icons for opening, saving, and filtering. The main pane displays a table of services:

Name	Description	Status	Startup Type	Log On As
ActiveX Installer (AxInstSV)	Provides User...		Manual	Local System
Adobe Acrobat Update Servi...	Adobe Acro...	Running	Automatic	Local System
Agent Activation Runtime_b...	Runtime for ...		Manual	Local System
AllJoyn Router Service	Routes Alljo...		Manual (Trigg...	Local Service
App Readiness	Gets apps re...		Manual	Local System
Application Identity	Determines ...		Manual (Trigg...	Local Service
Application Information	Facilitates th...	Running	Manual (Trigg...	Local System
Application Layer Gateway S...	Provides sup...		Manual	Local Service
Application Management	Processes in...		Manual	Local System
AppX Deployment Service (A...	Provides infr...	Running	Manual (Trigg...	Local System
AssignedAccessManager Ser...	AssignedAcc...		Manual (Trigg...	Local System
Auto Time Zone Updater	Automaticall...	Disabled	Local Service	
AVCTP service	This is Audio...	Running	Manual (Trigg...	Local Service
Background Intelligent Tran...	Transfers file...		Manual	Local System
Background Tasks Infrastruc...	Windows inf...	Running	Automatic	Local System
Base Filtering Engine	The Base Filt...	Running	Automatic	Local Service
BitLocker Drive Encryption S...	BDESVC hos...	Running	Manual (Trigg...	Local System
Block Level Backup Engine S...	The WBENGI...		Manual	Local System

- ❖ Windows services can be started **AUTOMATICALLY** or manually.
- ❖ You can also manually pause, stop and restart Windows services.

Chapter 12 : .NET Framework - Garbage Collection

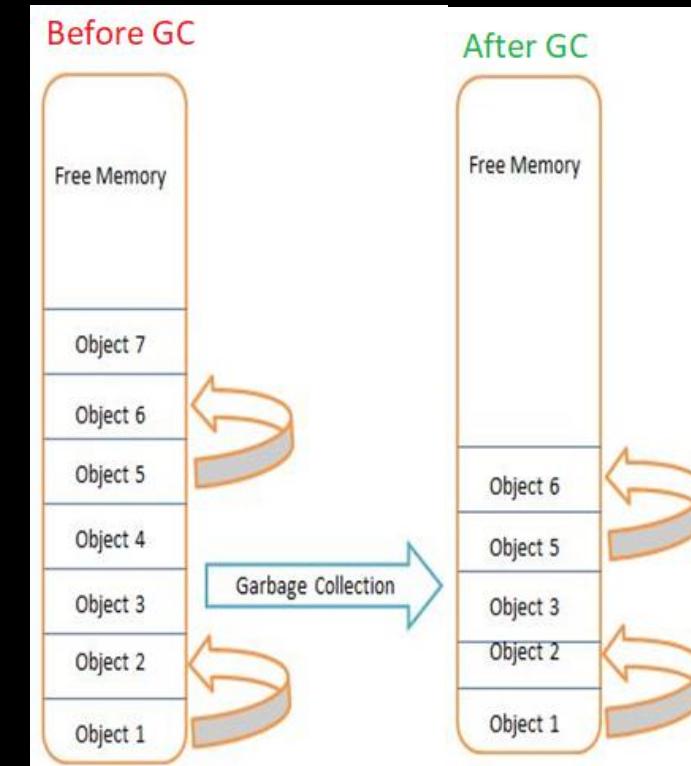
Q110. What is Garbage Collection(GC)? V Imp

Q111. What are Generations in garbage collection?

Q112. What is the difference between “Dispose” and “Finalize”? V Imp

Q113. What is the difference between “Finalize” and “Finally” methods?

Q114. Can we force Garbage Collector to run?

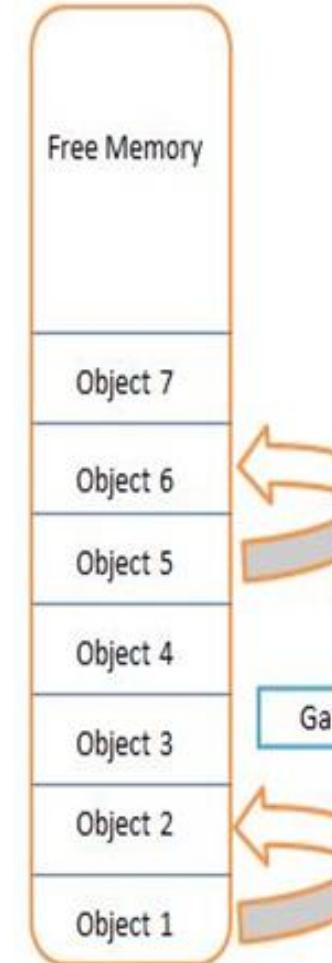


- ❖ The garbage collector (GC) **manages** the allocation and release of memory in .NET framework.
- ❖ Garbage collection is one of the responsibilities of CLR only.

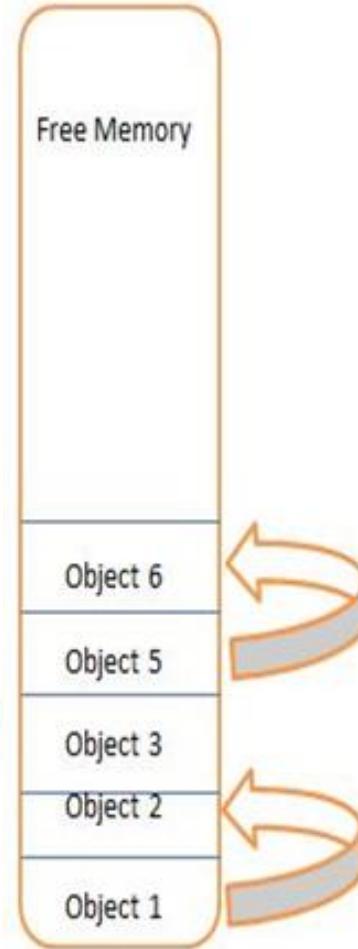
```
static void Main(string[] args)
{
    Employee employee = new Employee();
    employee.GetSalary();
}
```

- ❖ *Garbage collector will dispose this employee object automatically when it is no longer needed.*

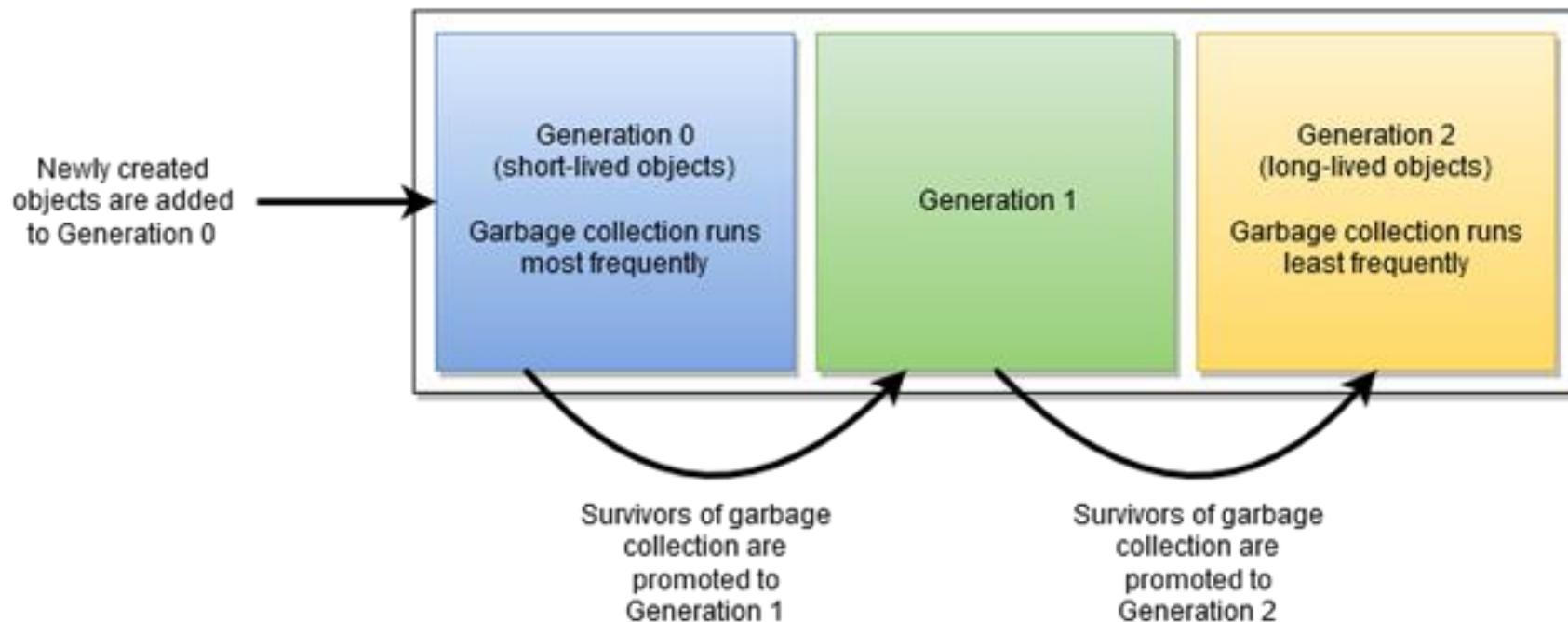
Before GC



After GC



- ❖ Garbage collection release the objects which are no longer needed.
- ❖ .NET framework has a mechanism to decide which objects are no longer needed and that mechanism we call generations.
- ❖ Generation is a mechanism to collect the short-lived objects more frequently than the longer-lived object.



- ❖ Both Dispose and Finalize methods are used to release the unmanaged objects which are no longer required.
- ❖ Finalize method is called automatically by the garbage collector.
- ❖ But the Dispose method is called explicitly by the code to release any unmanaged object.

```
using (SqlConnection connection = new SqlConnection(conString))  
{  
    // use the connection here  
    connection.Open();  
  
    // execute a query, etc.  
}  
// the connection is automatically closed here
```

```
public class Employee : IDisposable  
{  
    private bool disposed = false;  
  
    public void Dispose()  
    {  
        Dispose(true);  
        GC.SuppressFinalize(this);  
    }  
  
    protected virtual void Dispose(bool disposing)  
    {  
        if (!disposed)  
        {  
            if (disposing)  
            {  
                // Release managed resources  
            }  
  
            // Release unmanaged resources  
            disposed = true;  
        }  
    }  
}
```

- ❖ Finalize method is used for **garbage collection**. So before destroying an object this method is called as part of clean up activity by GC.
- ❖ Finally block is used in **exception handling** for executing the code irrespective of exception occurred or not.

```
try
{
    SqlConnection con = new SqlConnection(connectionString);
    //Some logic
    //Error occured
}
catch(ExceptionName ex) {
    //Error handled
}
finally
{
    //Connection closed
    con.Close();
}
```

- ❖ Yes, by calling **GC.COLLECT()** method we can force garbage collector to run, but this is not recommended, instead use Dispose method.

```
static void Main(string[] args)
{
    // Allocate some memory
    var obj1 = new object();
    var obj2 = new object();

    // Set obj1 and obj2 to null to make
    // them eligible for garbage collection
    obj1 = null;
    obj2 = null;

    // Trigger a garbage collection cycle
    System.GC.Collect();

    // The memory used by obj1 and obj2
    // should now be freed
}
```

Chapter 13 : .NET Framework - Threading

Q115. What is the difference between Process and Thread?

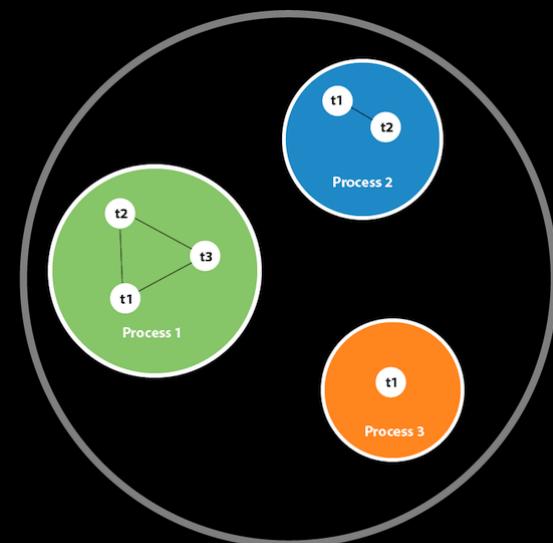
Q116. Explain Multithreading? V Imp

Q117. What is the difference between synchronous and asynchronous programming?

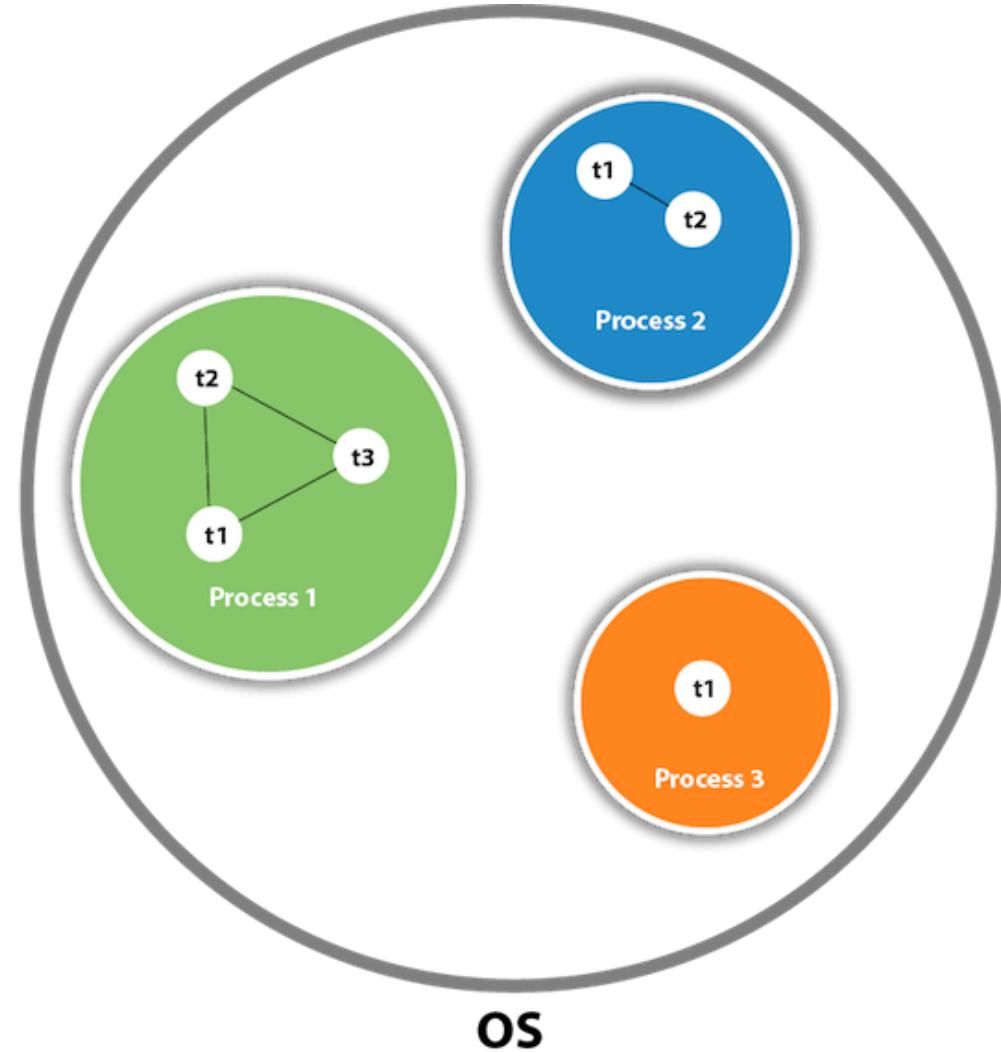
What is the role of Task? V Imp

Q118. What is the difference between Threads and Tasks? What are the advantages of Tasks over Threads?

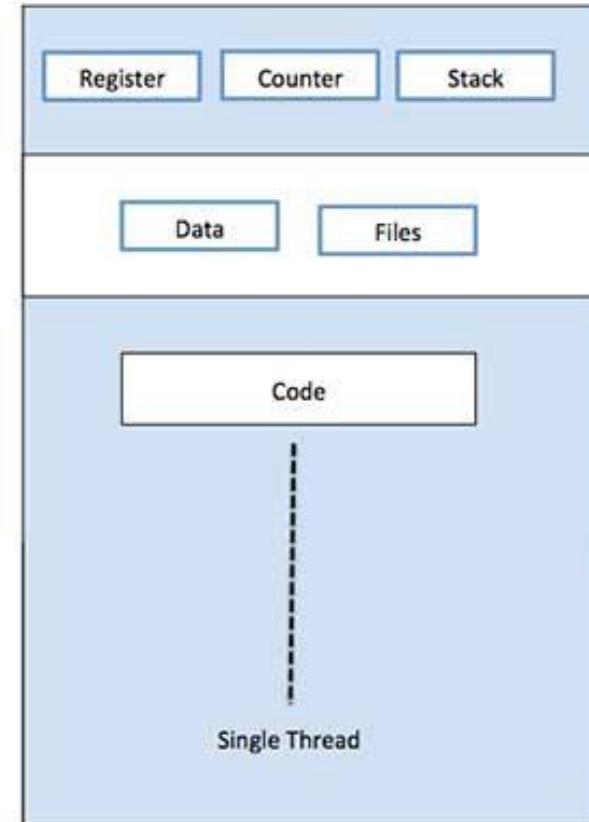
Q119. What is the role of Async and Await? V Imp



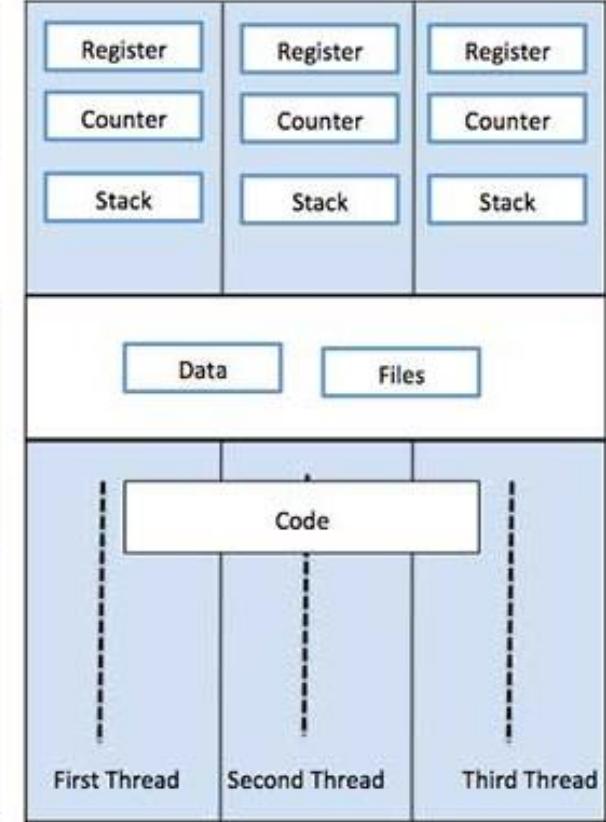
- ❖ A process is an instance of a program with its **own** memory space and system resources.
- ❖ A thread is the smallest unit of process, that **shares** memory and resources with other threads within the same process.



- ❖ Multithreading refers to the ability to execute multiple threads of code **concurrently** within a single process.
- ❖ Multithreading allows you to perform multiple tasks simultaneously, such as downloading data while displaying a progress bar.
- ❖ To create multithreaded application in C#, we need to use **SYSTEM.THREADING** namespace.



Single Process P with single thread



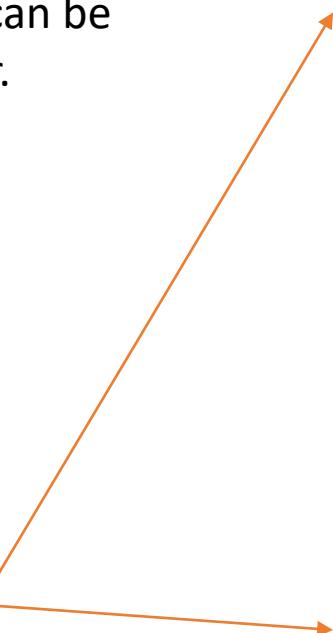
Single Process P with three threads

- ❖ In synchronous programming, task is executed in a sequential manner.
- ❖ In asynchronous programming, tasks can be executed independently of each other.

```
public static int Method1()
{
    Thread.Sleep(500);
    return 10;
}

public static int Method2()
{
    return 20;
}

public static int Method3()
{
    return 30;
}
```



```
public static void Main(string[] args)
{
    Console.WriteLine(Method1());
    Console.WriteLine(Method2());
    Console.WriteLine(Method3());
}
//Output 10 20 30
```

Synchronous
Programming


```
public static void Main(string[] args)
{
    Task task1 = Task.Run(() => {
        Console.WriteLine(Method1());
    });

    Task task2 = Task.Run(() => {
        Console.WriteLine(Method2());
    });

    Task task3 = Task.Run(() => {
        Console.WriteLine(Method3());
    });
    Console.Read();
}
//Output: 20 30 10
```

Asynchronous
Programming

- ❖ In .NET, threads and tasks are two different ways for doing **multithreading**.
- ❖ Thread is a general programming concept. On the other hand, Microsoft created Task in .NET to **simplify** the use of Threads.
- ❖ Tasks are like a **wrapper** over Threads.
- ❖ Tasks **internally** uses threads only.



```
public class ExampleThread
{
    public void DoWork()
    {
        Thread thread = new Thread(new
            ThreadStart(LongRunningMethod));
        thread.Start();
    }

    private void LongRunningMethod()
    {
        // simulate a long-running operation
        Thread.Sleep(5000);
        // continue with the rest of the method
    }
}
```

```
public class ExampleTask
{
    public async Task DoWorkAsync()
    {
        await Task.Delay(5000);
        // continue with the rest of the method
    }
}
```

- ❖ In .NET, threads and tasks are two different ways for doing **multithreading**.
 - ❖ Thread is a general programming concept. On the other hand, Microsoft created Task in .NET to **simplify** the used of Threads.
 - ❖ Tasks are like a **wrapper** over Threads.
 - ❖ Tasks **internally** uses threads only.
- ❖ Advantages of Tasks over Threads:
 1. Simplified Code.
 2. Exception handling.
 3. A Task can **return a result**, but there is no proper way to return a result from Thread.
 4. We can apply **chaining** and **parent/ child** on multiple tasks, but it can be very difficult in threads.

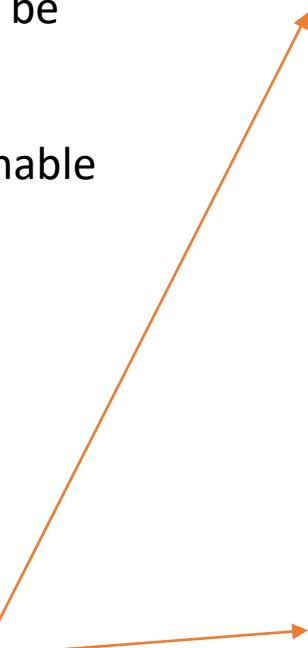


- ❖ In synchronous programming, task is executed in a sequential manner.
- ❖ In asynchronous programming, tasks can be executed independently of each other.
- ❖ Async and Await are keywords used to enable asynchronous programming.

```
public static int Method1()
{
    Thread.Sleep(500);
    return 10;
}

public static int Method2()
{
    return 20;
}

public static int Method3()
{
    return 30;
}
```



```
public static void Main(string[] args)
{
    Console.WriteLine(Method1());
    Console.WriteLine(Method2());
    Console.WriteLine(Method3());
}
//Output 10 20 30
```

Synchronous
Programming



```
public static void Main(string[] args)
{
    Task task1 = Task.Run(() => {
        Console.WriteLine(Method1());
    });

    Task task2 = Task.Run(() => {
        Console.WriteLine(Method2());
    });

    Task task3 = Task.Run(() => {
        Console.WriteLine(Method3());
    });
    Console.Read();
}
//Output: 20 30 10
```

Asynchronous
Programming

- ❖ The `async` keyword is used to mark a method as an asynchronous method.
- ❖ The `await` keyword is used to pause the execution of an asynchronous method until a specified operation completes.

```
static int Method1()
{
    Thread.Sleep(500);
    return 10;
}

static int Method2(int i)
{
    return 20 * i;
}

static int Method3()
{
    return 30;
}

static void Main(string[] args)
{
    Method1_2();

    int k = Method3();
    Console.WriteLine(k);
    Console.Read();
}
```

```
//Asynchronous with Task async/await

static async void Method1_2()
{
    Console.WriteLine("Test");

    var i = await Task.Run(() =>
    {
        return Method1();
    });

    Console.WriteLine(i);

    int j = Method2(i);
    Console.WriteLine(j);
}

//Output: Test 30 10 200
```

Chapter 14 : SQL - Basics

Q120. What is the difference between DBMS and RDBMS?

Q121. What is a Constraint in SQL? What are the types of constraints? V Imp

Q122. What is the difference between Primary key and Unique key?

Q123. What are Triggers and types of triggers?

Q124. What is a View? V Imp

Q125. What is the difference between Having clause and Where clause? V Imp

Q126. What is Sub query or Nested query or Inner query in SQL?

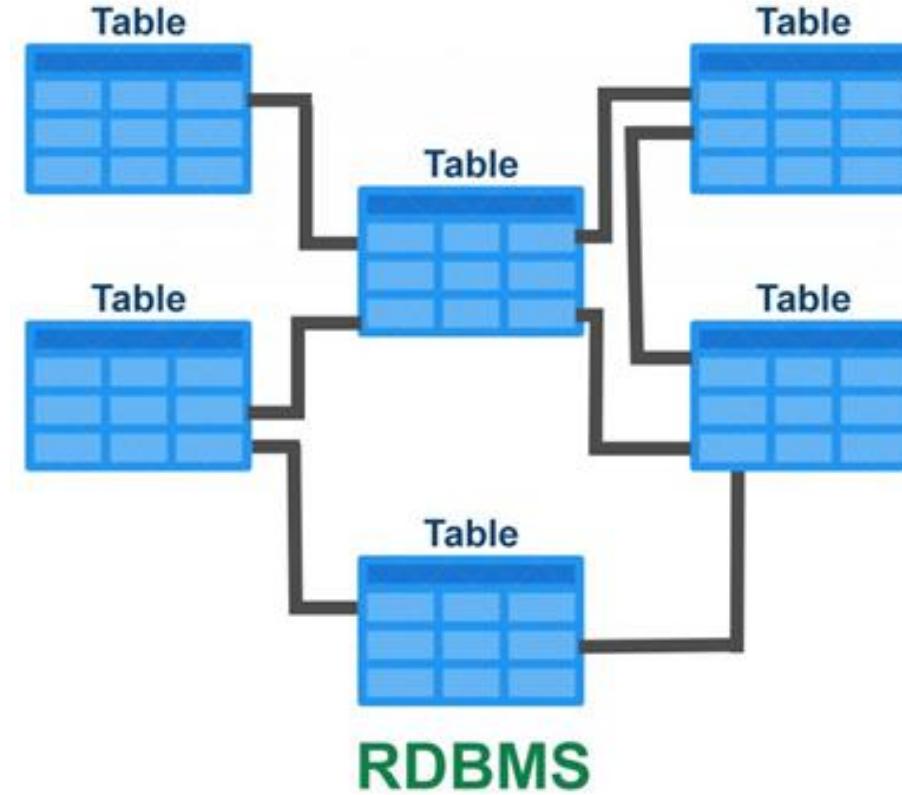
Q127. What is Auto Increment/ Identity column in SQL Server?



What is the difference between DBMS and RDBMS?



DBMS



DBMS	RDBMS
1. DBMS stores data as file.	RDBMS stores data in TABULAR form.
2. No relationship between data.	Data is stored in the form of tables which are RELATED to each other. Eg: Foreign key relationship.
3. Normalization is not present.	NORMALIZATION is present.
4. It deals with small quantity of data.	It deals with LARGE amount of data.
5. Examples: XML	Examples: MySQL, PostgreSQL, SQL Server, Oracle, Microsoft Access etc.

- ❖ SQL constraints are used to specify **rules** for the data in a table.

1. A **PRIMARY KEY** is a field which can uniquely identify each row in a table.

```
CREATE TABLE Students (
    ID int NOT NULL PRIMARY KEY,
```

2. **NOT NULL** constraint tells that we cannot store a null value in a column.

3. A **FOREIGN KEY** is a field which can uniquely identify each row in another table.

```
    Name varchar(255) NOT NULL,
```

```
    CourseID int FOREIGN KEY REFERENCES Courses(CourseID),
```

4. **CHECK** constraint helps to validate the values of a column to meet a particular condition.

```
    Age int NOT NULL CHECK (AGE >= 18),
```

5. **DEFAULT** constraint specifies a default value for the column when no value is specified by the user.

```
    AdmissionDate date DEFAULT GETDATE(),
```

```
    CONSTRAINT UC_Student UNIQUE (ID,Name)
```

6. **UNIQUE** constraint tells that all the values in the column must be unique.

Primary Key

1. Primary key can't accept null values

2. Automatically create clustered index.

3. Only one primary key can be present in a table.

Unique Key

Unique key can accept only one null value

Create non clustered index.

More than one unique key can be present in a table.

- ❖ Triggers are stored programs, which are **AUTOMATICALLY** executed or fired when some events (insert, delete and update) occur.

❖ Locations → LocationHist

❖ Example of After(DML) Trigger

```

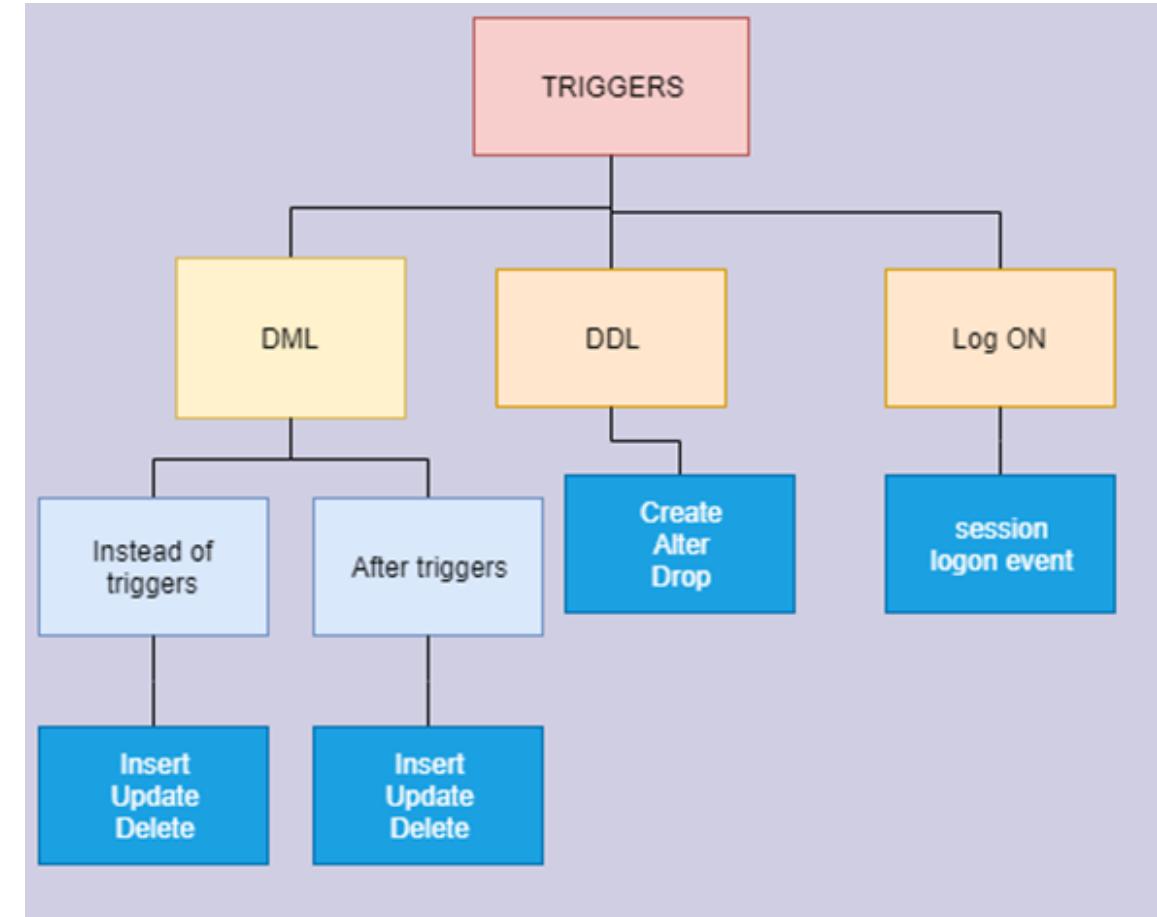
CREATE TRIGGER TR_UPD_Locations ON Locations
FOR UPDATE
NOT FOR REPLICATION
AS
BEGIN
  INSERT INTO LocationHist
  SELECT LocationID
    ,getdate()
   FROM inserted
END
  
```

Table Name → Locations

Trigger Name → TR_UPD_Locations

DML Event → FOR UPDATE

T-SQL block that runs against specified DML Event → Insert, Update, Delete



- ❖ In after trigger, update on the table executed first and then after trigger will be fired.

- ❖ Triggers are stored programs, which are **AUTOMATICALLY** executed or fired when some events (insert, delete and update) occur.

- ❖ Example of Instead of(DML) Trigger

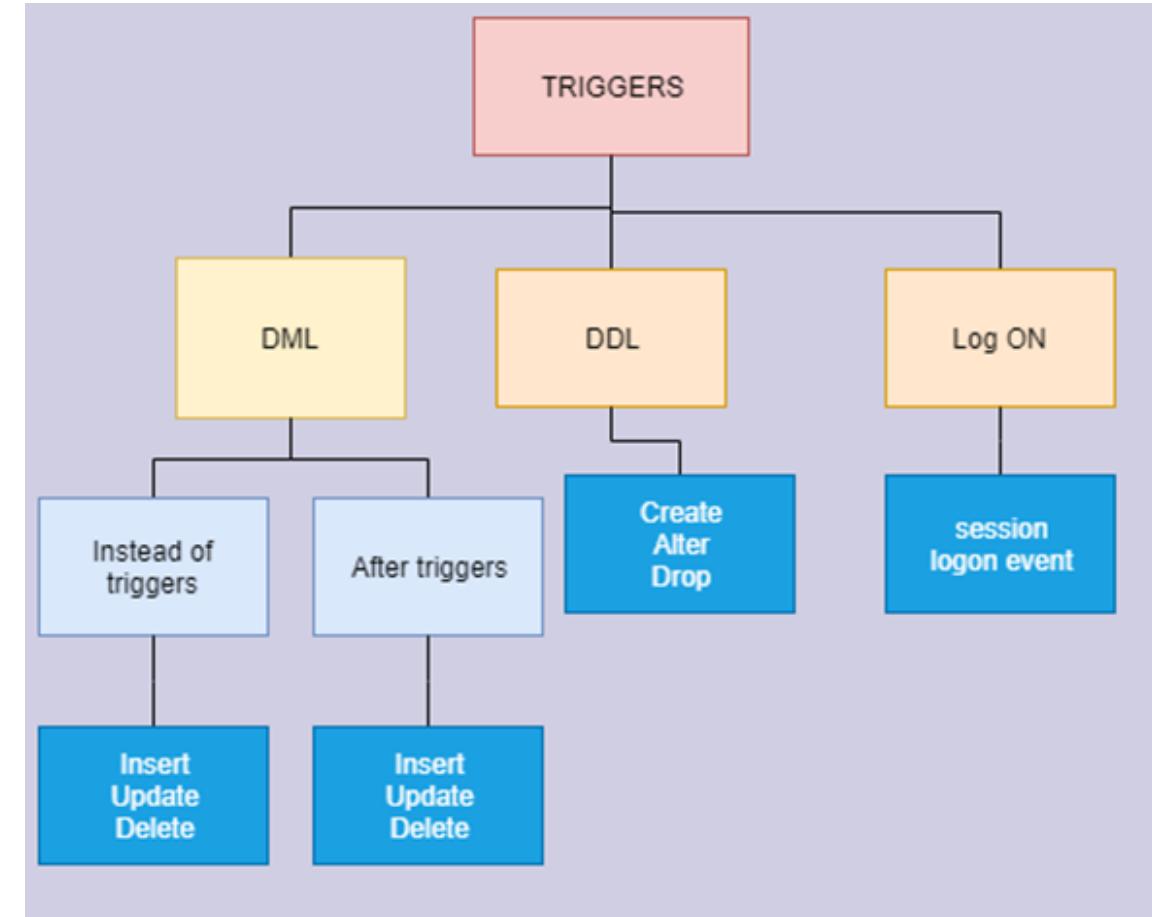
```
CREATE TRIGGER [dbo].[TRG_VM_EMPDETAILS]
ON [dbo].[vw_empdetails]
INSTEAD OF INSERT
AS
BEGIN
-- LOGIC HERE
END
```

Trigger Name: INSTEAD OF

View Name: vw_empdetails

INSTEAD OF Trigger

This trigger is only for INSERT



- ❖ An INSTEAD OF trigger is a trigger that allows you to **skip** an INSERT , DELETE , or UPDATE statement to a table or a view and execute other statements defined in the trigger.

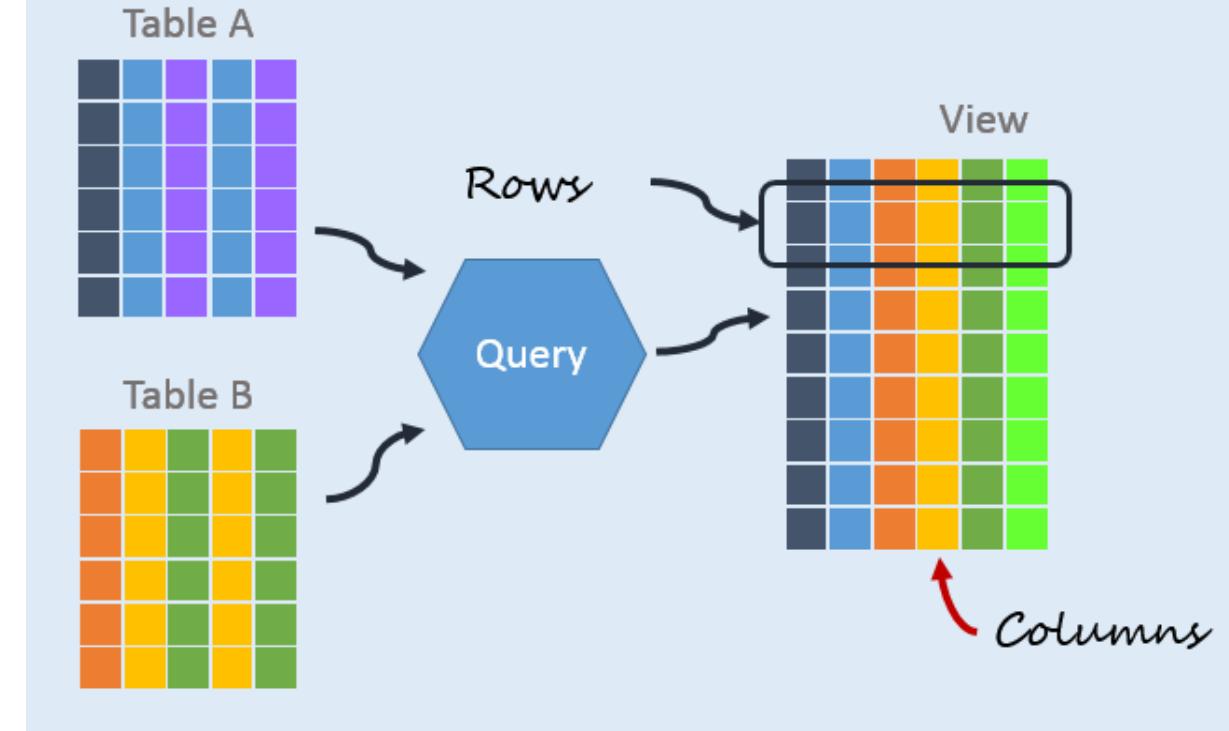
- ❖ A view is a VIRTUAL table which consists of a subset of data contained in single table or more than one table.

```
CREATE VIEW [India-Customers] AS
```

```
SELECT CustomerName, ContactName  
FROM Customers  
WHERE Country = 'India';
```

- ❖ Advantages of Views

1. Indexed Views to improve the performance.
2. Extra security – DBA can hide the actual table names and expose views for Read operations only.



- ❖ Remember, in case of view, only query is stored but the actual data is never stored like a table.

1. WHERE Clause is used before GROUP BY Clause.

HAVING Clause is used after GROUP BY Clause.

```
SELECT COUNT(CustomerID), Country  
FROM Customers  
WHERE Country = "India"  
GROUP BY Country  
HAVING COUNT(CustomerID) > 5;
```

2. WHERE Clause cannot contain AGGREGATE function.

HAVING Clause can contain aggregate function.

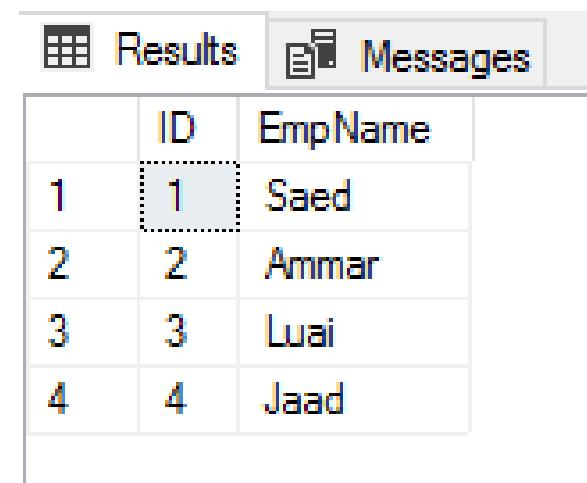
```
SELECT EmpName FROM Employee  
GROUP BY EmpName  
HAVING SUM(EmpSalary) < 30000
```

- ❖ A Subquery/ Inner query/ Nested query is a query within another SQL outer query and **embedded** within the **WHERE** clause.



- ❖ Auto-increment allows a unique number to be **generated automatically** when a new record is inserted into a table.
- ❖ Mostly auto increment is set on the primary key only.

```
CREATE TABLE Employee (
    ID int IDENTITY(1,1) PRIMARY KEY,
    EmpName varchar(255) NOT NULL
);
```



	ID	EmpName
1	1	Saed
2	2	Ammar
3	3	Luai
4	4	Jaad

Chapter 15: SQL - Joins & Indexes

Q128. What are **Joins** in SQL?

Q129. What are the **types of Joins** in SQL Server? V Imp

Q130. What is **Self-Join**?

Q131. What are **Indexes** in SQL Server?

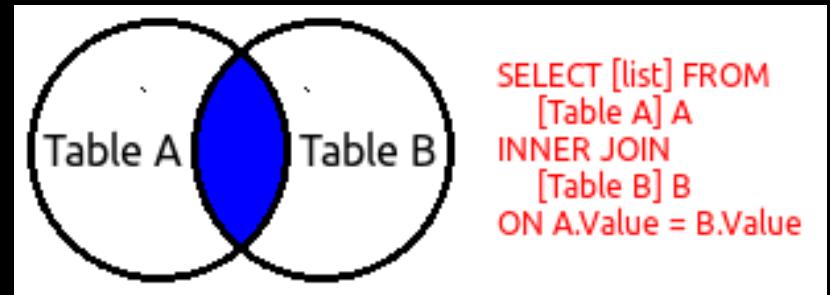
Q132. What is **Clustered** index?

Q133. What is **Non-Clustered** index?

Q134. What is the difference between Clustered and Non-Clustered index? V Imp

Q135. How to create Clustered and Non-Clustered index in a table?

Q136. In which column you will apply the indexing to optimize this query?



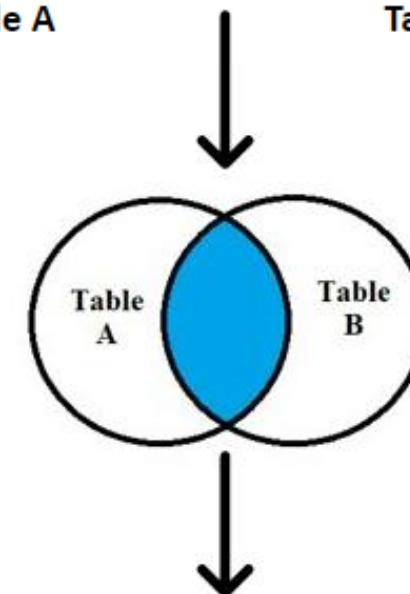
- ❖ A join clause is used to **combine** rows from two or more tables, based on a related column between them.

Student ID	Name
1001	A
1002	B
1003	C
1004	D

Student ID	Department
1004	Mathematics
1005	Mathematics
1006	History
1007	Physics
1008	Computer Science

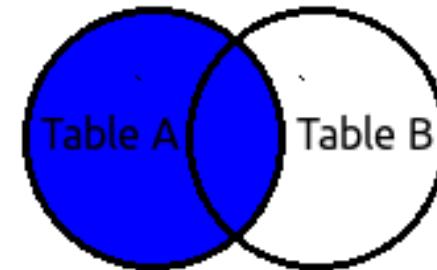
Table A

Table B



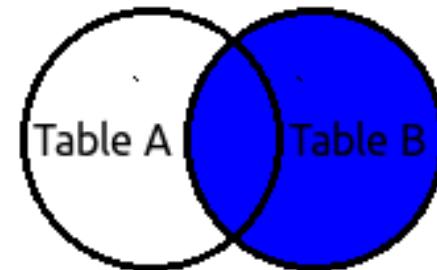
Student ID	Name	Department
1004	D	Mathematics

- ❖ **Left outer join** - A left join returns all the rows from the left table, along with any matching rows from the right table.



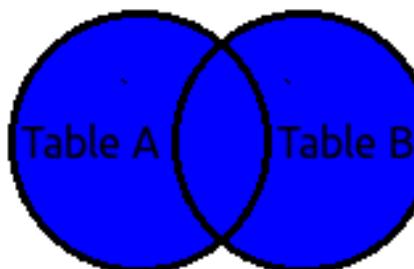
```
SELECT [list] FROM  
[Table A] A  
LEFT JOIN  
[Table B] B  
ON A.Value = B.Value
```

- ❖ **Right outer join** - A right join returns all the rows from the right table, along with any matching rows from the left table.



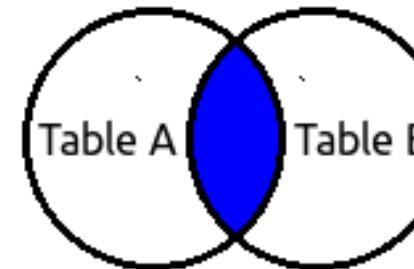
```
SELECT [list] FROM  
[Table A] A  
RIGHT JOIN  
[Table B] B  
ON A.Value = B.Value
```

- ❖ **Full outer join** - A full outer join returns all the rows from both the left and right tables in the join.



```
SELECT [list] FROM  
[Table A] A  
FULL OUTER JOIN  
[Table B] B  
ON A.Value = B.Value
```

- ❖ **Inner join** - An inner join returns only the common rows from both tables that meet the join condition.



```
SELECT [list] FROM  
[Table A] A  
INNER JOIN  
[Table B] B  
ON A.Value = B.Value
```

- ❖ A self join is a join of a table to **itself**.
- ❖ When to use Self Join??

employees	
*	employee_id
	first_name
	email
	phone_number
	hire_date
	job_id
	salary
	manager_id
	department_id

This manager id is
employee id of the
manager of an
employee

SELECT

e.first_name AS employee,
m.first_name AS manager

FROM

employees e LEFT JOIN
employees m

ON m.employee_id = e.manager_id

ORDER BY manager;

	employee	manager
▶	Steven King	NULL
	Bruce Ernst	Alexander Hunold
	David Austin	Alexander Hunold
	Valli Pataballa	Alexander Hunold
	Diana Lorentz	Alexander Hunold
	Alexander Khoo	Den Raphaely

- ❖ Now your task is to get the employees name with their manager names??

