

**What is C#?**

C# is one of the OOPS programming language used for developing .net Framework Applications it is developed by Microsoft.

**What is .NET?**

.NET is a framework that provides a programming guideline that can be used to develop a wide range of applications—from web to mobile to Windows-based applications. The .NET framework can work with several programming languages such as C#, VB.NET, C++ and F#.

**What are the types of Data Types?**

There are two types of data types they are

1) Value type also called primitive type also these are stored in Stack Memory.

such as: int, long, short, bool, byte, char, Date, double, float etc.,

2) reference type also called as non-primitive type also these are stored in Heap Memory.

Such as: classes, string, array

**What are Variables?**

A variable is a way to store data in the computer's memory to be used later in the program. C# is a type-safe language, meaning that when variables are declared it is necessary to define their data type.

EX: 1) `int i=10;`

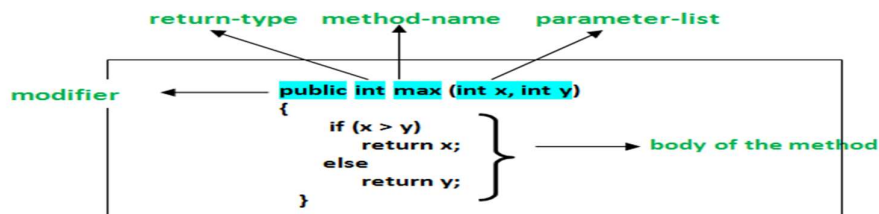
2) `string s="Hello";`

3) `char ch='A';`

**What are Methods?**

A method contains set of statements which gets executed only when we call the method

**Method syntax:**

**What is return type?**

It is data type specified in method signature before Method name that a method must give out value to it after method is called.

**What is return statement?**

Return statement is the one that assigns value or returns a value of return type\* during method body execution.

**What is Type Casting?**

Converting data from one datatype to another datatype is called Type Casting and are of 2 types

**Implicit Type Casting (automatic):** -> Converting data from smaller datatype to larger type.

Ex: char -> int -> long -> float -> double

```
int myInt = 9;
double myDouble = myInt;
```

**Explicit Type Casting (manual)** :-> Converting data from larger datatype to smaller type.

Ex: double -> float -> long -> int -> char

```
double myDouble = 9.78;
int myInt = (int)myDouble;
```

Here the compiler will not make the conversion automatically because there is a chance of data loss.

Int capacity is 4 bytes where-in double capacity is 8bytes

so if we are okay with data loss, we can do it manually.

**What is Boxing?**

Boxing is a process of converting value type to reference type is called Boxing

```
int i = 123;
object o = i; // Boxing happens manually no need to mention the datatype
```

**What is Un-Boxing?**

Unboxing is a process of converting reference type to value type is called unboxing

```
int o = 123;
object i = (int)o; // unboxing
```

**What is Up-Casting?**

Creating Subclass object and Casting into the super class reference is called Upcasting

Ex: Animal a=new Dog();

**What is Down-Casting?**

Creating Super Class reference and Casting into the Sub Class reference is called Down-casting

Ex: Dog d=(Dog)new Animal();

**What is method signature?**

Method signature consists of access modifier, type of method, return type, method Name, parameters.

**What is method body?**

Method body consists of block of statements wrapped under curly braces {}. And it must contain return type. (optional if return type is void).

**What are Parameters?**

Parameters are nothing but variables defined with method signature under the parenthesis () we can create multiple parameters separated by comma

whoever calls the method that has parameters then callers should compulsorily pass the values/arguments to described parameters.

**Note: Practice on Basic Programs with instance usage and method usage, before proceeding plus learn how to use all of the statements\***

Refer this link: <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/statements-expressions-operators/statements>

### what are the types of variables?

There are 2 types of variables global variables and local variables

- 1) **Global variables** are created inside the class and can be anywhere within that class.
- 2) **Local variables** are created inside the method which cannot be access outside that method's body.

