**Encapsulation**

It is the process of wrapping functions and data members together in a class like a capsule.

encapsulation is the wrapping up of class members.

holding of class members.

to an object's ability to hide data and behaviour that are not necessary to its user.

• The meaning of encapsulation, is “hiding sensitive data”

• Declare the variables as private. (data hiding).

Used to prevent unwanted change of data outside of the function

Encapsulation is implemented **by using access specifiers.**

**polymorphism**

Polymorphism is the one of the primary pillars of object oriented programming.

polymorphism allows you to invoke derived class methods through a base class reference.

polymorphism having many forms/Behaviours

ex:

+ operator is used for addition of two integers

10+20=30

it is also used for joining two strings

"spider “+” man"=Spiderman

Real Time Example: 1. **A person behaves as an employee in the office, that the same person behaves as a father in the house, that the same person behaves as a customer in the shopping malls**.

**Class**

Class is blue print of an Object.

ex- **apple, guava, banana**, etc. are the examples of object and collection of the fruits is the class

**Object**

object is an entity that has state and behaviour. Here, state means data and behaviour means functionality.

Object is a runtime entity; it is created at runtime.

**Inheritance**

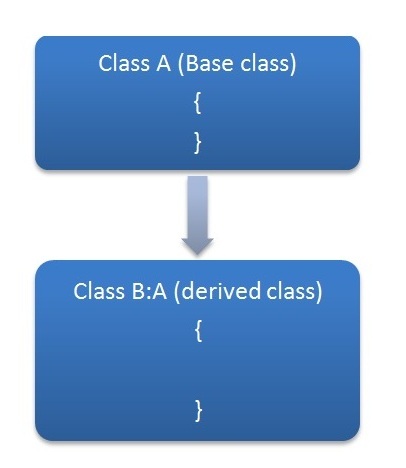
In C#, inheritance allows us to create a new class from an existing class. It is a key feature of Object-Oriented Programming (OOP)

The class from which a new class is created is known as the base class (parent or superclass). And, the new class is called derived class (child or subclass)

The derived class inherits the fields and methods of the base class. This helps with the code reusability in C#.

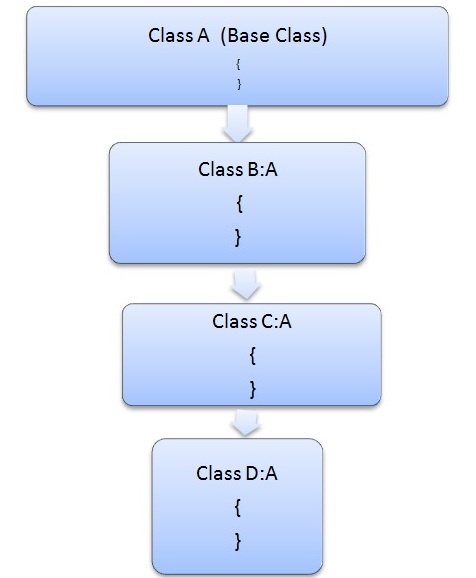
## Single inheritance in C#

It is the type of inheritance in which there is one base class and one derived class.



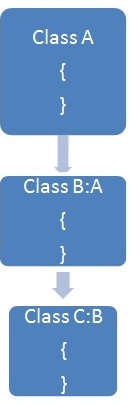
## Hierarchical inheritance in C#

This is the type of inheritance in which there are multiple classes derived from one base class. This type of inheritance is used when there is a requirement of one class feature that is needed in multiple classes.



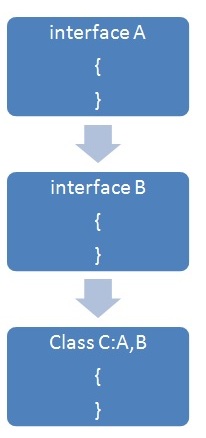
## Multilevel inheritance in C#

When one class is derived from another, this type of inheritance is called multilevel inheritance.



## Multiple inheritances using Interfaces

C# does not support multiple inheritances of classes. To overcome this problem, we can use interfaces.



**Method Overriding**

If the same method is present in both the base class and the derived class, the method in the derived class overrides the method in the base class.

This is called method overriding in C#.