- ❖ SQL Indexes are used in relational databases to retrieve data **VERY FAST**.
- ❖ They are similar to indexes at the start of the BOOKS, which purpose is to find a topic quickly.

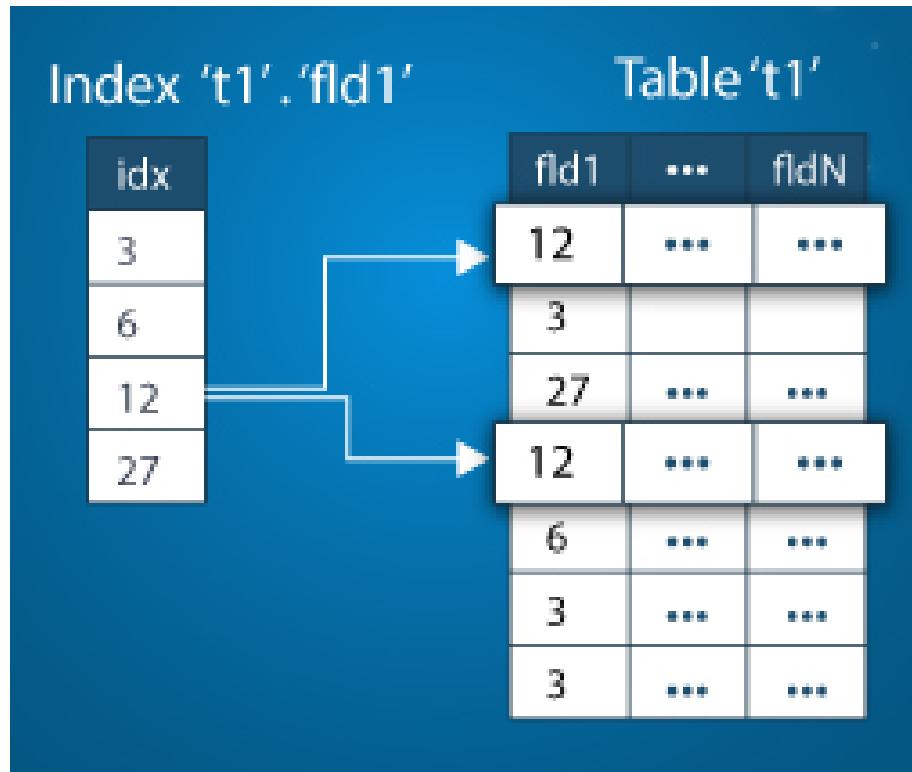
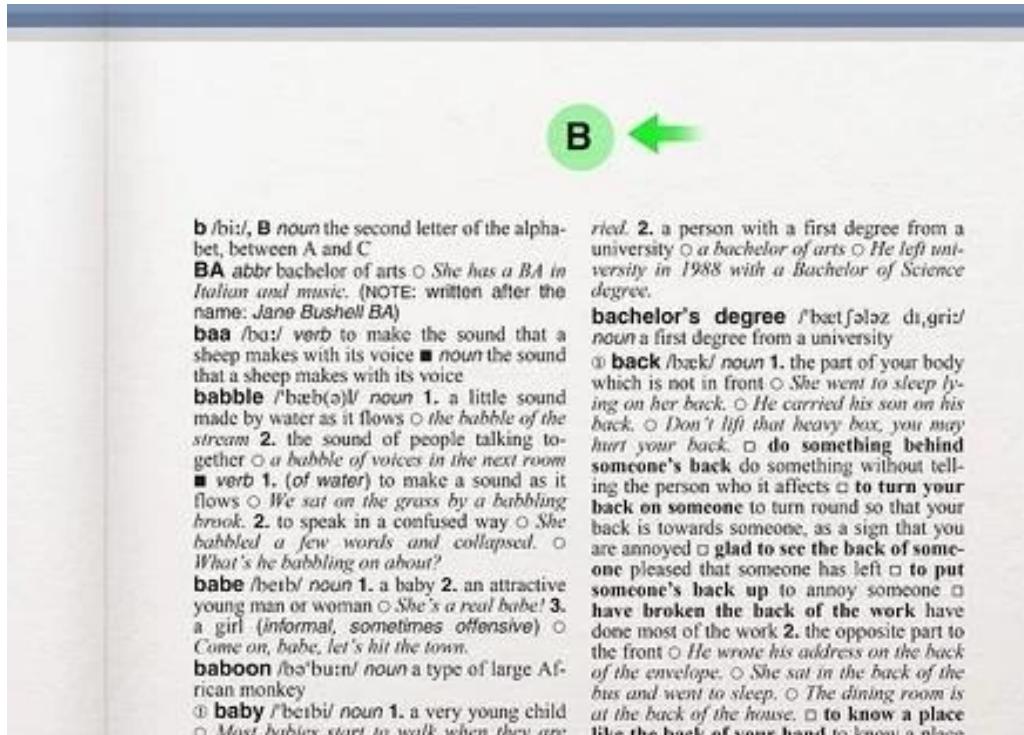


Table of Contents

Acknowledgments.....	ix
Introduction	xi
Part I Envision the Possibilities	
1 Welcome to Office 2010	3
Features that Fit Your Work Style	3
Changes in Office 2010	4
Let Your Ideas Soar	5
Collaborate Easily and Naturally	5
Work Anywhere—and Everywhere	6
Exploring the Ribbon.....	7

- ❖ A clustered index is a type of index that determines the **physical order** of data in a table.
- ❖ Table data can be sorted in only way, therefore, there can be **only one** clustered index per table.
- ❖ In SQL Server, if you set a primary key on a column, then it will **automatically create** a clustered index on that particular column.

Dictionary



- ❖ A non-clustered index is stored at one place and table data is stored in another place.
- ❖ A table can have multiple non-clustered index in a table.

Index

A

accordion, layouts
about 128
movie form, adding 131
nesting, in tab 128, 129
toolbar, adding 129-131
adapters, Ext
about 18
using 18, 20
Adobe AIR 285
Adobe Integrated Run time. *See* Adobe AIR
AJAX 12
Asynchronous JavaScript and XML.
See AJAX

B

built-in features, Ext
client-side sorting 86
column, reordering 86, 87
columns, hidden 86

lookup data stores, creating 83
two columns, combining 84
classes 254
ComboBox, form
about 47
database-driven 47-50
component config 59
config object
about 28, 29
new way 28, 29
old way 28
tips 26, 29
content, loading on menu item click 68, 69
custom class, creating 256-259
custom component, creating 264-266
custom events, creating 262-264

D

data, filtering
about 238
remote, filtering 238-244

Book Index

Table of Contents

Acknowledgments	ix
Introduction	xi

Part I Envision the Possibilities

1 Welcome to Office 2010	3
Features that Fit Your Work Style	3
Changes in Office 2010	4
Let Your Ideas Soar	5
Collaborate Easily and Naturally	5
Work Anywhere—and Everywhere	6
Exploring the Ribbon	7

1. A clustered index is a type of index that determines the **physical order** of data in a table.

A non-clustered index is stored at one place and table data is stored in another place. For example, Book Index.

2. A table can have only one clustered index.

A table can have multiple non-clustered index.

3. Clustered index is faster.

Non-clustered index is slower.

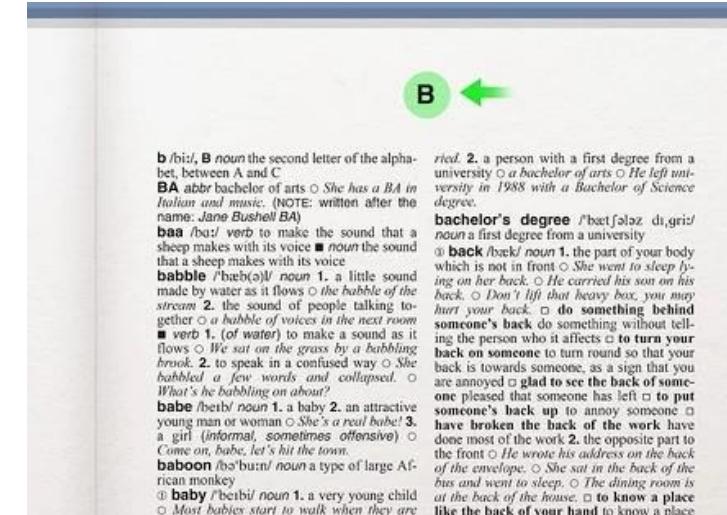


Table of Contents	
Acknowledgmentsix
Introductionxi
Part I Envision the Possibilities	
1 Welcome to Office 2010	3
Features that Fit Your Work Style	3
Changes in Office 2010	4
Let Your Ideas Soar	5
Collaborate Easily and Naturally	5
Work Anywhere—and Everywhere	6
Exploring the Ribbon	7

❖ Clustered Index

```
CREATE CLUSTERED INDEX <index_name>
ON <table_name>(<column_name> ASC/DESC)
```

❖ Non-clustered Index

```
CREATE NONCLUSTERED INDEX <index_name>
ON <table_name>(<column_name> ASC/DESC)
```

- ❖ When you create a PRIMARY KEY constraint, a clustered index on the column is automatically created.

In which column you will apply the indexing to optimize this query.

“select id, class from student where name=“happy””?



select id, class from student where name=“happy”

- ❖ The column **after WHERE** condition, which is “NAME” here.

Chapter 16 : SQL - Stored Procedure, Functions & Others

Q137. What is the difference between **Stored Procedure** and **Functions**? V Imp

Q138. How to optimize a Stored Procedure or SQL Query?

Q139. What is a **Cursor**? Why to avoid them?

Q140. What is the difference between **scope_identity** and **@@identity**?

Q141. What is **CTE** in SQL Server?

Q142. What is the difference between **Delete**, **Truncate** and **Drop** commands? V Imp

Q143. How to get the **Nth highest salary** of an employee?

Q144. What are **ACID properties**?

Q145. What are **Magic Tables** in SQL Server?

```
CREATE PROCEDURE proc_name  
(@Ename varchar(50),  
@EId int output)  
AS  
BEGIN  
  
    INSERT INTO Employee (EmpName)  
    VALUES (@Ename)  
    SELECT @EId= SCOPE_IDENTITY()  
  
END
```

Stored Procedure	Function
1. SP may or may not return a value	Function must return a value
2. Can have input/ output parameters	Only has input parameters
3. We can call function inside SP	Cannot call SP inside a function
4. We cannot use SP in SQL statements like SELECT, INSERT, UPDATE, DELETE, MERGE, etc.	We can use them with function. <i>SELECT *, dbo.fnCountry(city.long) FROM city;</i>
5. We can use try-catch exception handling in SP	We can not use try-catch in functions
6. We can use transactions inside SP.	We can not use transactions inside functions.

```

CREATE PROCEDURE proc_name
(@Ename varchar(50),
@EId int output)
AS
BEGIN

INSERT INTO Employee (EmpName)
VALUES (@Ename)
SELECT @EId= SCOPE_IDENTITY()

END

CREATE FUNCTION function_name
(parameters) --only input parameter
RETURNS data_type AS
BEGIN

-- SQL statements
RETURN value

END;
  
```

1. Use SET NOCOUNT ON

2. Specify column names instead of using * .

~~SELECT * FROM table1~~

~~SELECT col1, col2 FROM table1~~

3. Use schema name before objects or table names.

~~SELECT EmpID, Name FROM Employee~~

~~SELECT EmpID, Name FROM dbo.Employee~~

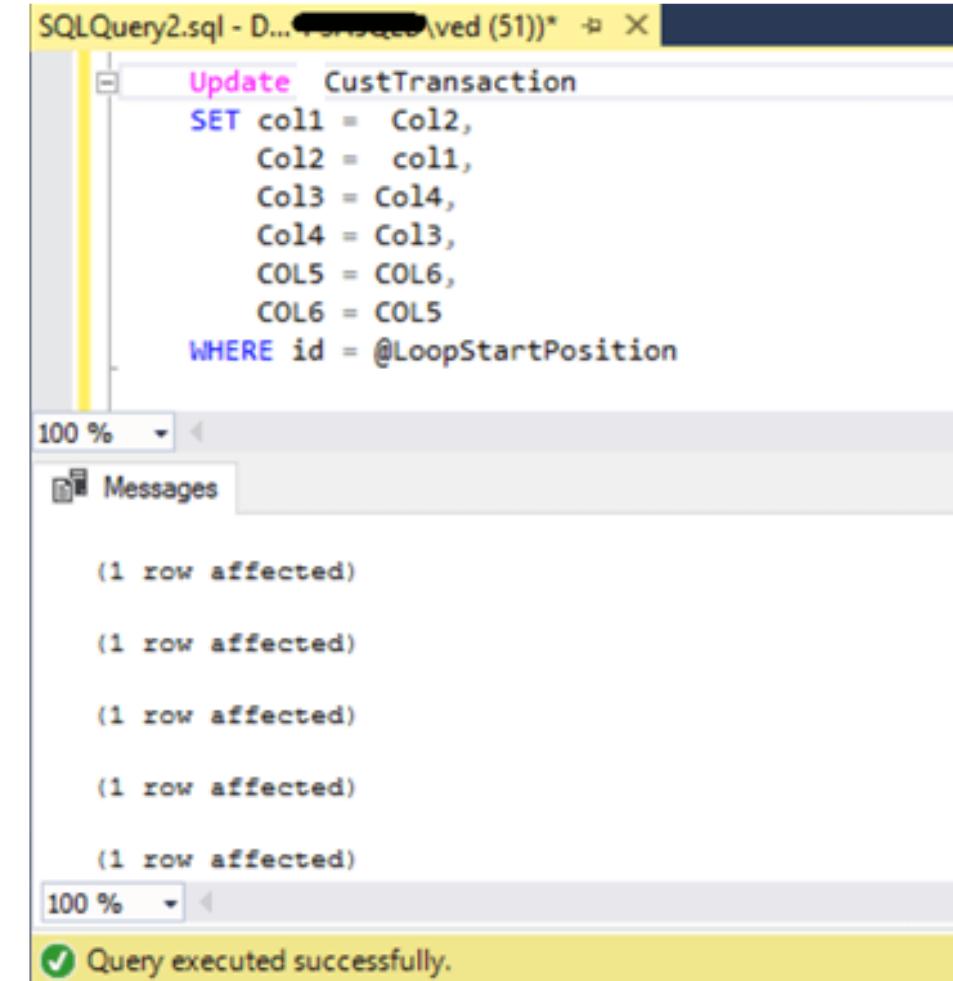
4. Use EXISTS () instead of COUNT ().

~~SELECT Count(1) FROM dbo.Employee~~

~~IF(EXISTS (SELECT 1 FROM db.Employees))~~

5. Use TRANSACTION when required only.

6. Do not use DYNAMIC QUERIES. They are vulnerable to SQL Injections.



The screenshot shows a SQL Server Management Studio window titled "SQLQuery2.sql - D:\[REDACTED]\ved (51)*". The query is:

```
Update CustTransaction
SET col1 = Col2,
    Col2 = col1,
    Col3 = Col4,
    Col4 = Col3,
    COL5 = COL6,
    COL6 = COL5
WHERE id = @LoopStartPosition
```

The results pane shows five rows affected by the update statement. The status bar at the bottom right indicates "Query executed successfully."

- ❖ A database Cursor is a control which enables traversal/ iteration over the rows or records in the table.

- ❖ 5 step process:

1. Declare
2. Open
3. Fetch using while loop
4. Close
5. Deallocate

- ❖ LIMITATION

A cursor is a MEMORY resident set of pointers. Meaning it occupies lots of memory from your system which is not good for performance.

```

DECLARE
    @product_name VARCHAR(MAX),
    @list_price    DECIMAL;

DECLARE cursor_product CURSOR
FOR SELECT
    product_name,
    list_price
FROM
OPEN cursor_product;
FETCH NEXT FROM cursor_product INTO
    @product_name,
    @list_price;

WHILE @@FETCH_STATUS = 0
BEGIN
    PRINT @product_name + CAST(@list_price AS varchar);
    FETCH NEXT FROM cursor_product INTO
        @product_name,
        @list_price;
CLOSE cursor_product;
DEALLOCATE cursor_product;
  
```

- ❖ Both are used to get the last value generated in the **identity** column of the table.
- ❖ @@IDENTITY function returns the last identity value generated within the **current session**, regardless of the scope.
- ❖ This will return a value generated by an INSERT statement in a **trigger, stored procedure or batch** of T-SQL statements.

```
SELECT @@IDENTITY
```

- ❖ The **scope_identity()** function returns the last identity created in the same session and the same **scope**.

```
SELECT SCOPE_IDENTITY()
```

- ❖ Mostly we use **scope_identity()** function inside stored procedures.

- ❖ A Common Table Expression, is a TEMPORARY named result set, that you can reference within a SELECT, INSERT, UPDATE, or DELETE statement.

```
WITH  
with engineers as (  
    select *  
    from employees  
    where dept='Engineering'  
)  
  
select *  
from engineers ← CTE Usage  
where ...
```

Diagram annotations:

- WITH**: Points to the keyword **WITH** in the first line of the CTE definition.
- CTE name**: Points to the alias **engineers** in the **as** clause.
- CTE Body**: Points to the **select**, **from**, and **where** clauses enclosed in parentheses, representing the body of the CTE.
- CTE Usage**: Points to the **from** clause in the outer query, indicating where the CTE is being used.

DELETE	TRUNCATE	DROP
1. It is a DML. 2. It is used to delete one or all rows from the table based on where condition, but it will not delete schema.	1. It is a DDL. 2. It is used to delete all rows from the table, but it will not delete schema.	1. It is a DDL. 2. It is used to delete all rows from the table with structure/schema.
3. It can be rollback.	3. It can not be rollback.	3. It can not be rollback.

```
DELETE FROM Employees  
WHERE Emp_Id = 7;
```

```
TRUNCATE TABLE Employees;
```

```
DROP TABLE Employees;
```

1. The logic is first select TOP 3 salaries in descending order.

```
SELECT DISTINCT TOP 3 SALARY
FROM tbl_Employees
ORDER BY SALARY DESC
```

Salary
5000
10000
6000
4000
2000
7000

10000
7000
6000

2. Put the result in “Result” and then do order by asc

```
SELECT SALARY
FROM (
```

```
SELECT DISTINCT TOP 3 SALARY
FROM tbl_Employees
ORDER BY SALARY DESC
```

```
) RESULT
ORDER BY SALARY
```

Result
6000
7000
10000

3. Select top 1 salary from result set

```
SELECT TOP 1 SALARY
FROM (
```

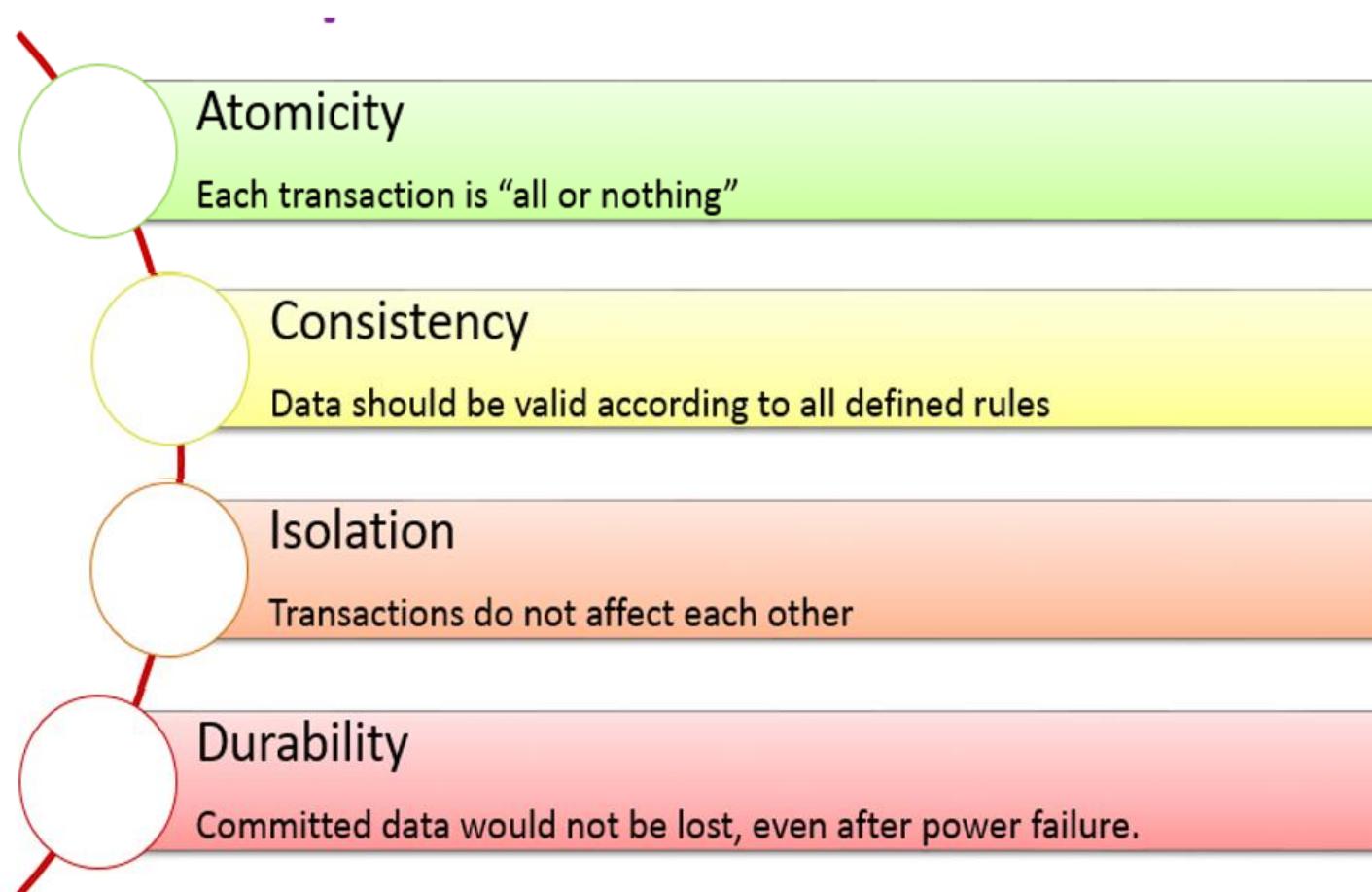
```
SELECT DISTINCT TOP 3 SALARY
FROM tbl_Employees
ORDER BY SALARY DESC
```

```
) RESULT
ORDER BY SALARY
```

Result
6000

- ❖ ACID properties are used when you are handling **transactions** in SQL.

For example, multiple inserts in a table are happening at the same point of time.



- ❖ **Magic tables** are the temporary logical tables that are created by the SQL server, whenever there are **insertion or deletion or update(D.M.L)** operations.

- ❖ Types of magic tables
 - 1. **INSERTED** – When any insert query executed, then the recently inserted row gets added to the INSERTED magic table.

 - 2. **DELETED** – When any delete query executed, then the recently deleted row gets added to the DELETED magic table.

In update case, the updated row gets stored in INSERTED magic table and the old row or previous row gets stored in the DELETED magic table.

- ❖ The use of magic tables are TRIGGERS.

Chapter 17 : ASP.NET – MVC - Part 1

Q146. What is **MVC**? Explain MVC Life cycle.

Q147. What are the **advantages of MVC over Web Forms?** V Imp

Q148. What are the different **return types** of a controller Action method? V Imp

Q149. What are **Filters** and their types in MVC? V Imp

Q150. What is **Authentication** and **Authorization** in ASP.NET MVC? V Imp

Q151. What are the **types of Authentication** in ASP.NET MVC?

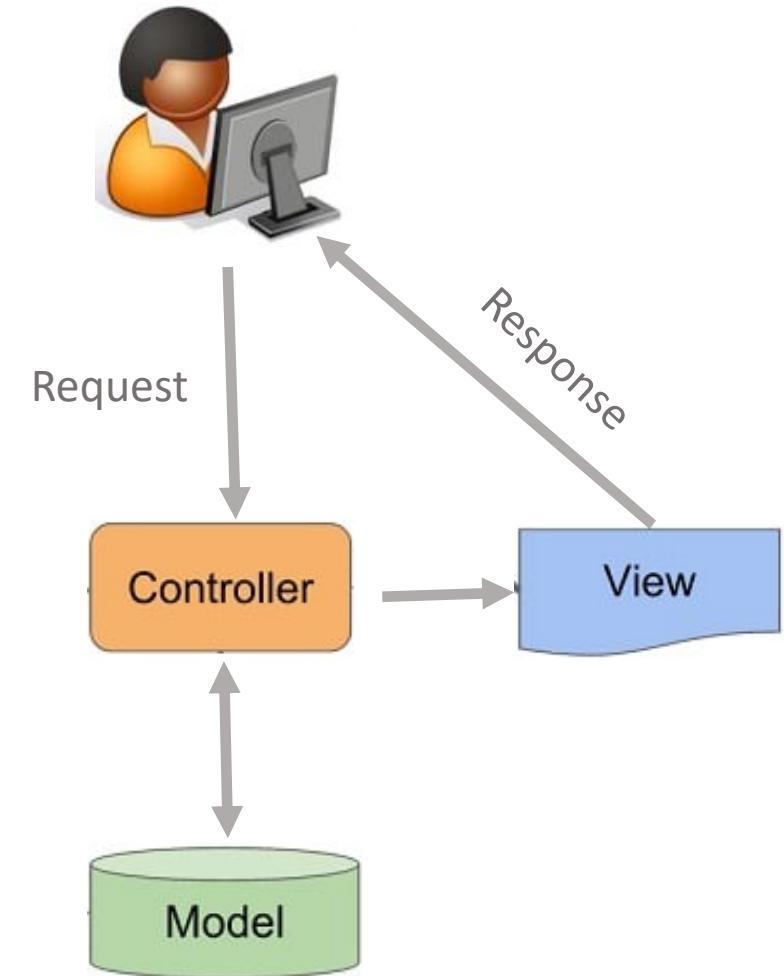
Q152. What is **Output Caching** in MVC? How to implement it?

Q153. What is **Routing** in MVC?

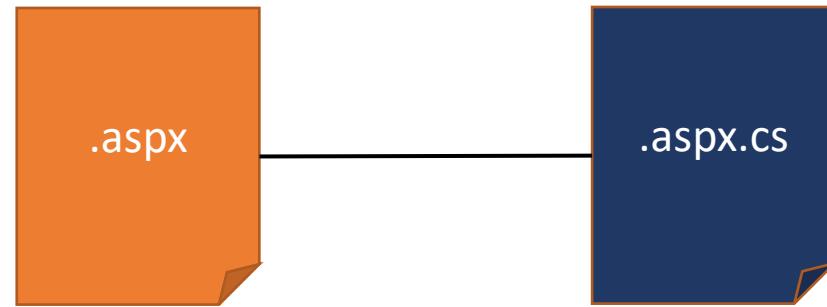
Q154. Explain **Attribute Based Routing** in MVC? V Imp



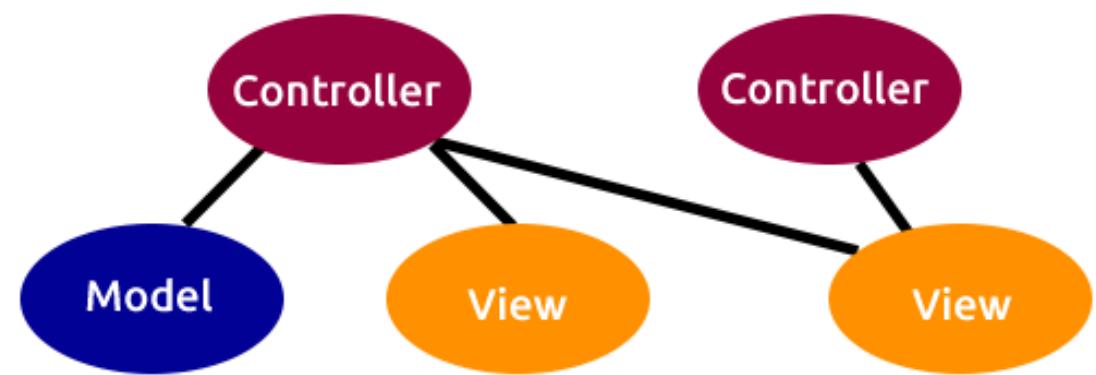
- ❖ MVC is a framework for building web applications using MVC (Model View Controller) architecture.
 1. The **Model** represents the data.
 2. The **View** displays the data.
 3. The **Controllers** act as an interface between Model and View components to process all the business logic.



- In web forms one aspx file will have one aspx.cs file, which means UI(aspx) is **tightly coupled** with logic in code-behind (.aspx.cs).

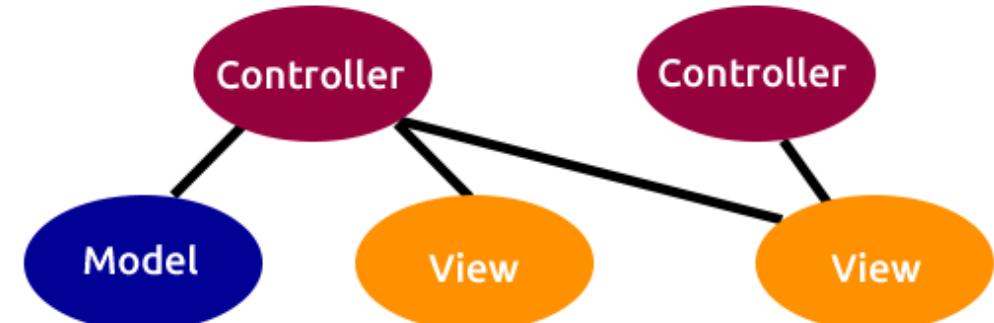


- In MVC, one controller can interact with multiple views and one view can interact with multiple controllers and therefore they are **loosely coupled**.



Advantage of MVC over Web-Forms:

- 1. Separation of concerns** - The MVC framework provides a clean separation of the UI, business logic and data with the help of MVC architecture.
- 2. Multiple view support** - Due to the separation of the model from the view, the user interface can display multiple views of the same data at the same time.
- 3. Change accommodation** - UI change more frequently than business rules. Now because the model does not depend on the views, modifying view will not affect the model.
- 4. Testability** - ASP.NET MVC provides better support for test driven development.
- 5. Light-weight** – In MVC framework, the request and response are bit lighter than webforms.
- 6. Full features of Asp.Net** – Almost all the features of web-forms are present in ASP.NET MVC.

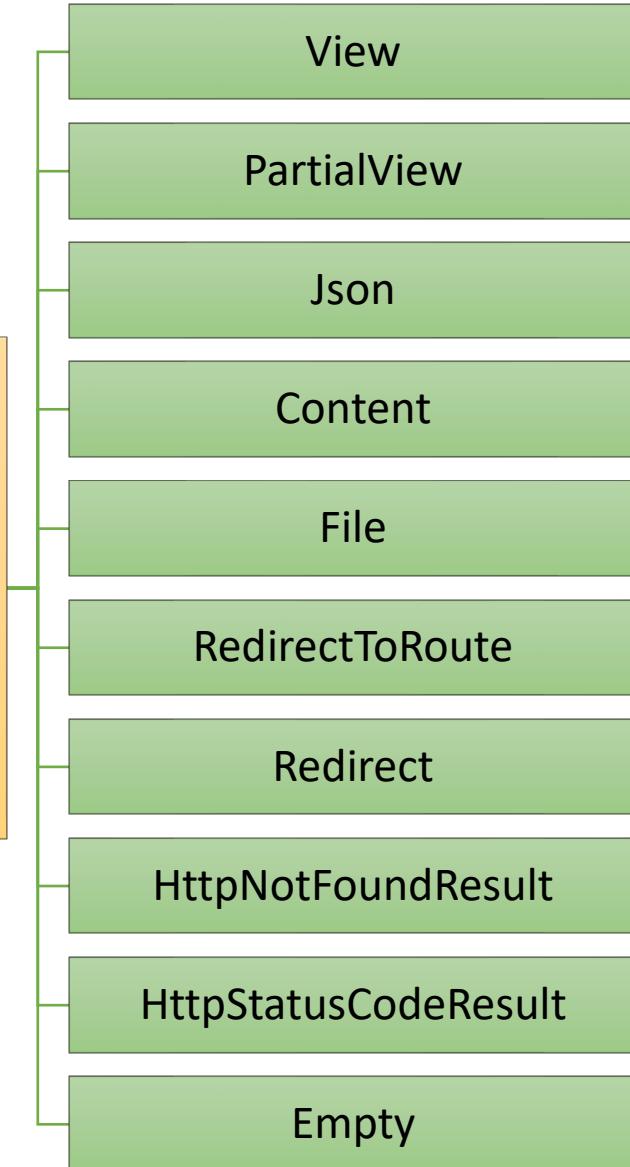


```
public ViewResult Index()
{
    // Do some processing to get data
    var data = SomeService.GetData();

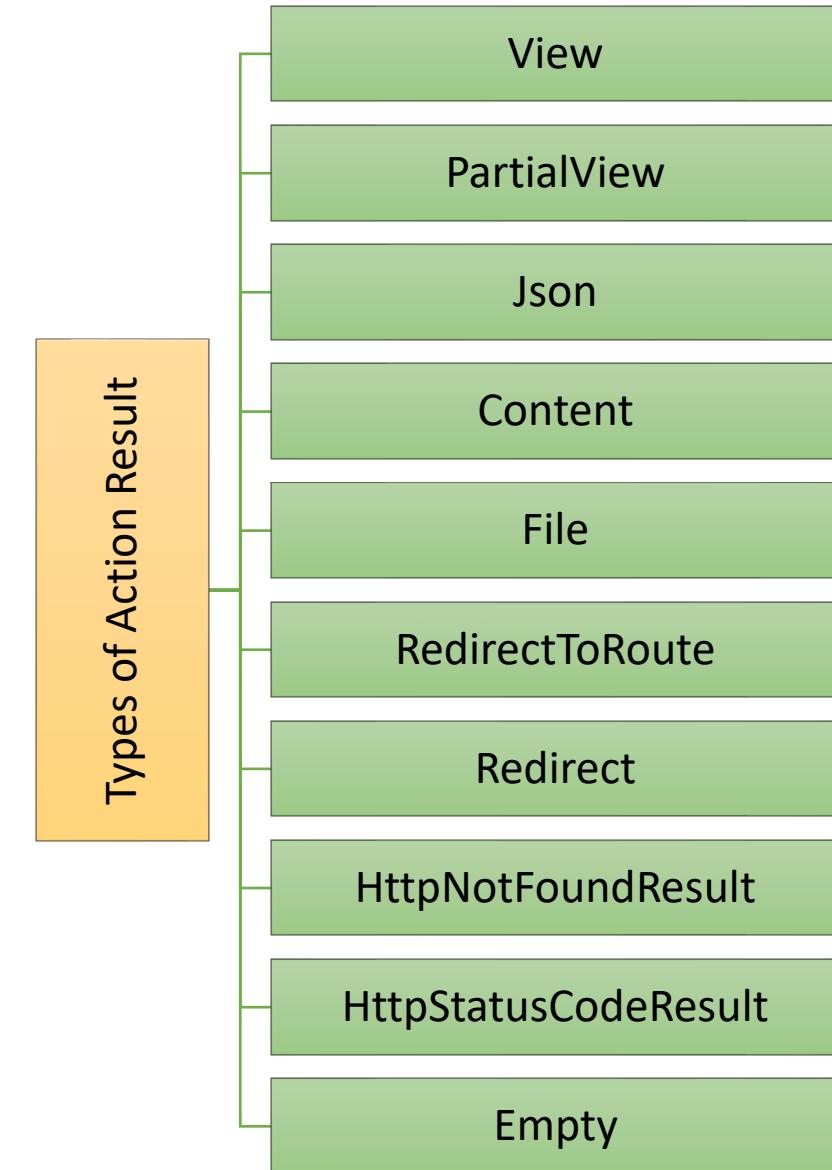
    return View(data);
}
```

1. **ViewResult:** It returns a view that renders HTML to be sent to the client.
2. **PartialViewResult:** It returns a partial view that is rendered within another view.
3. **JsonResult:** It returns JSON data that can be consumed by client-side JavaScript code.

Types of Action Result

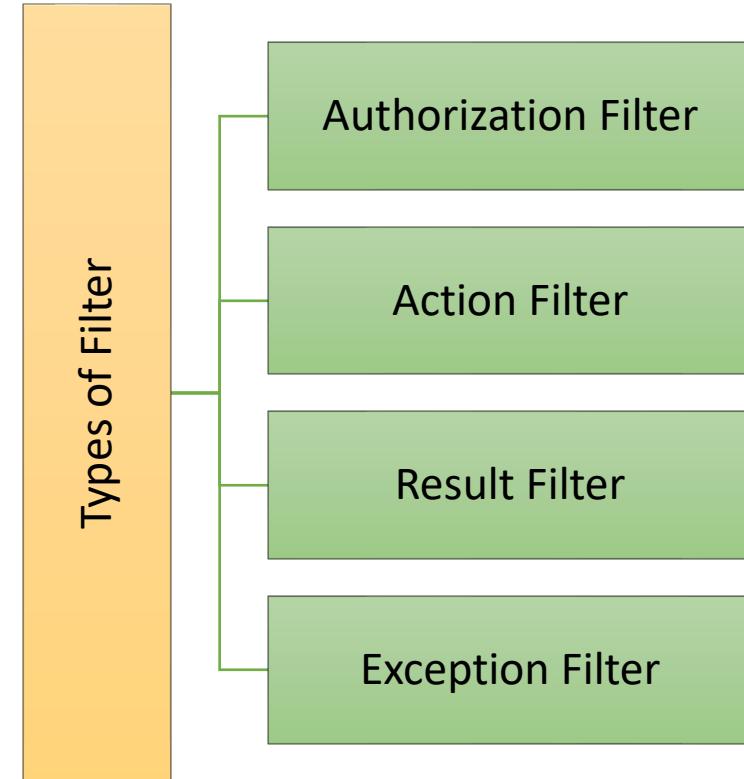


4. **ContentResult**: It returns a string of content that is directly written to the response stream. This is useful for returning plain text, XML, or other non-HTML content.
5. **FileResult**: It returns a file to the client for download, such as a PDF, image, or video.
6. **RedirectToRouteResult**: It redirects the client to another action method based on a specified route.
7. **RedirectResult**: It redirects the client to a specified URL.
8. **HttpNotFoundResult**: It returns a 404 Not Found status code to the client.
9. **HttpStatusCodeResult**: It returns an HTTP status code to the client, such as 401 Unauthorized or 500 Internal Server Error.
10. **EmptyResult**: It returns an empty response to the client without any content.



- ❖ Filters in ASP.NET MVC are used to inject logic that runs before or after an action method is executed.

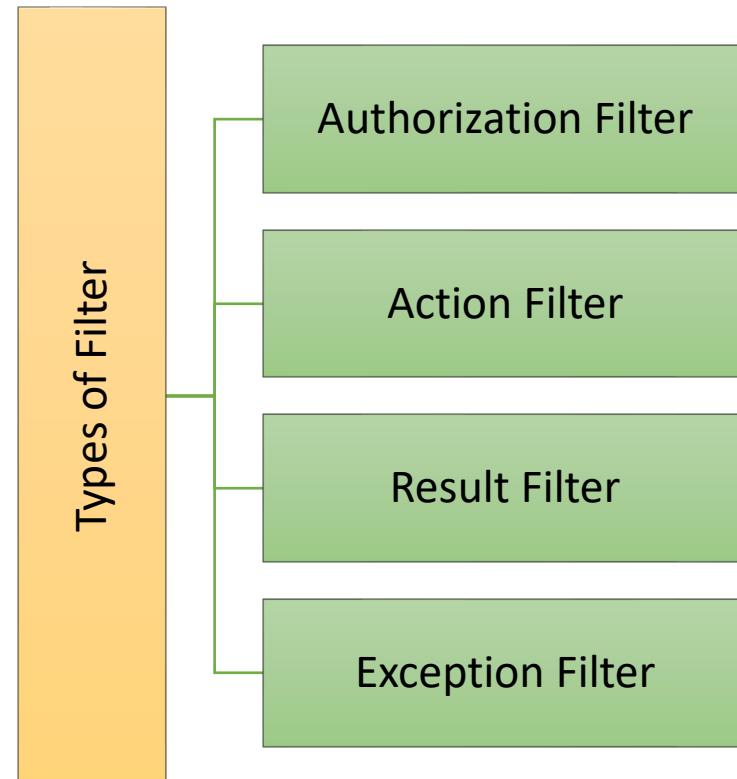
```
[Authorize(Roles = "Admin")]
public ActionResult Create()
{
    // Logic for creating a new
}
```



- 1. Authorization Filters:** Authorization filters are used to enforce authentication and authorization rules.

They run before the action method is executed and can be used to verify whether the user is authenticated and authorized to perform the requested action.

```
[Authorize(Roles = "Admin")]
public ActionResult Create()
{
    // Logic for creating a new
}
```



2. Action Filters: Action filters run before and after an action method is executed. They can be used to modify the behavior of the action method or perform pre-processing or post-processing tasks.

```
[LogActionFilter]  
  
public ActionResult Index()  
{  
    // Logic for displaying a  
}
```

```
public class LogActionFilter : ActionFilterAttribute  
{  
    public override void OnActionExecuting(ActionExecutingContext filterContext)  
    {  
        // This code will be executed before the action method is called  
    }  
  
    public override void OnActionExecuted(ActionExecutedContext filterContext)  
    {  
        // This code will be executed after the action method is called  
    }  
}
```

3. Result Filters: Result filters run before and after the result of an action method is executed. They can be used to modify the behavior of the result or perform post-processing tasks.

```
[OutputCache(Duration = 3600)]  
[LogResultFilter]  
public ActionResult Index()  
{  
    // Your action logic here  
    return View();  
}
```

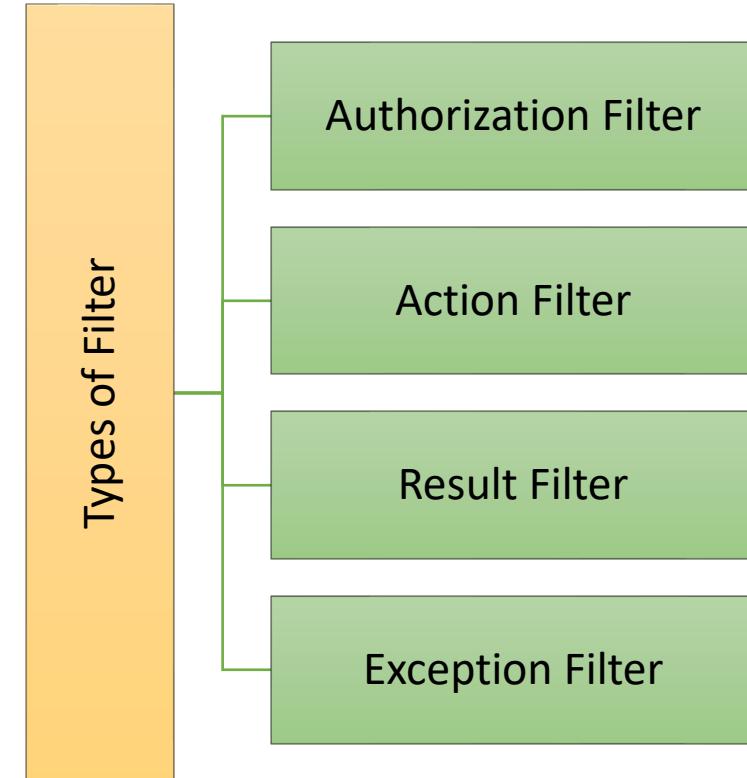
```
public class LogResultFilter : FilterAttribute, IResultFilter  
{  
    public void OnResultExecuting(ResultExecutingContext filterContext)  
    {  
        // This code will be executed before the action result  
    }  
  
    public void OnResultExecuted(ResultExecutedContext filterContext)  
    {  
        // This code will be executed after the action result  
    }  
}
```

4. Exception Filters: Exception filters are used to handle exceptions that occur during the execution of an action method. They run only when an unhandled exception occurs.

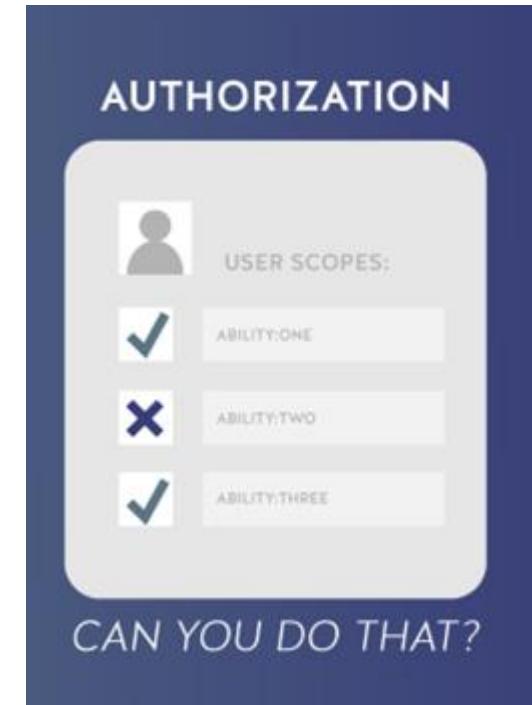
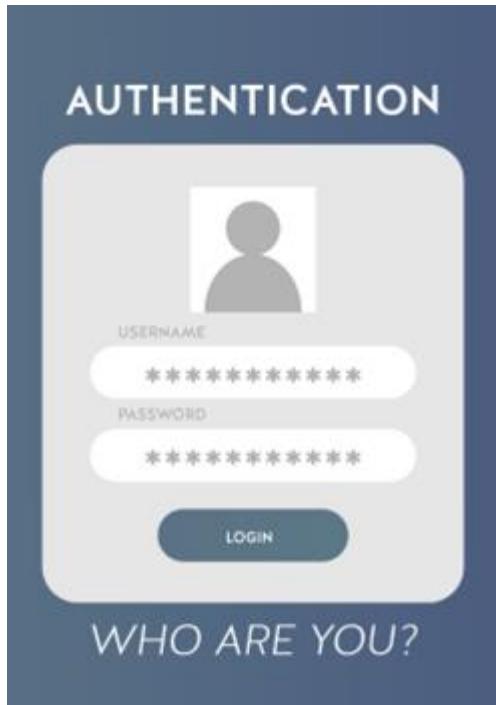
```
[LogExceptionFilter]
public ActionResult Index()
{
    // Logic for displaying
}
```

```
public class LogExceptionFilter : FilterAttribute, IExceptionFilter
{
    public void OnException(ExceptionContext filterContext)
    {
        // This code will be executed when an exception is thrown in
    }
}
```

- ❖ Filters in ASP.NET MVC are used to inject logic that runs before or after an action method is executed.



- ❖ Authentication is the process of verifying the identity of a user by validating their credentials such as username and password.
- ❖ Authorization is the process of allowing an authenticated user **ACCESS** to resources. Authentication is always precedes to Authorization.