### What is variable shadowing?

Variable shadowing means when we create a new method in child class with same as which is available in Parent class but we use new keyword while defining the method makes it a shadow over the parent class method

```
class A
{
    public int Foo() { return 5; }
    public virtual int Bar() { return 5; }
}
class B : A
{
    public new int Foo() { return 1; } // using new keyword here will act as shadow
    for the Foo() from Parent class
    public override int Bar() { return 1; }
}
```

### What are the types of Methods?

There are 2 types of methods static and non-static

### What is Constructor?

It is a special type of method which doesn't need return type neither return statement. and the constructor's name must be same as the class in which we create the constructor

When we call the constructor with **new** keyword it creates an Object of the class in heap memory.

### What are the types of Constructors?

**Default constructor:** if we don't create any constructor in a class, by-default compiler creates a default constructor but, if you manually create any constructor compiler will not create any constructor.

```
ex: public class MyClass
{
    public MyClass()
    {
    } // this is the constructor compiler injects to our class if there is no
    constructor defined
}
```

- 1) **Parameterized Constructor:** we can create constructors with parameters then while calling this kind of constructors we need to give arguments to those parameters
- 2) **Static Constructor:** we can only create static constructors in static classes only.
- 3) **Private constructors:** these constructors cannot be called from outside the class in which it is created.

4) **Copy Constructor:**

A parameterized constructor that contains a parameter of same class type is called as copy constructor.

Ex for Copy constructor:

```
public employee(employee emp)
{
    Name = emp.name;
    age = emp.age;
}
```

### What Is class?

Class is a blueprint or a template which can have states and behaviors ex: variables and methods which can be used to create objects.

Real World **Ex:** a blueprint of a house by which we can build multiple real houses.

### What is Object?

Objects are real world entities which contains states and behaviors and are instances of class

We can create an instance/object of a class using constructor with new keyword.

### What is Namespace?

Namespace is just a folder/directory which can have classes, interfaces etc. and we can group the classes, interfaces etc.,

### Difference between static vs non-static members?

**static members** are class specific which can be called with creating objects

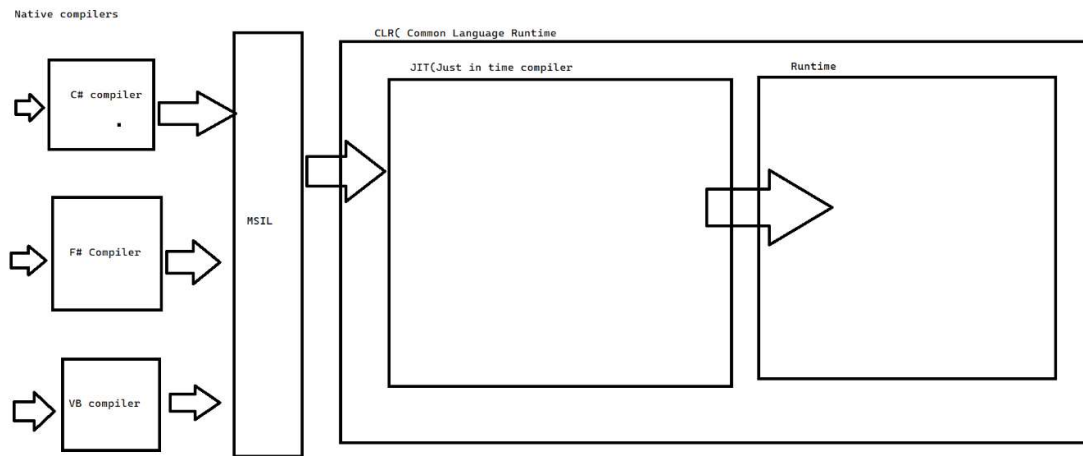
**non-static members** are Object specific which means we need to create object of these member's existence.

### What are the types of classes?

- 1) **Partial class**  
if a class has huge code base, we can share the code base with 2 files with same class name. compiler will merge these 2 classes and their code to single class during compile-time.
- 2) **Sealed class**  
If we don't want someone to inherit the class, we use sealed class.
- 3) **Static class**  
static class cannot have non-static members(variables and methods), it can have only one constructor which must be default/ zero params constructor.
- 4) **Abstract class**  
cannot create an instance of abstract class directly.  
it can have concrete method as well as abstract method.

