**Inheritance:**

inheritance is a process in which one object acquires all the properties and behaviours of its parent object automatically.

**Derived class:**

the class which inherits the members of another class is called derived class

**base class:**

the class whose members are inherited is called base class.

**Polymorphism:**

The word polymorphism means having many forms. This can be achieved by Method Overloading & Method Overriding which is also called as compile time polymorphism & runtime polymorphism.

**Method Overloading:**

Having two or more methods with same name but different in parameters, is known as method overloading.

**Method Overriding:**

If derived class defines same method as defined in its base class, it is known as method overriding

**Encapsulation:**

Encapsulation is the concept of wrapping data into a single unit. It collects data members and member functions into a single unit called class. This can be restrict the alteration from the outside. Public, Private, Internal, Protected, Internal-Protected.

**Abstraction:**

Abstraction is the process to hide the internal details and showing functionality only. Abstract class is a class which is declared abstract and it can have abstract and non-abstract methods. This can be achieved by the Abstraction & Interface.

**Abstract Method:**

A method which is declared abstract and has no body is called abstract method. It can be declared inside the abstract class only. Its implementation must be provided by derived classes.

**Interface:**

It cannot have method body and cannot be instantiated. Its implementation must be provided by class or struct. The class or struct which implements the interface, must provide the implementation of all the methods declared inside the interface.

**Constructor:**

constructor is a special method which is invoked automatically at the time of object creation. It is used to initialize the data members of new object generally. It has 2 types Default Constructor & Parameterized Constructor.

**Static Constructor:**

static constructor is used to initialize static fields. It can also be used to perform any action that is to be performed only once. It is invoked automatically before first instance is created.

**Base:**

base keyword is used to access fields, constructors and methods of base class.

**Sealed:**

sealed keyword applies restrictions on the class and method. If you create a sealed class, it cannot be derived. If you create a sealed method, it cannot be overridden.

**Collection:**

collection represents group of objects. By the help of collections, we can perform various operations on objects such as Store, Update, Delete, Retrieve, Sort etc actions. Ex: List, Stack, Queue, LinkedList, Hashset, Dictionary, SortedDictionary etc.,

**Generics:**

generics allows us to write a class or method that can work with any data type.

public class GenericClass<T> {} || public void Show<T>(T msg) {}

**Reflection:**

reflection is a process to get metadata of a type at runtime. Type, MemberInfo, MethodInfo, PropertyInfo etc.,

**Anonymous Methods:**

Anonymous function is a type of function that does not has name. In other words, we can say that a function without name is known as anonymous function. Types: Lambda Expressions & Anonymous Methods.

**Delegates:**

Delegates are similar to pointers to functions, in C or C++. A delegate is a reference type variable that holds the reference to a method.

**asEnumerable vs asIQueryable:**

Enumerate means it will loop through the collection. This mean IEnumerable will loop through all elements where as IQueryable will loop based on condition only.

**Anonymous Method:**

An anonymous method is a method without a name. Anonymous methods in C# can be defined using the delegate keyword and can be assigned to a variable of delegate type.

**Extension Method:**

Extension methods allow us to inject additional methods without modifying or deriving the original class.

**Dictionary:**

A Dictionary stores Key-Value pairs where the key must be unique. Foreach or for loop is used for iterate a dictionary. Use dictionary indexer to access individual item.

**Dispose and Finalize:**

The main difference between dispose () and finalize () is that the method dispose() has to be explicitly invoked by the user whereas, the method finalize() is invoked by the garbage collector, just before the object is destroyed.

**Garbage Collection:**

When a class object is created at runtime, certain memory space is allocated to it in the heap memory. However, after all the actions related to the object are completed in the program, the memory space allocated to it is a waste as it cannot be used. In this case, garbage collection is very useful as it automatically releases the memory space after it is no longer required.

**Heap vs Stack:**

Reference types (classes, interfaces, delegates) are always allocated on the heap.Value types (derived from System.ValueType, e.g. int, bool, char, enum and any struct) can be allocated on the stack.

**Constant vs Read-only:**

Constant variables are declared and initialized at compile time. The value can't be changed afterward. Read-only is used only when we want to assign the value at run time.

**Value types & Reference types:**

A value type holds a data value within its own memory space. Reference type stores the address of the Object where the value is being stored. It is a pointer to another memory location.

**Object pool:**

An object pool is a container having objects ready to be used. It tracks the object that is currently in use, total number of objects in the pool.

**Is & As:**

"is" operator is used to check the compatibility of an object with a given type, and it returns the result as Boolean. "as" operator is used for casting of an object to a type or a class.

>> class BasicAuthAttr: Inherit from AuthorizationFilter (System.Web.Http.Filters)

>> Then override with OnAuthorization with parameters

Ex: Override void OnAuthorizatin(HttpActionContext actionContext)

>> If actionContext has no headers, send Unathorized status else check user credentials by actionContext having the access or not.

>> else just decode the actionContext headers as username:password (as token) and send for the other class to test the access authorization.

>> if token has access then create genericPrincipal which holds genericIdentity & roles.

>> else throw the unathorized status.

------------\*------------- basicAuthAttr is ready

>> decorate this attribute to the method level and test.

1. Client (Uname & Pw) >> Server.
2. Server validates (Token) >> Client.
3. Client (Token saved) >> Server.
4. Server (Token Val & Data) >> Clients