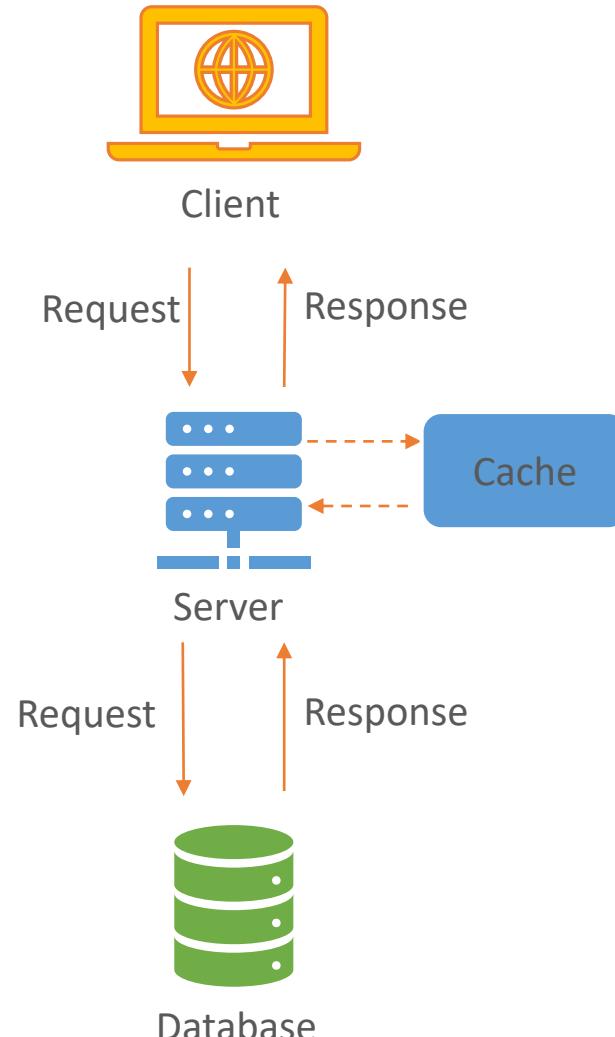
1. **Form Authentication** - Forms authentication is a cookie-based authentication mechanism that uses user credentials stored in a database to authenticate users.
2. **Passport Authentication** - Passport authentication is a centralized authentication service provided by Microsoft.
3. **Token Authentication**: Token authentication is a mechanism that uses JSON Web Tokens (JWT) to authenticate users. It is commonly used in Single Page Applications (SPA) or APIs where the user's credentials are not stored on the server.
4. **Windows Authentication** - Windows authentication is a mechanism that uses the user's Windows credentials to authenticate users. It is typically used in intranet scenarios where all users are part of the same Windows domain.

```
<authentication mode="Forms">
  <forms loginUrl="Accounts/Login"></forms>
</authentication>
```



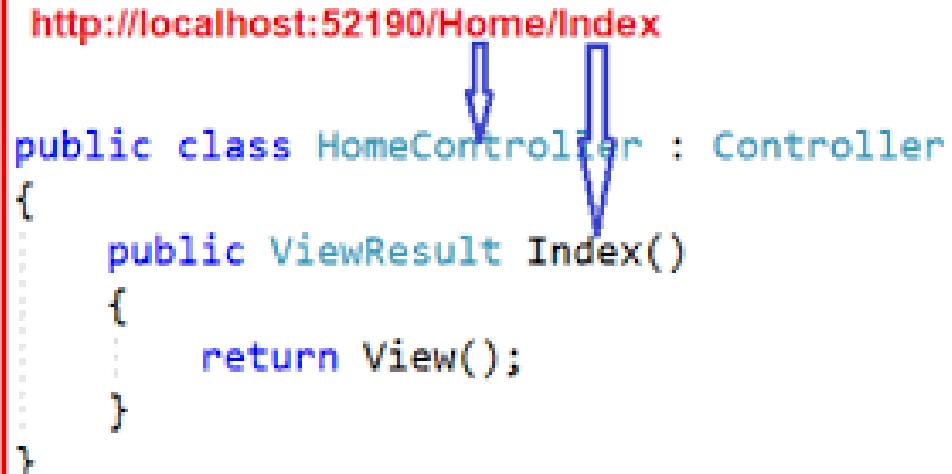
- ❖ Output caching in MVC is a technique used to improve the performance of web applications by storing the output of an action method in cache.

```
[OutputCache(Duration = 3600)]  
public ActionResult Index()  
{  
    return View();  
}
```



- ❖ Outputcache is one of the type of ResultFilter only

- ❖ Routing in MVC is the process of mapping a URL request to a specific controller action in a web application.
- ❖ The routing system is responsible for directing incoming requests to the appropriate controller, based on the URL.
- ❖ The routing system is typically configured using a **routing table**, which maps URLs to controller actions.



The diagram illustrates the routing process. At the top, the URL `http://localhost:52190/Home/Index` is shown in red. Two blue arrows point downwards from the URL to the corresponding code in a C# file. The first arrow points to the word `HomeController`, and the second arrow points to the method name `Index()`. The code itself is displayed in a syntax-highlighted editor:

```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        return View();
    }
}
```

- ❖ Attribute based routing is used to manipulate the default behavior of routing in ASP.NET MVC.

`http://localhost:1234/home/about`

```
public class HomeController: Controller
{
    [Route("Users/about")]
    Public ActionResult GotoAbout()
    {
        return View();
    }
}
```

Chapter 18 : ASP.NET – MVC - Part 2

Q155. What is the difference between ViewData, ViewBag & TempData? V Imp

Q156. How can we pass the data from controller to view in MVC?

Q157. What is Partial View?

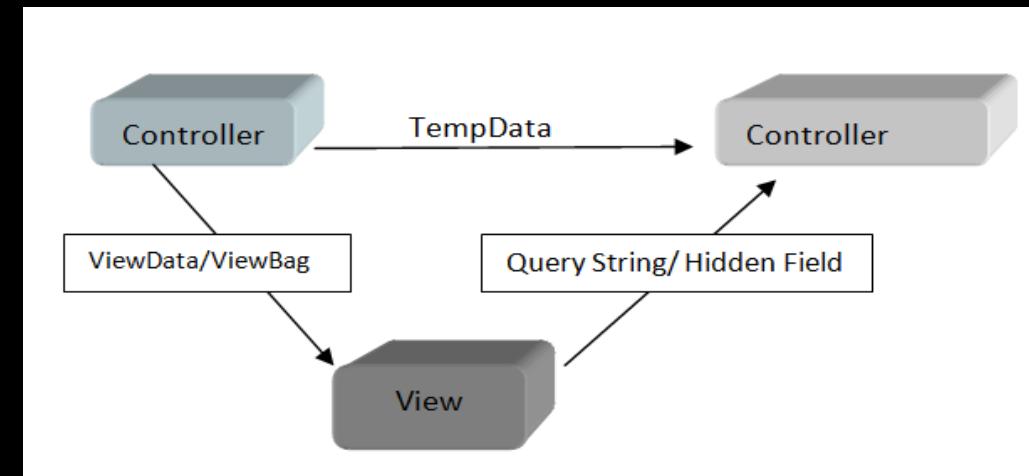
Q158. What are Areas in MVC?

Q159. How Validation works in MVC? What is Data Annotation?

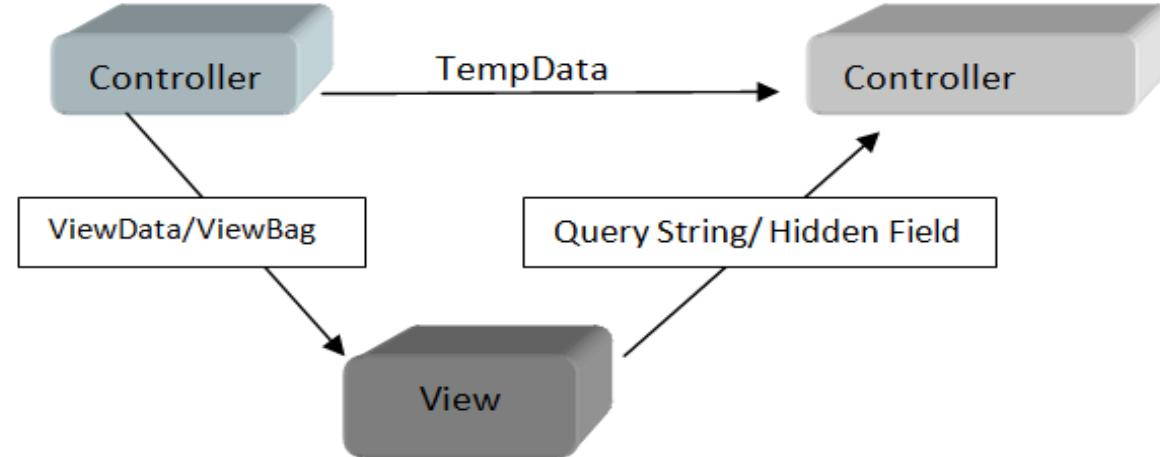
Q160. Explain the concept of MVC Scaffolding?

Q161. What is Bundling and Minification in MVC?

Q162. How to implement Security in web applications in MVC? V Imp



- ❖ ViewData, ViewBag, and TempData are all ways to **pass data** between a controller and a view.
1. ViewData and ViewBag are used to transfer data from controller to view.
 2. TempData is used to pass data from controller to controller(pass data between two action methods).
 3. ViewBag doesn't require typecasting, whereas ViewData require the typecasting.



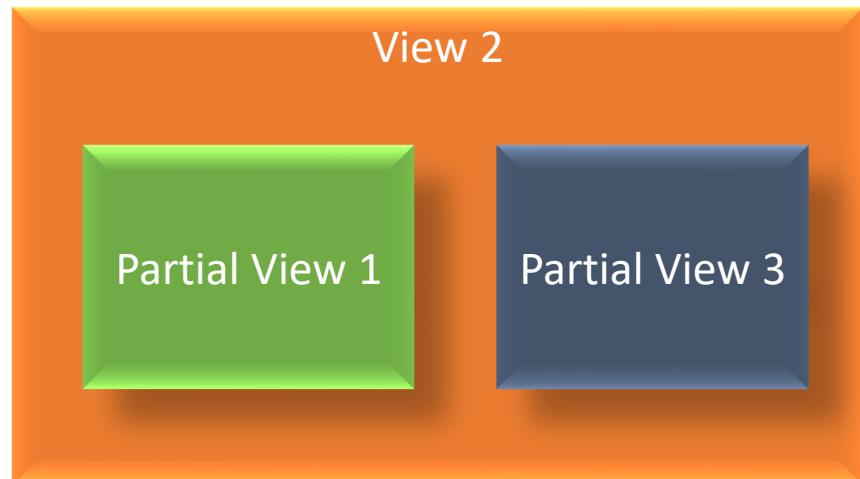
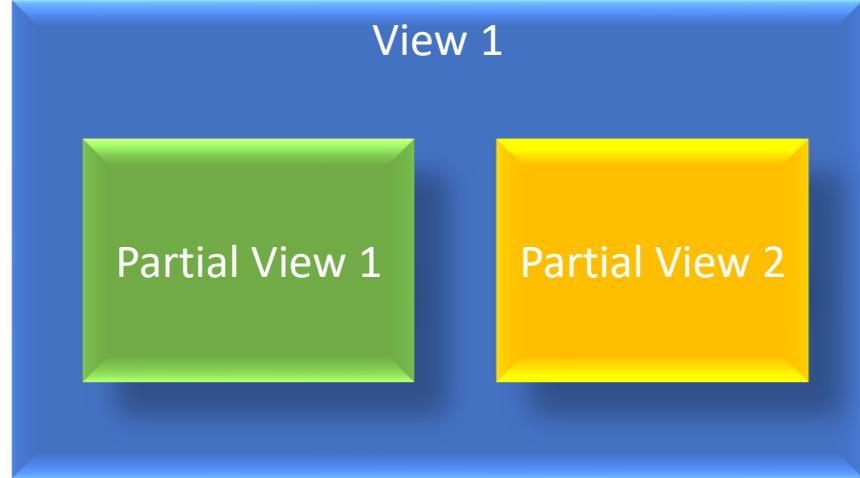
```
public ActionResult Index()
{
    ViewBag.Number = 42;
    ViewData["Number"] = 42;
    return View();
}
```

```
<p>Using ViewBag: @(ViewBag.Number)</p>
<p>Using ViewData: @((int)ViewData["Number"])</p>
```

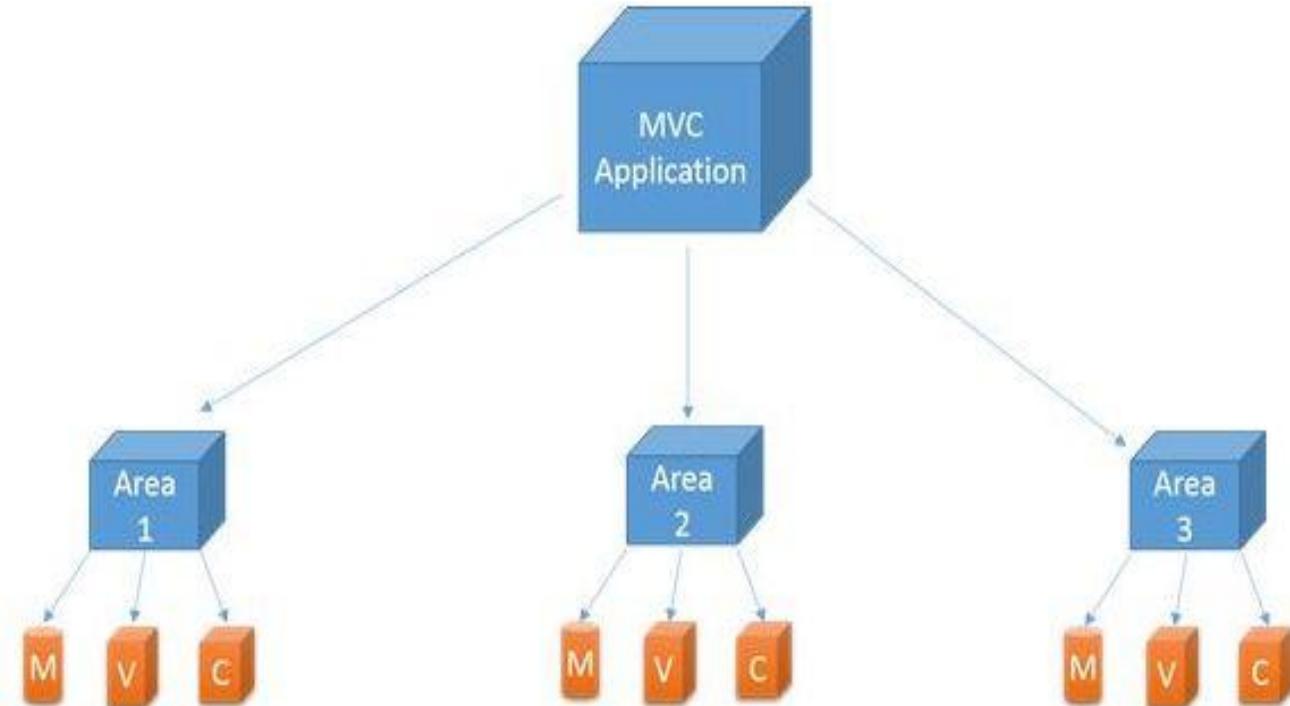
- ❖ By using ViewData and ViewBag

- ❖ A partial view is a reusable view component that can be rendered within other views or shared across multiple pages in a web application.
- ❖ It helps us to reduce code duplication.

```
<div class="container">  
    <h1>Products</h1>  
    @Html.Partial("_ProductList")  
</div>
```



- ❖ Areas are just a way to divide the modules of large applications in multiple or separated MVC.



- ❖ Validation can be done in MVC using **DATA ANNOTATION** Attributes.

```
public class Student
{
    public int StudentId { get; set; }

    [Required]
    public string StudentName { get; set; }

    [Range(10, 20)]
    public int Age { get; set; }
}
```

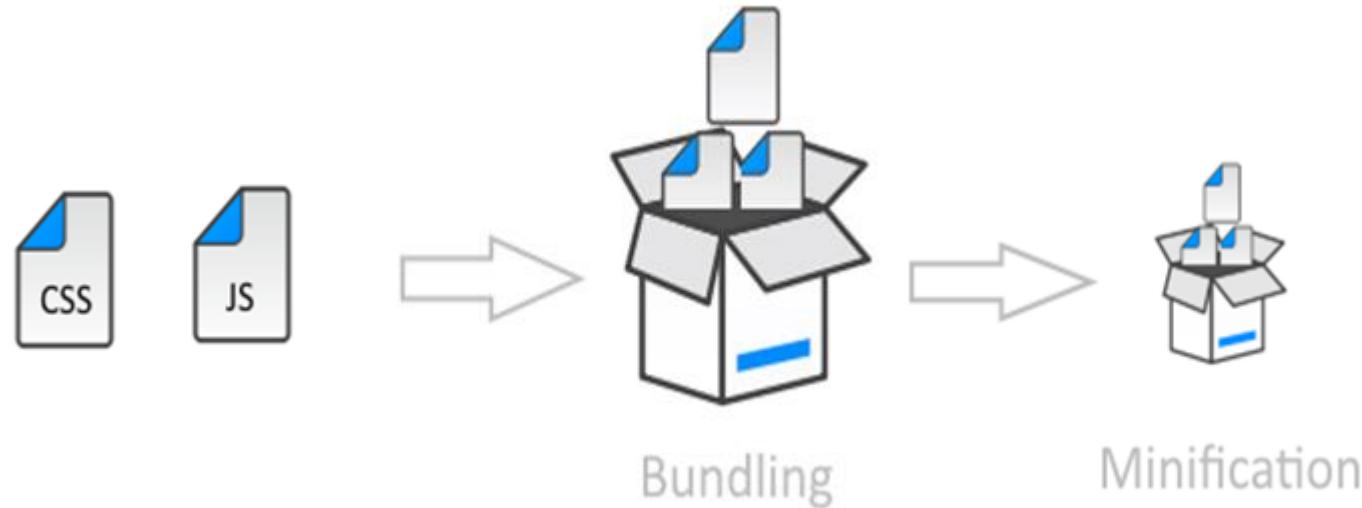
- ❖ [Required]: Specifies that a property is required and cannot be null or empty.
- ❖ [StringLength]: Specifies the minimum and maximum length of a string property.
- ❖ [RegularExpression]: Specifies a regular expression pattern that a string property must match.
- ❖ [EmailAddress]: Specifies that a string property must be a valid email address format.
- ❖ [Range]: Specifies the minimum and maximum allowed values for a numeric property.
- ❖ [Compare]: Compares the value of a property with the value of another property in the same model.
- ❖ [DataType]: Specifies the data type of a property, which can be used for formatting and validation purposes.

- ❖ [Display]: Provides metadata for displaying the property, such as the name and order of the property in a view.
- ❖ [Editable]: Specifies whether a property can be edited or displayed in a view.
- ❖ [ReadOnly]: Specifies that a property can be displayed in a view but cannot be edited.
- ❖ [HiddenInput]: Specifies that a property should be hidden in a view.
- ❖ [DisplayName]: Provides a friendly name for a property that can be used in a view.
- ❖ [DisplayFormat]: Specifies a format string for displaying the value of a property.
- ❖ [RequiredIf]: Specifies that a property is required if another property in the model has a specific value.
- ❖ [CreditCard]: Specifies that a string property must be a valid credit card number format.

[back to chapter index](#)

- ❖ In ASP.NET MVC, scaffolding is a technique that **generates code** for common web application.
- ❖ The scaffolding code can include controllers, views, and data access code automatically generated that can help developers to create web applications more quickly and with less manual coding.





- ❖ **BUNDLING** - It lets us combine multiple JavaScript (.js) files or multiple cascading style sheet (.css) files, so that they can be downloaded as a unit, rather than making individual HTTP requests for different js and css files.
- ❖ **MINIFICATION** - It squeezes out whitespace and performs other types of compression to make the downloaded files as small as possible.

1. Error handling - Must setup custom error page.
2. Cross-Site-Request-Forgery (CSRF) – Antiforgery token
3. Cross-Site-Scripting (XSS) attacks – Must validate input controls
4. Malicious file upload – Must validate the extension of files.
5. SQL injection attack - Validate inputs, use parameterized queries, use ORM (e.g. dapper , entity framework), use stored procedures and avoid dynamic queries.
6. Save the password in encrypted form so that even developer can't access it.
7. Use https



Your Name :

Your comment :

Submit

C:\Documents and Setting Upload Jpegs and Gifs only.

Chapter 19 : ASP.NET - Web Forms

Q163. What are the events in Page Life Cycle? V Imp

Q164. What is the difference between Server.Transfer() and Response.Redirect()?

Q165. What are the different types of Caching?

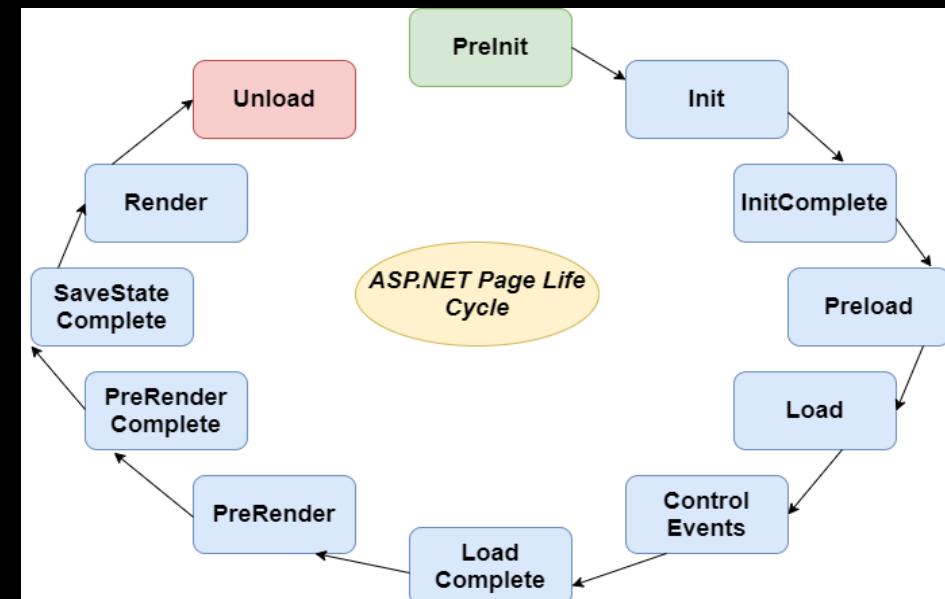
Q166. What are the types of state management? V Imp

Q167. Where the ViewState is stored after the page postback?

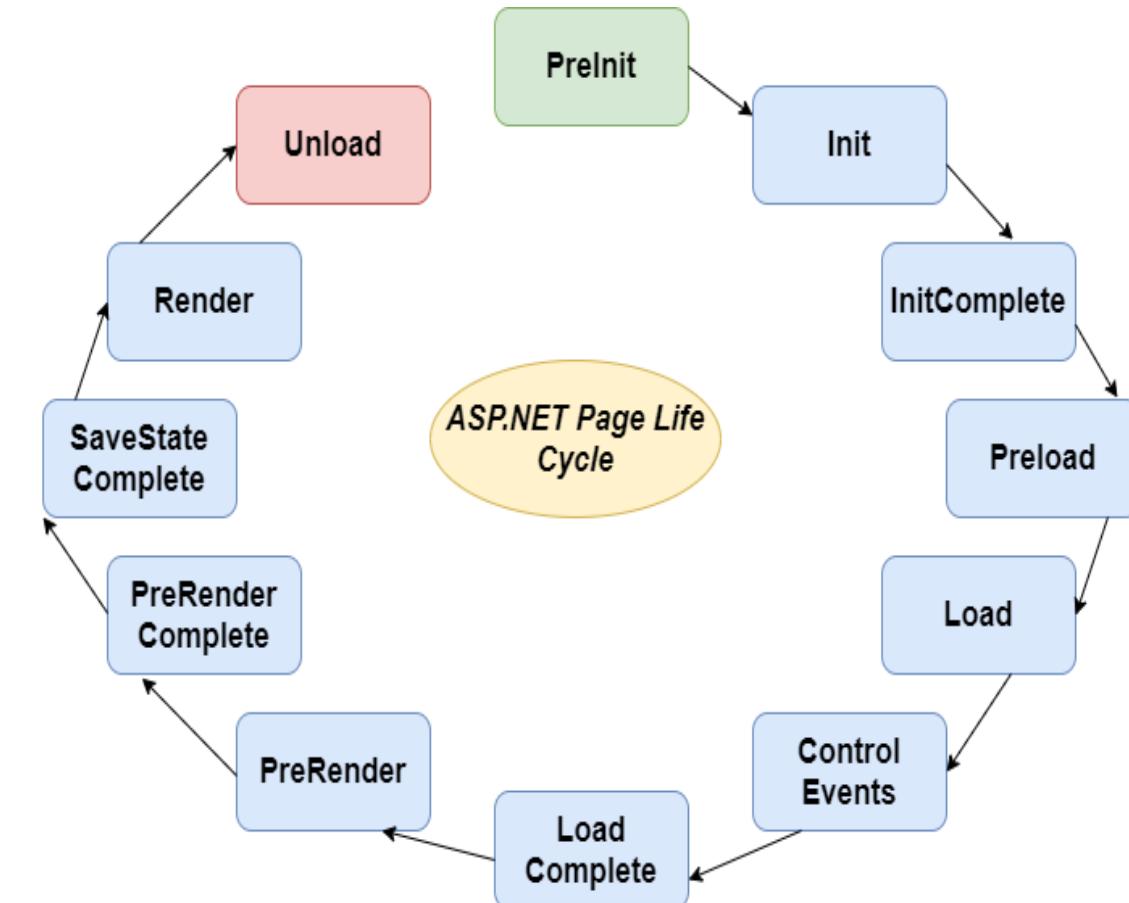
Q168. What are the different ways to store session state in asp.net?

Q169. What is cookie less session?

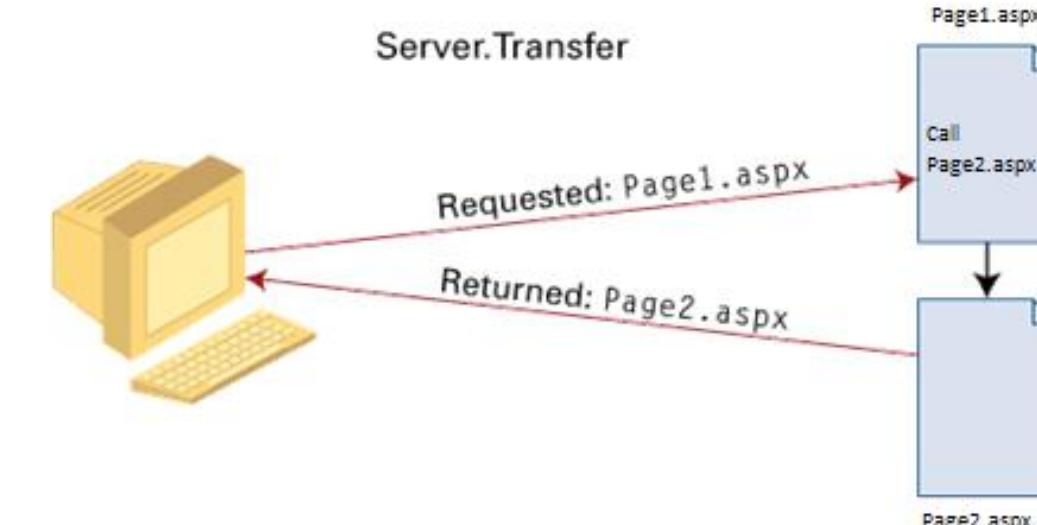
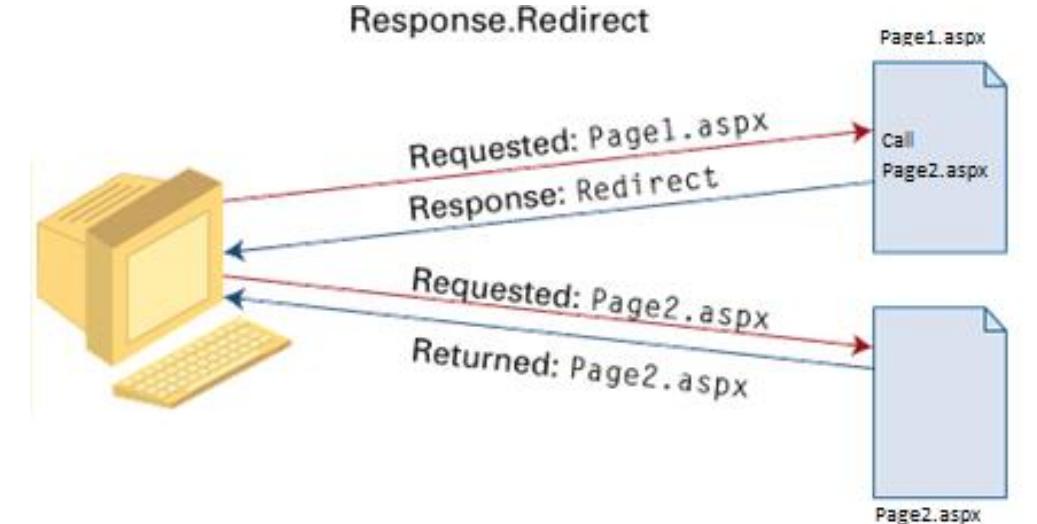
Q170. How to force all the validation controls to run in a page in web forms?



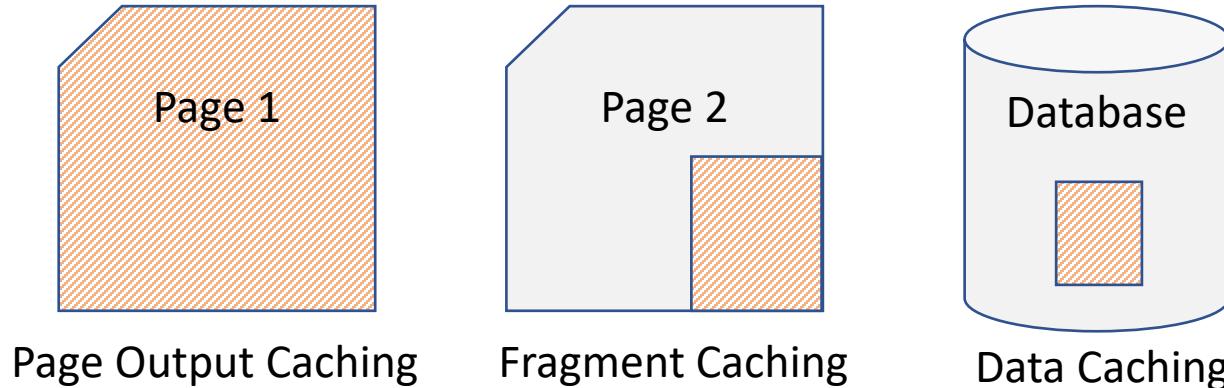
1. **Init** - This event fires after each control has been initialized.
 2. **Preload** – Set properties or default values of the control.
 3. **Load** – All the controls are ready at this event.
 4. **PreRender** - Allows final changes to the page or its control.
 5. **Render** - The Render event generates the client-side HTML
 6. **Unload** - This event is used for cleanup code.
- ❖ In PAGE LOAD event controls are fully loaded.



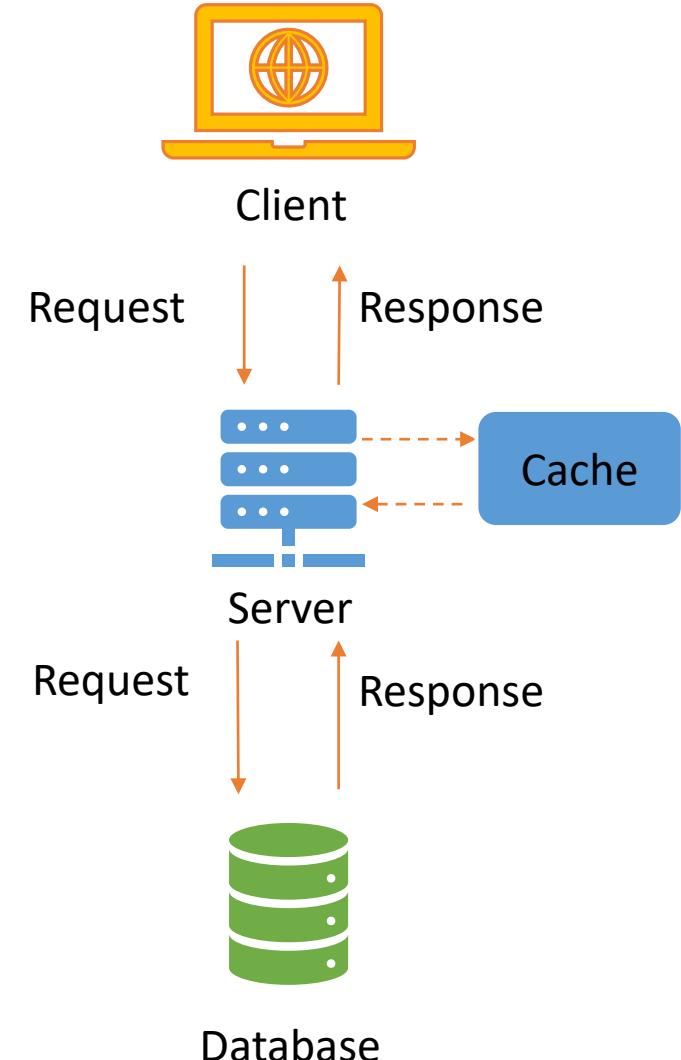
- ❖ Both Response.Redirect() and Server.Transfer() methods are used to **redirect** the user's browser from one page to another page.
- ❖ **Response.Redirect** uses **round trip** back to the client for redirecting the page.
- ❖ It is a **slow** technique, but it maintains the URL history in the client browser for all pages.
- ❖ In **Server.Transfer** page processing transfers from one page to the other page without making a round-trip back to the client's browser.
- ❖ It is a **faster** technique, but it does not maintain the URL history in the client browser for all pages.



What are the different types of Caching?



1. **Page Output Caching:** Page output caching allows you to cache the output of an **entire page**. This is useful when a page contains all static content.
2. **Fragment Caching:** Fragment caching allows you to cache a **portion or fragment** of a page. This is useful when a page contains dynamic content that can be separated from the rest of the page.
3. **Data Caching:** Data caching allows you to cache **data** retrieved from a database or other data source. This can improve performance by reducing the number of database calls.



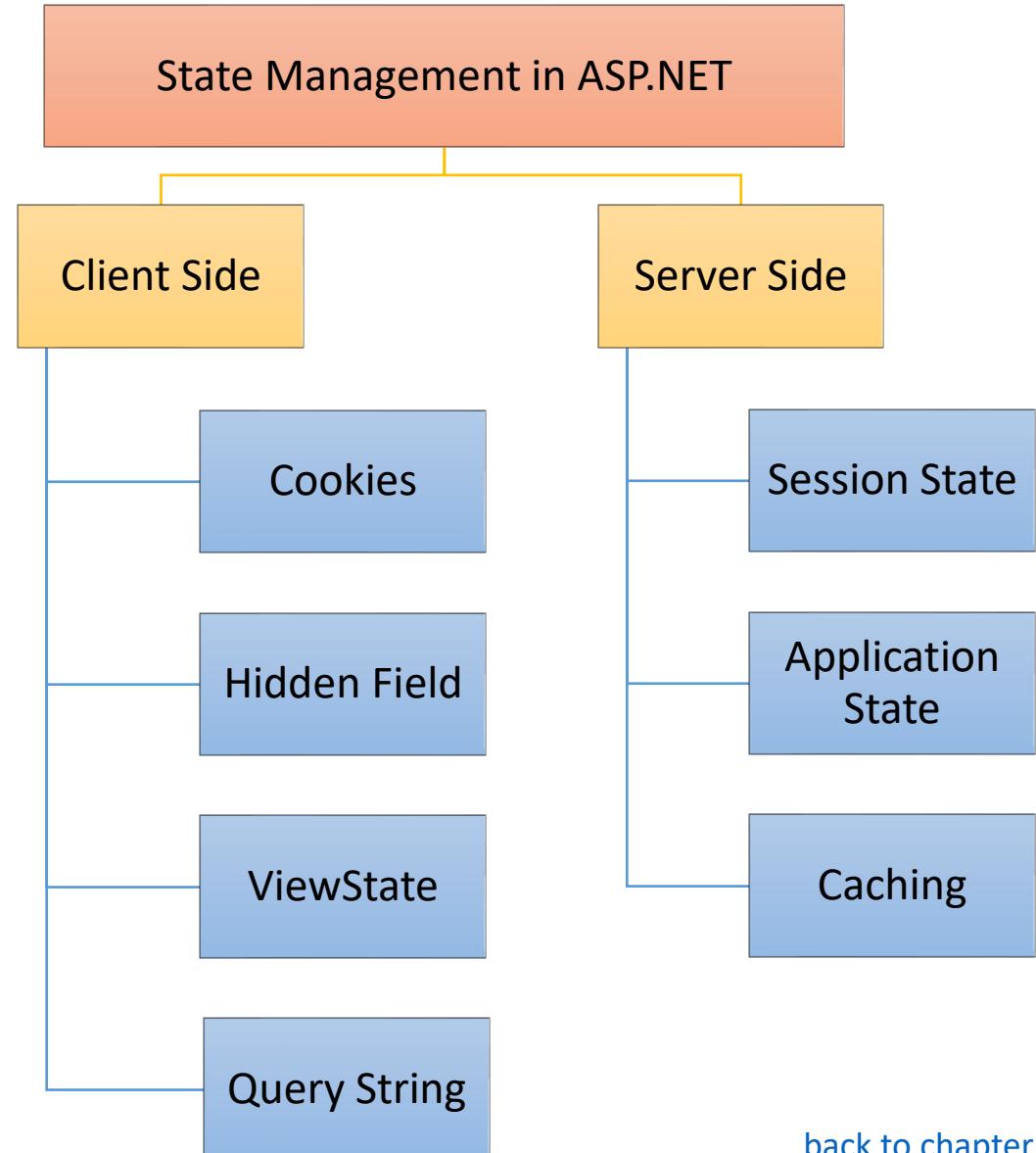
❖ State management refer to **storing** data that is required by a web application to track information between user requests and server responses.

1. Cookies: Cookies is a mechanism for storing small amounts of data on the client's machine. For example, it is used to save usernames.

2. Hidden Fields: Hidden fields are HTML form elements that can be used to store data on the client side. They are often used to pass data between pages.

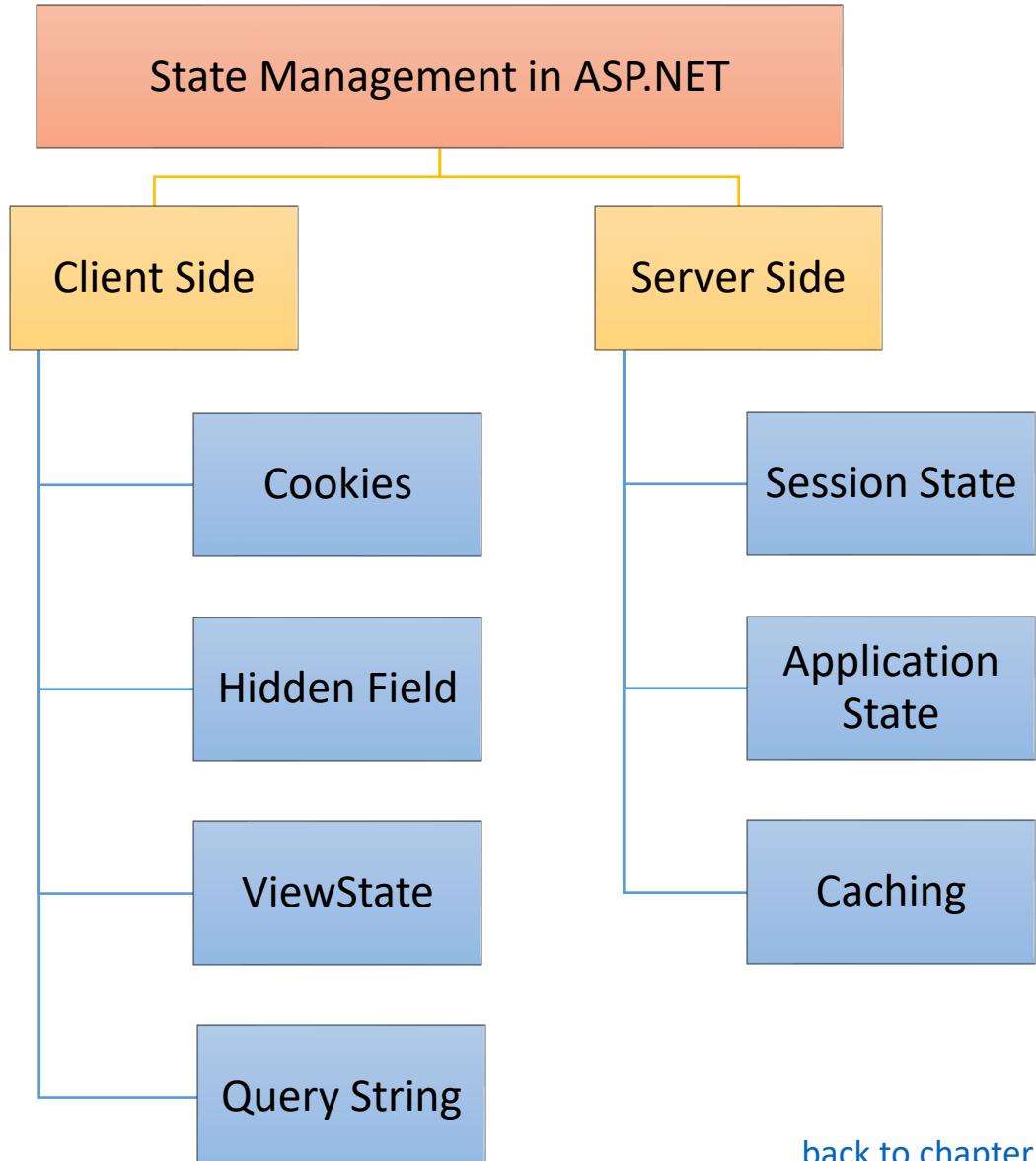
3. ViewState: ViewState is a mechanism that stores the data on the client side as a hidden field. ViewState is useful for persisting data between postbacks in a WebForms application.

4. QueryString: QueryString is a mechanism for passing data from one page to another through the URL. But it is not secure as it is visible to all.



❖ Server-side state management techniques:

1. Session State: Session state is a server-side mechanism that allows you to store user-specific data on the server. Session state uses a session ID to associate data with a specific user and can be used to persist data between page requests.
2. Application State: Application state is a server-side mechanism that allows you to store data that is shared by all users of an application.
3. Caching: Cache is a server-side mechanism that allows you to store frequently accessed data in memory.



- ❖ ViewState is stored in a HIDDEN FIELD on the page at client side.

- Session state is a mechanism that enables you to store data on server side for multiple requests. It allows you to persist data between pages.

1. In-Process Session State: This is the default session state management mode in ASP.NET, where session data is stored in memory on the same **web server**.
2. State Server Session State: This mode stores session data in a separate process called the ASP.NET State Service, which runs **outside** the web server process.
3. SQL Server Session State: This mode stores session data in a **SQL Server database**, which provides the highest level of reliability and scalability.

Types of storage for session state

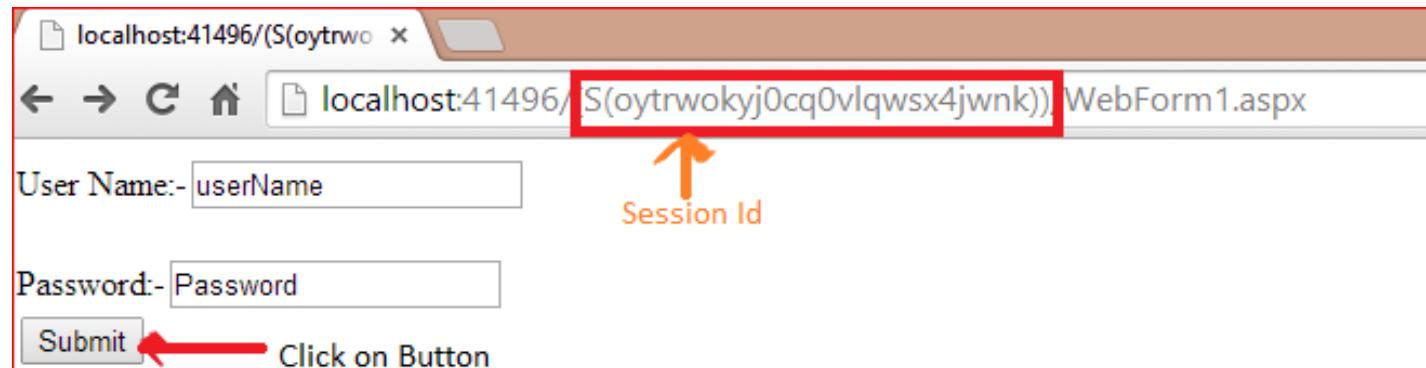


```
<sessionState mode="InProc"/>
```

```
<sessionState mode="StateServer"  
stateConnectionString="tcpip=127.0.0.1:42424"/>
```

```
<sessionState mode="SQLServer"  
sqlConnectionString="data source=myServerAddress;  
initial catalog=myDataBase;user id=myUsername;  
password=myPassword;"/>
```

- ❖ By default, a session uses a browser cookie in the background.
- ❖ In cookie less, the session is passed via **url** instead of cookie.



```
<sessionState mode="InProc" cookieless="true" timeout="20" />
```

- ❖ Page.Validate()

Chapter 20 : ADO.NET & EF

Q171. What are the main **components** of ADO.NET? V Imp

Q172. What is **Connected** architecture and **Disconnected** architecture?

Q173. What are the different **Execute Methods** of ADO.NET?

Q174. What are the **Authentication techniques** used to connect to SQL Server? V Imp

Q175. What is **ORM**? What are the different types of **ORM**?

Q176. What is **Entity Framework**?

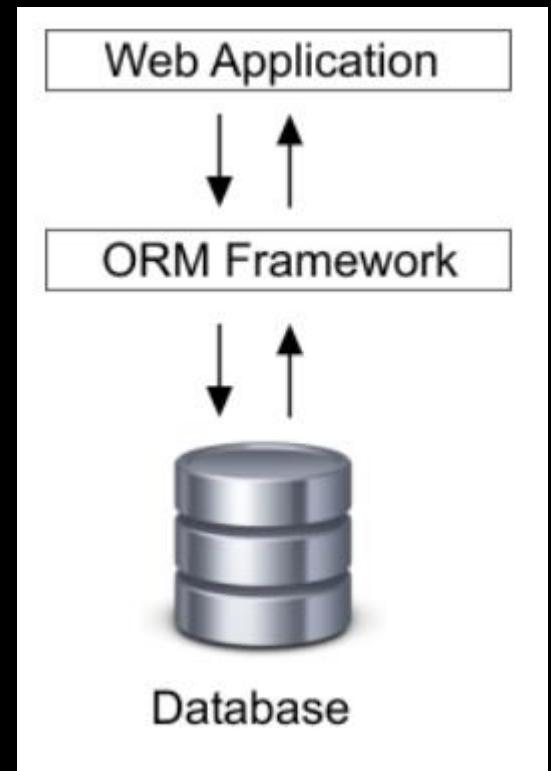
Q177. How will you differentiate ADO.NET from Entity Framework? V Imp

Q178. How **Entity Framework** works? OR How to setup EF?