// base class

class Animal

{

public virtual void eat()

{

Console.WriteLine("I eat food");

}

}

// derived class of Animal

class Dog : Animal

{

// overriding method from Animal

public override void eat()

{

Console.WriteLine("I eat Dog food");

}

}

class Program

{

static void Main(string[] args)

{

// object of derived class

Dog labrador = new Dog();

// accesses overridden method

labrador.eat();

}

}

**Method Overloading**

In C#, there might be two or more methods in a class with the same name but different numbers, types, and order of parameters, it is called method overloading.

For example:

void display() { ... }

void display(int a) { ... }

float display(double a) { ... }

float display(int a, float b) { ... }

class Program

{

// method with one parameter

void display(int a)

{

Console.WriteLine("Arguments: " + a);

}

// method with two parameters

void display(int a, int b)

{

Console.WriteLine("Arguments: " + a + " and " + b);

}

static void Main(string[] args)

{

Program p1 = new Program();

p1.display(100);

p1.display(100, 200);

Console.ReadLine();

}

}

**Constructor Overloading**

Similar to [method overloading](https://www.programiz.com/csharp-programming/method-overloading), we can also overload constructors. For constructor overloading, there must be two or more constructors with the same name but different

* number of parameters
* types of parameters
* order of parameters

class Car

{

// constructor with no parameter

Car()

{

Console.WriteLine("Car constructor");

}

// constructor with one parameter

Car(string brand)

{

Console.WriteLine("Car constructor with one parameter");

Console.WriteLine("Brand: " + brand);

}

static void Main(string[] args)

{

// call with no parameter

Car car = new Car();

Console.WriteLine();

// call with one parameter

Car car2 = new Car("Bugatti");

Console.ReadLine();

}

}

**Object**

Object is an instance of a class that is created dynamically.

**Abstract Class**

We use the abstract keyword to create an abstract class.

We cannot create objects of an abstract class, we must create a derived class from it.

Then we can access members of the abstract class using the object of the derived class.

An abstract class can have constructors as well.

**Abstract Method**

A method that does not have a body is known as an abstract method. We use the abstract keyword to create abstract

Abstraction is one of the important concepts of object-oriented programming. It allows us to hide unnecessary details and only show the needed information.

This helps us to manage complexity by hiding details with a simpler, higher-level idea.

Note: Unlike the C# inheritance, we cannot use virtual with the abstract methods of the base class.

This is because an abstract class is implicitly virtual.

**Nested Class**

In C#, we can define a class within another class. It is known as a nested class. For Example

// outer class

public class Car

{

public void displayCar()

{

Console.WriteLine("Car: Bugatti");

}

// inner class

public class Engine

{

public void displayEngine()

{

Console.WriteLine("Engine: Petrol Engine");

}

}

}

class Program

{

static void Main(string[] args)

{

// create object of outer class

Car sportsCar = new Car();

// access method of outer class

sportsCar.displayCar();

// create object of inner class

Car.Engine petrolEngine = new Car.Engine();

// access member of inner class

petrolEngine.displayEngine();

Console.ReadLine();

}

}

**Note**: We cannot access the members of the inner class using the object of the outer class.

**Sealed Class**

In C#, when we don't want a class to be inherited by another class, we can declare the class as **a sealed class**.

A sealed class cannot have a derived class. We use the sealed keyword to create a sealed class. For example,

sealed class Animal

{

}

// trying to inherit sealed class

// Error Code

class Dog : Animal

{

}

class Program

{

static void Main(string[] args)

{

// create an object of Dog class

Dog d1 = new Dog();

Console.ReadLine();

}

}

O/P error CS0509: 'Dog': cannot derive from sealed type 'Animal'

During method overriding, if we don't want an overridden method to be further overridden by another class, we can declare it as a **sealed method**.

Why Sealed Class?

We use sealed classes to prevent inheritance. As we cannot inherit from a sealed class, the methods in the sealed class cannot be manipulated from other classes.

It helps to prevent security issue

**Interface**

An interface is similar to abstract class. However, unlike abstract classes, all methods of an interface are fully abstract (method without body).

We use the interface keyword to create an interface. For example,

interface IPolygon

{

// method without body

void calculateArea();

}

 We cannot use access modifiers inside an interface.

 All members of an interface are public by default.