**Very Important\***

Code Compilation and Execution process of any program in .net framework



**.Net architecture**

**What is native compiler?**

these compilers compile the programs written in their own native language to MSIL.

**What is MSIL?**

Micro-Soft Intermediate Language native compilers compile native language to MSIL because CLR only Accepts MSIL

**What is CLR?**

Common Language Runtime .net engine/component which only accepts MSIL and it contains 2 important sub-components they are JIT and Execution runtime.

**what is JIT?**

JUST-IN-TIME compiler which resides in CLR which compiles the MSIL and converts it into bytecodes or machine-readable instructions and feed to Execution Runtime where the Program Execution Starts.

**Note: .net has 2 step compilation process 1<sup>st</sup> native compilation 2<sup>nd</sup> JIT compilation.**

**What is Assembly?**

assembly is nothing but the MSIL which will be in the form of EXE or DLL

**What is DLL?**

Dynamic Link Library is a library project where we can build libraries which can be re-used in other projects

**What is EXE?**

Exe is nothing but an assembly which is an executable file in windows environment.

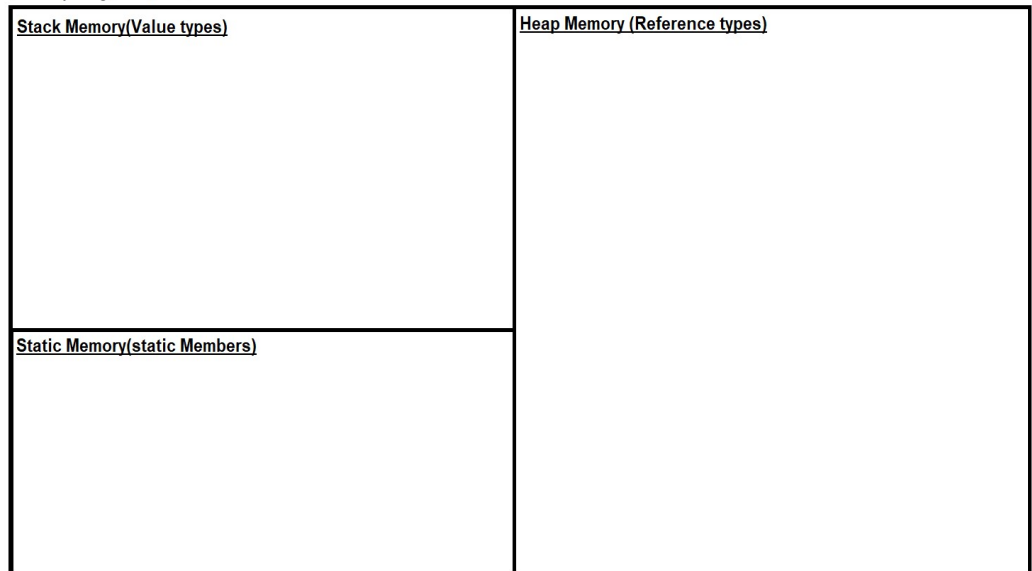
**Very Important\***

How C# Code Execution happens from Coding to Execution to result with **Memory Diagram\*\***

Example Code and Explanation required.

Plain Memory Diagram:

Plain Memory Diagram



Plain Memory Diagram

**Why do we need Main method?\*\***

Static Members are Loaded even before the class loading Happens

CLR is programmed to search for Main method in Static Memory and call Main method right after class loading happens

**Why Main method needs to be static?\*\***

if the Main Method is non-static then we will have to create an instance of the class that contains main method but who will create an instance of that class hence Main method static.

## Object Oriented Programming Concepts

### What are the OOPS concepts?

Object? Class?

Inheritance,

Polymorphism,

Abstraction,

Encapsulation

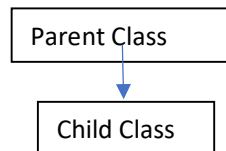
### Inheritance

#### What is Inheritance?

Inheritance is obtaining the states and behaviors from one class to another/ parent class to child class is nothing but inheritance.