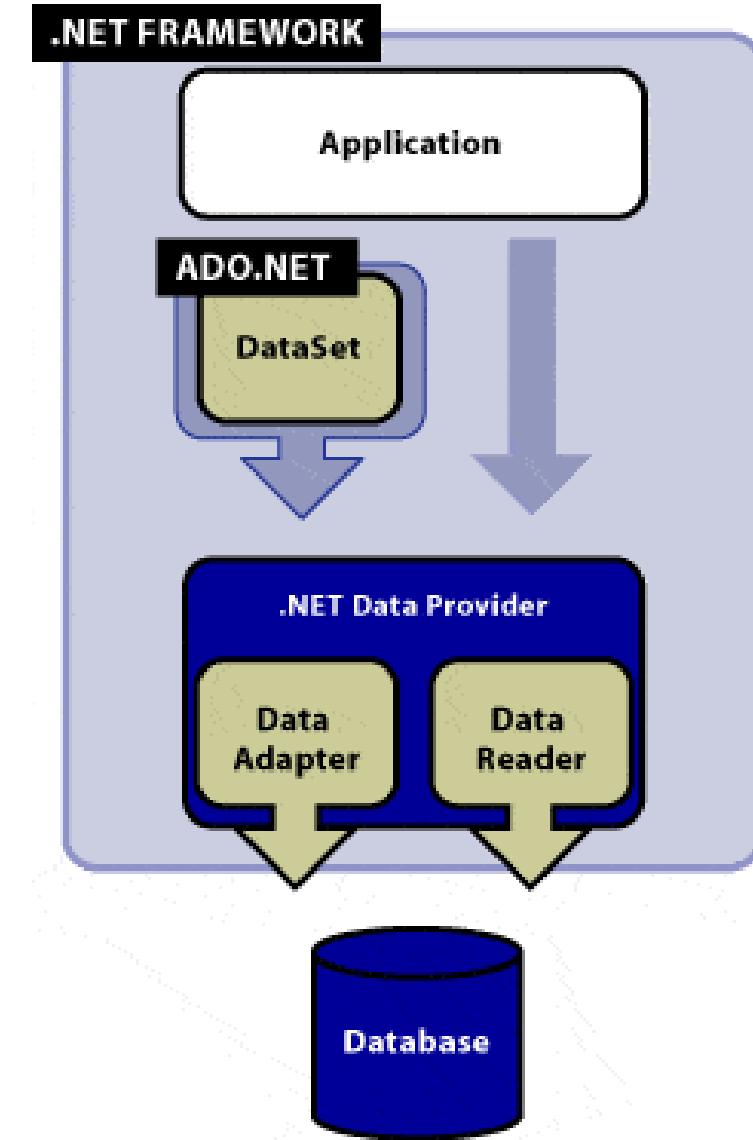
Q179. What is meant by **DbContext** and **DBSet**?

Q180. What are the different **types of application** development approaches used with EF?

Q181. What is the difference between **LINQ to SQL** and **Entity Framework**?



- ❖ **DataSet class** - A DataSet is basically a container which gets the data from one or more tables from the database. It follows disconnected architecture.
- ❖ **DataAdapter class** - A DataAdapter bridges the gap between the disconnected DataSet/ DataTable objects and the physical database.
- ❖ **DataReader Class** - The DataReader allows you to read the data returned by a SELECT command.
 1. It is read only.
 2. Unlike dataset we cannot update the database via this.
 3. It follows connected architecture.



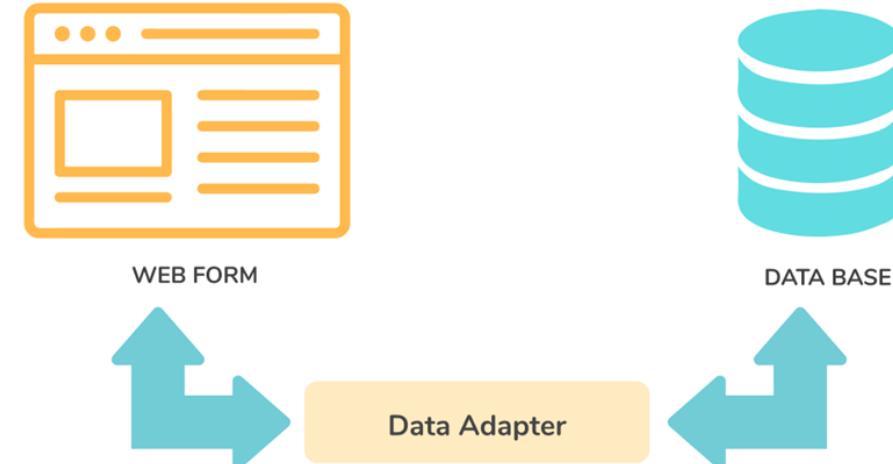
- ❖ **Disconnected Architecture** - Disconnected architecture means, you don't need to connect always to get data from the database.

1. Get data into DataAdapter.
2. Manipulate the DataAdapter
3. Resubmit the data.

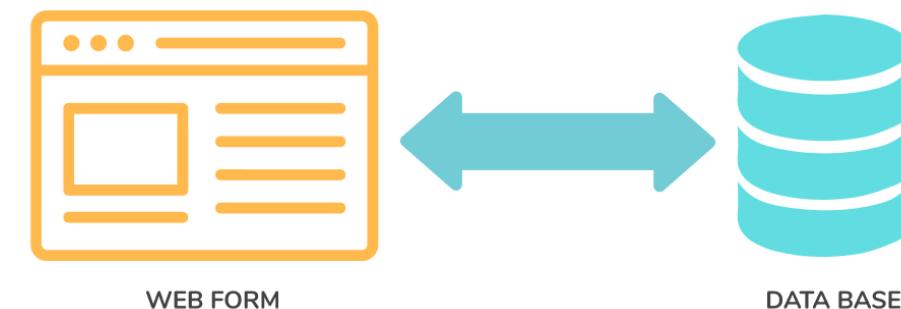
- ❖ It is fast and robust(data will not lose in case of any power failure).

- ❖ **Connected Architecture** - Connected architecture means you are directly interacting with database, but it is less secure and not robust.

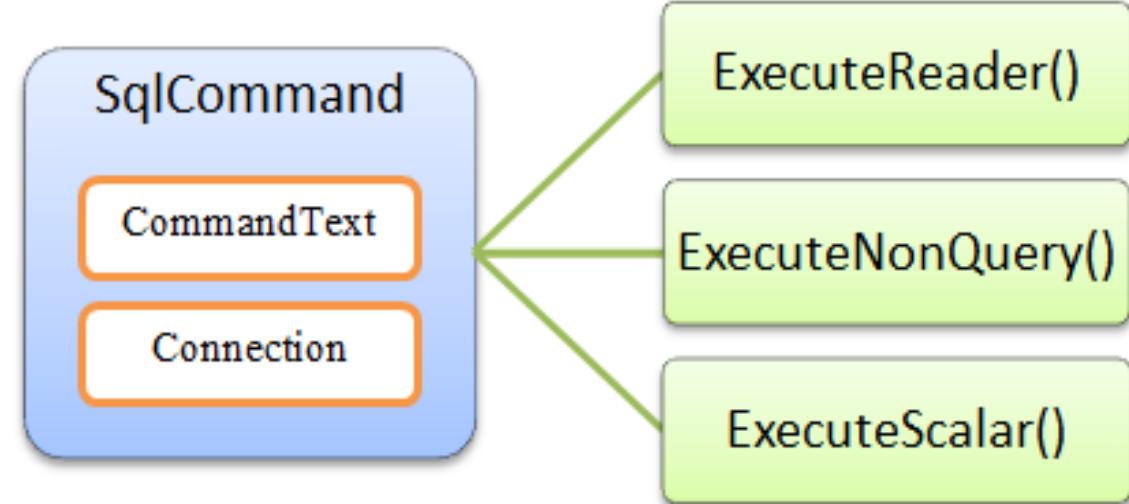
DISCONNECTED ARCHITECTURE

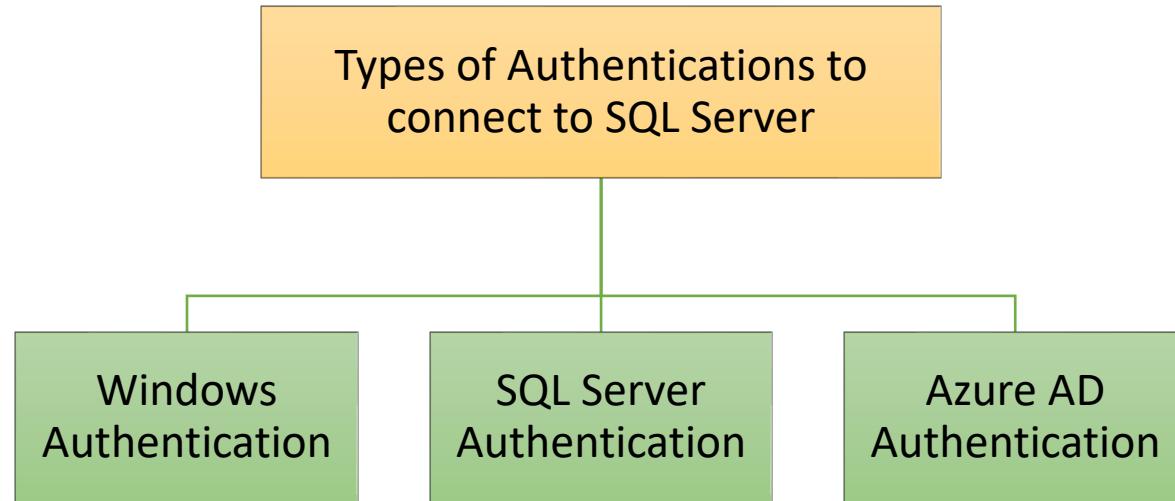


CONNECTED ARCHITECTURE



1. **EXECUTESCALAR()** - It is used to return SINGLE value from the database.
2. **EXECUTENONQUERY()** – It returns a RESULTSET from database and it has multiple values. It can be used for insert, update and delete.
3. **EXECUTEREADER()** - It only retrieves data from database. No update, insert. It is readonly.







❖ The main authentication techniques are:

1. Windows Authentication: This technique uses the current Windows user account to authenticate to SQL Server. It is the most secure and recommended technique for applications running on a Windows domain.

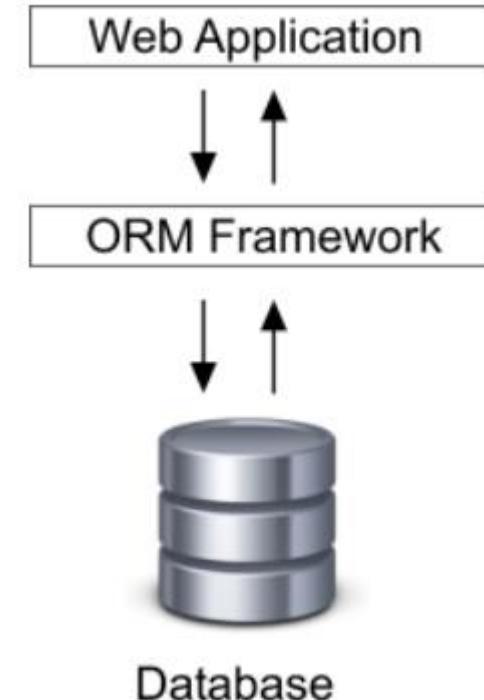
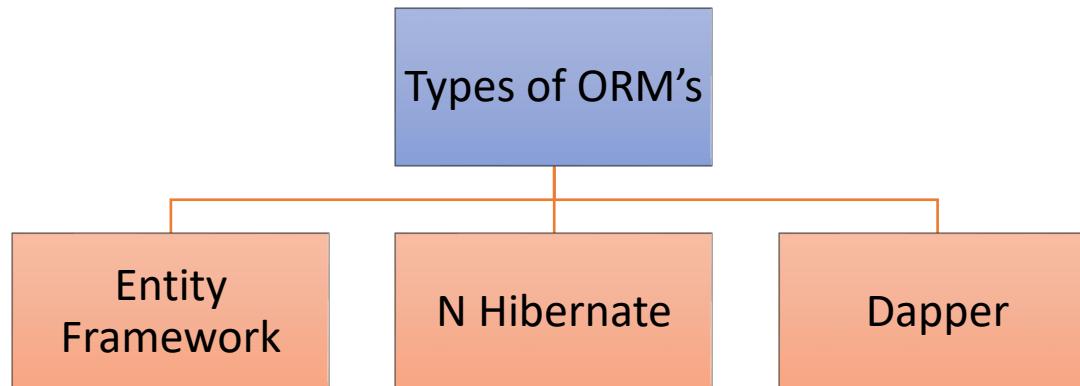
2. SQL Server Authentication: This technique uses a SQL Server user account and password to authenticate to SQL Server.

3. Azure Active Directory Authentication: This technique allows for authentication using Azure Active Directory (AAD) identities.

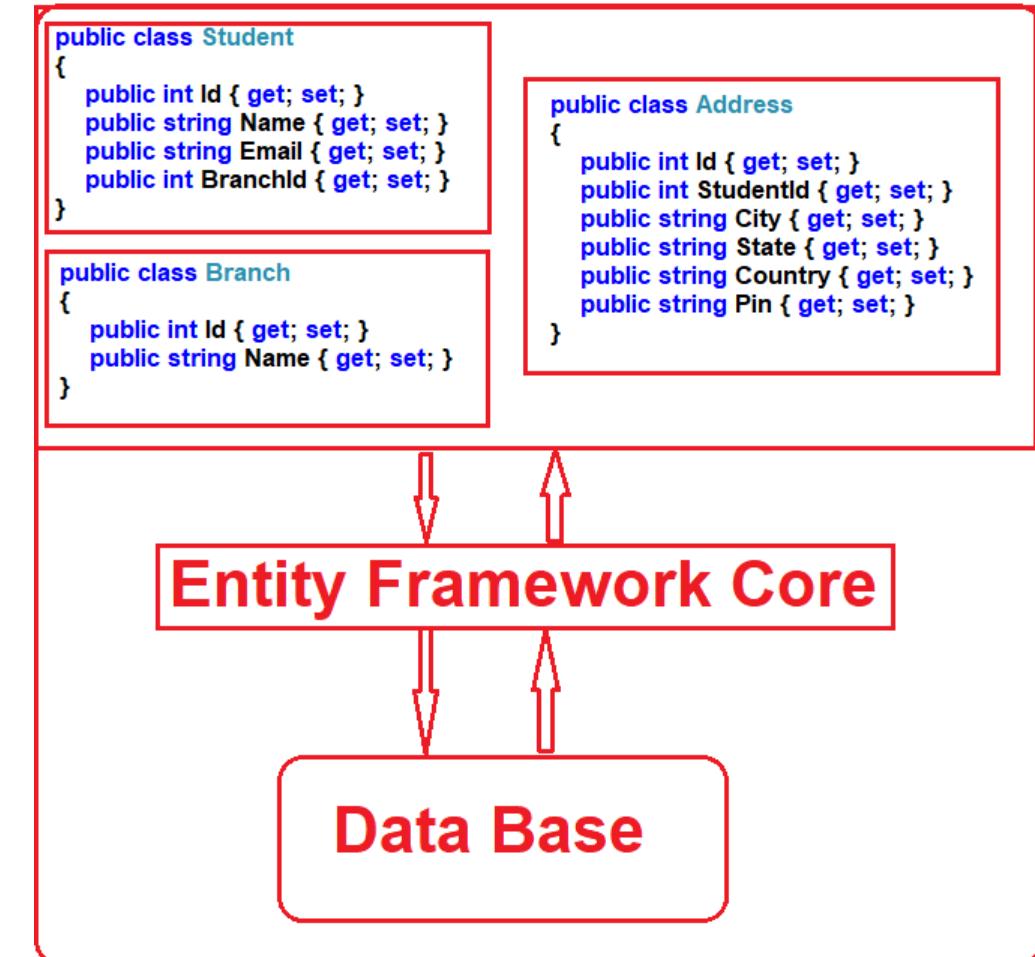
```
<connectionStrings>
  <add name="MyDbConnection"
    connectionString="Data Source=MySqlServer;
      Initial Catalog=MyDatabase;
      Integrated Security=True;" />
</connectionStrings>
```

```
<connectionStrings>
  <add name="MyDbConnection"
    connectionString="Data Source=MySqlServer;
      Initial Catalog=MyDatabase;
      User ID=MyUser;Password=MyPassword;" />
</connectionStrings>
```

- ❖ ORM (Object-Relational Mapper) is for mapping objects in your application with database tables.
- ❖ It is like a wrapper to make database calls simple and easy.



- ❖ An Entity Framework (EF) is an open-source **ORM** (Object-Relational Mapper) from Microsoft.
- ❖ It's like a **wrapper** on ADO.NET.
- ❖ Entity Framework minimizes the coding effort.



ADO.NET

```

public class UserRepository
{
    public DataSet GetAllUsersList()
    {
        DataSet ds = new DataSet(); // Initializes disconnected dataset to fetch results

        //Create and initialize the connection using connection string
        using (SqlConnection sqlConn = SqlConnection ("DataSource=localhost;
                                                Initial Catalog=TestDB; User ID=sa;Password=admin;"))
        {

            sqlConn.Open(); //Open connection

            string sql = "select * from tblusers"; // sql query to access user from table
            SqlCommand sqlCmd = new SqlCommand(sql, sqlConn);

            sqlCmd.CommandType = CommandType.Text; // Define Command Type

            SqlDataAdapter sqlAdapter = new SqlDataAdapter(sqlCmd);

            sqlAdapter.Fill(ds); //Get the data in disconnected mode
        }

        return ds; // return dataset result
    }
}

```

EF

```

public class UserRepository
{
    public static void Main(string[] args)
    {
        // Initializes the dbContext class User Entity
        using (var userContextDB = new UserContext())
        {

            // Create a new user object
            var user = new User() { UserID = "USER-01" };

            // Call Add Command
            userContextDB.User.Add(user);

            // Execute the save command to make changes
            userContextDB.SaveChanges();
        }
    }
}

```

Entity Framework

1. In EF, code is simpler and short.

2. EF can generate SQL statements automatically based on LINQ queries.

3. EF is designed to be database independent, which means same code can work with different databases.

4. EF is slightly slower than ADO.NET because internally EF uses ADO.NET only.

ADO.NET

In ADO.NET, same code is bigger.

ADO.NET requires developers to write SQL statements themselves.

ADO.NET is tightly coupled to SQL Server and requires different code for different database systems.

❖ How to add records to Student table in database using EF?

1. Install-Package EntityFramework

2. Create the data model for the student entity

3. Create the database context

In the *Models* folder, create a class file named *Student.cs*

```
public class Student
{
    public int ID { get; set; }
    public string Name { get; set; }
}
```

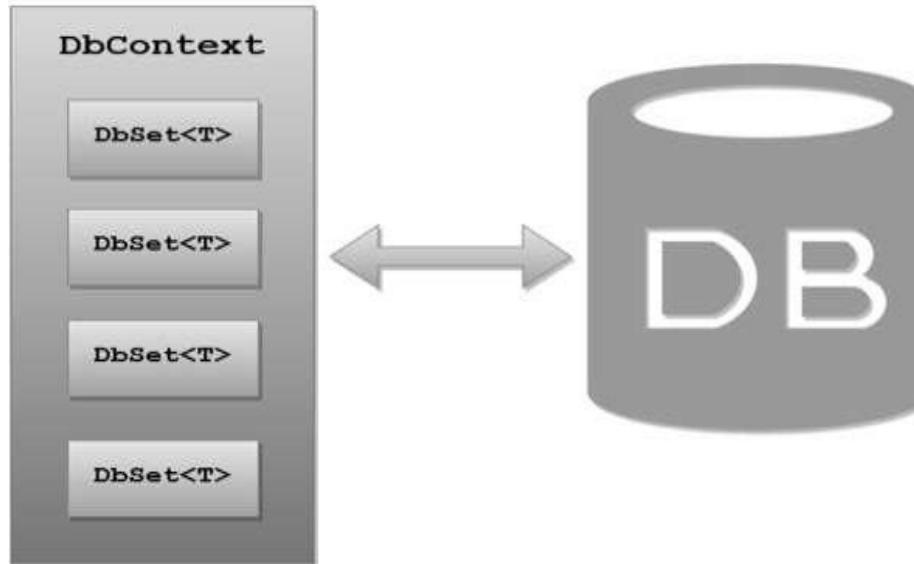
```
public class SchoolContext : DbContext
{
    public DbSet<Student> Students { get; set; }
}
```

3. Add data to student table

```
protected void AddStudents(SchoolContext context)
{
    var students = new List<Student>
    {
        new Student{Name="Happy"},
        new Student{Name="John"},
        new Student{Name="Amit"},
    };

    students.ForEach(s => context.Students.Add(s));

    context.SaveChanges()
}
```

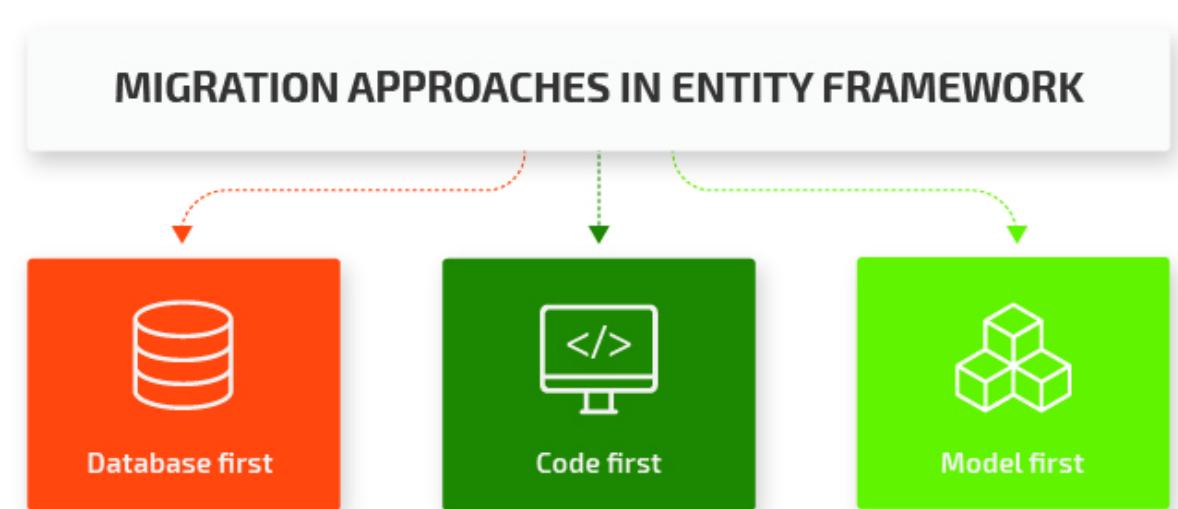


- ❖ DbContext is a class in the Entity Framework that helps in creating the **communication** between the database and the domain/entity class.
- ❖ The DbSet class represents an **entity set** that can be used for create, read, update, and delete operations.

```
public class MyDbContext : DbContext
{
    public MyDbContext() : base("MyDbConnectionString")
    {
        // optional: disable database initialization
        Database.SetInitializer<MyDbContext>(null);
    }

    public DbSet<Customer> Customers { get; set; } // Db
    // add more DbSet properties for other tables here
}
```

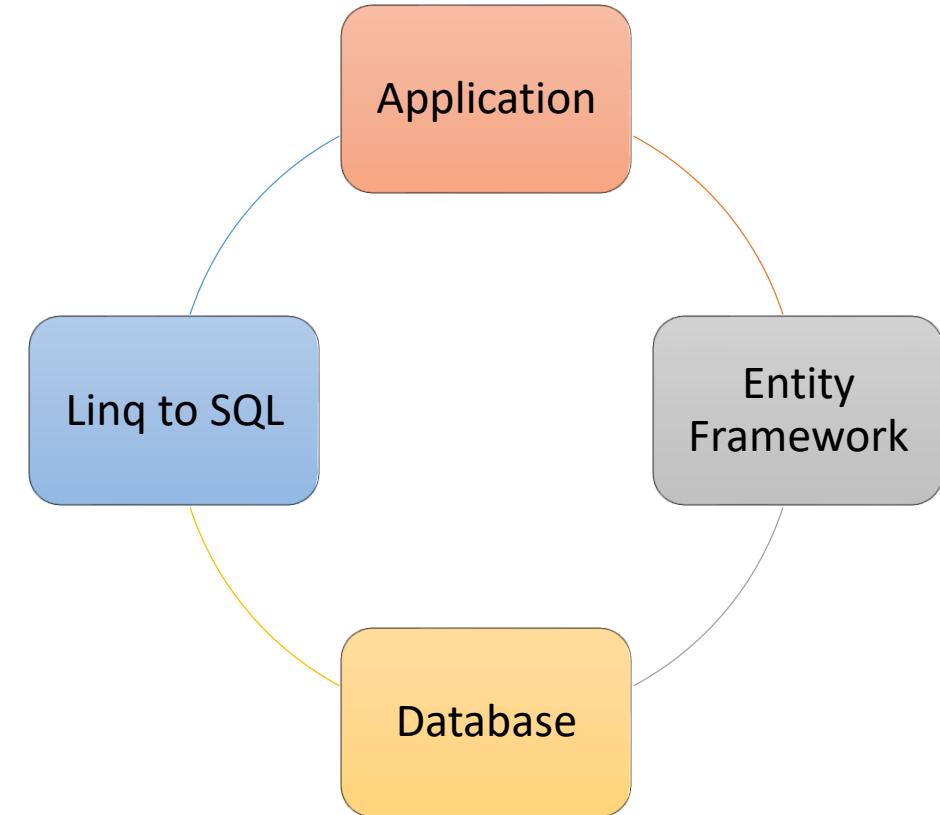
1. **Database First** – In Database First approach first the database is created and then the entity model is generated from it.
2. **Code First** - The Code First approach involves creating the data model using C# or VB.NET classes. Developers can define entities, relationships, and other schema elements using code, and then generate the database schema from the code.
3. **Model First** - The Model First approach involves creating the data model using visual tools such as Entity Data Model Designer in Visual Studio. Developers can create entities, relationships, and other schema elements visually, and then generate the database schema from the model.



- ❖ LINQ to SQL and Entity Framework are both Object-Relational Mapping (ORM) frameworks that allow developers to interact with relational databases using object-oriented programming techniques.

- ❖ Differences between LINQ to SQL and EF:
 1. LINQ to SQL is a lightweight ORM that supports only SQL Server databases.
 2. Entity Framework is a more robust ORM that supports multiple database providers, including SQL Server, Oracle, MySQL etc.

 3. LINQ to SQL is suitable for small to medium-sized applications.
 4. Entity Framework is suitable for large and complex applications.



Chapter 21 : Web API - Basics

Q182. What is **Web API**? What is the purpose of Web API?

Q183. What are Web API advantages over **WCF** and web services? V Imp

Q184. What are **HTTP verbs or HTTP methods**?

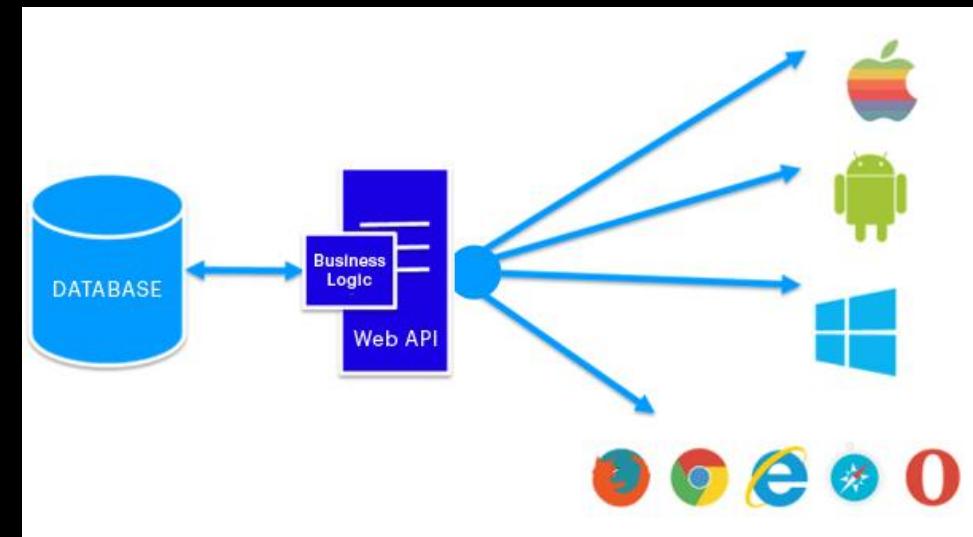
Q185. What is the difference **Rest API** and **Web API**?

Q186. What are **REST guidelines**? What is the difference between Rest and Restful?

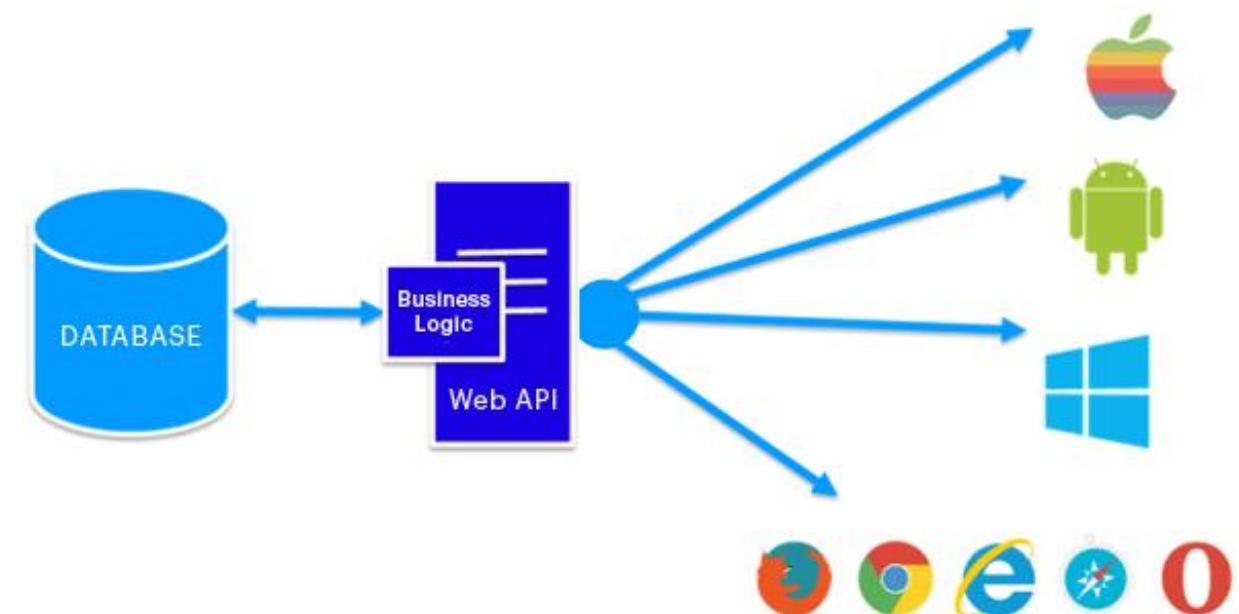
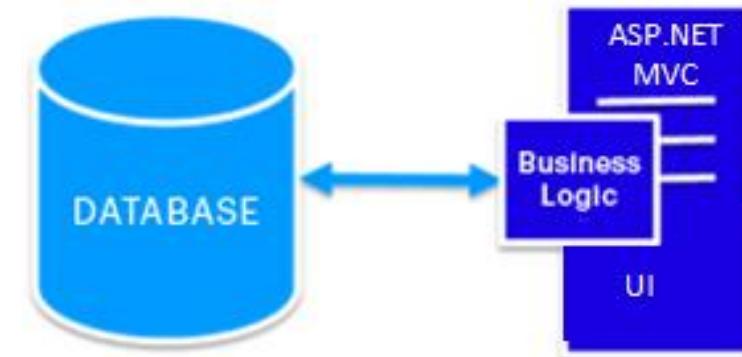
Q187. Is it possible to use **WCF** as Restful services?

Q188. How to consume Web API from a .NET MVC application?

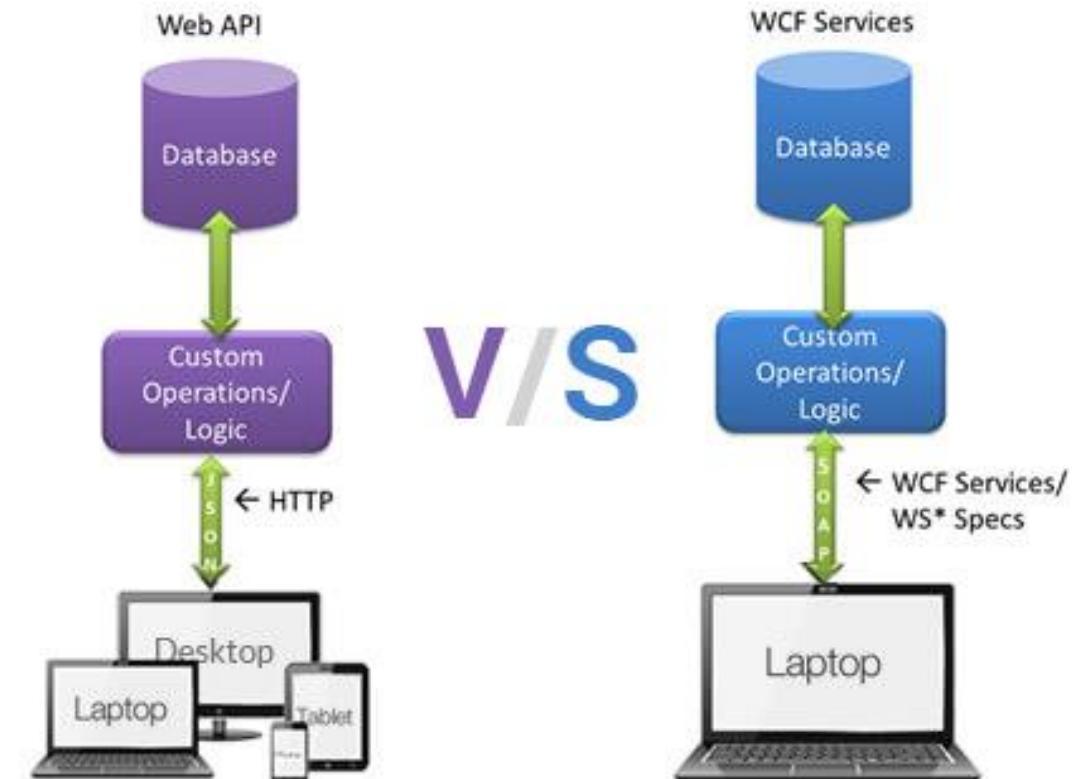
Q189. What is the difference between **Web API** and **MVC Controller**?

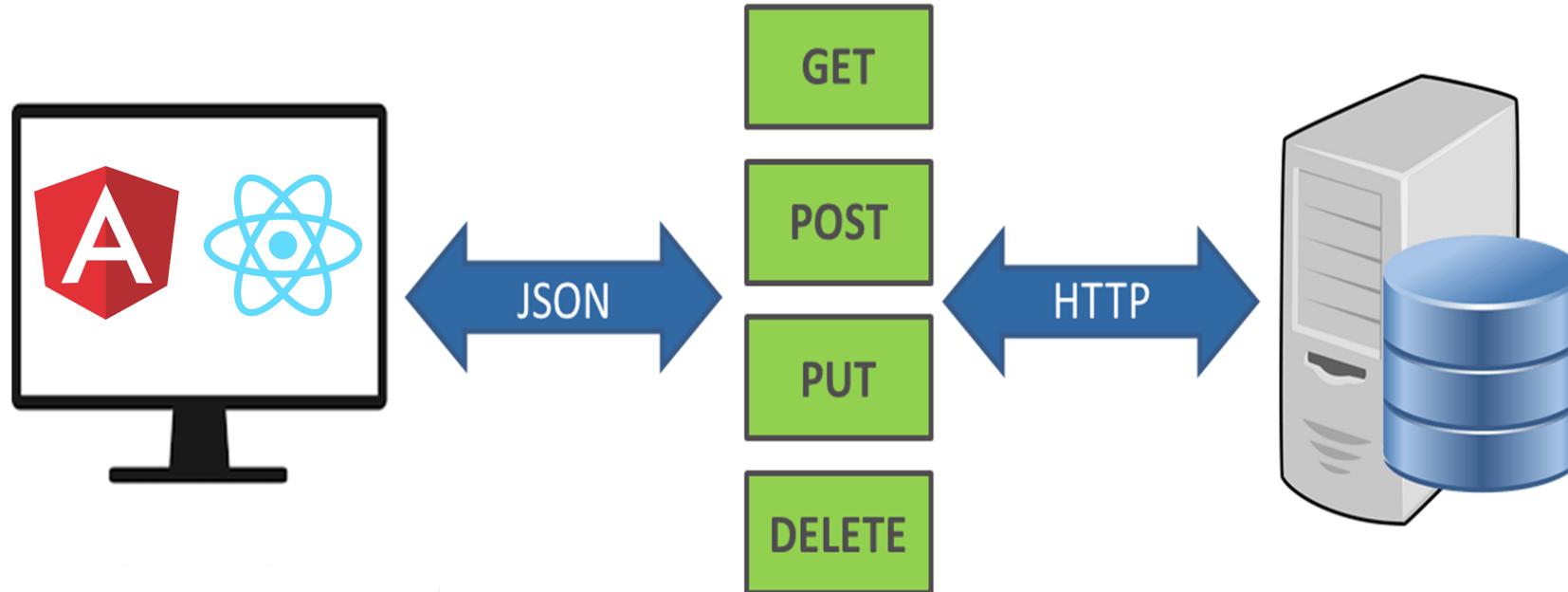


- ❖ Web API(Application Programming Interface) are HTTP based services that can be accessed from browser or mobile-apps.
- ❖ Web API enables different software applications to communicate with each other through the internet.
- ❖ Web API's mostly use JSON and XML formats to transfer data between different applications.

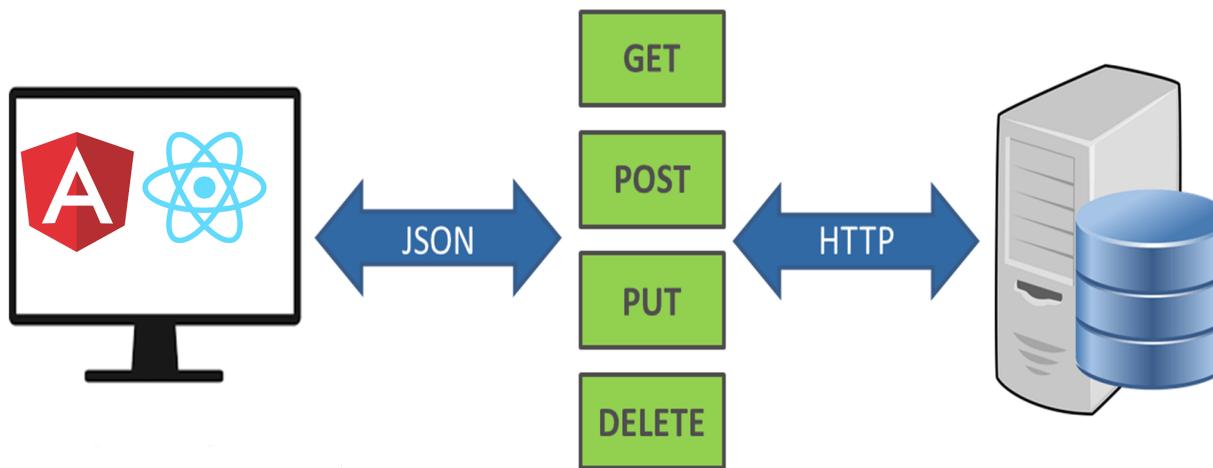


1. Web APIs are **simpler** to use and learn, and use familiar web technologies like HTTP, JSON and XML.
2. Web APIs are more **flexible**, can be hosted on a wide range of platforms, and are compatible with many devices including mobile devices, web browsers, and IoT devices.
3. Web APIs are more **scalable** and can handle a large number of requests without consuming too many resources.
4. Web APIs are **faster** than WCF because they use lightweight protocols like JSON and HTTP.





HTTP Method	Action	Examples
GET	Obtain information about a resource	<code>http://example.com/api/orders</code> (retrieve order list)
GET	Obtain information about a resource	<code>http://example.com/api/orders/123</code> (retrieve order #123)
POST	Create a new resource	<code>http://example.com/api/orders</code> (create a new order, from data provided with the request)
PUT	Update a resource	<code>http://example.com/api/orders/123</code> (update order #123, from data provided with the request)
DELETE	Delete a resource	<code>http://example.com/api/orders/123</code> (delete order #123)



Parsed Raw Scratchpad Options

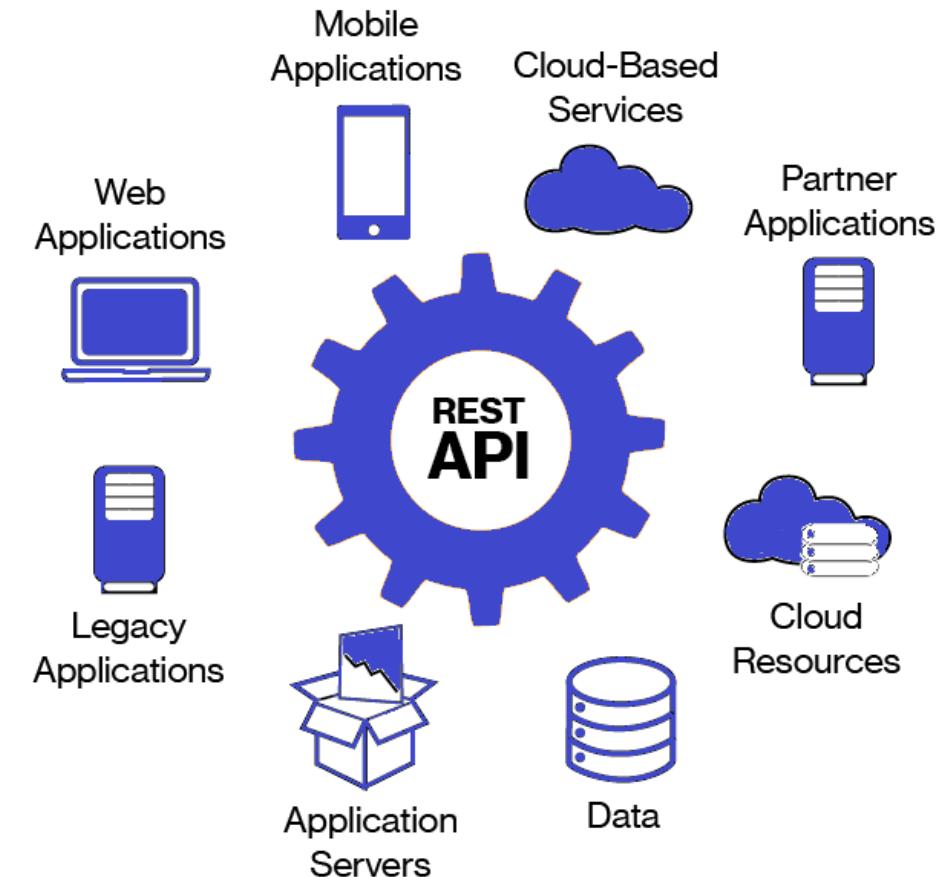
POST http://localhost:60464/api/student?age=25

User-Agent: Fiddler
Host: localhost:60464
Content-Type: application/json
Content-Length: 31

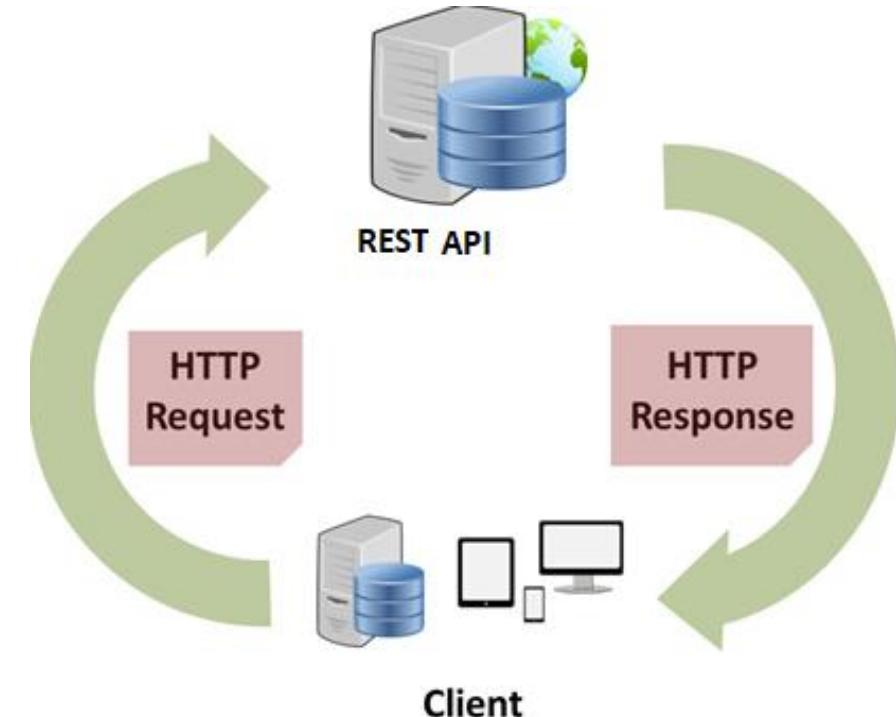
Request Body

```
{  
  id:1,  
  name:'steve'  
}
```

- ❖ REST is a **set of guidelines** which helps in creating a system where applications can easily communicate with each other.
- ❖ REST stands for Representational State Transfer.



1. **Separation of Client and Server** - The implementation of the client and the server must be done independently.
 2. **Stateless** - The server will not store anything about the latest HTTP request the client made. It will treat every request as new request.
 3. **Uniform interface** – Identify the resources by URL
www.abc.com/api/questions
 4. **Cacheable** – The API response should be cacheable to improve the performance.
 5. **Layered system** - The system should follow layered pattern.
For example, MVC is a layered system.
- ❖ **RESTFUL Service:** If a system written by applying REST guidelines, then it is also called REST service or RESTful services.



Is it possible to use WCF as Restful services?



- ❖ Yes, we can develop RESTful services with WCF by following the REST guidelines in WCF service.

- ❖ Web API methods can be consumed with the help of **HttpClient** class.

2. Set the BaseAddress property of client object.

4. GetAsync method will send request to find api resource GetAllEmployees. And then store the response in HttpResponseMessage class object.

6. If successful then result is saved in a variable

1. Create the object of HttpClient class.

3. Add media type, the request format of data. Here it is json.

5. Checking the response is successful or not.

7. Then the variable will be deserialized from json format to Employee object.

```
public class HomeController : Controller
{
    string Baseurl = "http://facebook.com/api";

    public async Task<ActionResult> Index()
    {
        List<Employee> EmpInfo = new List<Employee>();

        using (var client = new HttpClient())
        {
            client.BaseAddress = new Uri(Baseurl);
            client.DefaultRequestHeaders.Clear();

            client.DefaultRequestHeaders.Accept.Add(new
                MediaTypeWithQualityHeaderValue("application/json"));

            HttpResponseMessage Res = await client.GetAsync("Employee/GetAllEmployees");

            if (Res.IsSuccessStatusCode)
            {
                var EmpResponse = Res.Content.ReadAsStringAsync().Result;
                EmpInfo = JsonConvert.DeserializeObject<List<Employee>>(EmpResponse
                );
            }
            return View(EmpInfo);
        }
    }
}
```

[back to chapter index](#)

Web API Controller	MVC Controller
1. Web api controller derives from SYSTEM.WEB.HTTP.APICO NTROLLER class.	ASP.NET MVC controller derives from SYSTEM.WEB.MVC.CON TROLLER class.
2. Web API controller does not give view support.	ASP.NET MVC gives view support.