 An interface doesn't allow fields.

 Similar to abstract classes, interfaces help us to achieve **abstraction in C#**.

 Interfaces are used to achieve multiple inheritance in C#.

 Interfaces provide **loose coupling**(having no or least effect on other parts of code when we change one part of a code).

We cannot create objects of an interface. To use an interface, other classes must implement it.

**Note**: We must provide the implementation of all the methods of interface inside the class that implements it.

**Using**

We use the using keyword to import external resources (namespaces, classes, etc) inside a program. For example :

// using System namespace

using System;

namespace Program

{

class Program1

{

static void Main(string[] args)

{

Console.WriteLine("Hello World!");

}

}

}

**Type Conversion**

The process of converting the value of one type (int, float, double, etc.) to another type is known as type conversion.

There are two basic types of type conversion:

1. **Implicit Type Conversions**
2. **Explicit Type Conversions**
3. **Implicit Type Conversions**

Generally, smaller types like int (having less memory size) are automatically converted to larger types like double (having larger memory size).

**Note**: In implicit type conversion, smaller types are converted to larger types. Hence, there is no loss of data during the conversion.

1. **Explicit Type Conversions**

Generally, larger types like double (having large memory size) are converted to smaller types like int (having small memory size).

**Note**: The explicit type conversion is also called type casting.

|  |  |
| --- | --- |
| **Method** | **Description** |
| ToBoolean() | converts a type to a Boolean value |
| ToChar() | converts a type to a char type |
| ToDouble() | converts a type to a double type |
| ToInt16() | converts a type to a 16-bit int type |
| ToString() | converts a type to a string |

| **Array** | **ArrayList** |
| --- | --- |
| Must include **System** namespace to use array. | Must include **System.Collections** namespace to use ArraList. |
| Array Declaration & Initialization: int[] arr = new int[5] int[] arr = new int[5]{1, 2, 3, 4, 5}; int[] arr = {1, 2, 3, 4, 5}; | ArrayList Declaration & Initialization: ArrayList arList = new ArrayList(); arList.Add(1); arList.Add("Two"); arList.Add(false); |
| Array stores a fixed number of elements. The size of an Array must be specified at the time of initialization. | Array List grows automatically and you don't need to specify the size. |
| Array is strongly typed. This means that an array can store only specific type of items\elements. | Array List can store any type of items\elements. |
| No need to cast elements of an array while retrieving because it is strongly typed and stores a specific type of items only. | The items of Array List need to be cast to an appropriate data type while retrieving. So, boxing and unboxing happens. |
| Performs faster than ArrayList because it is strongly typed. | Performs slows because of [boxging and unboxing](https://www.tutorialsteacher.com/articles/boxing-unboxing-in-csharp). |
| Use static helper class Array to perform different tasks on the array. | ArrayList itself includes various utility methods for various tasks. |

# **Enum**

An enum is a special "class" that represents a group of **constants**(unchangeable/read-only variables).

using System;

namespace MyApplication

{

enum Level

{

Low,

Medium,

High

}

class Program

{

static void Main(string[] args)

{

Level myVar = Level.High;

Console.WriteLine(myVar);

Console.ReadLine();

}

}

}

**Try Catch Finally**

const numerator= 100, denominator = 'a';

try {

console.log(numerator/denominator);

console.log(a);

}

catch(error) {

console.log('An error caught');

console.log('Error message: ' + error);

}

finally {

console.log('Finally will execute every time');

}

* **try** − A try block identifies a block of code for which particular exceptions is activated. It is followed by one or more catch blocks.
* **catch** − A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The catch keyword indicates the catching of an exception.
* **finally** − The finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.

**Garbage Collection**

Garbage collection is the process of freeing up memory that is captured by unwanted objects

**Middleware** is a piece of code in an application pipeline used to handle requests and responses.

Eg

// This middleware is used to authorizes a user to access secure resources.

app.UseAuthorization();

Public – the code is accessible for all classes.

Private - the code is accessible for with in the same class.

Protected – the code is accessible for with in the class, or in a class that is inherited from that class.

Important Points

* A Class with private constructor cannot be inherited.
* We cannot create an object of the class which has private constructor.