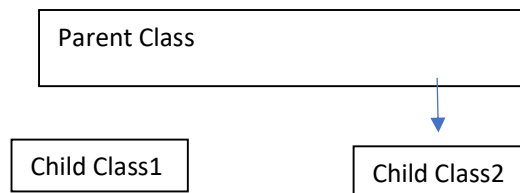
#### Types of Inheritance?

##### Single-Level Inheritance (Possible)

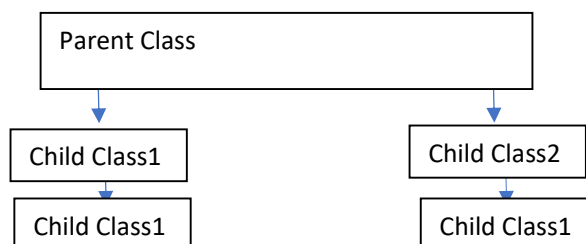


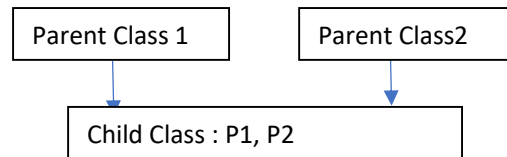
##### Multi-Level Inheritance

(Possible)



##### Hierarchical Inheritance (Possible)



**Multiple Inheritance (Not possible with class but possible with interface)****What is Object class in C#?**

In C# we have a built-in class called Object which is available in System namespace and every class we create in C# **by default** inherits object class.

**Note: Object of a class is different and Object class is different**

**What is constructor Chaining?**

Let's say: we have Parent and a child class which inherits parent class

```

class Program
{
    static void Main(string[] args)
    {
        Console.WriteLine("Hello World!");
        Child childInstance = new Child();
        Console.WriteLine(childInstance.Name) ;
        Console.ReadLine();
    }
}

public class Parent
{
    public Parent()
    {
    }
    public string Name;
}

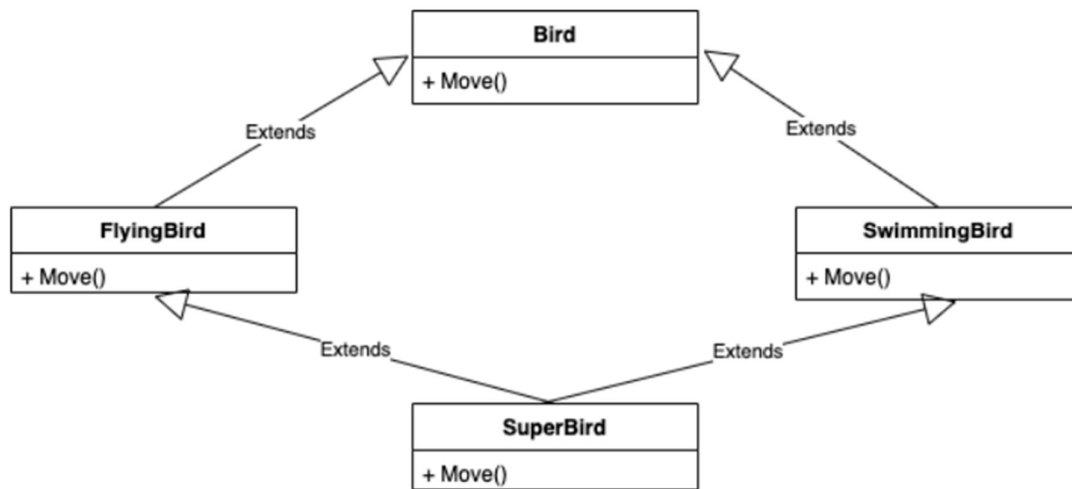
public class Child: Parent
{
    public Child():base()
    {
    }
}
  
```

Here child constructor by default calls its parent class constructor.

So before even creating child object parent object gets created.



**What is Diamond Problem and Constructor chaining Problem regards to multiple inheritance in class?**



Now the child constructor fails to decide which parent's Move() has to call.. this is called Diamond Problem.

#### Reason for Constructor chaining problem:

The child class constructor is instructed to call its base class constructor by default but if it inherits 2 or more classes it gets confused on which base class constructor it needs to call...