❖ Web API Controller

```
[ApiController]  
[Route("[controller]")]  
public class WeatherForecastController : ControllerBase  
{  
    ...  
}
```

```
[Route("[controller]")]  
public class WeatherForecastController : ApiController  
{  
    ...  
}
```

❖ MVC Controller

```
public class HomeController : Controller  
{  
    ...  
}
```

```
... public abstract class Controller : ControllerBase, IActionFilter, IFilterMetadata,  
{  
    ...  
}
```

Chapter 22 : Web API – Authentication & JWT

Q190. What are the **types of authentication** techniques in web api?

Q191. What is **Basic Authentication** in Web API?

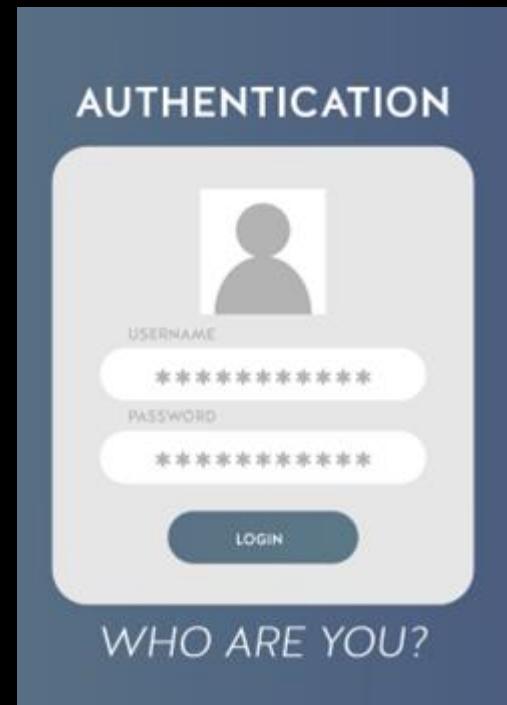
Q192. What is **API Key Authentication** in Web API?

Q193. What is **Token** based authentication? V Imp

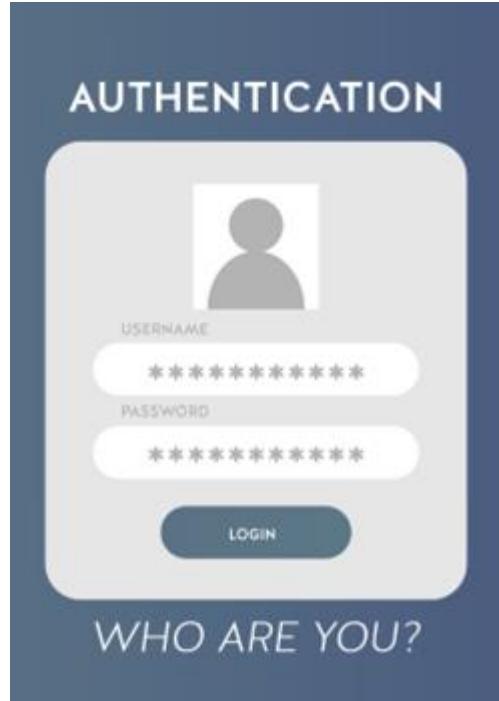
Q194. What is **JWT** Authentication? V Imp

Q195. What are the **parts of JWT** token?

Q196. Where JWT token reside in the request?



- ❖ Authentication is the process of verifying the identity of a user by validating their credentials such as username and password.



Types of Authentication techniques

Basic Authentication

API Key Authentication

Token-based Authentication

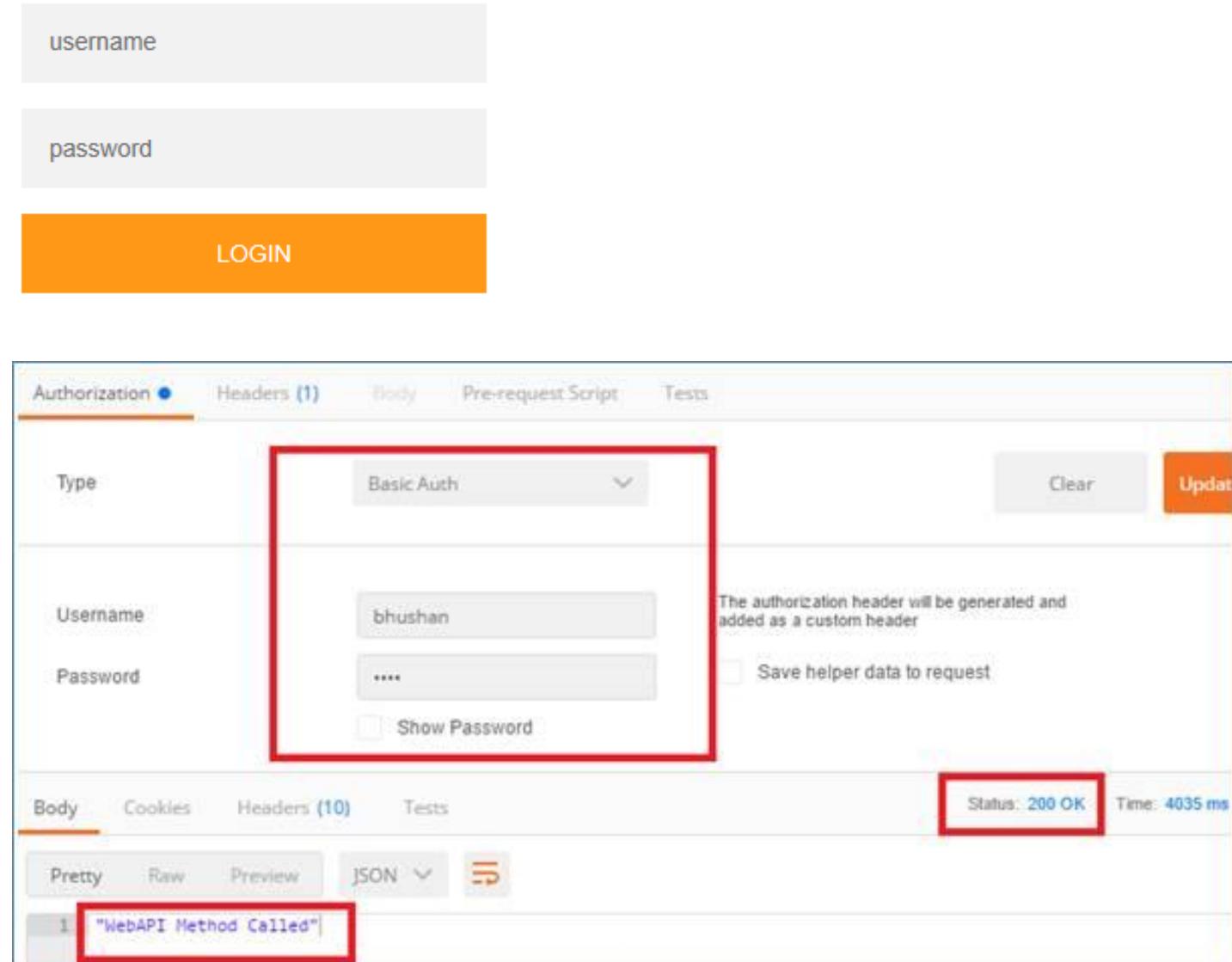
Certificate-based Authentication

Windows Authentication

Multi-Factor Authentication

- ❖ In Basic Authentication, the user passes their credentials on a post request. At the WebAPI end, credentials are verified, and response is sent back.

- ❖ The disadvantage of it is, Basic Authentication sends credentials in plain text over the network, so it is not considered a secure method of authentication.



The screenshot shows a Postman interface for a basic auth request. The 'Authorization' tab is selected, displaying a dropdown set to 'Basic Auth'. Below it, the 'Username' field contains 'bhushan' and the 'Password' field contains '....'. A red box highlights this entire section. To the right, a note says 'The authorization header will be generated and added as a custom header' and there's a checked 'Save helper data to request' checkbox. The 'Body' tab is also visible at the bottom. The response status is shown as 'Status: 200 OK' with a time of '4035 ms'. A red box highlights the status code. The response body, which contains the string 'WebAPI Method Called', is also highlighted with a red box.

[back to chapter index](#)

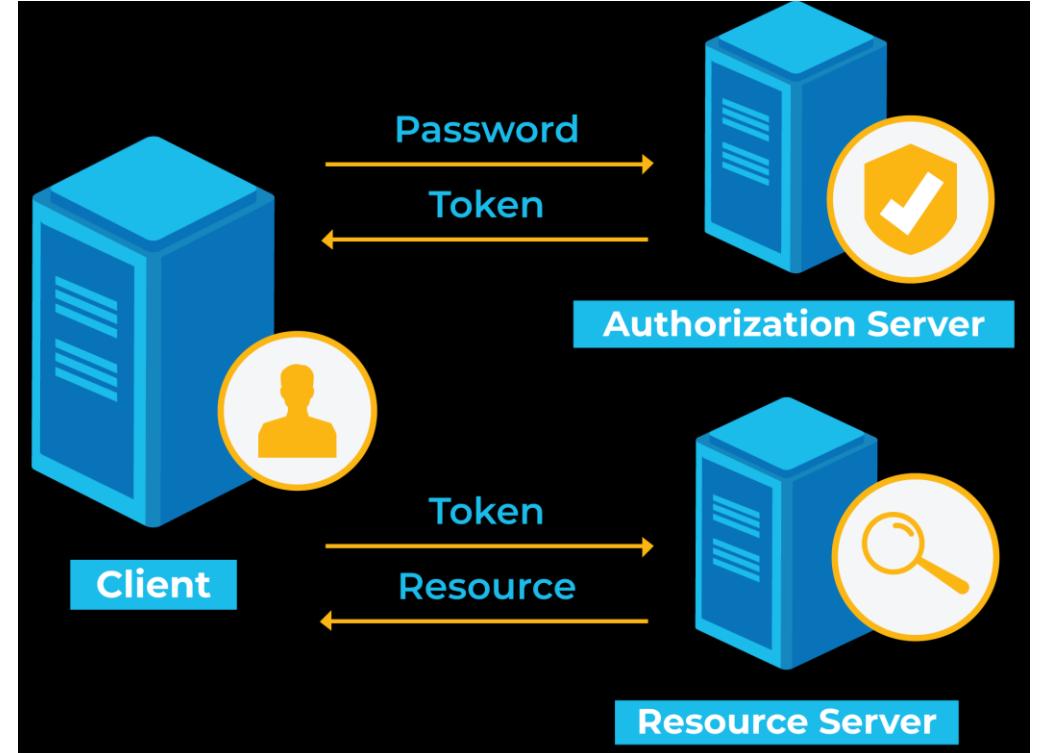
- ❖ **API Key Authentication** - In API Key Authentication, the API owner will share an API key with the users and this key will authenticate the users of that API.
- ❖ The disadvantage of it is, API keys can be shared or stolen therefore it may not be suitable for all scenarios.

The screenshot shows the Postman application interface. At the top, there is a header bar with 'GET' selected, the URL 'https://localhost:44395/weatherforecast', and buttons for 'Send', 'Save', and more. Below the header are tabs for 'Params', 'Authorization', 'Headers (7)', 'Body', 'Pre-request Script', 'Tests', and 'Settings'. The 'Headers' tab is active, showing a table with one row highlighted by a red box. This row contains the key 'ApiKey' and its value 'hL4bA4nB4yI0vI0fC8fH7eT6'. The 'Body' tab is also visible at the bottom, showing a JSON response:

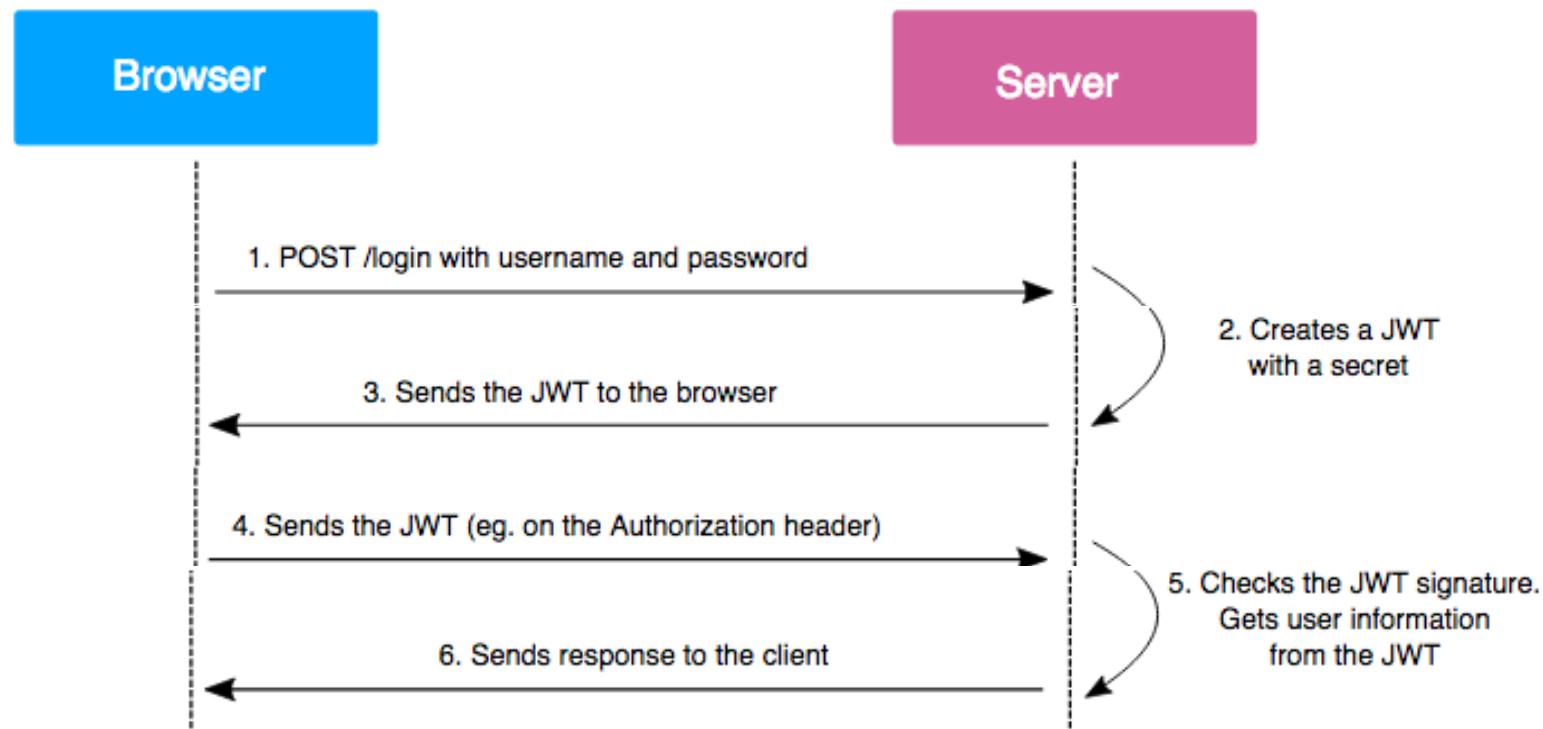
```
1 [  
2   {  
3     "date": "2020-09-11T01:16:58.032899+03:00",  
4     "temperatureC": 47,  
5     "temperatureF": 116,  
6     "summary": "Warm"  
7   },  
8 ]
```

The status bar at the bottom indicates 'Status: 200 OK', 'Time: 63 ms', and 'Size: 684 B'.

- ❖ Token-based authentication is a 4 step process:
 1. Client application first sends a request to Authentication server with valid **credentials**.
 2. The Authentication server/ Web API sends an **Access token** to the client as a response.
 3. In next request, the client uses the same token to access the restricted resources until the token is **valid** or not expired.
 4. If the Access token is expired, then the client application can request for a new access token by using **Refresh token**.



- ❖ JWT authentication is a token base authentication where JWT is a token format.
- ❖ JWT stands for JSON Web Token.



- ❖ JWT token has 3 parts:
 1. Header
 2. Payload
 3. Signature

The image shows a screenshot of the jwt.io website. A JWT token is displayed, split into an 'Encoded' section on the left and a 'Decoded' section on the right. The 'Encoded' section contains the base64-encoded string: eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6Ikpvag4gRG9lIiwiYWRtaW4iOnRydWV9.TJVA95OrM7E2cBab30RMHrHDcEfjoYZgeFONFh7HgQ. The 'Decoded' section shows the JSON structure:
HEADER:

```
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```


PAYLOAD:

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "admin": true  
}
```


VERIFY SIGNATURE

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  secret  
) secret base64 encoded
```

A red box labeled 'JWT Token' points to the encoded string. Four red arrows point from the 'Decoded' section to four separate callout boxes, each containing a description of one of the JWT parts:

- Algorithm used to generate the token.
- The type of the token, which is JWT here.
- The payload contains the CLAIMS.
- The signature is a string that is used to ensure the INTEGRITY of the token and verify that it has not been tampered with.

Signature Verified

❖ In REQUEST HEADER.

KEY - Authorization
VALUE - Token

The screenshot shows the Postman interface with a red box highlighting the 'Authorization' header in the list. The 'Headers' tab is selected, and the 'Value' column for the 'Authorization' header contains a JWT token.

KEY	VALUE	DESCRIPTION
Authorization	Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJpc3MiOiJodHRwOi8vZGV2ZWxvc2Uub3JnLmNvbS9hcmFzL2FjY291bnRzL2RhdGUiLCJhdWQiOiJodHRwOi8vZGV2ZWxvc2Uub3JnLmNvbS9hcmFzL2FjY291bnRzL2RhdGUiLCJleHAiOjE2MjUyOTUwNjMsImV4cCI6MTk4NTIwODA2M30.5f3d4... [REDACTED]	
Cookie	AWSALB=+GxIFUIZ9u1N9MX49/1cntFkXyB9O1xiN(...	
Postman-Token	<calculated when request is sent>	
Content-Type	text/plain	
Content-Length	<calculated when request is sent>	
Host	<calculated when request is sent>	
User-Agent	PostmanRuntime/7.24.1	
Accept	*/*	
Accept-Encoding	gzip, deflate, br	
Connection	keep-alive	
Content-Type	application/vnd.vmware.horizon.manager.connect	

Chapter 23 : Web API - More

Q197. How to test Web API? What are the tools?

Q198. What are main **Return Types** supported in Web API? V Imp

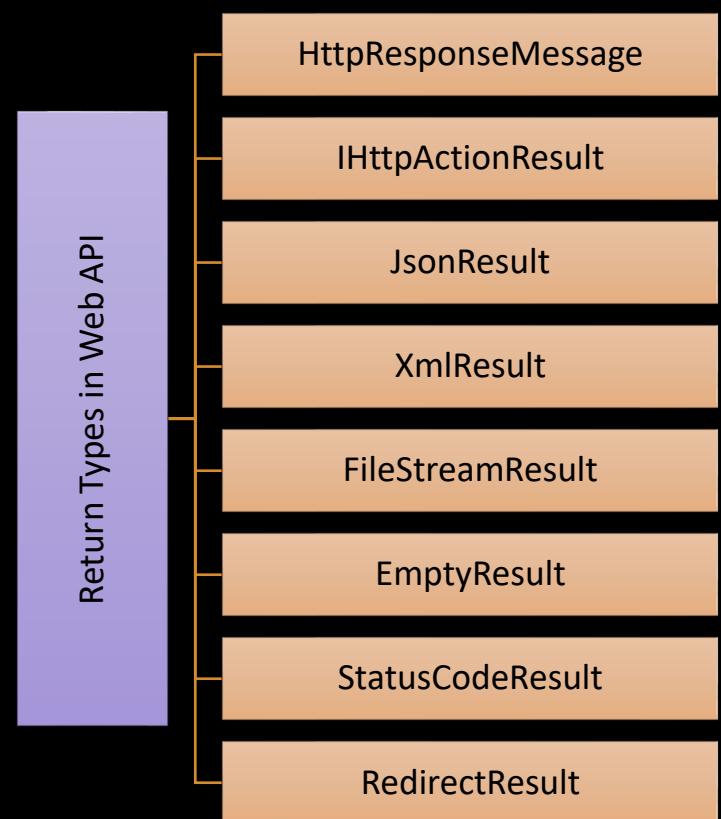
Q199. What is the difference between **HttpResponseMessage** and **IHttpActionResult**?

Q200. What is the difference between **IActionResult** and **IHttpActionResult**?

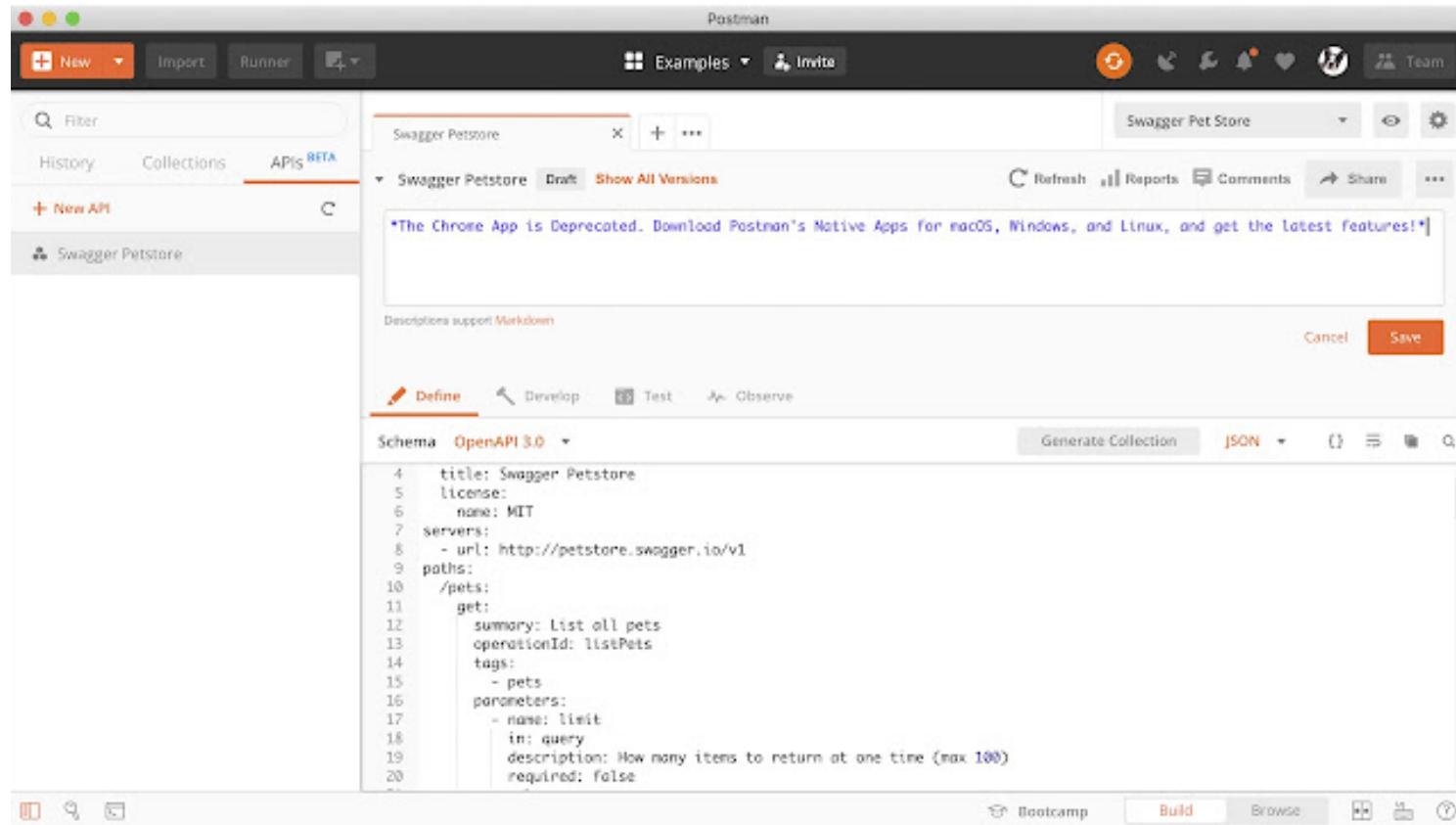
Q201. What is **Content Negotiation** in Web API? V Imp

Q202. What is **MediaTypeFormatter** class in Web API?

Q203. What are **Response Codes** in Web API?



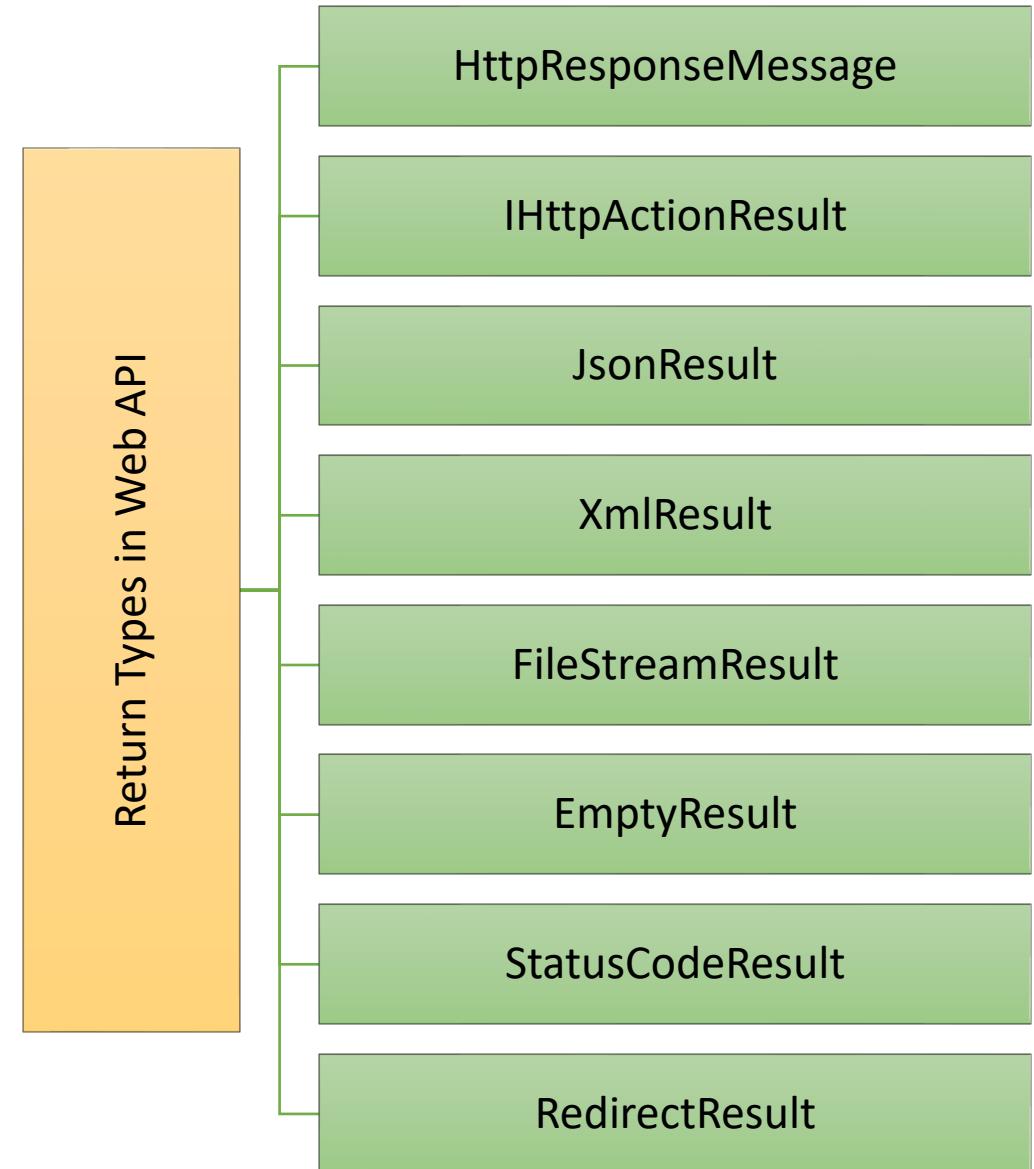
❖ Postman, Swagger and Fiddler tools



1. HttpResponseMessage: This class allows developers to create and return HTTP responses with a customized status code, headers, and content.

```
public HttpResponseMessage Get(int id)
{
    // create an HTTP response message with a status code
    // of 200 OK and the resource as the content
    var response = new HttpResponseMessage(HttpStatusCode.OK);
    response.Content = new StringContent("Resource found.");

    return response;
}
```

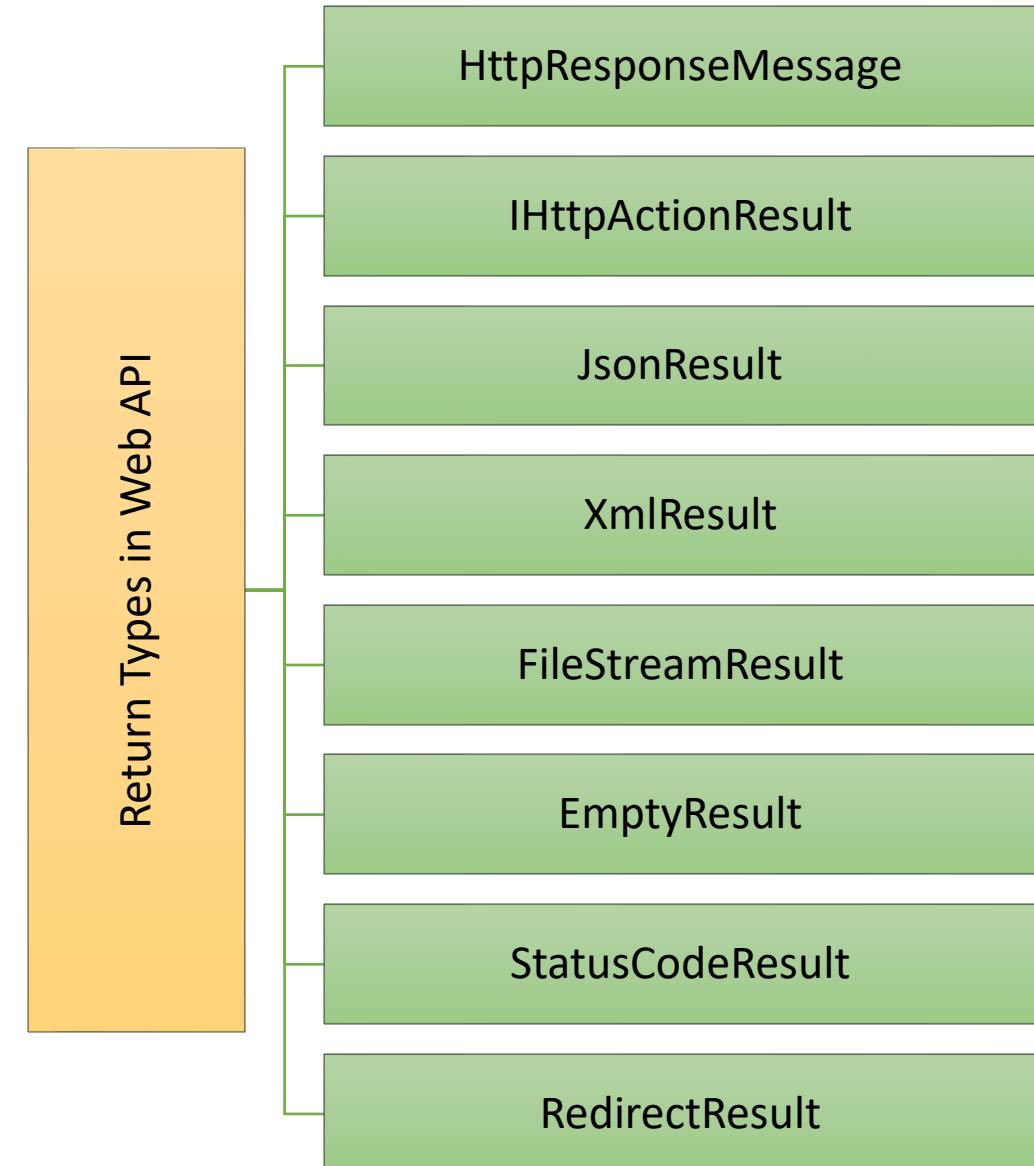


2. `IHttpActionResult`: This interface provides a level of abstraction between the controller and the HTTP response, allowing developers to return a variety of response types that implement the interface.

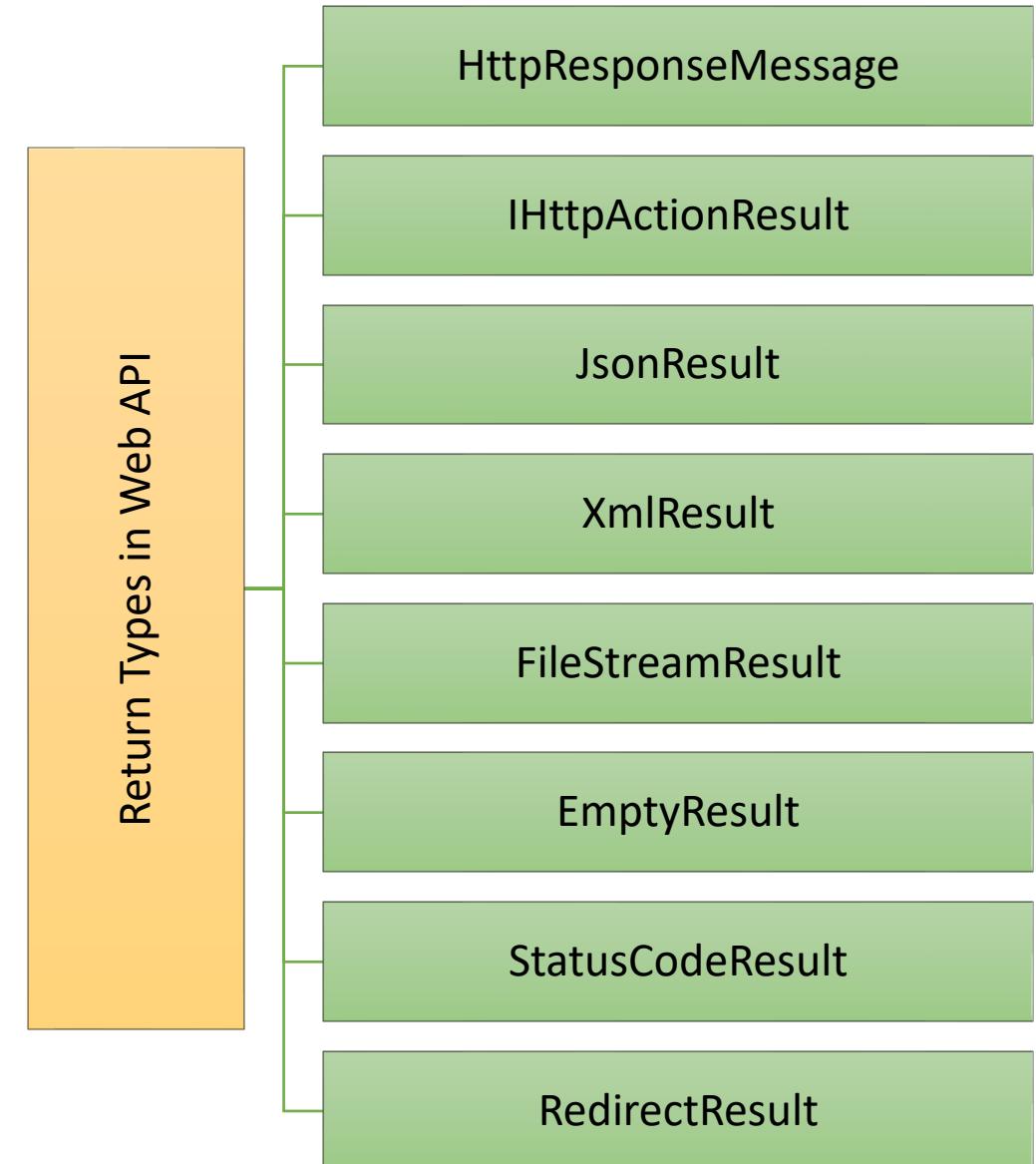
```
public IHttpActionResult Get(int id)
{
    // return an HTTP response with a status code
    // of 200 OK and the resource as the content
    return Ok("Resource found.");
}
```

3. `JsonResult`: This class is used to serialize an object to JSON format and return it as an HTTP response message.

```
public JsonResult Get(int id)
{
    // return the resource as JSON
    return Json(new { message = "Resource found." });
}
```



4. **XmlResult:** This class is used to serialize an object to XML format and return it as an HTTP response message.
5. **FileStreamResult:** This class is used to return a file as an HTTP response message by reading the file as a stream.
6. **EmptyResult:** This class is used to return an empty HTTP response message with a specified status code.
7. **StatusResult:** This class is used to return an HTTP response message with a specified status code and no content.
8. **RedirectResult:** This class is used to redirect the client to a different URL by returning an HTTP response message with a redirect status code.



- ❖ In web API version 1.0, We have a response type class called **HttpResponseMessage** for returning Http response message from API.
- ❖ In Web API version 2.0 **IHttpActionResult/ IActionResult** is introduced which is basically the replacement of HttpResponseMessage.

```
[HttpGet]
public HttpResponseMessage Get(int id)
{
    var entity = _repository.Get(id);
    if (entity == null)
    {
        return new HttpResponseMessage(HttpStatusCode.NotFound)
    {
        Content = new StringContent("Entity not found")
    };
    }
    else
    {
        var response = new HttpResponseMessage(HttpStatusCode.OK);
        response.Content = new ObjectContent<Entity>(entity, new
            JsonMediaTypeFormatter());
        return response;
    }
}
```



```
[HttpGet]
public IHttpActionResult Get(int id)
{
    var entity = _repository.Get(id);
    if (entity == null)
    {
        return NotFound();
    }
    else
    {
        return Ok(entity);
    }
}
```

Benefits of IHttpActionResultResult:

1. **Readability:** The IHttpActionResultResult implementation is more readable as it uses simple NotFound() and Ok() methods.
2. **Testability:** The IHttpActionResultResult implementation is more testable, as it allows for easier mocking of the return values.
3. **Flexibility:** The IHttpActionResultResult implementation is more flexible, as it allows for easy customization of the HTTP response without needing to construct a HttpResponseMessage object manually.

- ❖ In Web API version 2.0 **IHttpActionResult/ IActionResult** is introduced which is basically the replacement of HttpResponseMessage.

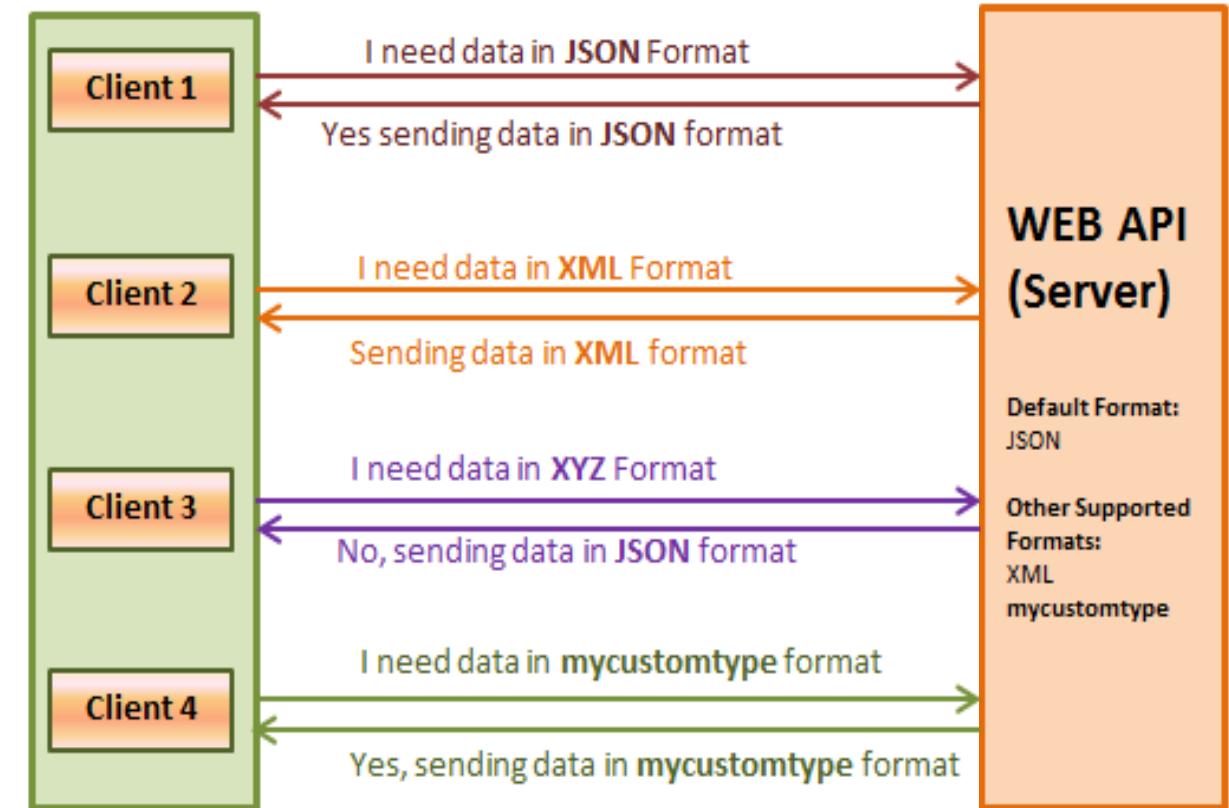
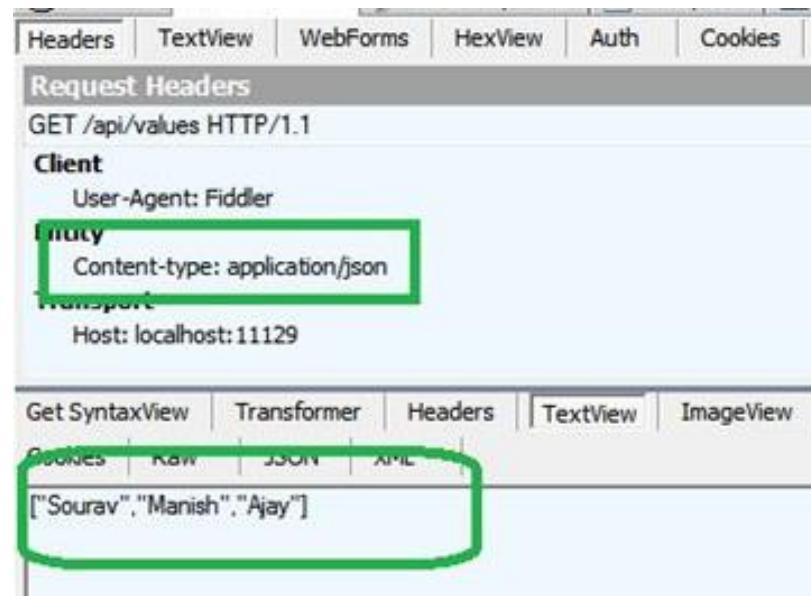
```
[HttpGet]
public IHttpActionResult Get(int id)
{
    var entity = _repository.Get(id);
    if (entity == null)
    {
        return NotFound();
    }
    else
    {
        return Ok(entity);
    }
}
```

What is the difference between IActionResult and IHttpActionResult?

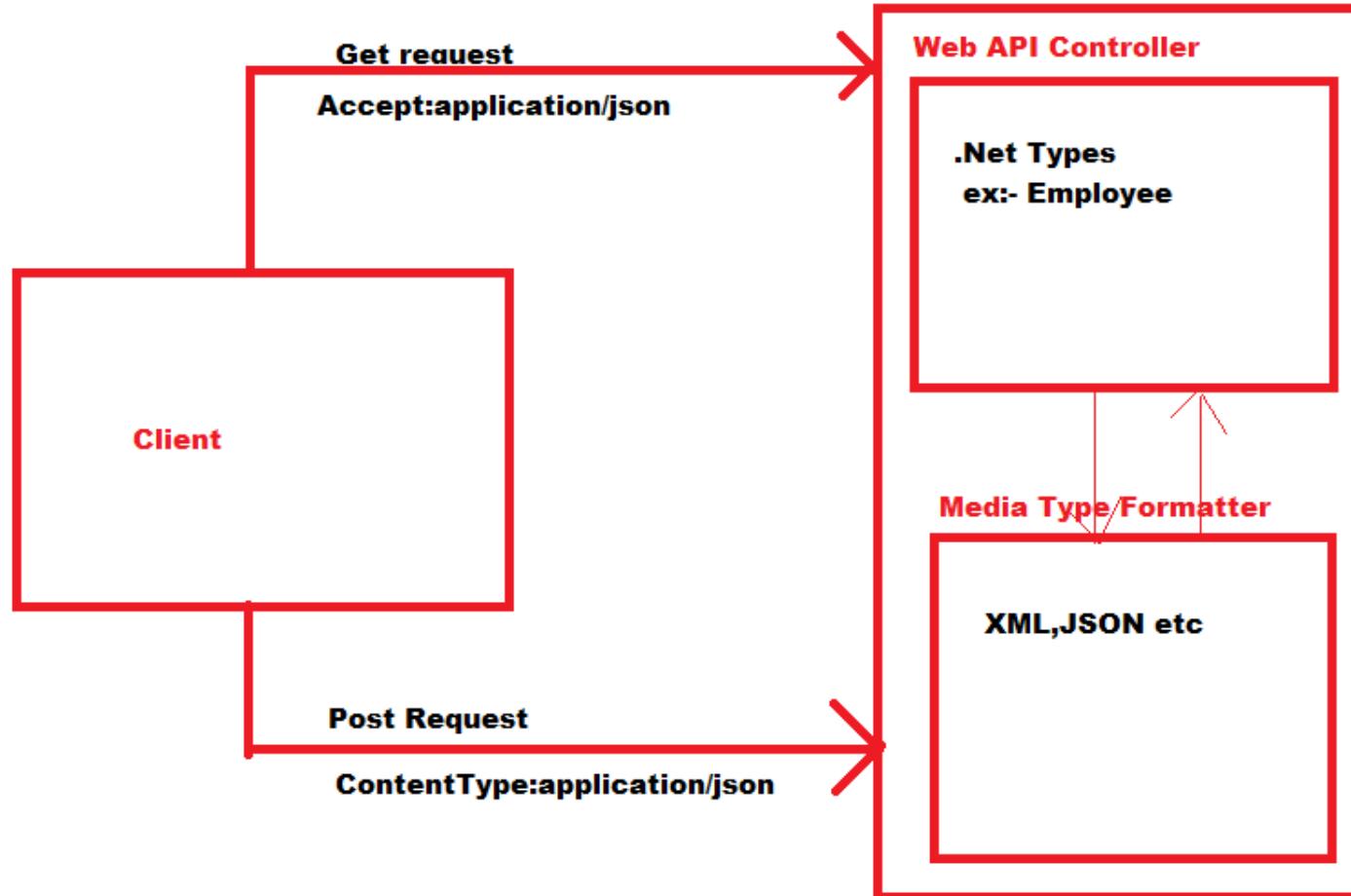


- ❖ IHttpActionResult is used in ASP.NET Web API,
while IActionResult is used in ASP.NET Core.

- Content negotiation is a way for the client and server to agree on the format(json/ xml) of the data being exchanged.



- Content negotiation can be implemented with the help of MediaTypeFormatter class.



- ❖ MediaTypeFormatter is a class in ASP.NET Web API that handles the conversion of objects into serialized formats(JSON, XML, and other media) that can be transmitted over the web.