*What is the use of static class?*

* *A static class is a class which object cannot be created, and which cannot be inherited. That is for code security.*
* *All Members are static inside the static class.*
* *Static classes are used as containers for static members like methods, constructors and others.*

*What are the advantages of OOPS in C#?*

1. *Reuse The Code using Inheritance.*
2. *Flexibility of code using* Polymorphism*. (Method Overloading, Method Overriding)*
3. *Secure Application by using Encapsulation.*
4. *Easy to scalable from small to large applications.*

*What are the limitations of OOPS?*

1. *OOPS are not suitable for small applications. (to implement oops concept you need more time)*

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

*Using*

*Public abstract class name*

*Private class name*

*Public static class name*

*What are namespaces?*

*A Namespace is a container for a set of related classes and other types.*

***Static Constructor***

* *Static constructor is used to be called before any static member of a class is called.*
* *Static constructor cannot have parameters or access modifiers.*

***Copy constructor***

* *The constructor which creates an object by copying variables from another object is called a copy constructor.*
* *used to create a new object as a copy of an existing object*

*Difference between Sealed Class and Static Class*

*You can create an object of sealed class, but you cannot create an object of static class.*

***How to return more than one value from a method in C#?***

* *By using ref or out parameter.*
* *By using Tuples.*

*Eg,*

(int Age, string Name) employee1 = (23, "Yohan");

Console.WriteLine($"{employee1.Name} is {employee1.Age} years old");

* *By using a List,ArrayList(any collection) in the return type.*

***What is params keyword in C#?***

* *Params keyword is used as a parameter which can take variable number of parameters.*

// User defined function

public void Show(params int[] val) // Params Paramater

{

for (int i = 0; i < val.Length; i++)

{

Console.WriteLine(val[i]);

Console.ReadKey();

}

}

static void Main(string[] args)

{

Program p = new Program();

p.Show(1, 2, 3, 5, 4, 3, 2);

}

***Why Partial Class?***

*Multiple developers can work simultaneously with a single class in separate files.*

***List***

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

***Dictionary***

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

***What Is the use of IEnumarable?***

* *IEnumarable interface is used when want to iterate among our collection classes using foreach loop.*

***Yield***

*A strong feature of C# is the “yield” keyword, which* ***enables programmers to build iterator blocks that may be used to loop around a group of objects****.*

***LINQ***

Language-Integrated Query (LINQ) is the name for a set of technologies based on the integration of query capabilities directly into the C# language

* *From variable in object\_set.*
* *Select variable.*

# **SQL Stored Procedures**

Stored procedure is a set of statement(s) that perform some defined actions. We make stored procedures so that we can reuse statements that are frequently.

Stored procedures are similar to functions in programming. they can accept parameters, and perform operations when we call them.

1. Declarative Part - Creation
2. Executable Part - This is a mandatory part and contains statements that perform the designated action.
3. Exception-handling : This is again an optional part. It contains the code that handles run-time errors.

CREATE [OR REPLACE] PROCEDURE procedure\_name

[(parameter\_name [IN | OUT | IN OUT] type [, ...])]

{IS | AS}

BEGIN

< procedure\_body >

END procedure\_name;

**Functions** − These subprograms return a single value; mainly used to compute and return a value.

**Sql Index**

Indexes are used to retrieve data from the database more quickly than otherwise. The users cannot see the indexes, they are just used to speed up searches/queries.

CREATE INDEX index\_name

ON table\_name (column1, column2, ...);

**Primary key**

A primary key is used to ensure data in the specific column is unique. It is a column cannot have null values.

**Foreign key**

A foreign key is a column or group of columns in a relational database table that provides a link between data in two tables.

 [UNIQUE](https://www.w3schools.com/sql/sql_unique.asp) - Ensures that all values in a column are different

 [PRIMARY KEY](https://www.w3schools.com/sql/sql_primarykey.asp) - A combination of a NOT NULL and UNIQUE. Uniquely identifies each row in a table.

**Different Types of SQL JOINs**

* **(INNER)JOIN**: Returns records that have matching values in both tables
* **LEFT (OUTER) JOIN**: Returns all records from the left table, and the matched records from the right table
* **RIGHT (OUTER)** **JOIN**: Returns all records from the right table, and the matched records from the left table
* **FULL (OUTER) JOIN**: Returns all records when there is a match in either left or right table

What is ASP.NET?