**EX:**

```

public class ParentA
{
    public ParentA()
    {
    }
}
public class ParentB
{
    public ParentB()
    {
    }
}
public class Child : ParentA, ParentA
{
    public Child()
    {
        // this constructor don't know which is its base class because there are 2
        parents
    }
}
  
```

#### What if I don't want my class to be inherited by other classes?

we can use sealed keyword which will restrict it to be inherited by other classes.

#### What is Method Over-Loading?

Creating multiple methods in same class with same name but change in the signature is nothing but method over-loading

change is signature in the sense change in number of Parameters or change in datatype of parameters or change in sequence of parameters.

### What is Method Overriding?

An override method provides a new implementation of the method inherited from a base class.

Code:

```
class Program
{
    static void Main(string[] args)
    {
        // prints: Parent works in Railway Dept
        Parent parent = new Parent();
        parent.Work();

        //prints: Child is goes to school
        Parent parent1 = new Child();
        parent1.Work();
        Console.ReadLine();
    }
}

public class Parent
{
    public virtual void Work()
    {
        Console.WriteLine("Parent works in Railway Dept");
    }
}

public class Child: Parent
{
    public override void Work()
    {
        Console.WriteLine("Child goes to school");
    }
}
```

Note: virtual keyword In parent class method states that the method is allowed to be overridden in the child class

if we don't specify virtual keyword compiler will not allow us to override the method in child class.

### Difference between Method Overloading and Method Overriding

Method Overloading	Method Overriding
Method overloading is a compile-time polymorphism.	Method overriding is a run-time polymorphism.
It occurs within the class.	it occurs only in the inheritance
In method overloading, the return type can or cannot be the same, but we just have to change the parameter.	In method overriding, the return type must be the same
No need to use any keyword to create method overloading	Need override keyword to create method overriding.
we can overload static methods	we can't override static methods

**What are Access Modifiers?**

access modifiers are used to specify or restrict the accessibility of a member

**public:** it can be accessed anywhere.

**protected:** it can only be accessed in inherited members.

**Private:** it can only be accessed within the class where member is defined.

**Internal:** it can be accessed within the assembly.

**Private protected:** A private protected member is accessible by types derived from the containing class, but only within its containing assembly.

**Protected internal:** Access is limited to the current assembly or types derived from the containing class.

## Polymorphism

### What is Polymorphism?

Doing the same thing in many forms is nothing but Polymorphism. There are two types

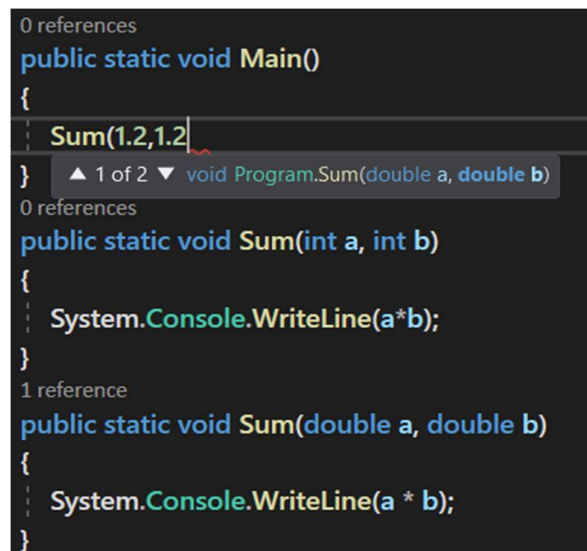
#### 1) Compile-time polymorphism

Whenever we call a method then which method overload to call based on the arguments passed is decided by compiler hence it is called compile-time polymorphism. It is also called as Early Binding or static binding.

things available in Intellisense are nothing but Early Binding.