```
...public class JsonMediaTypeFormatter : BaseJsonMediaTypeFormatter
```

```
public abstract class BaseJsonMediaTypeFormatter : MediaTypeFormatter
```

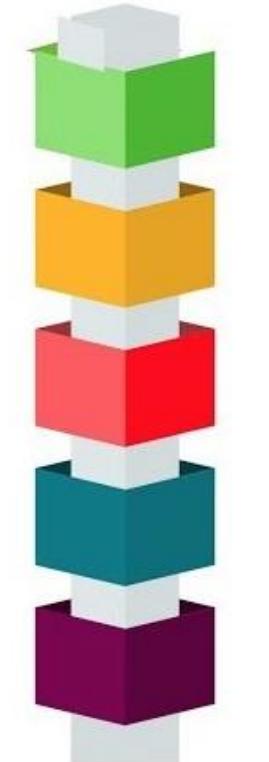
```
// set the request headers to accept XML format
client.DefaultRequestHeaders.Accept.Clear();

client.DefaultRequestHeaders.Accept.Add
(new MediaTypeWithQualityHeaderValue("application/xml"));
```

1. **1xx: Informational** – Communicates transfer protocol-level information.
2. **2xx: Success** – Indicates that the client's request was accepted successfully.
3. **3xx: Redirection** – This means request is not complete. The client must take some additional action in order to complete their request.
4. **4xx: Client Error** – This means there is some error in API code.
5. **5xx: Server Error** – This means the error is not due to web api code but due to some environment settings.



- ❖ 200 OK: The request was successful, and the server has returned the requested data in the response body.
- ❖ 204 No Content: The request was successful, but the server has no data to return in the response body.
- ❖ 400 Bad Request: The client's request was invalid or could not be understood by the server.
- ❖ 401 Unauthorized: The client is not authorized to access the requested resource. The client should include authentication credentials in the request header to authenticate itself.
- ❖ 403 Forbidden: The client is authenticated, but not authorized to access the requested resource.
- ❖ 404 Not Found: The requested resource could not be found on the server.
- ❖ 500 Internal Server Error: An error occurred on the server while processing the request.



1XX	INFORMATIONAL
2XX	SUCCESS
3XX	REDIRECTION
4XX	CLIENT ERROR
5XX	SERVER ERROR

Chapter 24 : .NET Core - Basics

Q204. What is .NET Core?

Q205. What is .NET Standard?

Q206. What are the advantages of .NET Core over .NET framework? V Imp

Q207. What is the role of Program.cs file in ASP.NET Core?

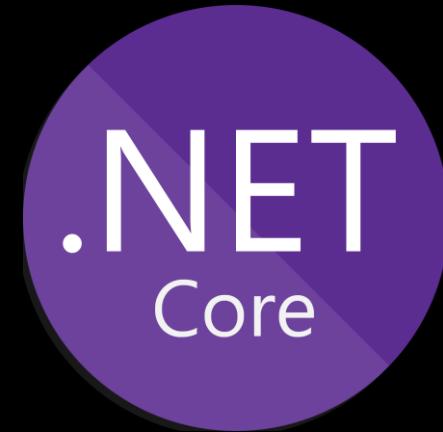
Q208. What is the role of ConfigureServices method?

Q209. What is the role of Configure method?

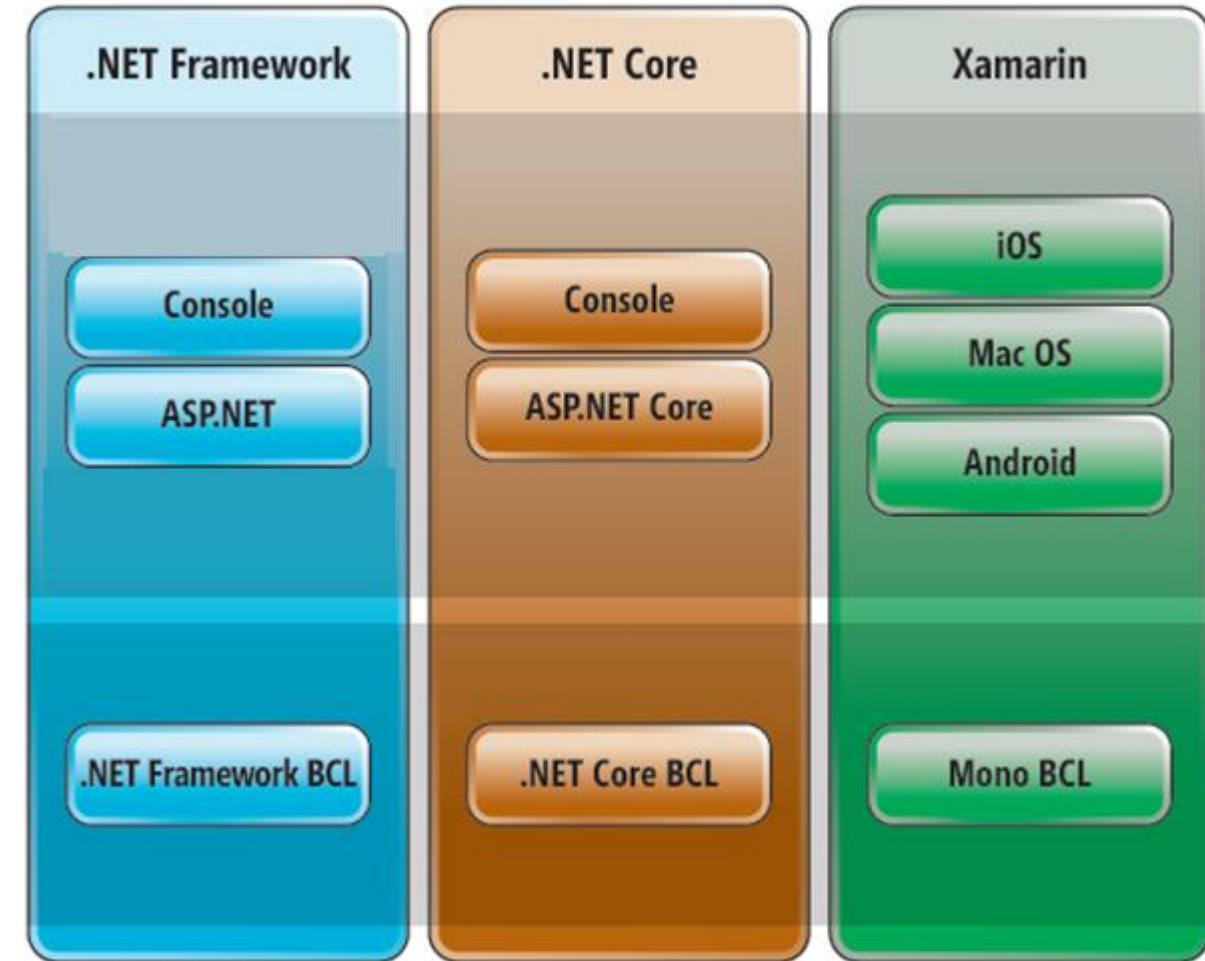
Q210. Describe the complete Request Processing Pipeline for ASP.NET Core MVC?

Q211. What is the difference between .NET Core and .NET 5?

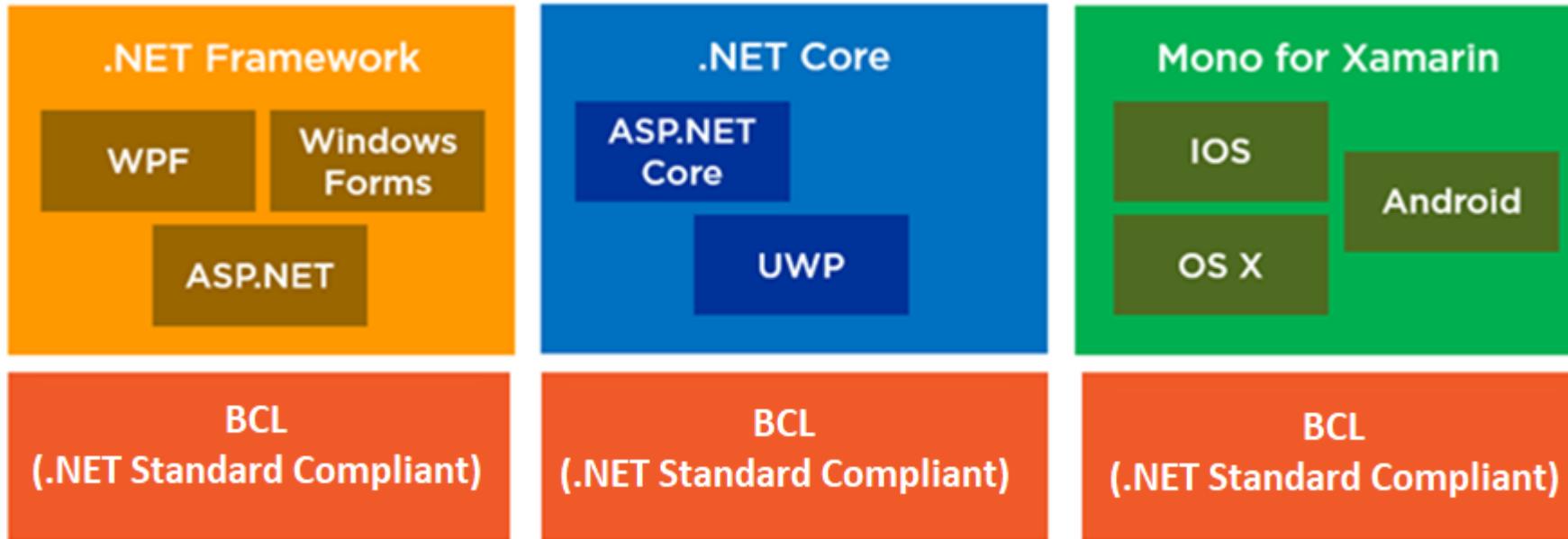
Q212. What is Metapackage? What is the name of Metapackage provided by ASP.NET Core?



- ❖ .NET Core is completely a **NEW** framework.
- ❖ It is **FREE** and **OPEN-SOURCE** platform.
- ❖ It is developed by Microsoft.



- ❖ .NET Standard is a set of rules that is common across all .NET frameworks.



CROSS PLATFORM

- Windows
- Linux
- MacOS

.NET Framework only supports Windows

OPEN SOURCE

- Free to use
- Modify
- Distribute

.NET Framework is paid

HOSTING

- Kestrel
- IIS
- Nginx

.NET Framework only support IIS Hosting

BUILT-IN DEPENDENCY INJECTION

- Loosely Coupled Design
- Reusability
- Testability

.NET framework don't have built in dependency injection

SUPPORT MULTIPLE IDE

- Visual Studio
- Visual Studio for Mac
- Visual Studio Code

.NET framework only support Visual Studio IDE

- ❖ Program.cs file contains the application startup code.
- ❖ The main method of Program.cs file is the entry point of application.

```
0 references
public class Program
{
    0 references
    public static void Main(string[] args)
    {
        CreateHostBuilder(args).Build().Run();
    }

    1 reference
    public static IHostBuilder CreateHostBuilder(string[] args) =>
        Host.CreateDefaultBuilder(args)
            .ConfigureWebHostDefaults(webBuilder =>
            {
                webBuilder.UseStartup<Startup>();
            });
}
```

```
2 references
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime.
    // Use this method to add services to the container.
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
    }

    // This method gets called by the runtime.
    // Use this method to configure the HTTP request pipeline.
    0 references
    public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
    {
        if (env.IsDevelopment())
        {
            app.UseDeveloperExceptionPage();
        }
        else
        {
            app.UseExceptionHandler("/Home/Error");
            // The default HSTS value is 30 days.
            // You may want to change this for production scenarios, see https://aka.ms/aspnet/https
            app.UseHsts();
        }
        app.UseHttpsRedirection();
        app.UseStaticFiles();

        app.UseRouting();

        app.UseAuthorization();
    }
}
```

- ❖ From .NET 6.0, the startup class code is merged in Program.cs file and therefore there is no separate Startup.cs class.

```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

        // Add services to the container.
        builder.Services.AddControllers();
        builder.Services.AddSwaggerGen();

        var app = builder.Build();

        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseAuthorization();
        app.MapControllers();
        app.Run();
    }
}
```

1. ConfigureServices is used to add services to the application.
2. ConfigureServices method always execute before Configure method.
3. ConfigureServices is an optional method. It's not necessary that all its methods will be execute.



```
2 references
public class Startup
{
    0 references
    public Startup(IConfiguration configuration)
    {
        Configuration = configuration;
    }

    1 reference
    public IConfiguration Configuration { get; }

    // This method gets called by the runtime.
    // Use this method to add services to the container.
    0 references
    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllersWithViews();
    }
}
```

```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

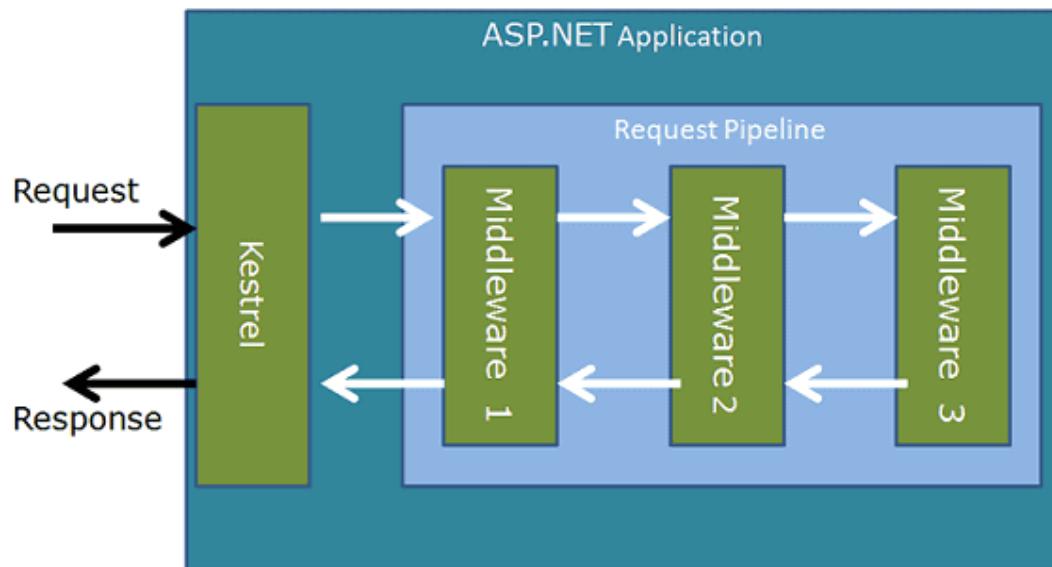
        // Add services to the container.
        builder.Services.AddControllers();
        builder.Services.AddSwaggerGen();
    }
}
```

[back to chapter index](#)

What is the role of Configure method?

1. Configure method will configure the request pipeline.
2. Configure method executes after ConfigureServices method.
3. For each and every request, all the methods will execute inside the Configure method and in the same sequence.

Middleware in ASP.NET Request Pipeline



```

// This method gets called by the runtime.
// Use this method to configure the HTTP request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
        // The default HSTS value is 30 days.
        // You may want to change this for production scenarios, see https://go.microsoft.com/fwlink/?LinkID=205236
        app.UseHsts();
    }
    app.UseHttpsRedirection();
    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();
}

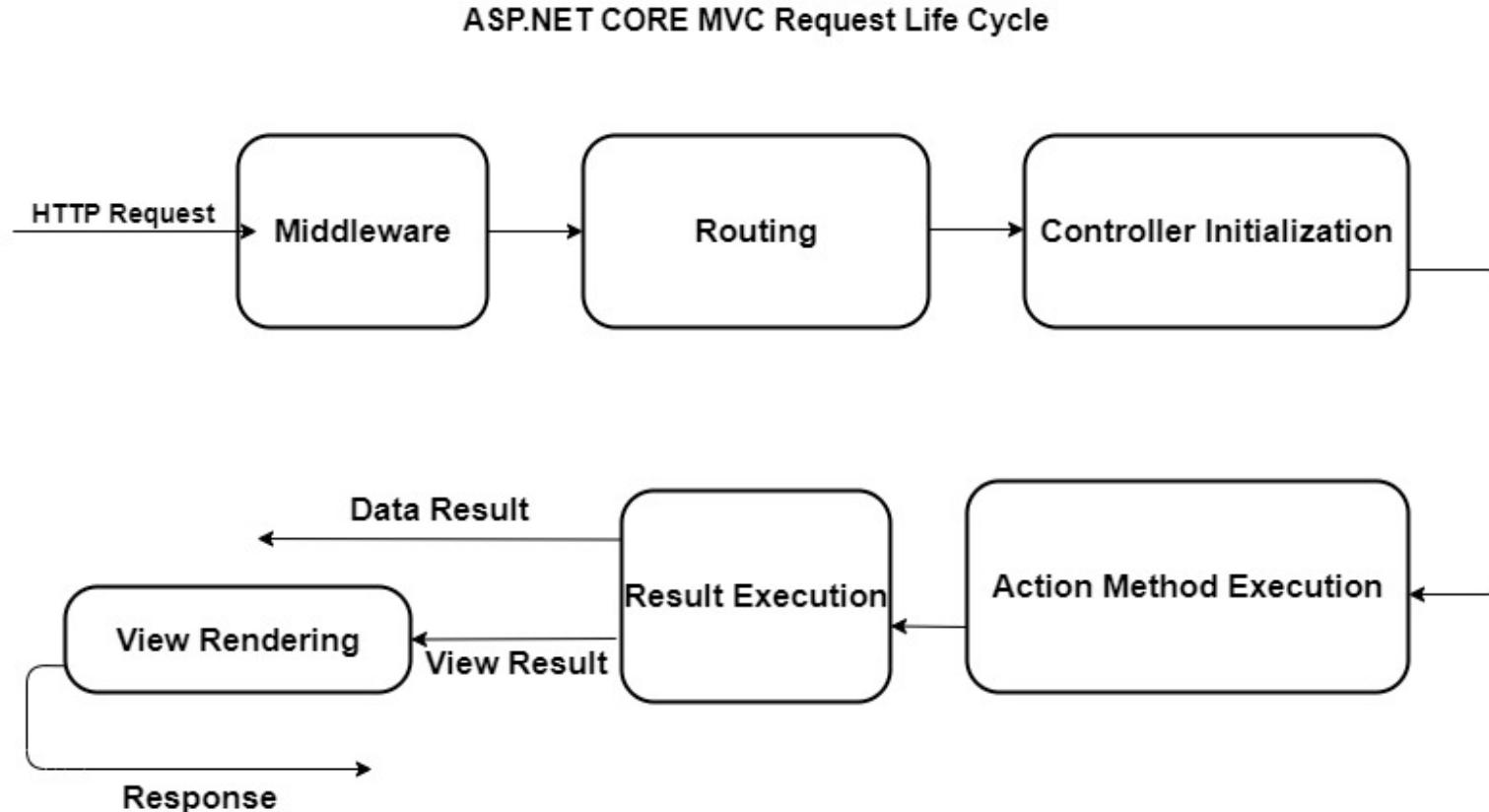
```

```

// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())
{
    app.UseSwagger();
    app.UseSwaggerUI();
}

app.UseAuthorization();

```



<http://localhost:1234/home/about>

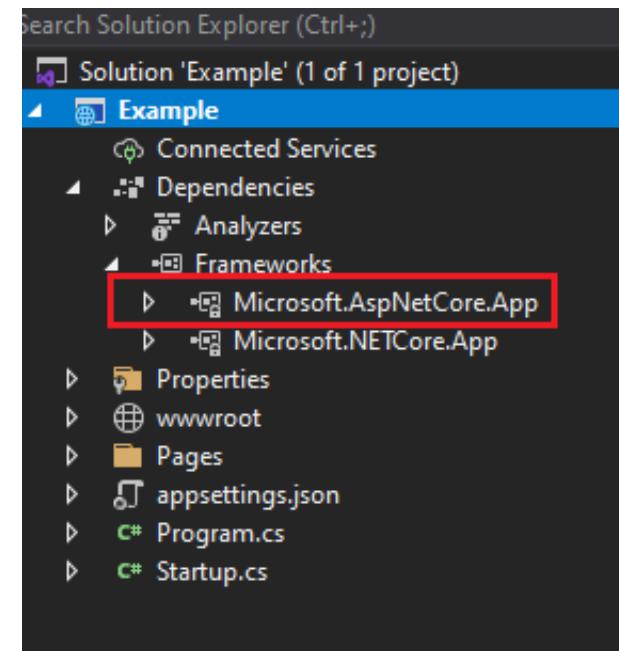
What is the difference between .NET Core and .NET 5?



1. .NET 5 is the next major release after .NET Core 3.1.
2. The word 'Core' is dropped from the name to emphasize that .NET 5 is the future of all earlier versions of .NET Core.
3. Recently, Microsoft has introduced .NET 6.

- ❖ Metapackage is a consolidated package of many dependencies.

```
dependencies:  
  Microsoft.AspNetCore.Diagnostics  
  Microsoft.AspNetCore.Hosting  
  Microsoft.AspNetCore.Routing  
  Microsoft.AspNetCore.Server.IISIntegration  
  Microsoft.AspNetCore.Server.Kestrel  
  Microsoft.Extensions.Configuration.EnvironmentVariables  
  Microsoft.Extensions.Configuration.FileExtensions  
  Microsoft.Extensions.Configuration.Json  
  Microsoft.Extensions.Logging  
  Microsoft.Extensions.Logging.Console  
  Microsoft.Extensions.Options.ConfigurationExtensions  
  NETStandard.Library
```



Chapter 25 : .NET Core - Dependency Injection

Q213. What is **Dependency Injection**? V Imp

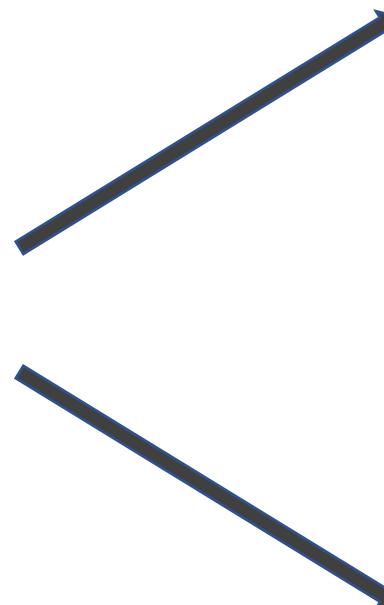
Q214. How to **implement** Dependency injection in .NET Core? V Imp

Q215. What are the **advantages** of Dependency Injection in .NET Core? V Imp

Q216. How to **use** Dependency Injection in Views in ASP.NET Core?

- ❖ Dependency Injection (DI) is a software **design pattern** in which we inject the dependency object of a class into another class.

```
public class Salary
{
    public int CalculateSalary()
    {
        return 1000000;
    }
}
```



```
public class Employee
{
    public void GetSalary()
    {
        Salary sal = new Salary();

        int salary = sal.CalculateSalary();
    }
}

public class Employee
{
    private readonly Salary _salary;

    public Employee(Salary salary)
    {
        _salary = salary;
    }

    public void GetSalary()
    {
        int salary = _salary.CalculateSalary();
    }
}
```

```
0 references
public class FirstController : Controller
{
    0 references
    public int Index()
    {
        //Create object of math class
        MathStudent cls = new MathStudent();
        return cls.GetStudentCount();
    }
}
```



```
4 references
public class MathStudent
{
    2 references
    public int GetStudentCount()
    {
        return 50;
    }
}
```

```
0 references
public class SecondController : Controller
{
    0 references
    public int Index()
    {
        //Create object of math class
        MathStudent cls = new MathStudent();
        return cls.GetStudentCount();
    }
}
```



Tight Coupling

```
1 reference
public class ScienceStudent
{
    0 references
    public int GetStudentCount()
    {
        return 100;
    }
}
```

100 Controllers
(Big/Enterprise Level Application)

[back to chapter index](#)

```
1 reference
public class FirstController : Controller
{
    private IStudent _student;
    0 references
    public FirstController(IStudent student)
    {
        _student = student;
    }
    0 references
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```

```
1 reference
public class SecondController : Controller
{
    private IStudent _student;
    0 references
    public SecondController(IStudent student)
    {
        _student = student;
    }
    0 references
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```



```
1 reference
public interface IStudent
{
    0 references
    public int GetStudentCount();
}
```

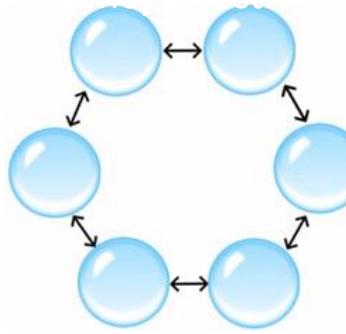
```
4 references
public class MathStudent : IStudent
{
    3 references
    public int GetStudentCount()
    {
        return 50;
    }
}
```

```
1 reference
public class ScienceStudent : IStudent
{
    1 reference
    public int GetStudentCount()
    {
        return 100;
    }
}
```

```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<IStudent, ScienceStudent>();
}
```



Tight Coupling



Loose Coupling

- 1. Flexibility:** DI allows you to easily change the behavior of an application without modifying its code. By injecting dependencies, you can easily switch between different implementations of a service/ class.

- 2. Easier unit testing:** DI makes it easy to unit test your code by allowing you to easily replace real dependencies with mock with the help of interfaces.

- 3. Independent modules:** By separating dependencies, it becomes easier to make changes to your code without affecting the rest of the system.

- 4. Reusability:** DI promotes reuse of components by making them independent of their environment. This allows you to reuse the same components in different applications without modification.

- ❖ A service can be injected into a view using the **@inject directive**.

```
public interface IStudent
{
    2 references
    public int GetStudentCount();
}
```

```
public class MathStudent:IStudent
{
    2 references
    public int GetStudentCount()
    {
        return 100;
    }
}
```

```
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();

    services.AddScoped<IStudent, MathStudent>();
}
```

```
@inject WebApplication1.IStudent _student

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    Total Student: @_student.GetStudentCount();
</div>
```

```
0 references
public class FirstController : Controller
{
    private IStudent _student;
    0 references
    public FirstController(IStudent student)
    {
        _student = student;
    }
    0 references
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
```

Chapter 26 : .NET Core - Service Lifetimes, Middleware & Hosting

Q217. What are the types of [Service Lifetimes](#) of an object in ASP.NET Core?

Q218. What is [AddSingleton](#), [AddScoped](#) and [AddTransient](#) method? V Imp

Q219. What is [Middleware](#) in ASP.NET Core? What is [custom middleware](#)? V Imp

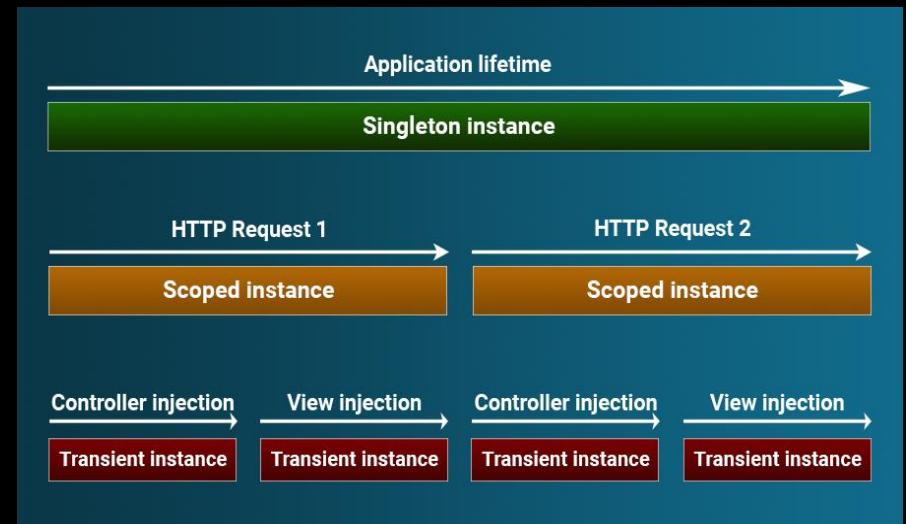
Q220. How [ASP.NET Core](#) [Middleware](#) is different from [HttpModule](#)?

Q221. What is [Request Delegate](#) in .NET Core?

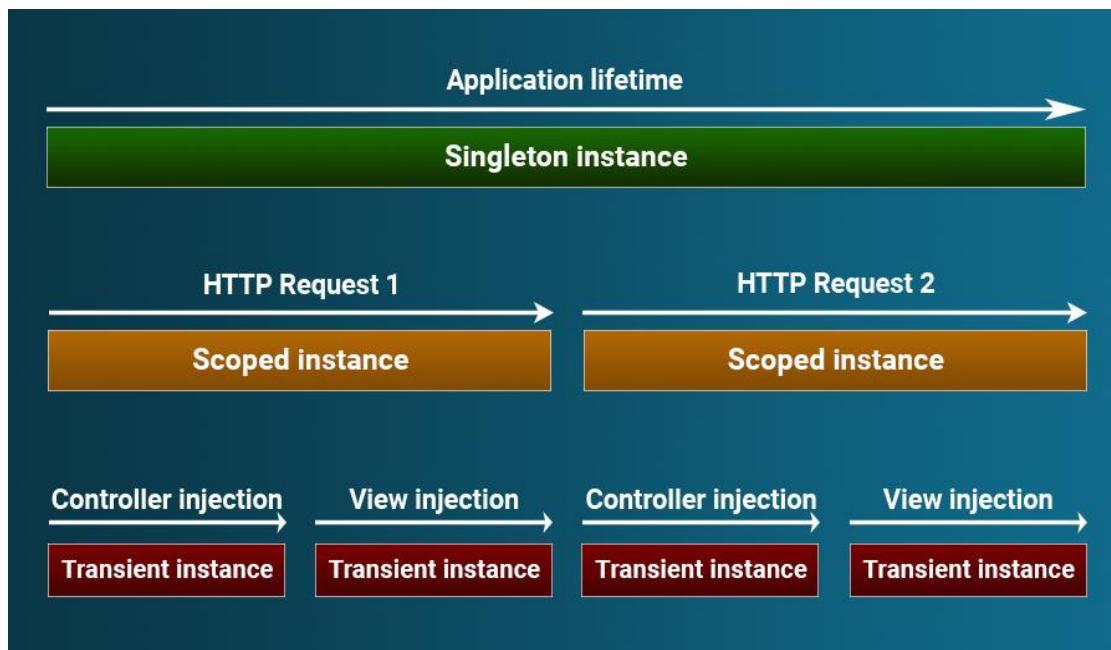
Q222. What is [Run\(\)](#), [Use\(\)](#) and [Map\(\)](#) method? V Imp

Q223. What are the [types of Hosting](#) in ASP.NET Core? What is In process and Out of process hosting?

Q224. What is [Kestrel](#)? What is the difference between Kestrel and IIS?



- ❖ The Service Lifetime refers to the lifespan of a registered service. Or
- ❖ The service lifetime controls how long a result object will live for after it has been created by the container.



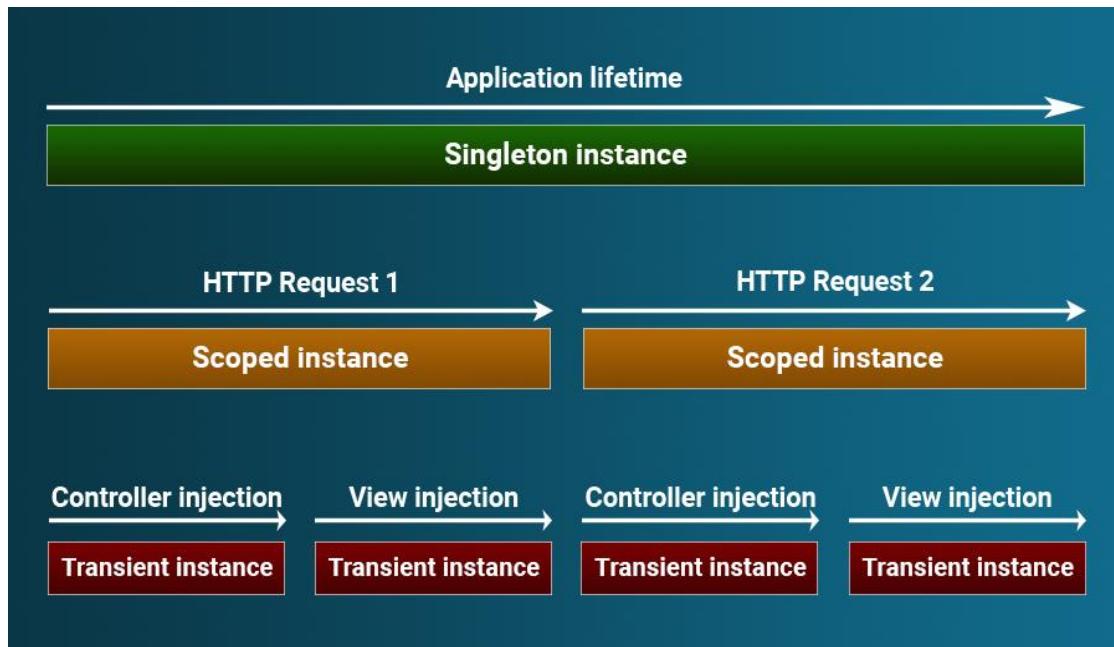
```

0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddMvc();
    services.AddSingleton<IStudent, ScienceStudent>();
}
  
```

```

1 reference
public class FirstController : Controller
{
    private IStudent _student;
    public FirstController(IStudent student)
    {
        _student = student;
    }
    public int Index()
    {
        return _student.GetStudentCount();
    }
}
  
```

- ❖ AddSingleton method create only **one instance** when the service is requested for first time. And then the same instance will be shared by all different http requests.

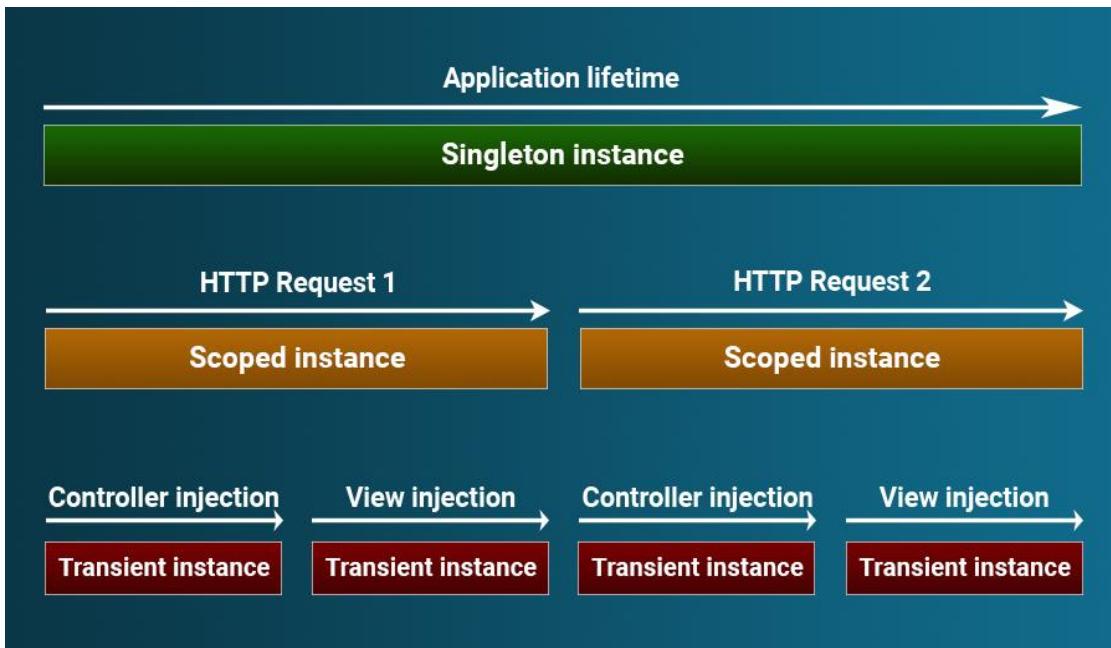


```
// This method gets called by the runtime.
// Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    services.AddSingleton<ILogger, Logger>();
    //services.AddSingleton<IPayment, Payment>();
    //services.AddTransient<IDataAccess, DataAccess>();

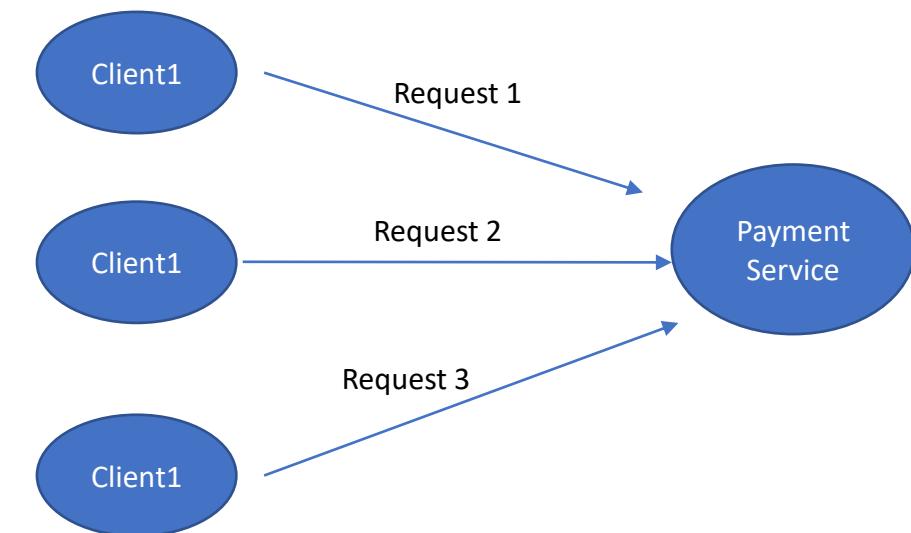
}
```

```
Jul 6 08:42:01 ip-10-17-12-120 rsyslogd: [origin software="rsyslog.com"] rsyslog was HUPed
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST f
Jul 6 12:24:33 ip-10-17-12-120 dhclient[2486]: bound to
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST f
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: bound to
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST f
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPACK f
Jul 7 08:44:47 ip-10-17-12-120 dhclient[2486]: bound to
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST f
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPACK f
Jul 7 19:41:23 ip-10-17-12-120 dhclient[2486]: bound to
Jul 8 02:47:00 ip-10-17-12-120 yum[31369]: Installed: p
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST f
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPACK f
Jul 8 07:06:13 ip-10-17-12-120 dhclient[2486]: bound to
```

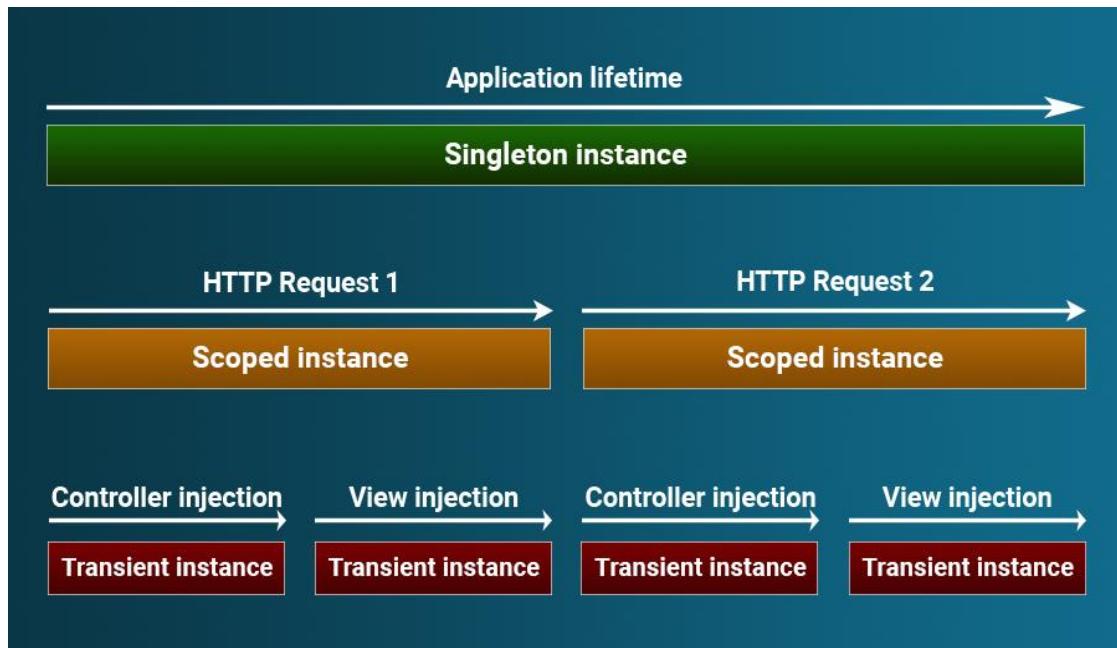
- ❖ AddScoped method create single instance per request. For every individual request there will be a single instance or object.



```
// This method gets called by the runtime.
// Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    //services.AddSingleton<ILogger, Logger>();
    services.AddScoped<IPayment, Payment>();
    //services.AddTransient<IDataAccess, DataAccess>();
}
```

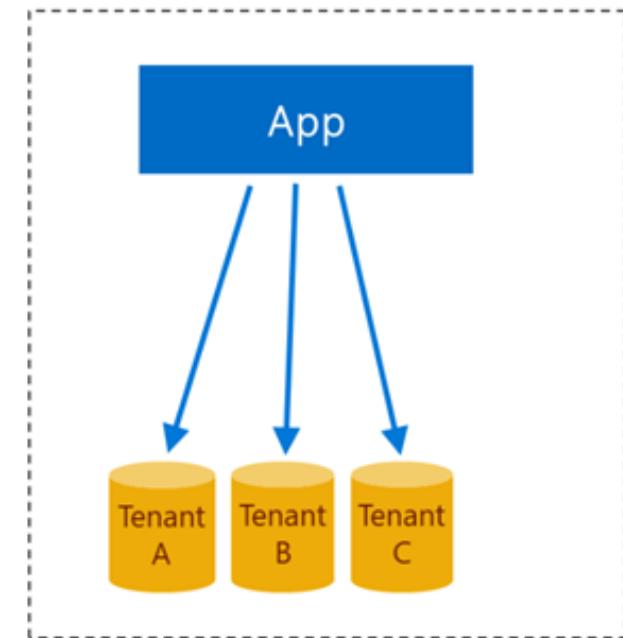


- ❖ AddTransient instance will not be shared at all, even with in the same request. Every time a new instance will be created.

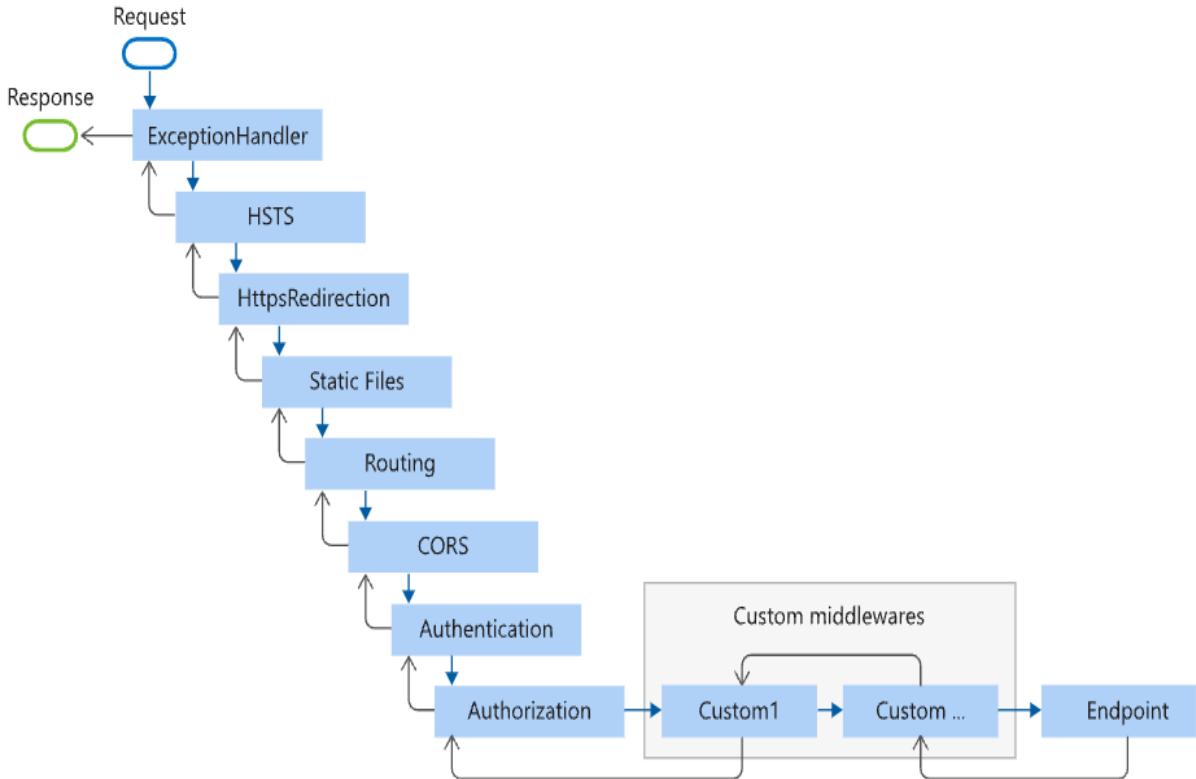


```
// This method gets called by the runtime.
// Use this method to add services to the container.
0 references
public void ConfigureServices(IServiceCollection services)
{
    //services.AddSingleton<ILogger, Logger>();
    //services.AddScoped<IPayment, Payment>();
    services.AddTransient<IDataAccess, DataAccess>();

}
```

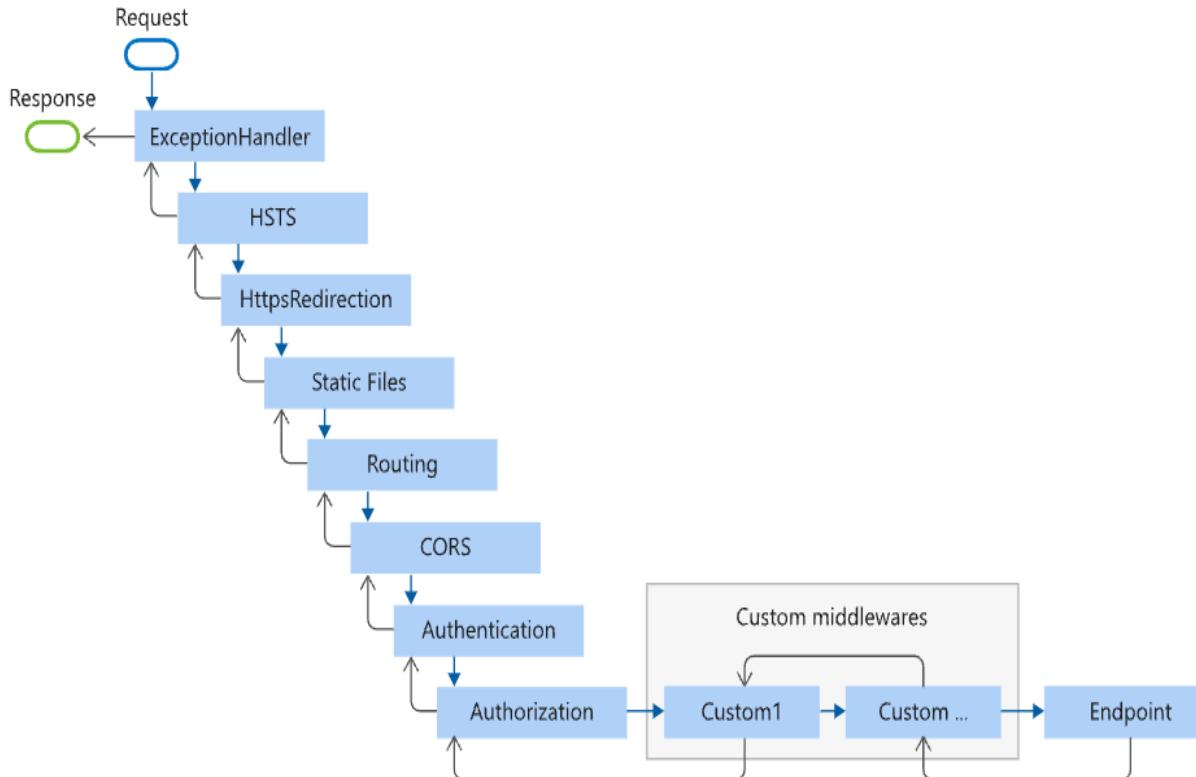


- ❖ A middleware is a component that is executed on every request in the ASP.NET Core application.