ASP.NET is a Server side technology used for developing dynamic web sites and web applications on the internet.

What platform does asp.net use?

ASP.NET is a cross-platform which works on the .NET Framework and runs on Linux, macos, windows, and Docker.

C# is a programming Language developed by Microsoft that runs on the dotnet framework.

Vb.net is a modern object oriented programming language developed by Microsoft to combine the features of CLR and Dotnet.

What type of Library is used in asp.net?

A library is a pre-coded Object oriented programming template collections is used to developing web based applications. ASP.NET Uses Web Library.

**.Net Framework**

What is the purpose of web API?

Web API (Application Programing Interface) provides interaction between software applications.

What is the difference between Web API and MVC Controller?

|  |  |
| --- | --- |
| **Web API Controller** | **MVC Controller** |
| Web API Controller derives from SYSTEM.WEB.HTTP.APICONTROLLER Class. | MVC controller derives from SYSTEM.WEB.MVC.CONTROLLER class. |
| Web API controller does not give view support | ASP.NET MVC gives view support. |

XML is WSDL (Web Service Description Language)

**Differences between ViewState and SessionState:**

|  |  |
| --- | --- |
| ViewState | SessionState |
|  |  |
| Maintained at page level only. | Maintained at session level. |
| View state can only be visible from a single page and not multiple pages. | Session state value availability is across all pages available in a user session. |
| It will retain values in the event of a postback operation occurring. | In session state, user data remains in the server. Data is available to user until the browser is closed or there is session expiration. |
| Information is stored on the client’s end only. | Information is stored on the server. |
| used to allow the persistence of page-instance-specific data. | used for the persistence of user-specific data on the server’s end. |
| ViewState values are lost/cleared when new page is loaded. | SessionState can be cleared by programmer or user or in case of timeouts. |

ViewState and SessionState helps to maintain states.

Important

* View state values are accessible with the same page.
* Session variables can be accessed across pages.
* View state information is stored in hidden fields in client side.
* View state is base 64 encoded so not good for critical information.
* Session data is stored on the server and keys are stored in cookies files.
* If cookie Is disabled, then session id key passed via query string.
* As data stored on the server the critical information can be stored.
* View state client side and session is server side.

|  |  |  |
| --- | --- | --- |
|  | **VIEW STATE** | **SESSION STATE** |
| Data storage location | Browser Hidden fields | On server |
| Accessible | In the same page. | Across pages |
| Use cookies | No (NA) | Yes. If cookie is disabled will use URL Query string. |

**Usage:**

* **SessionState:** It can be used to store information that you wish to access on different web pages.
* **ViewState** It can be used to store information that you wish to access from same web page.

**PostBack**

* PostBack happens when data is sent to the server.
* IsPostback helps you to identify if PostBack has happened or not.
* AutoPostBack helps to you post as soon as data changes on the control.

### **What is IIS?**

IIS stands for Internet Information Services. It is created by Microsoft to provide Internet-based services to ASP.NET Web applications.

**.Net Framework**

**NET was supposed to make it easy both to write and consume web services**. In particular, it was supposed to make it easier to call the web services that Microsoft was going to provide, and that everyone would then use: the ". NET My Services".

. NET Framework is **a software development framework for building and running applications on Windows**.

**Authentication** is the process of determining a user’s identity.

**Authorization** is the process of determining whether a user has access to a resource.

In ASP.NET Core, authentication is handled by the **authentication service, IAuthenticationService,** which is used by authentication middleware.

**Q.1) What is a web.config file.**  
Web. Config is an application configuration file of asp.net application which is written in XML Format.

### **What is caching? Explain.**

Caching is the technique which facilitates you to store frequently used items in memory so that they can be accessed more quickly.

### **Explain uses of Global.asax**

Global.asax is used to handle high-level applications like Application\_Start, Session\_End, Session\_Start.

Global file is also used to execute the application-level events and set application-level variables.

**Dependency Injection (DI)**

* Dependency injection is a software design pattern that allows us to develop **loosely coupled application.**
* This is a process in which we are injecting the object of a class into another class that depends on that class
* Dependency injection there are three types

Constructor Injection

Property Injection

Method Injection