#### Ex: Method Overloading

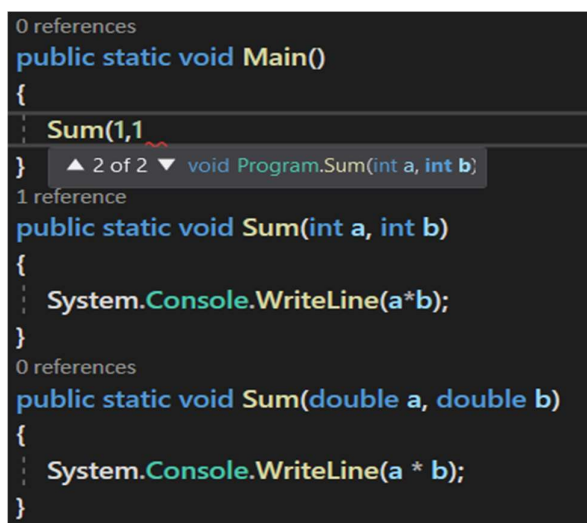
Here when we pass arguments in decimal compiler decided to call Sum method that has double datatype as parameter



```

0 references
public static void Main()
{
    Sum(1.2, 1.2)
}
▲ 1 of 2 ▼ void Program.Sum(double a, double b)
0 references
public static void Sum(int a, int b)
{
    System.Console.WriteLine(a*b);
}
1 reference
public static void Sum(double a, double b)
{
    System.Console.WriteLine(a * b);
}
  
```

Here when we pass arguments in int compiler decided to call Sum method that has int datatype as Parameter



```

0 references
public static void Main()
{
    Sum(1, 1)
}
▲ 2 of 2 ▼ void Program.Sum(int a, int b)
1 reference
public static void Sum(int a, int b)
{
    System.Console.WriteLine(a*b);
}
0 references
public static void Sum(double a, double b)
{
    System.Console.WriteLine(a * b);
}
  
```

**2) Runtime polymorphism**

- the method name is same in base class as well as Derived class, but which method's logic must execute at invocation is decided by the Runtime. i.e., Method overriding.
- if the method is not overridden then the Base class logic is executed.
- if the method is overridden then the Derived class logic is executed and this binding of method with the logic is decided by the runtime.
- it is also called as late binding or Dynamic Binding

**Ex: for Runtime Polymorphism is Method Overriding**

-----End of Polymorphism-----

## **Abstraction**

### **What is Abstraction?**

Abstraction is hiding the internal implementation and providing/exposing only the necessary functionality

### **Ways to Achieve abstraction:->**

- 1) **Interface**  
with interface we can achieve 100% abstraction.
- 2) **Abstract class**  
With abstract class we can achieve partial abstraction.

### **What is Interface?**

Interface is a template where we can create declared methods with no body or definition

**Or**

An interface is a completely "abstract class", which can only contain abstract methods and properties (with empty bodies):

<b>Interface</b>	<b>Abstract class</b>
We use interface keyword to define an interface	We use abstract key before class keyword to define abstract class
We cannot create instance of an interface	We cannot create object but can be accessed by inherited classes
It cannot have concrete method	It can have concrete methods as well as declared(abstract) methods
Multiple inheritance is possible.	Not possible
We can achieve 100% abstraction	We can achieve partial abstraction

### **Why do we need interface?**

It is used to achieve loose coupling, Unit testing becomes easy with interfaces

To enforce business rules, we need interfaces

### **Why do we need abstract class?**

- as interface only provides methods with declarations.
- in abstract class we can write the concrete methods which will be common for all the sub classes.
- if we need some functions or variables to be common for all the derived classes then we use abstract class.
- to instantiate some fields in the abstract class before sub class we use the abstract class constructor.

### Example Usage of abstract class

```
using System;
abstract class BaseClass
{
    public void SaveTheDataToDataBase(string data)
    {
        //definition to save the data-----
    }

    public abstract void ConnectToDatabase(string connectionString);
}
class SqlServerDbSource : BaseClass
{
    public override void ConnectToDatabase(string connectionString)
    {
        //Logic to connect to SqlServer-----
    }
}
class MySqlDBSource : BaseClass
{
    public override void ConnectToDatabase(string connectionString)
    {
        //-----Logic to connect to mySql-----
    }
}
```

## Encapsulation

### What is Encapsulation?

is a concept that binds the data member with the functions to make it accessible and to avoid misuse of data member.

Ex:

```
private string name;
public string Name { get { return name; } set { value = name; } }
```