```
// This method gets called by the runtime.  
// Use this method to configure the HTTP request pipeline.  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/Home/Error");  
    }  
    app.UseStaticFiles();  
  
    app.UseRouting();  
  
    app.UseAuthorization();  
  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapControllerRoute(  
            name: "default",  
            pattern: "{controller=Home}/{action=Index}/{id?}");  
    });  
}
```

- ❖ A middleware a component that is executed on EVERY REQUEST in the ASP.NET Core application.
- ❖ We can set up the middleware in ASP.NET using the CONFIGURE method of our STARTUP class.



```
public class Program
{
    public static void Main(string[] args)
    {
        var builder = WebApplication.CreateBuilder(args);

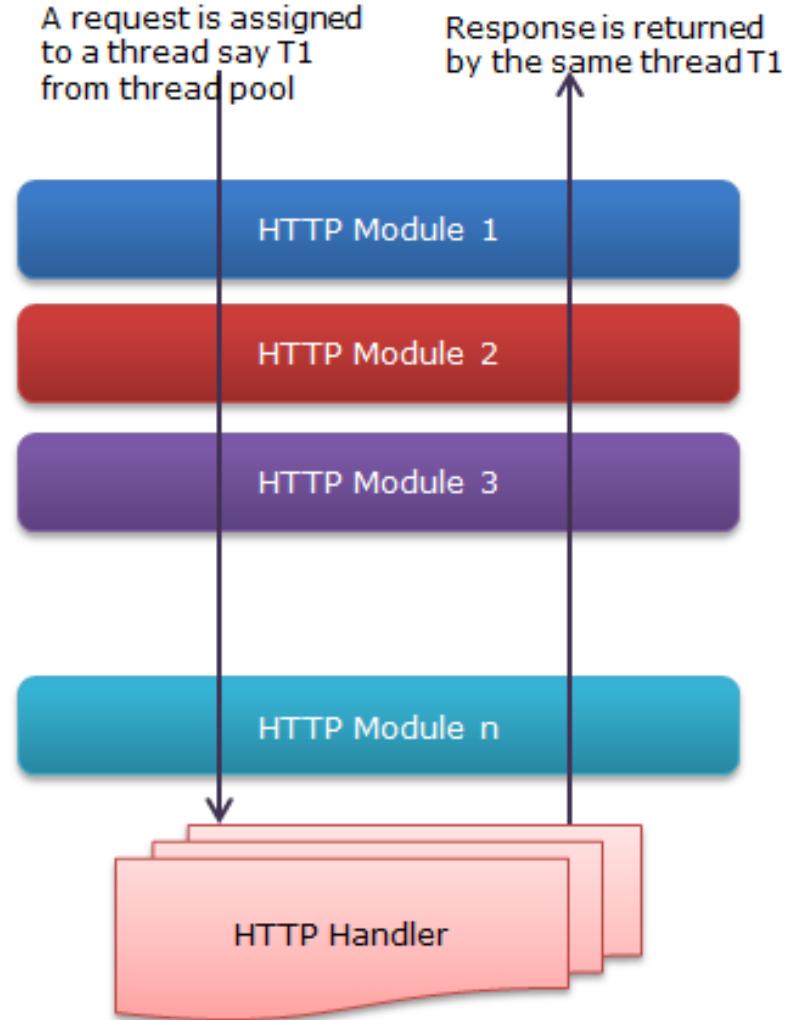
        // Add services to the container.
        builder.Services.AddControllers();
        builder.Services.AddSwaggerGen();

        var app = builder.Build();

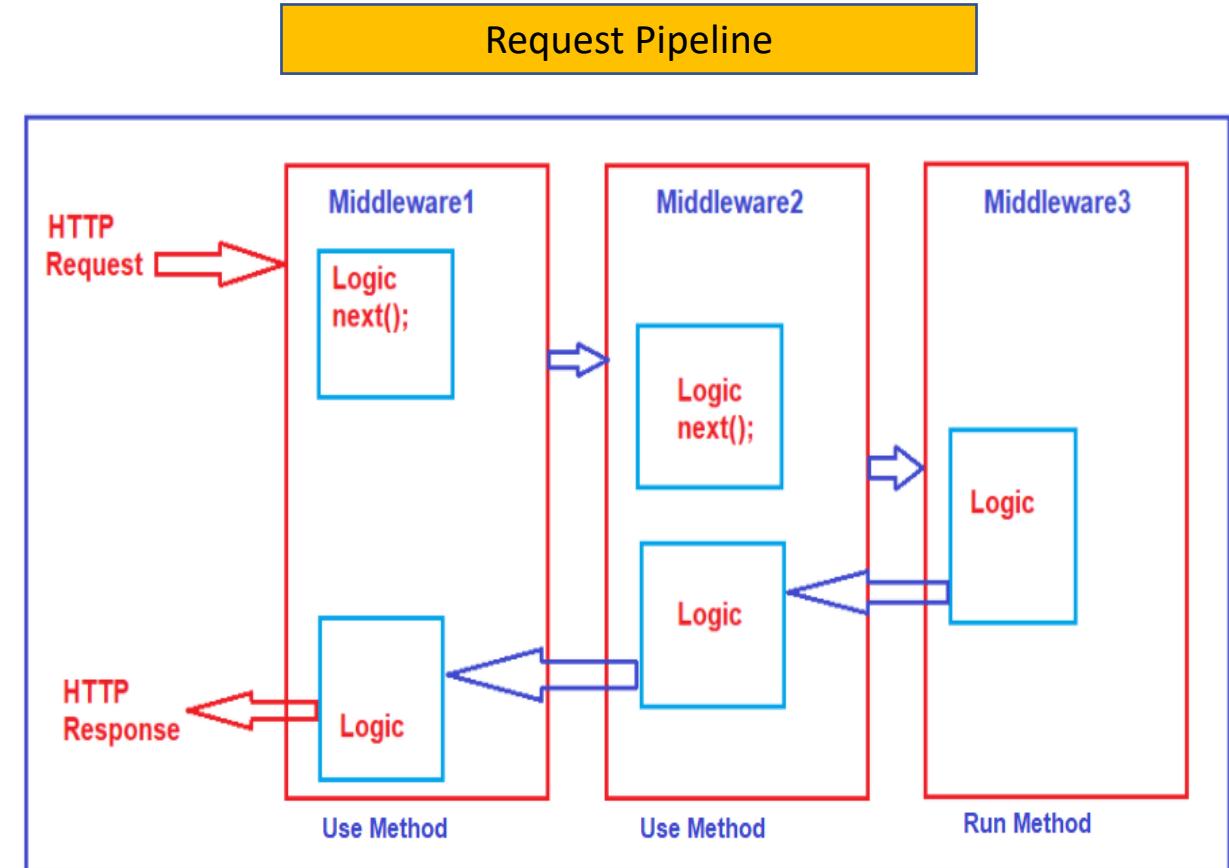
        // Configure the HTTP request pipeline.
        if (app.Environment.IsDevelopment())
        {
            app.UseSwagger();
            app.UseSwaggerUI();
        }

        app.UseAuthorization();
        app.MapControllers();
        app.Run();
    }
}
```

- ❖ HttpModules are nothing else, but they are very similar to Middlewares only.
- ❖ HttpModules are registered in the web.config or global.asax file of the ASP.NET framework, while a Middleware is registered via Configure method of the startup.cs class.

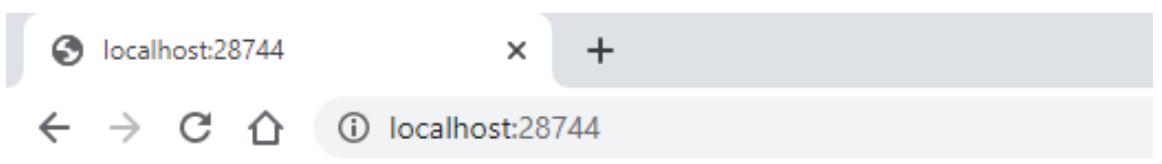


- ❖ Request delegates are used to **build** the request pipeline.
- ❖ Request delegates are configured using **Run**, **Map**, and **Use** extension methods.



- ❖ For any request, Use method will execute the middleware component and then pass the execution to the **next** middleware or component.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Use(async (context,next) =>
    {
        await context.Response.WriteAsync("Hello from 1st delegate.");
        await next();
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from 2nd Middleware");
    });
}
```



```
// This method gets called by the runtime.
// Use this method to configure the HTTP request pipeline.
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Home/Error");
    }
    app.UseStaticFiles();

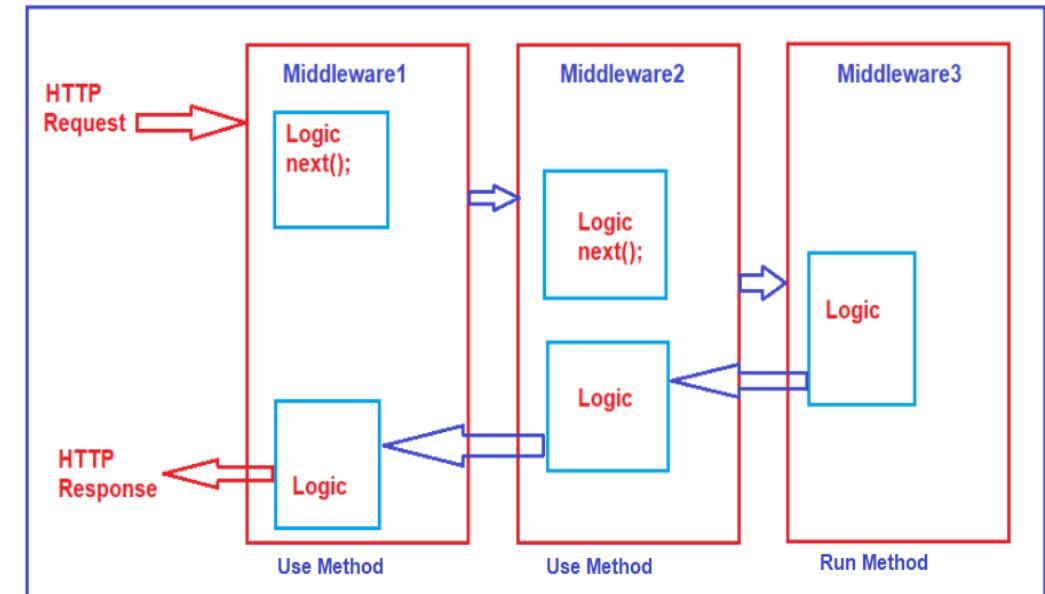
    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllerRoute(
            name: "default",
            pattern: "{controller=Home}/{action=Index}/{id?}");
    });
}
```

- ❖ Run method will execute the middleware component and then **terminate** the execution.
- ❖ It should be placed at the end of any pipeline.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Run(async context =>
    {
        await context.Response.WriteAsync("Hello from 1st delegate.");
    });
    app.Run(async (context) =>
    {
        await context.Response.WriteAsync("Hello from 2nd Middleware");
    });
}
```

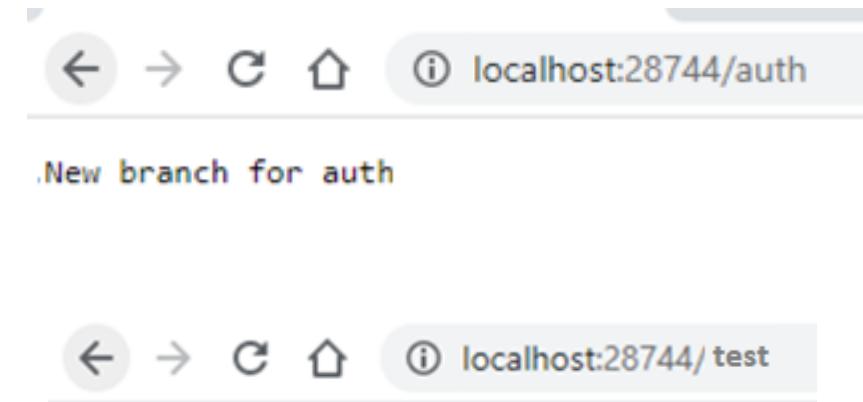


Hello from 1st delegate.

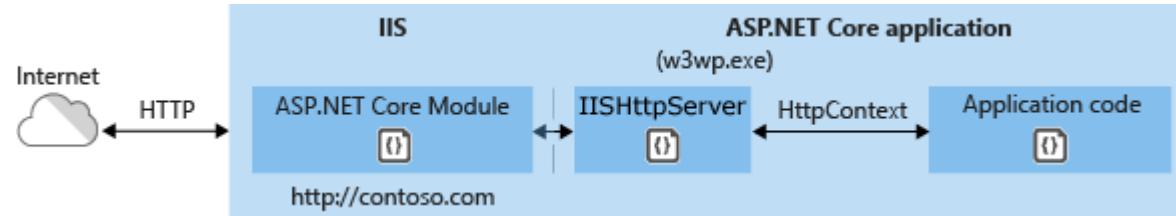
[back to chapter index](#)

- ❖ The Map method is used to **map** a specific request url path to a middleware component.

```
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.Map("/auth", a =>
    {
        a.Run(async (context) =>
        {
            await context.Response.WriteAsync("New branch for auth");
        });
    });
}
```

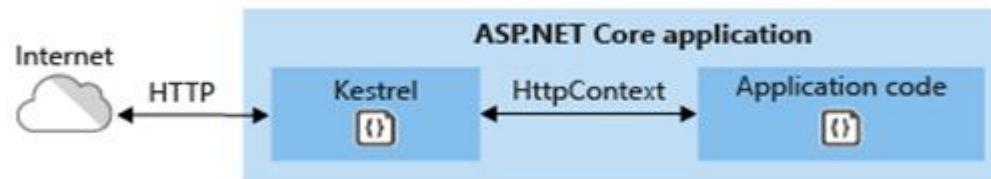


❖ In Process Hosting

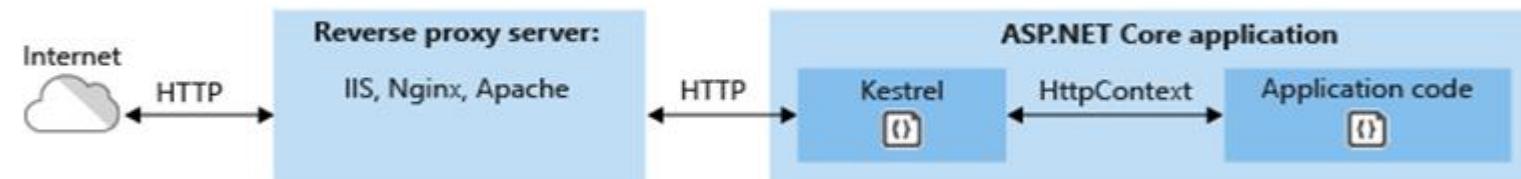


❖ Out of Process Hosting

Kestrel used as an edge (Internet-facing) web server:



Kestrel used in a reverse proxy configuration:



In-Process	Out of Process
Process name is w3wp.exe	Process name is dotnet.exe
Only one web server (IIS)	Two web server (IIS/Nginx/Apache + Kestrel) One web server (Kestrel)

- ❖ Kestrel is a **lightweight cross-platform** web server for asp.net core.
- ❖ It is a type of out of process hosting. It can be used alone, or it can be used with IIS or Nginx web servers.

Kestrel	IIS
1. Kestrel is a lightweight web server used for hosting.	IIS is a complete web server which is also used for hosting.
2. Kestrel is cross platform and can be used with other web servers like IIS, Nginx and Apache.	IIS is not cross platform and can only run in windows.
3. Kestrel is open source like .NET Core	IIS is not open source

Chapter 27 : .NET Core - Routing, Files, CORS & More...

Q225. What is **Routing**? Explain attribute routing in ASP.NET Core? V Imp

Q226. Explain **default project structure** in ASP.NET Core application?

Q227. How ASP.NET Core serve **static files**?

Q228. What are the roles of **Appsettings.Json** and **Launchsetting.Json** file?

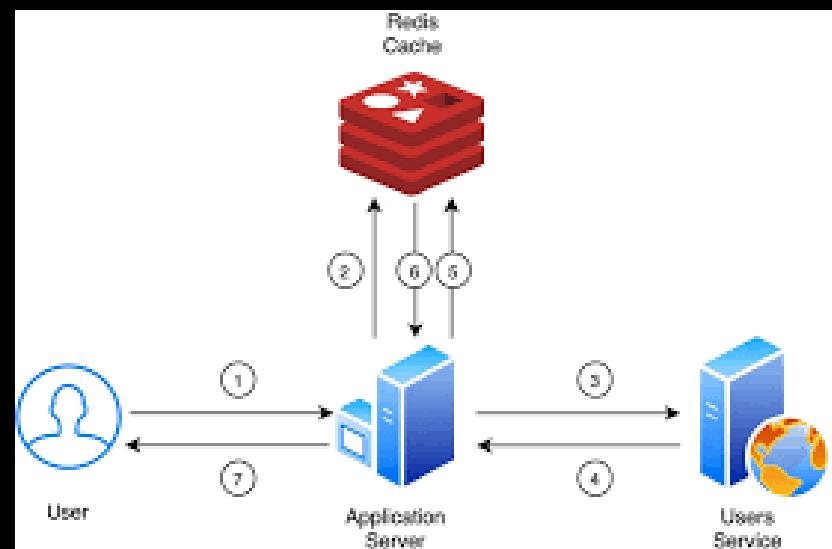
Q229. What are the various techniques to **save configuration** settings in .NET Core?

Q230. What is **CORS**? Why is CORS restriction is required? How to fix CORS error? V Imp

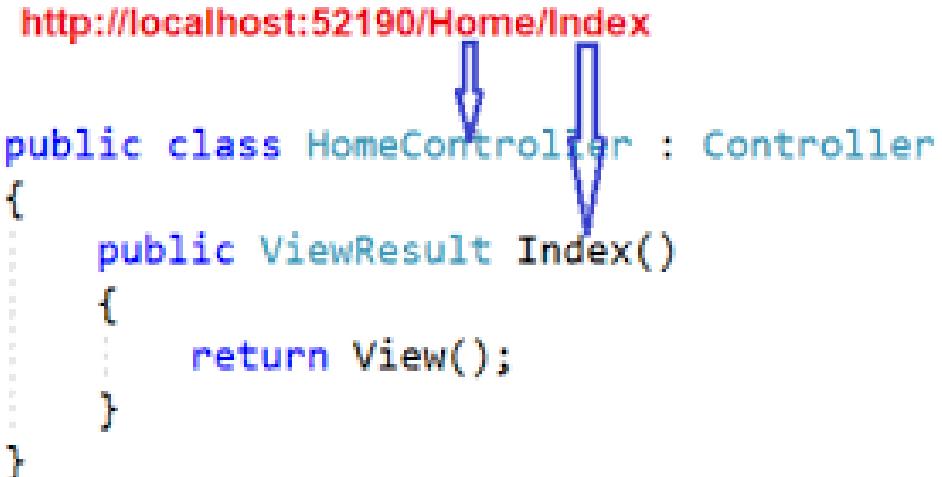
Q231. What is **In-Memory caching & Distributed Caching**?

Q232. How to handle **errors** in ASP.NET Core?

Q233. What are **Razor pages** in .NET Core?



- ❖ Routing in MVC is the process of mapping a URL request to a specific controller action in a web application.
- ❖ The routing system is responsible for directing incoming requests to the appropriate controller, based on the URL.
- ❖ The routing system is typically configured using a **routing table**, which maps URLs to controller actions.



The diagram illustrates the mapping of a URL to a controller action. A red box highlights the URL `http://localhost:52190/Home/Index`. Two blue arrows point from this URL to the corresponding code in a C# file. One arrow points to the word `HomeController` and another points to the method name `Index()`.

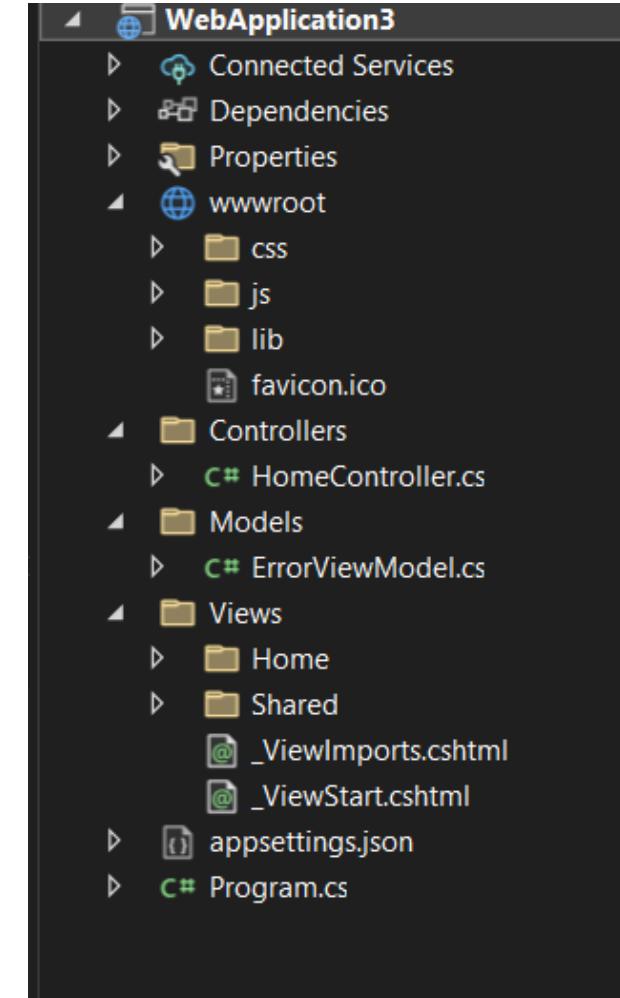
```
public class HomeController : Controller
{
    public ViewResult Index()
    {
        return View();
    }
}
```

- ❖ Attribute based routing is used to manipulate the default behavior of routing in ASP.NET MVC.

`http://localhost:1234/home/about`

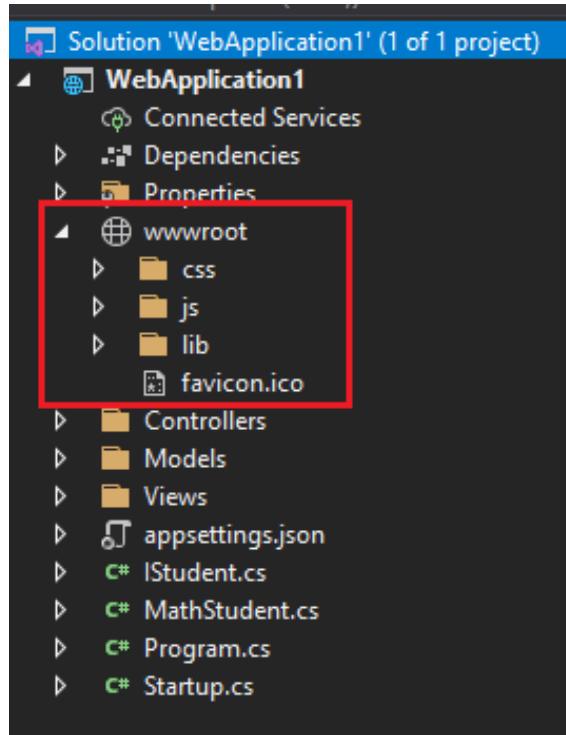
```
public class HomeController: Controller
{
    [Route("Users/about")]
    Public ActionResult GotoAbout()
    {
        return View();
    }
}
```

- ❖ wwwroot - Store static files of the application like js/css/images.
- ❖ Appsettings.json – Configuration settings like database connection strings and other things are saved here.
- ❖ Program.cs – Entry point of the application.



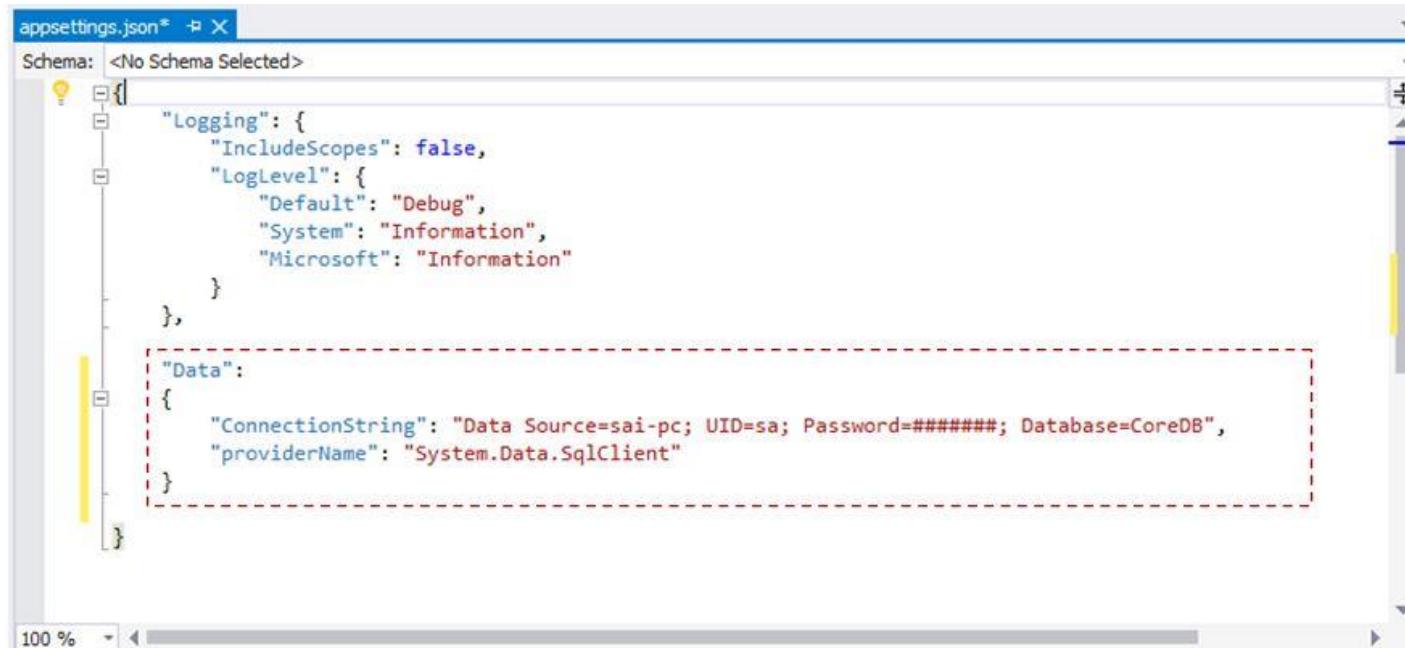
- ❖ WWWROOT contains all the static files.

- ❖ USESTATICFILES() enables the static files to be served to client.



```
// This method gets called by the runtime.  
// Use this method to configure the HTTP request pipeline.  
0 references  
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)  
{  
    if (env.IsDevelopment())  
    {  
        app.UseDeveloperExceptionPage();  
    }  
    else  
    {  
        app.UseExceptionHandler("/Home/Error");  
    }  
    app.UseStaticFiles();  
  
    app.UseRouting();  
  
    app.UseAuthorization();  
  
    app.UseEndpoints(endpoints =>  
    {  
        endpoints.MapControllerRoute(  
            name: "default",  
            pattern: "{controller=Home}/{action=Index}/{id?}");  
    });  
}
```

- The appsettings.json file is an application configuration file used to **store configuration settings** such as database connections strings etc.



```
appsettings.json*  # X
Schema: <No Schema Selected>
{
  "Logging": {
    "IncludeScopes": false,
    "LogLevel": {
      "Default": "Debug",
      "System": "Information",
      "Microsoft": "Information"
    }
  },
  "Data": {
    "ConnectionString": "Data Source=sai-pc; UID=sa; Password=#####; Database=CoreDB",
    "providerName": "System.Data.SqlClient"
  }
}
```



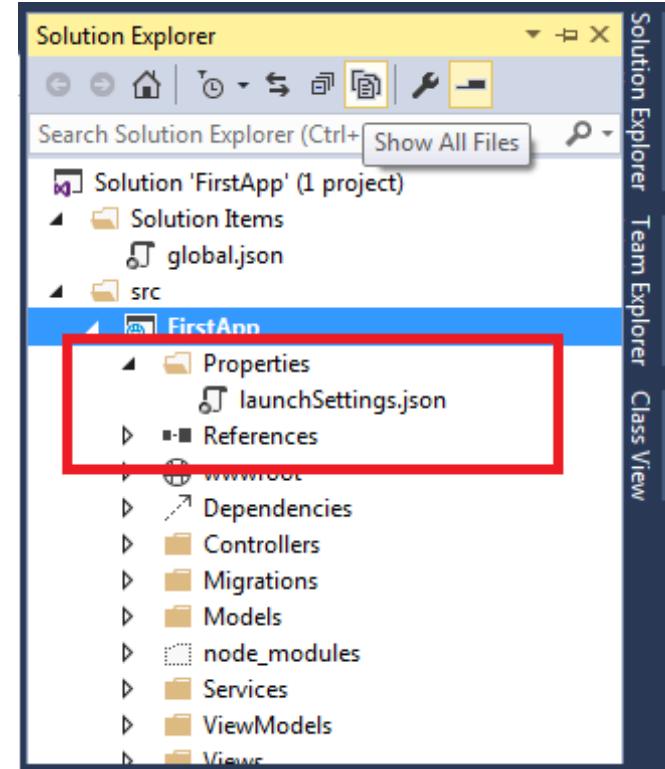
What are the roles of Appsettings.Json and Launchsetting.Json file in ASP.NET Core?

- The launchSettings.json file is used to store the configuration information, which is used to **start** the ASP.NET Core application.

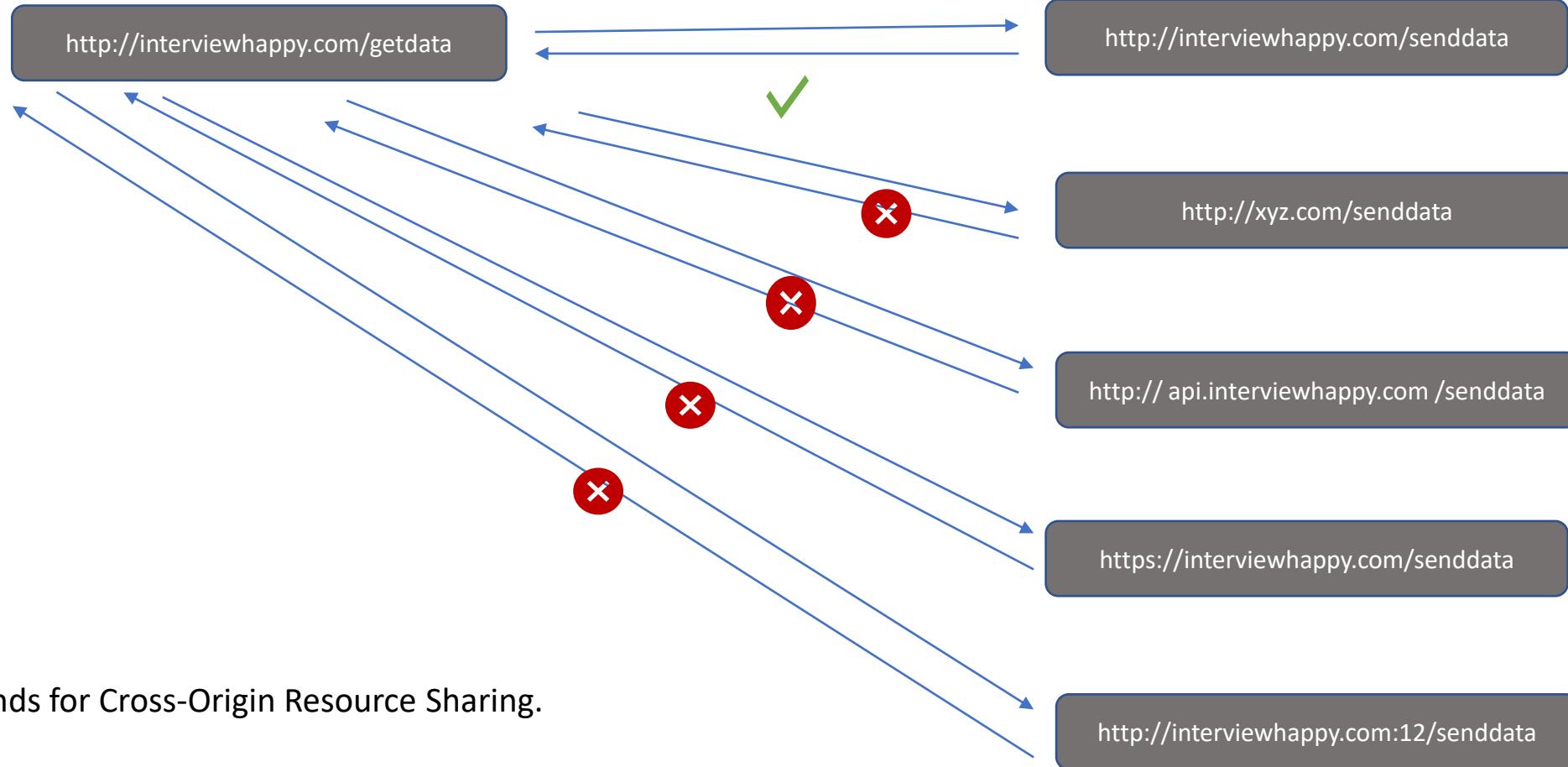
```

launchSettings.json
Schema: http://json.schemastore.org/launchsettings.json
1  {
2    "iisSettings": {
3      "windowsAuthentication": false,
4      "anonymousAuthentication": true,
5      "iisExpress": {
6        "applicationUrl": "http://localhost:50644",
7        "sslPort": 44309
8      }
9    },
10   "$schema": "http://json.schemastore.org/launchsettings.json",
11   "profiles": {
12     "IIS Express": {
13       "commandName": "IISExpress",
14       "launchBrowser": true,
15       "launchUrl": "api/values",
16       "environmentVariables": {
17         "ASPNETCORE_ENVIRONMENT": "Development"
18       }
19     },
20     "HeaderHelper": {
21       "commandName": "Project",
22       "launchBrowser": true,
23       "launchUrl": "api/values",
24       "environmentVariables": {
25         "ASPNETCORE_ENVIRONMENT": "Development"
26       },
27       "applicationUrl": "https://localhost:5001;http://localhost:5000"
28     },
29     "Docker": {
30       "commandName": "Docker",
31       "launchBrowser": true,
32       "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}/api/values",
33       "httpPort": 50645,
34       "useSSL": true,
35       "sslPort": 44310
36     }
37   }
38 }

```



- ❖ Appsettings.json (Default) (Mostly Used)
- ❖ Azure Key Vault (Application is hosted on Azure)
- ❖ Environment variables
- ❖ In-memory .NET objects
- ❖ Command Line Arguments
- ❖ Custom Providers



- ❖ CORS stands for Cross-Origin Resource Sharing.
- ❖ CORS is a security feature implemented in web browsers that restricts web pages or scripts from making requests to a different domain than the one that served the web page.

✖ Access to XMLHttpRequest at 'http://localhost:5000/global_config' step1:1 from origin '<http://localhost:8080>' has been blocked by CORS policy: Response to preflight request doesn't pass access control check: No 'Access-Control-Allow-Origin' header is present on the requested resource.

- ❖ CORS stands for Cross-Origin Resource Sharing.
- ❖ CORS is a security feature implemented in web browsers that restricts web pages or scripts from making requests to a different domain than the one that served the web page.

❖ Why CORS is required?

Because of security reason. So that you can make your api secure by default. Otherwise, any external website can try to access your data.

❖ How to fix CORS error?

1. Add Microsoft.AspNetCore.Cors nuget package to your project.
2. In startup class, edit the Configure and ConfigureServices method.

```
// This method gets called by the runtime. Use this method to add service
public void ConfigureServices(IServiceCollection services)
{
    services.AddCors();
    services.AddControllers();
}

// This method gets called by the runtime. Use this method to configure the
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    app.UseRouting();

    // global cors policy
    app.UseCors(x => x
        .AllowAnyMethod()
        .AllowAnyHeader()
        .SetIsOriginAllowed(origin => true) // allow any origin
        .AllowCredentials()); // allow credentials

    app.UseAuthentication();
    app.UseAuthorization();

    app.UseEndpoints(x => x.MapControllers());
}
```

In-Memory Caching

1. It's the normal way of caching.
In this cache is stored in the memory of a **single server** which is hosting the application.

2. It can be implemented with the `IMemoryCache` Interface in ASP.NET Core.

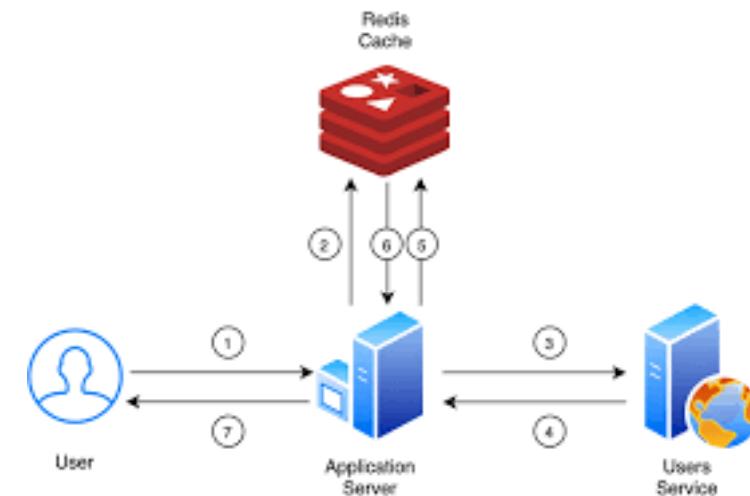
- ❖ Redis is an open-source, highly replicated, performant, non-relational kind of database and caching server.
- ❖ When to use which caching?

In normal cases where the application size is small, use in-memory cache.
But where application is very big or it's a microservices based architecture, then use distributed caching.

Distributed Caching

Distributed caching is when you want to handle caching outside of your application. A different server is used for store cached data.

It can be implemented with the help of Redis Cache.



- ❖ Error handling for development and other environments can be set in CONFIGURE method of Startup.cs class.

```
0 references
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }
    else
    {
        app.UseExceptionHandler("/Error");
    }

    app.UseStaticFiles();

    app.UseRouting();

    app.UseAuthorization();

    app.UseEndpoints(endpoints =>
    {
        endpoints.MapRazorPages();
    });
}
```

- ❖ The IsDevelopment() method will check the ASPNETCORE_ENVIRONMENT value in LaunchSettings.json file.

```
"Example": {
    "commandName": "Project",
    "dotnetRunMessages": "true",
    "launchBrowser": true,
    "applicationUrl": "http://localhost:5000",
    "environmentVariables": {
        "ASPNETCORE_ENVIRONMENT": "Development"
    }
}
```

- ❖ Razor pages follows a page-centric development model just like ASP.NET web forms.

Chapter 28 : SOLID Principles

Q234. What are **SOLID Principles**? What is the difference between SOLID Principles and Design Patterns? V Imp

Q235. What is **Single Responsibility Principle**? V Imp

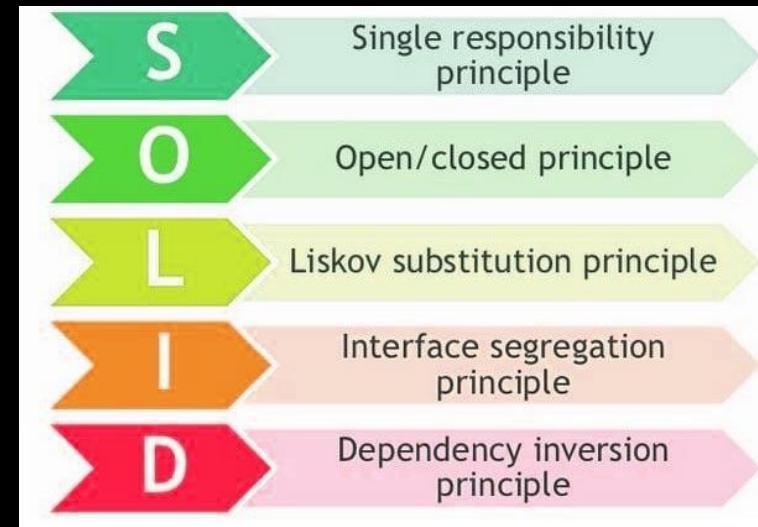
Q236. What is **Open-closed Principle**?

Q237. What is **Liskov Substitution Principle**? V Imp

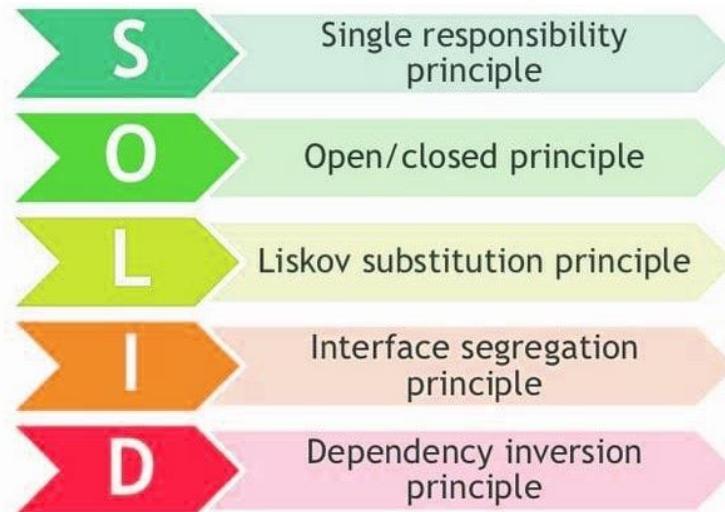
Q238. What is **Interface Segregation Principle**?

Q239. What is **Dependency Inversion Principle**?

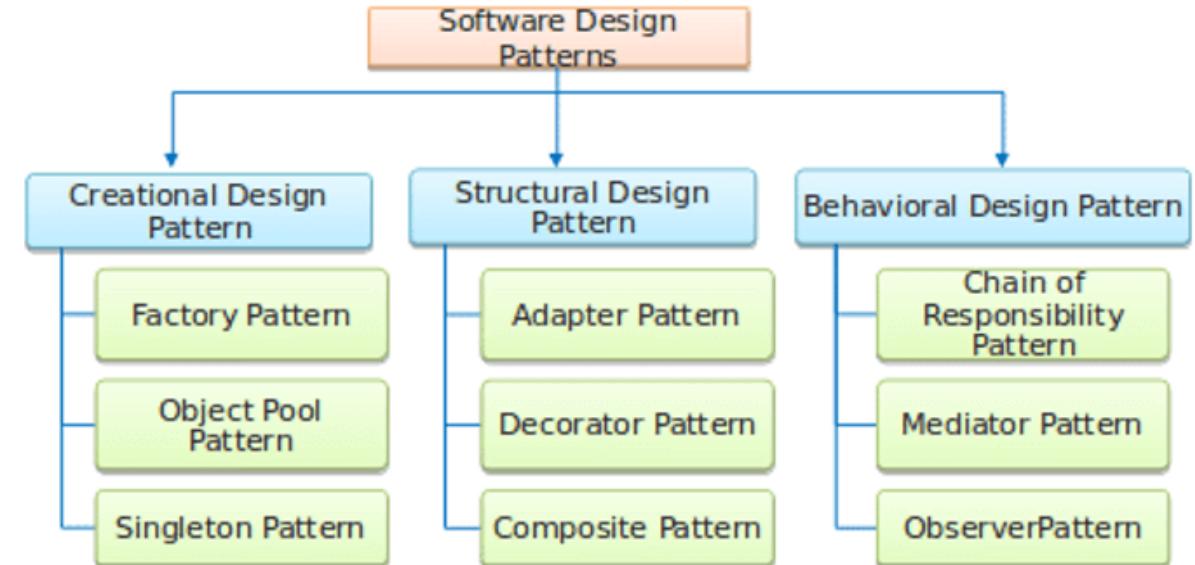
Q240. What is **DRY principle**?



- ❖ SOLID principles are a set of principles, which must be followed to develop flexible, maintainable, and scalable software systems.



- ❖ Design patterns are concrete and solve a particular kind of problem in software's.



- ❖ SOLID principles aren't concrete - rather abstract.

- ❖ Design patterns are concrete and solve a particular kind of problem in a specific and fixed manner.

- ❖ Single Responsibility Principle (SRP) states that a class should have only **one responsibility**.
- ❖ Or a class should have only **one reason** to change.

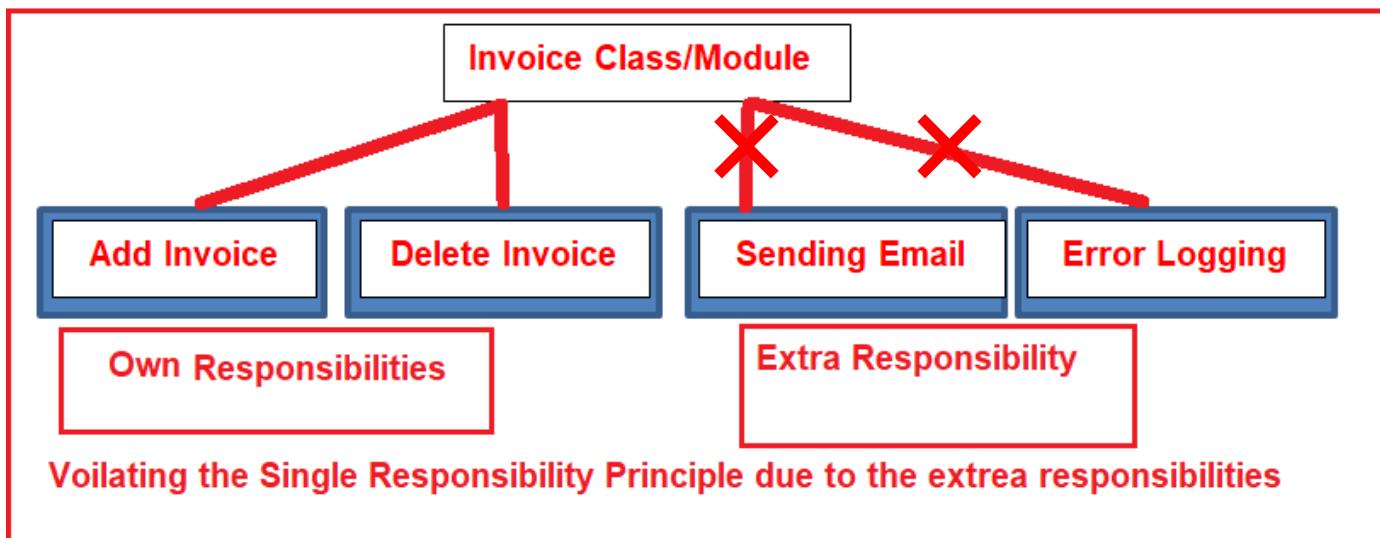
```
//Violating SRP, because the class  
//has extra responsibility

public class Employee
{
    //Own responsibility
    public int CalculateSalary()
    {
        return 100000;
    }

    //Own responsibility
    public string GetDepartment()
    {
        return "IT";
    }

    //Extra responsibility
    public void Save()
    {
        //Save employee to the database
    }
}
```

- ❖ Single Responsibility Principle (SRP) states that a class should have only **one responsibility**.
- ❖ Or a class should have only **one reason** to change.
- ❖ When a class has only one responsibility, it becomes easier to change and test. If a class has multiple responsibilities, changing one responsibility may impact others and more testing efforts will be required then.



```
//Following SRP
public class Employee
{
    public int CalculateSalary()
    {
        return 100000;
    }
    public string GetDepartment()
    {
        return "IT";
    }
}

public class EmployeeRepository
{
    public void Save(Employee employee)
    {
        //Save employee to the database
    }
}
```

- ❖ Open-Closed Principle (OCP) states that software entities(classes, modules) should be open for **extension**, but closed for **modification**.

Inserting one more else if{} will violate the open closed principle because you are modifying the class rather than extending it.

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

    public string Address { get; set; }

    public double Balance { get; set; }

    public double CalculateInterest(string accountType)
    {
        if(accountType == "Saving")
        {
            return Balance * 0.3;
        }
        else
        {
            return Balance * 0.5;
        }
    }
}
```

- ❖ Open-Closed Principle (OCP) states that software entities(classes, modules) should be open for **extension**, but closed for **modification**.
- ❖ SRP is the **prerequisite** for OCP.

```
public class Account
{
    public string Name { get; set; }
    public string Address { get; set; }
    public double Balance { get; set; }
}
```

```
interface IAccount
{
    double CalculateInterest(Account account);
}
```

- ❖ The benefit is **simple testing** is required to test individual classes, but if you will keep on adding and modifying in one class. Then even for the smallest modification, the whole class needs to be tested.

```
public class SavingAccount : IAccount
{
    public double CalculateInterest(Account account)
    {
        return account.Balance * 0.3;
    }
}

public class OtherAccount : IAccount
{
    public double CalculateInterest(Account account)
    {
        return account.Balance * 0.5;
    }
}

public class CurrentAccount : IAccount
{
    public double CalculateInterest(Account account)
    {
        return account.Balance * 0.7;
    }
}
```

- The Liskov Substitution Principle (LSP) states that an object of a child class must be able to **replace** an object of the parent class without **breaking** the application.

```
static void Main(string[] args)
{
    Employee employee = new Employee();

    PermanentEmployee pEmployee = new PermanentEmployee();

    ContractualEmployee cEmployee = new ContractualEmployee();

    Console.WriteLine(employee.CalculateSalary());
    //100000
    Console.WriteLine(employee.CalculateBonus());
    //10000

    Console.WriteLine(pEmployee.CalculateSalary());
    //200000
    Console.WriteLine(pEmployee.CalculateBonus());
    //10000
    Console.WriteLine(cEmployee.CalculateSalary());
    //150000
    Console.WriteLine(cEmployee.CalculateBonus());
    //Exception: The method or operation is not implemented
}
```

- All the base class methods must be applicable for the derived class.

```
//Base/ Parent/ Superclass
public class Employee
{
    public virtual int CalculateSalary()
    {
        return 100000;
    }
    public virtual int CalculateBonus()
    {
        return 10000;
    }
}

//Derived/ Child/ Subclass
public class PermanentEmployee : Employee
{
    public override int CalculateSalary()
    {
        return 200000;
    }
}

public class ContractualEmployee : Employee
{
    public override int CalculateSalary()
    {
        return 150000;
    }
    public override int CalculateBonus()
    {
        throw new NotImplementedException();
    }
}
```

[back to chapter index](#)

- ❖ The Interface Segregation (ISP) states that a class should not be forced to implement interfaces that it does not use.

```
public interface IVehicle
{
    void Drive();

    void Fly();
}
```

```
public class FlyingCar : IVehicle
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }

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

```
public class Car : IVehicle
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }

    public void Fly()
    {
        throw new NotImplementedException();
    }
}
```

- ❖ The Interface Segregation (ISP) states that a class should not be forced to implement interfaces that it does not use.
- ❖ It is better to have multiple smaller interfaces than larger interfaces.

```
public interface IDrive
{
    void Drive();
}

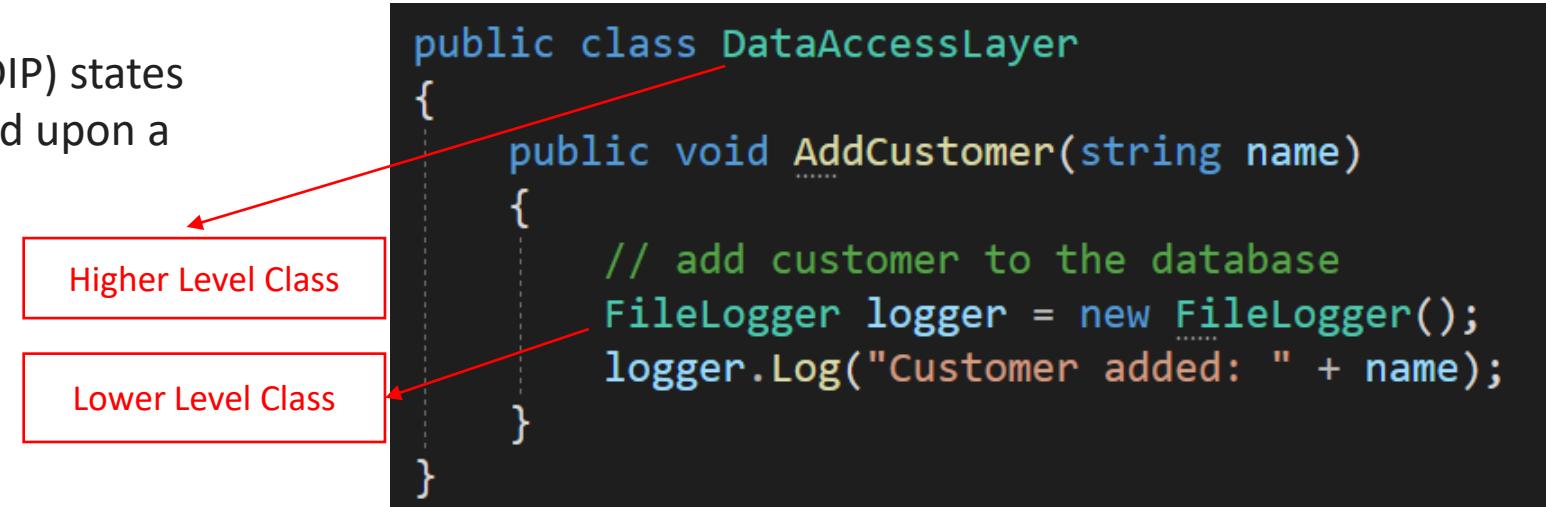
public interface IFly
{
    void Fly();
}
```

```
public class Car : IDrive
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }
}
```

```
public class FlyingCar : IDrive, IFly
{
    public void Drive()
    {
        Console.WriteLine("Drive car");
    }

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

- ❖ The Dependency Inversion Principle (DIP) states that a **high-level class** must not depend upon a **lower level class**.



```
public class FileLogger
{
    public void Log(string message)
    {
        // write message to a log file
    }
}
```

- ❖ The Dependency Inversion Principle (DIP) states that a **high-level class** must not depend upon a **lower level class**.

```
public interface ILogger
{
    void Log(string message);
}

public class FileLogger : ILogger
{
    public void Log(string message)
    {
        // write message to a log file
    }
}
```

```
public class DataAccessLayer
{
    private ILogger logger;

    public DataAccessLayer(ILogger logger)
    {
        this.logger = logger;
    }

    public void AddCustomer(string name)
    {
        // add customer to the database
        logger.Log("Customer added: " + name);
    }
}
```

- ❖ DRY stands for DON'T REPEAT YOURSELF.
- ❖ Don't write the same functionality multiple time.
- ❖ Instead write code once and then use it at multiple places using inheritance.

Chapter 29 : Design Patterns

Q241. What are **Design Patterns** and what problem they solve? **V Imp**

Q242. What are the types of Design Patterns? **V Imp**

Q243. What are **Creational** Design Patterns?

Q244. What are **Structural** Design Patterns?

Q245. What are **Behavioral** Design Patterns?

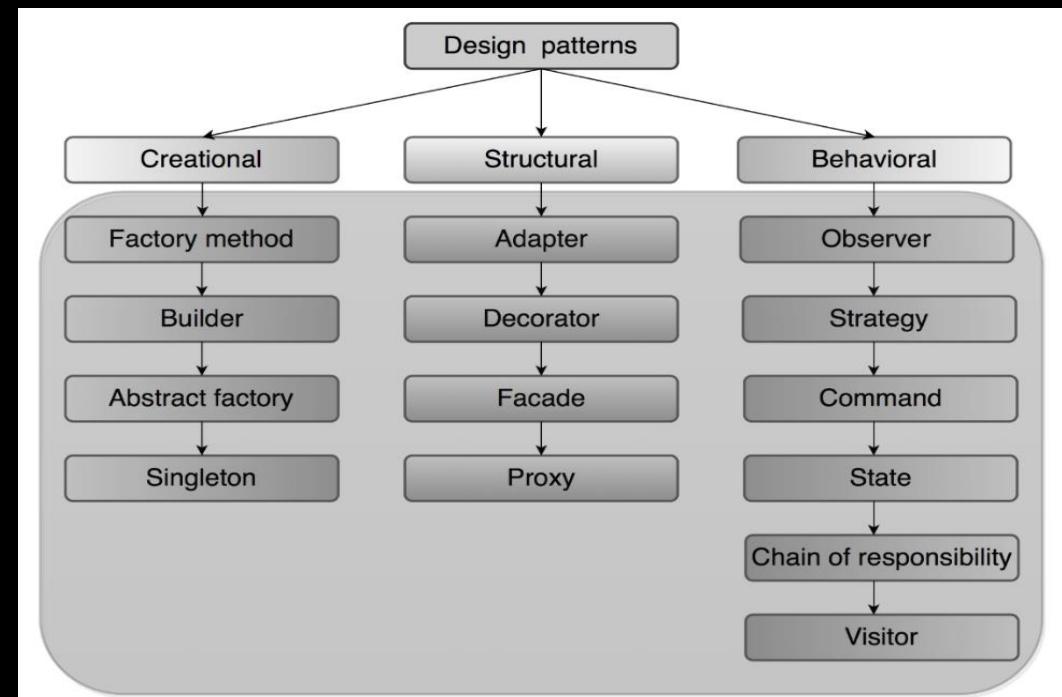
Q246. What is **Singleton** Design Pattern? **V Imp**

Q247. How to make singleton pattern **thread safe**? **V Imp**

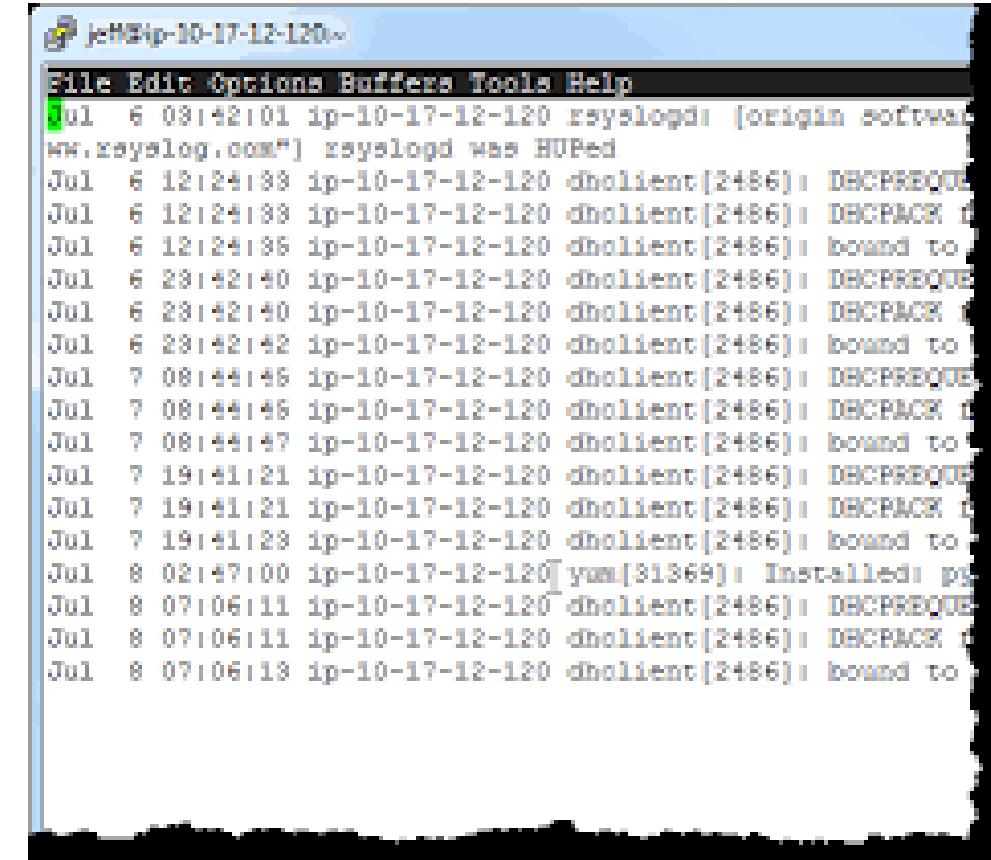
Q248. What is **Factory** pattern? Why to use factory pattern?

Q249. How to **implement** Factory method pattern?

Q250. What is **Abstract Factory** pattern?

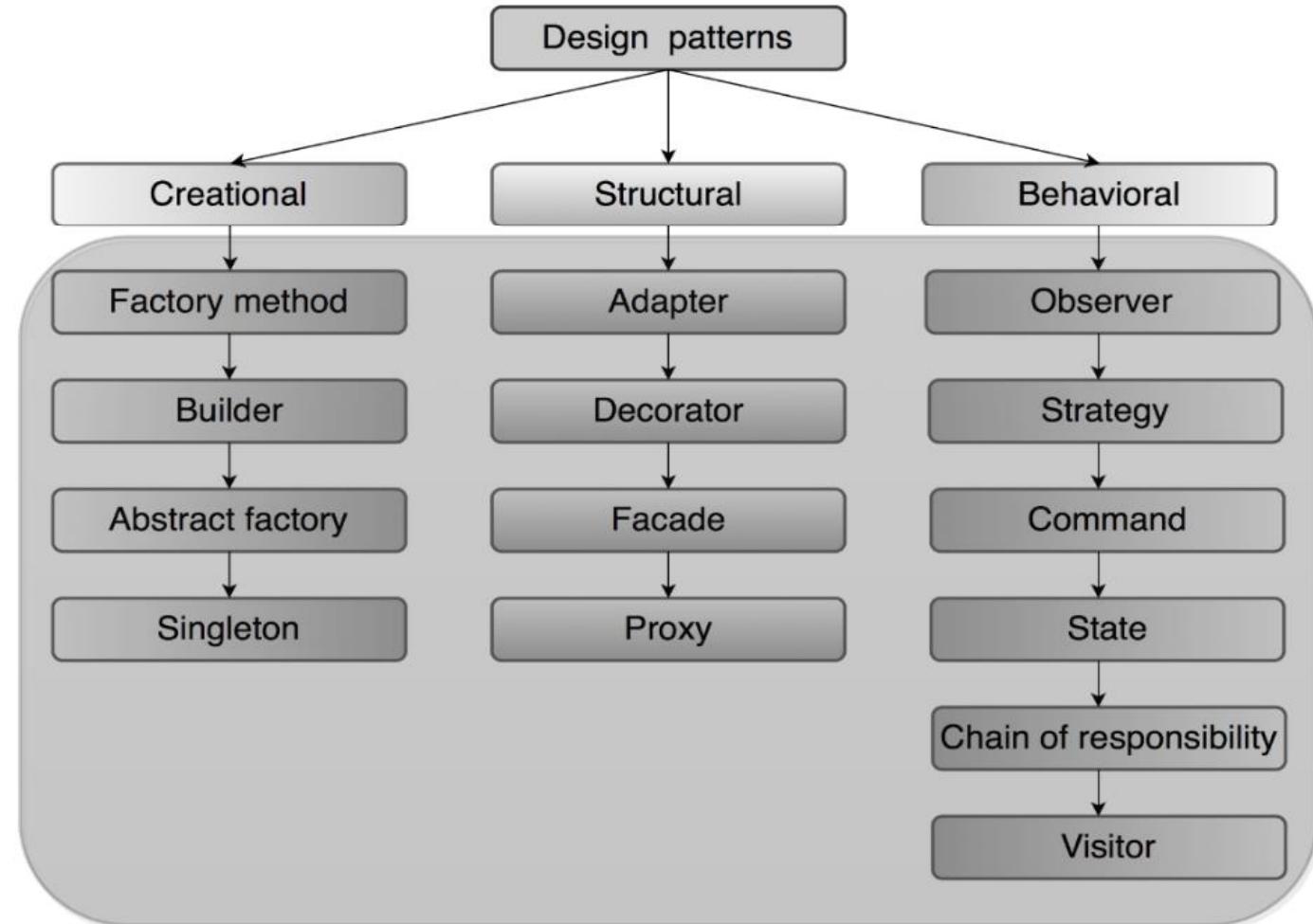


- ❖ Design patterns are **reusable solutions** for common problems in software design.



The screenshot shows a terminal window with the title "jet@ip-10-17-12-120:~". The window contains a list of log messages from the "rsyslogd" and "dhclient" daemons. The log entries are as follows:

```
File Edit Options Buffers Tools Help
Jul 6 09:42:01 ip-10-17-12-120 rsyslogd: [origin software="www.rsyslog.com"] rsyslogd was HUPed
Jul 6 12:24:39 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 6 12:24:39 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 6 12:24:39 ip-10-17-12-120 dhclient[2486]: bound to 10.17.12.12 via eth0
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 6 23:42:40 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 6 23:42:42 ip-10-17-12-120 dhclient[2486]: bound to 10.17.12.12 via eth0
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 7 08:44:46 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 7 08:44:47 ip-10-17-12-120 dhclient[2486]: bound to 10.17.12.12 via eth0
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 7 19:41:21 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 7 19:41:23 ip-10-17-12-120 dhclient[2486]: bound to 10.17.12.12 via eth0
Jul 8 02:47:00 ip-10-17-12-120 yum[31369]: Installed: ps
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPREQUEST on eth0 to 10.17.12.1 port 67
Jul 8 07:06:11 ip-10-17-12-120 dhclient[2486]: DHCPACK from 10.17.12.1
Jul 8 07:06:19 ip-10-17-12-120 dhclient[2486]: bound to 10.17.12.12 via eth0
```

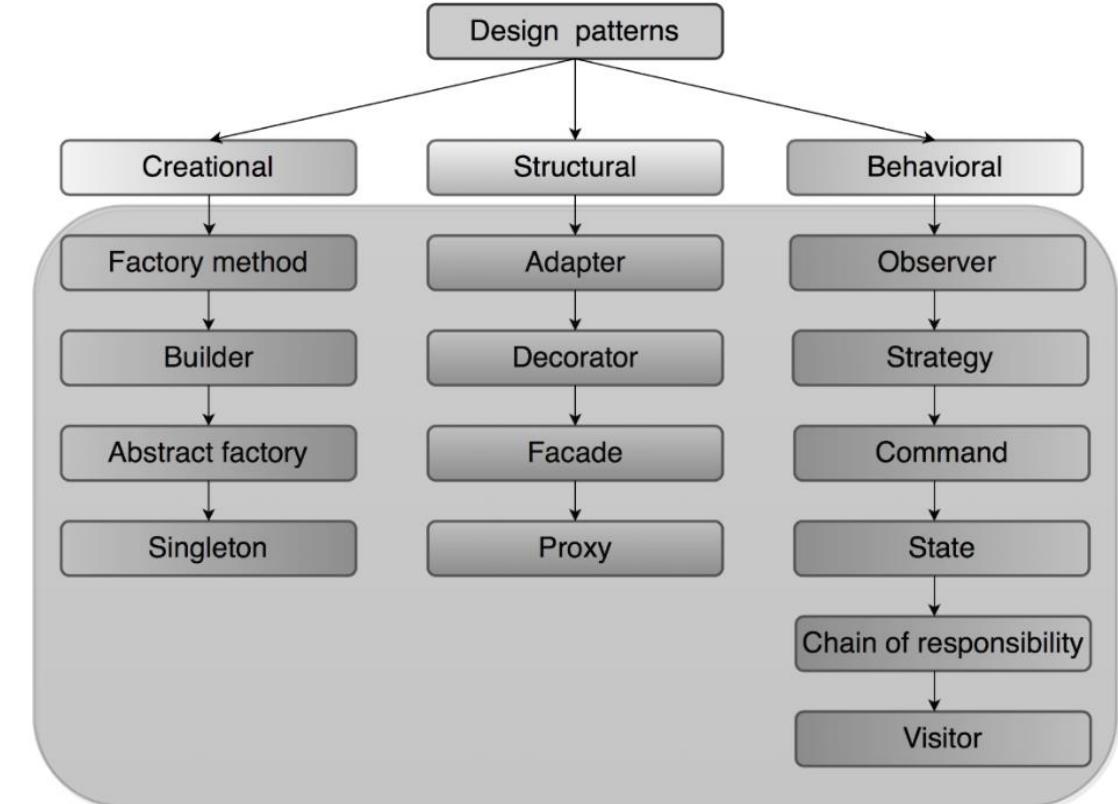


- ❖ Creational design patterns are design patterns that deal with **object creation** mechanisms.

- ❖ Why direct object creation is a problem?

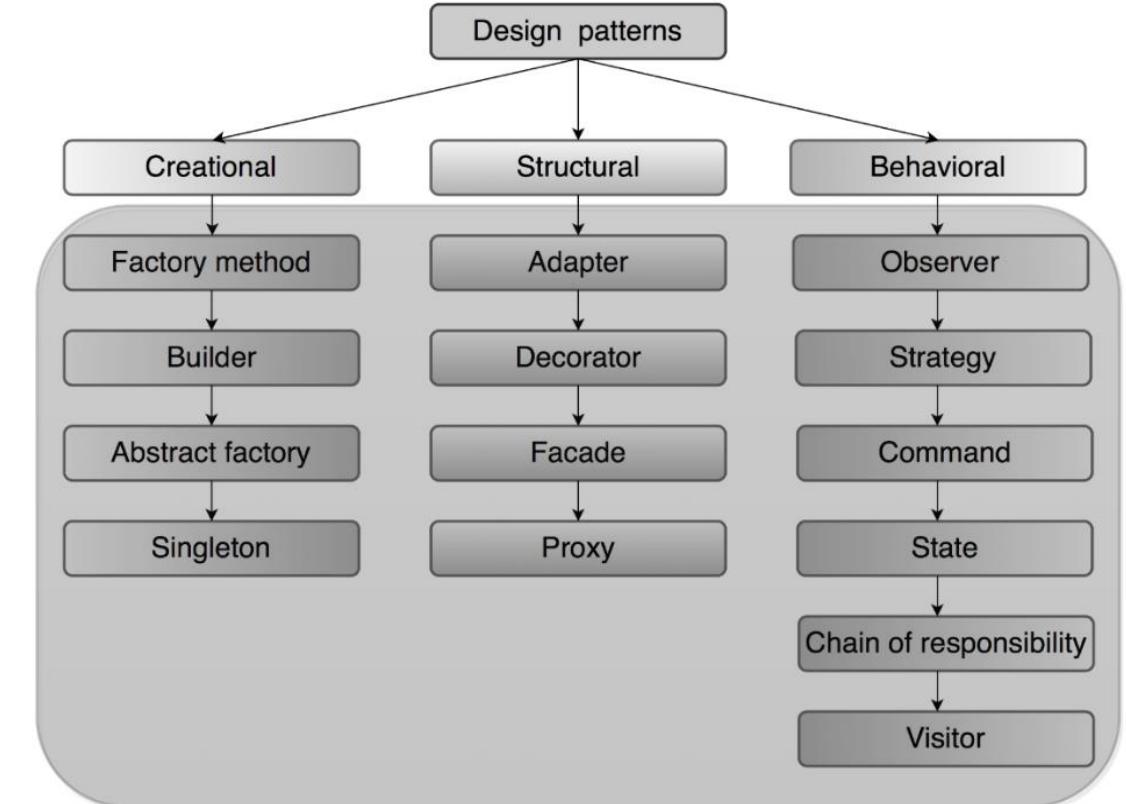
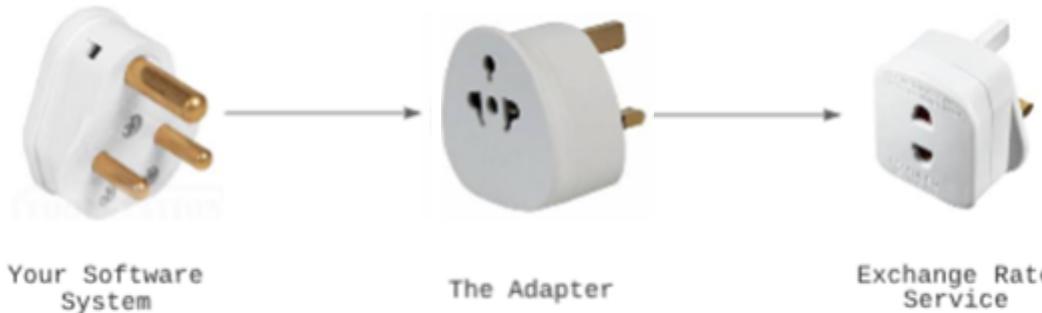
```
public Class Pet
{
    Cat b = new Cat();
    b.CountLegs();
}
```

- ❖ The reason is in a large/enterprise application, if you want to change this Class from Cat to Dog, then you have to change it everywhere wherever you have used it and then it will be a testing issue. You have to retest all these classes like Pet class.

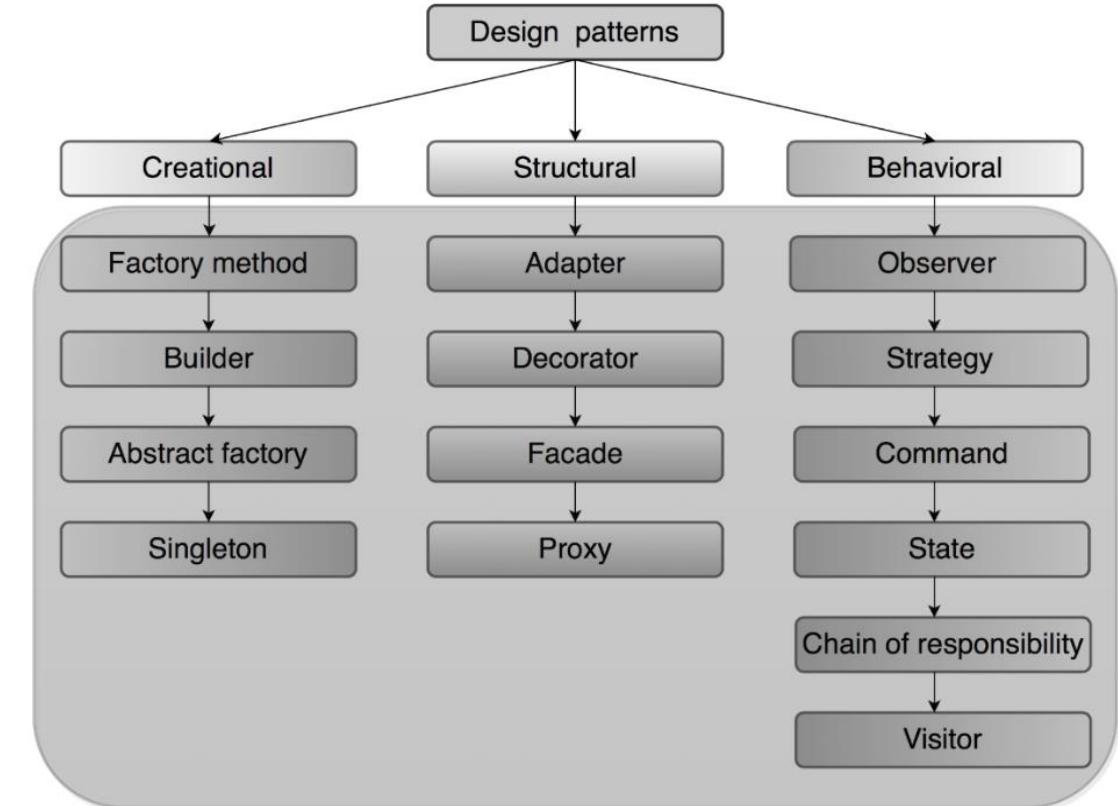
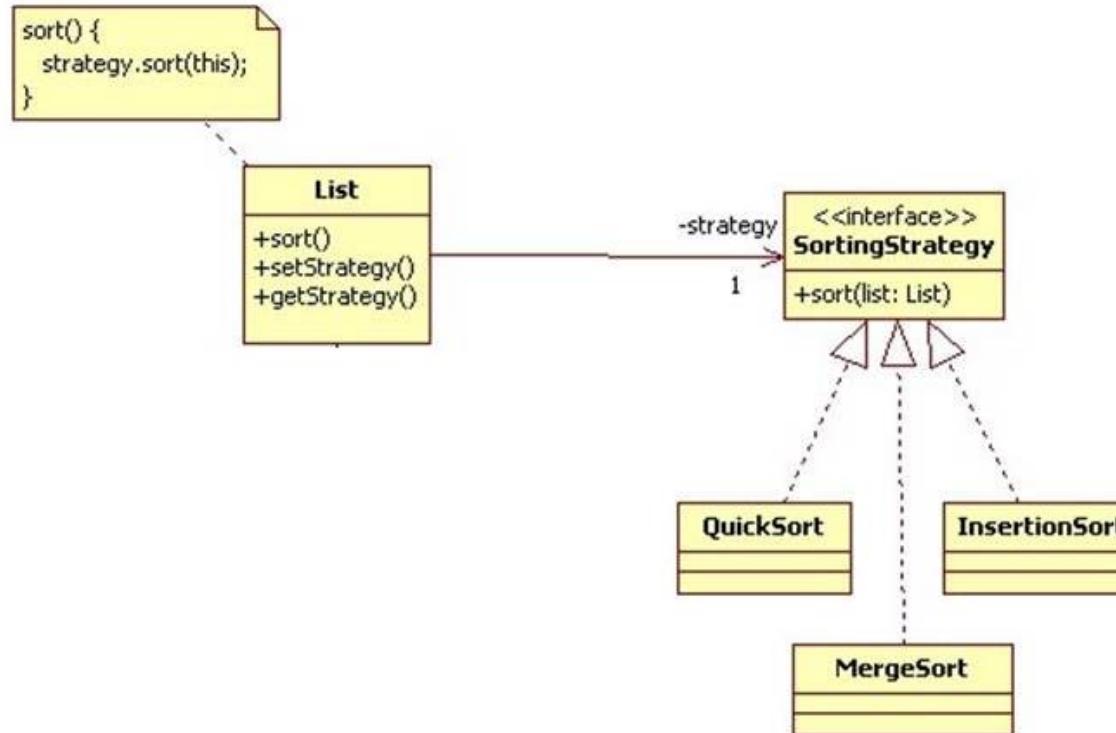


- ❖ Structural Design Pattern is used to manage the structure of classes and interfaces to manage the **relationship** between the classes.

- ❖ Adapter pattern



- ❖ Behavioral design patterns defines the way to **communicate** between classes and object.
- ❖ For example, in strategy pattern we can choose the best strategy at run time for doing any operation .



- The Singleton design pattern is a creational design pattern that ensures a class has only **one instance** of it.

```
public class NonSingleton
{
    public void Method()
    {
        //logic
    }
}

static void Main(string[] args)
{
    NonSingleton obj = new NonSingleton();
    obj.Method();
}
```

```
static void Main(string[] args)
{
    Singleton obj = Singleton.GetInstance();
    obj.Method();
}
```

1. Declare class as sealed to prevent any further inheritance

2. Declare a static instance of this class.

3. Create a private constructor so that no one can create the object of this class.

4. Create a method which will check whether the instance is null, if yes then it will create the new instance and if not null then it will return the same instance.

```
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }

    public static Singleton GetInstance()
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }

    public void Method()
    {
        //logic..
    }
}
```

❖ Normal Singleton Pattern

```
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }

    public static Singleton GetInstance()
    {
        if (instance == null)
        {
            instance = new Singleton();
        }
        return instance;
    }

    public void Method()
    {
        //logic..
    }
}
```



```
//Thread Safe Singleton Pattern

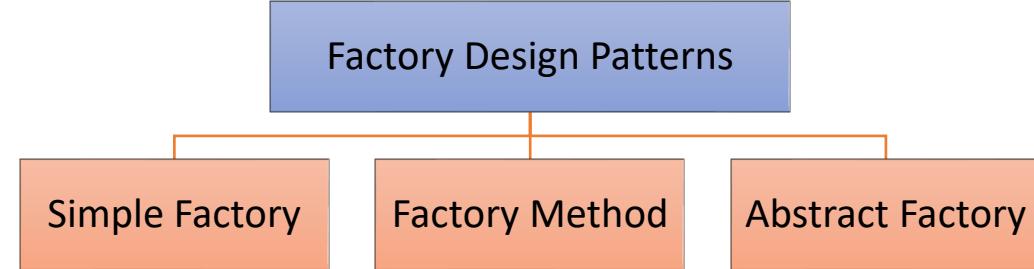
public sealed class Singleton
{
    private static Singleton instance = null;

    private Singleton()
    {
    }

    private static readonly object padlock = new object();

    public static Singleton GetInstance()
    {
        lock (padlock)
        {
            if (instance == null)
            {
                instance = new Singleton();
            }
            return instance;
        }
    }
}
```

- The Factory Pattern is a **creational** design pattern which manages object creation.



```
// Concrete Classes
class Java
{
    public string GetFramework()
    {
        return "Java Spring Boot";
    }
}

class DotNet
{
    public string GetFramework()
    {
        return ".NET Core";
    }
}
```

```
// Client code
class Program
{
    static void Main(string[] args)
    {
        // Creating Java object
        Java java = new Java();
        Console.WriteLine(java.GetFramework());
        // Output: Java Spring Boot

        // Creating DotNet object
        DotNet dotnet = new DotNet();
        Console.WriteLine(dotnet.GetFramework());
        // Output: .NET Core
    }
}
```

1. Lots of **new** keywords

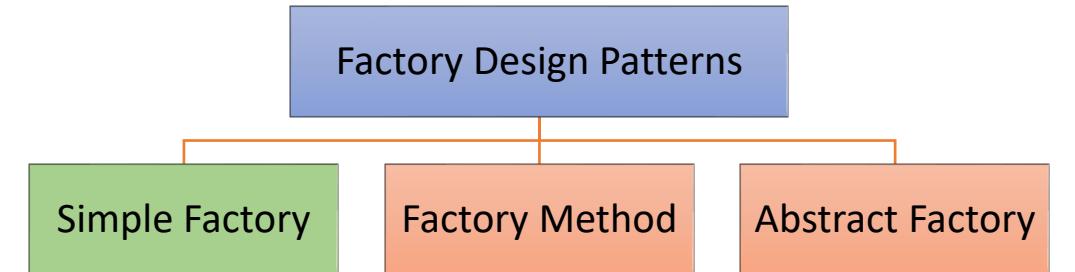
2. Client will know all the concrete classes name

- ❖ Factory Pattern provides an **interface** or abstract class for creating objects, but lets the subclasses decide which class to instantiate.
- ❖ Or In simple factory pattern, one factory class is responsible for creating all the objects for its derived or sub class.

```
// Interface
interface ITech
{
    string GetFramework();
}

// Concrete Classes
class Java : ITech
{
    public string GetFramework()
    {
        return "Java Spring Boot";
    }
}
class DotNet : ITech
{
    public string GetFramework()
    {
        return ".NET Core";
    }
}
```

```
// Simple Factory
class TechFactory
{
    public ITech CreateTech(string techType)
    {
        if (techType.Equals("Java"))
        {
            return new Java();
        }
        else if (techType.Equals("DotNet"))
        {
            return new DotNet();
        }
        else
        {
            throw new Exception("Invalid");
        }
    }
}
```



```
// Client code
class Program
{
    static void Main(string[] args)
    {
        TechFactory factory = new TechFactory();

        // Creating Java object
        ITech java = factory.CreateTech("Java");
        Console.WriteLine(java.GetFramework());
        // Output: Java Spring Boot

        // Creating DotNet object
        ITech dotnet = factory.CreateTech("DotNet");
        Console.WriteLine(dotnet.GetFramework());
        // Output: .NET Core
    }
}
```

- ❖ Simple Factory pattern provides a single factory class that creates objects, while the Factory Method pattern uses inheritance and polymorphism, so it provide more flexibility and extensibility.

```
// Interface
interface ITech
{
    string GetFramework();
}

// Concrete Classes
class Java : ITech
{
    public string GetFramework()
    {
        return "Java Spring Boot";
    }
}
class DotNet : ITech
{
    public string GetFramework()
    {
        return ".NET Core";
    }
}
```

```
// Simple Factory
class TechFactory
{
    public ITech CreateTech(string techType)
    {
        if (techType.Equals("Java"))
        {
            return new Java();
        }
        else if (techType.Equals("DotNet"))
        {
            return new DotNet();
        }
        else
        {
            throw new Exception("Invalid");
        }
    }
}
```

```
// Client code
class Program
{
    static void Main(string[] args)
    {
        TechFactory factory = new TechFactory();

        // Creating Java object
        ITech java = factory.CreateTech("Java");
        Console.WriteLine(java.GetFramework());
        // Output: Java Spring Boot

        // Creating DotNet object
        ITech dotnet = factory.CreateTech("DotNet");
        Console.WriteLine(dotnet.GetFramework());
        // Output: .NET Core
    }
}
```

- ❖ Simple Factory pattern provides a single factory class that creates objects, while the Factory Method pattern uses inheritance and polymorphism, so it provide more flexibility and extensibility.

```
// Product
interface ITech
{
    void GetFramework();
}

// Concrete Product
class Java : ITech
{
    public void GetFramework()
    {
        Console.WriteLine("Spring");
    }
}

// Concrete Product
class DotNet : ITech
{
    public void GetFramework()
    {
        Console.WriteLine("Core");
    }
}
```

```
abstract class Creator
{
    // The Factory Method
    public abstract ITech CreateTech();
}

// Concrete Creator
class JavaCreator : Creator
{
    // Implementation of Factory Method
    public override ITech CreateTech()
    {
        return new Java();
    }
}

// Concrete Creator
class DotNetCreator : Creator
{
    // Implementation of Factory Method
    public override ITech CreateTech()
    {
        return new DotNet();
    }
}
```

```
static void Main(string[] args)
{
    // Create a JavaCreator object
    Creator javaCreator = new JavaCreator();

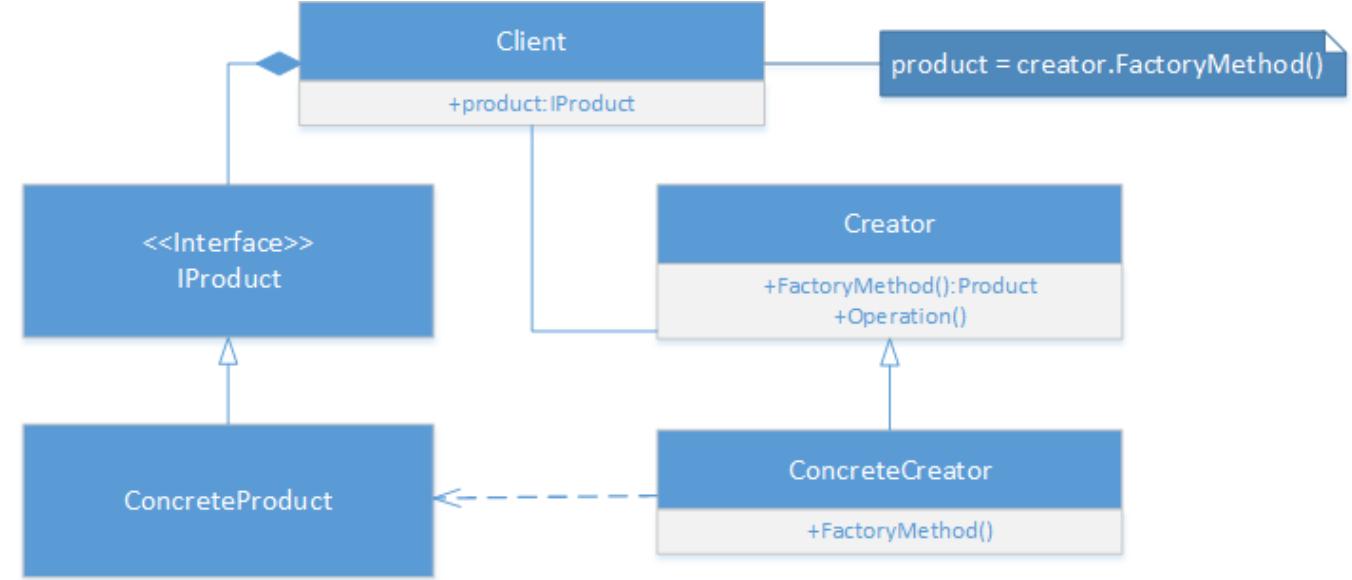
    // Call Factory Method to
    // create a JavaTech object
    ITech javaTech = javaCreator.CreateTech();
    javaTech.GetFramework();

    // Create a DotNetCreator object
    Creator dotNetCreator = new DotNetCreator();

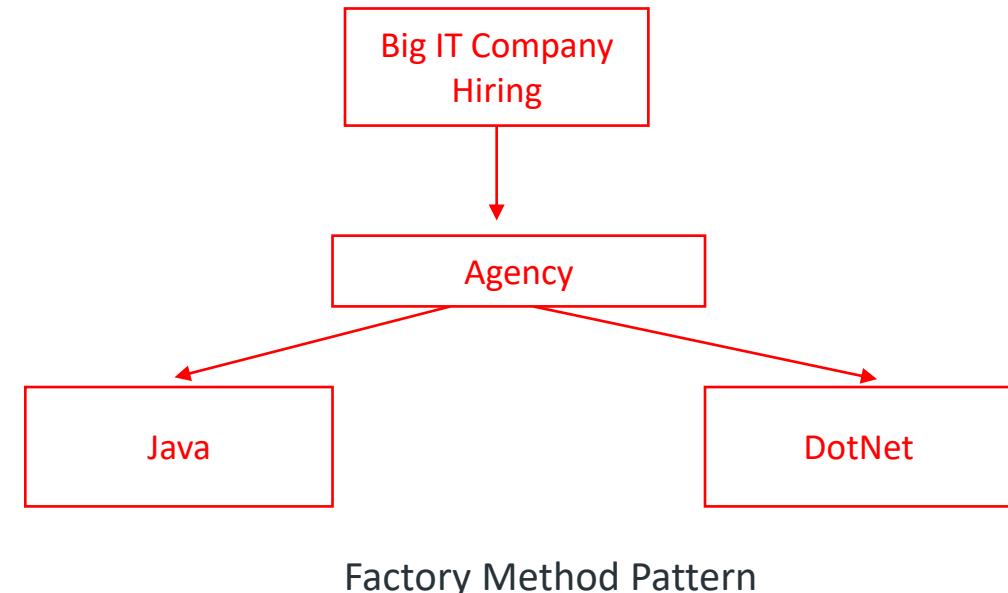
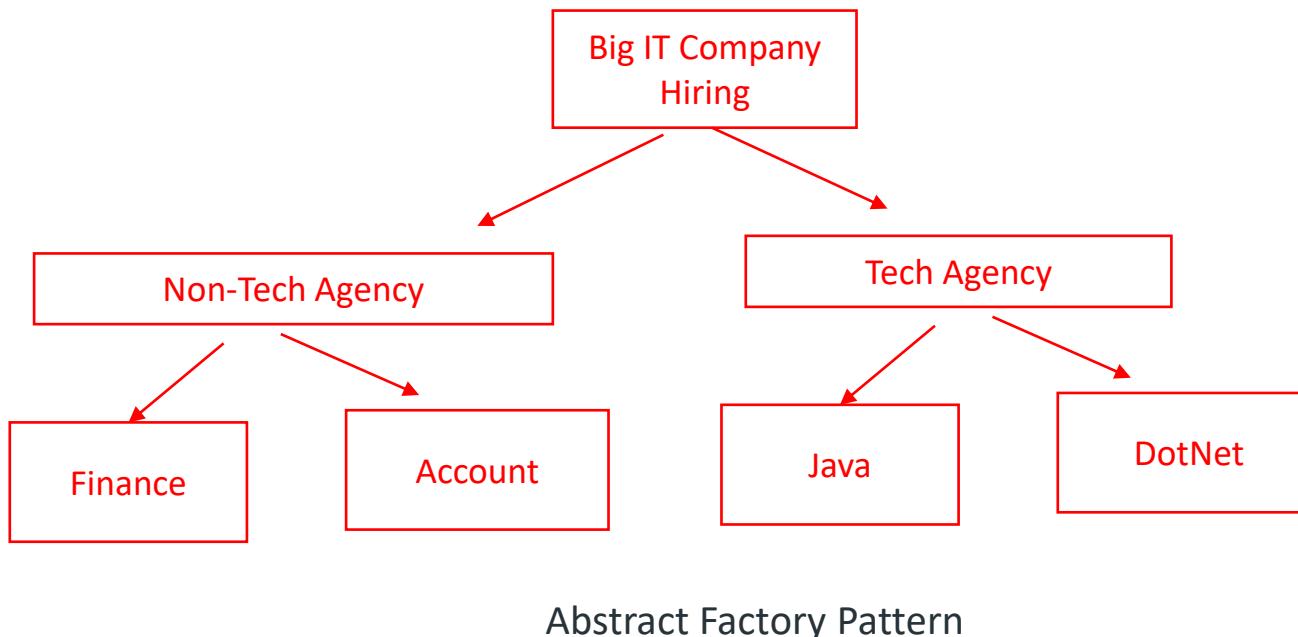
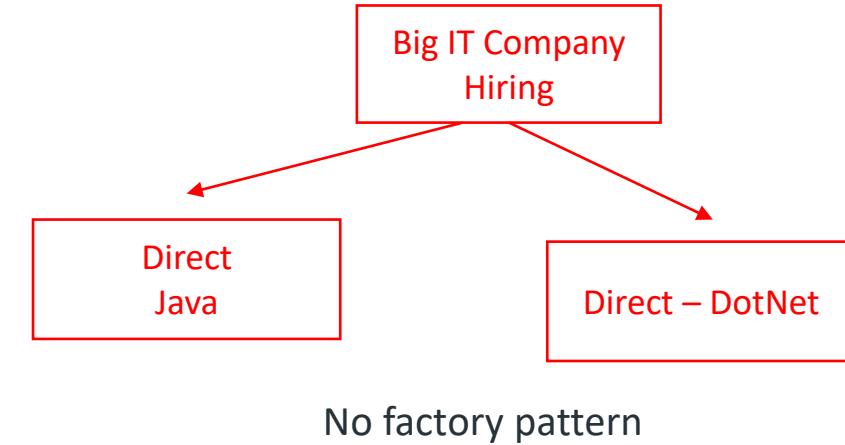
    // Call Factory Method to
    // create a DotNetTech object
    ITech dotNetTech = dotNetCreator.CreateTech();
    dotNetTech.GetFramework();

    Console.ReadLine();
}
//Output: Spring
//Output: Core
```

- ❖ Things to remember about Factory(simple factory and factory method) design patterns.
1. The Factory Pattern is a **creational** design pattern which manages object creation.
 2. Factory Pattern provides an **interface** or abstract class for creating objects, but lets the subclasses decide which class to instantiate.
 3. Or In simple factory pattern, one factory class is responsible for creating all the objects for its derived or sub class.
 4. Simple Factory pattern provides a single factory class that creates objects, while the Factory Method pattern uses inheritance and polymorphism, so it provide more flexibility and extensibility.



- The Abstract Factory pattern is a creational design pattern that provides an interface for creating related families of objects without specifying their concrete classes.
- Abstract Factory pattern can be thought of as a "factory of factories".





All the best for your interviews

Never ever give up

