**OOPS Concept**

Object Oriented Programming (OOP) is a programming model where programs are organized around objects and data rather than action and logic.

* Abstraction

To represent only essential feature without showing background details. *Abstract class is used to achieve abstraction*.

Eg calling in phone

* Encapsulation

Wrapping up of data members and methods into a single unit. To protect info from another object.

Eg Bag,Emp info

* Inheritance

Inheriting properties from another class.

1. Single inheritance A->B
2. Multilevel A->B->C
3. Hierarchical A->B, A->C, A->D
4. Multiple is only possible through interfaces. A->C,B->C
5. Hybrid is only possible through interfaces. A->B->D, A->C->D

* Polymorphism

Multiple forms of a single object.

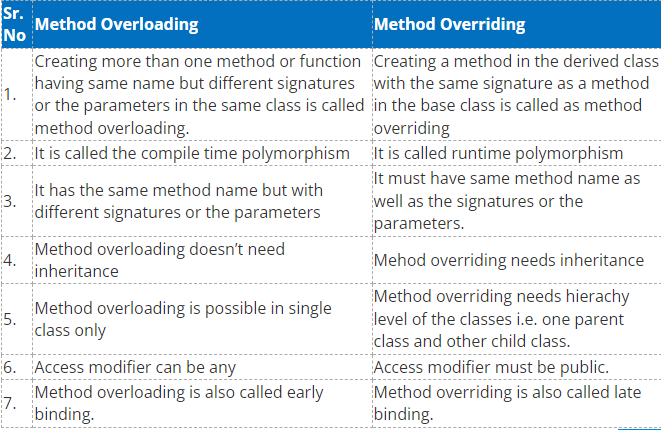
Eg phone

1. Compile time or static-Method **overloading**
2. Run time or dynamic or late binding **overriding**

**Virtual** keyword is used in **base** class where we want to make method overriden.

**Override** keyword is used in child class to override the method of base class

**Base** keyword is used to call the base class method



**Abstract Class**- atleast one abstract method(only declaration no implementation)

**Interfaces**- Only declaration of all members. Only public access modifier allowed.



When abstract and when interface

1. when constructor is required.

2. to implement multiple inheritance.

3. interfaces are always public so to implement abstraction we can use abstract class.

4. In interfaces we need to implement all methods while in abstract class only abstract methods needs to be implemented.

5. Variables cant be declared.

Mutable- *StringBuilder* is a mutable type, that means we are using the same memory location and keep on appending/modifying the stuff to one instance.  
Immutable- *Strings* are immutable, which means we are creating new memory everytime instead of working on existing memory.

* If a string can change (example: lots of logic and operations in the construction of the string) then using a StringBuilder is the best option.

Immutable Class means the variables are readonly means we can assign them only once.

== compares reference identity

.Equals() method compares data it will throw error if string is null

**Static** **Key**

* *Static class*-only static members,static methods and static constructor are allowed. Object cannot be created for static class.
* *Static variable-* scope remains until end of program(single copy is created and shared). Variable are accessed using className.Variabe
* *Static method-* static fields are directly accessed by the static method without class name whereas non-static fields require objects.
* *Static Constructor-* It can’t be called directly. No  access modifiers or any parameters.It is called automaticallycalled only once to initialize the class when the first instance created. Static Constructor has to be invoked only once in the class.

Checked exceptions are not present in c#

Const should be assigned while declaration only readonly can be assigned in constructor also.Const fields cannot be declared in methods while readonly can be declared.

**CONST**

const values assigned during declaration. Cannot be reassigned.

**READONLY**

readonly can be assigned during declaration or in non static constructor.

**STATIC READONLY**

readonly static can be assigned during declaration and can only be changed through a static constructor

<https://www.c-sharpcorner.com/UploadFile/c210df/difference-between-const-readonly-and-static-readonly-in-C-Sharp/>

Collections in c#

<https://www.tutorialsteacher.com/csharp/csharp-collection>

Explicit type variable(int a=10)

Implicit type variable(var a=10)

Serialization and Deserialization(check created program)

Entity Framework

1. Schema First approach-Database created first
2. Model First- Models are created from visual studio
3. Code First

Inversion of control- Class does not create which class to use. it depend on something else to classify which class it should use.

Eg-There are 2 databases Sql and MS Access so DataService class will have a generic method only to perform operation and something else will tell which database it should use to store the data.

**Tightly coupled**- if we change one component it might impact other components as well

DI pros

* Loosely coupled
* Separetes components cleanly
* Increases testability
* Allows use of IOC(Inversion of control)

Cons

Complexity increases.

DI implemented using Constructors or container. A container will contain object of all the Registered class and we can call for the objects for these classes from this container.

Unity Container- Register class and resolve objects. No need to create objects.

*Inversion of control is a design principle which****helps to invert****the control of object creation.* *Dependency Injection is a design pattern which implements IOC principle. DI provides objects that an object needs.*

**Solid Principles:-**

* Single Responsibility Principle(SRP)
  + Every class should have single job to do. Means only one purpose should be there for each class
  + If a class is doing multiple jobs then split it into multiple classes with single responsibility of each class.
* Open Closed Principle(OCP)
  + Open for new functionality but closed for modification on the existing functionality.
  + Functionalities should be written in such a way that if we need to add a new functionality it can be done without changing the existing functionality.
* Liskov Substitution Principle(LSP)
  + you should be able to use any derived class instead of a parent class and have it behave in the same manner without modification
* Interface Segregation Principle(ISP)
  + Instead of one big interface we should use multiple small interfaces based on the implementation.
  + Segregate big interface into small ones.
* Dependency Inversion Principle(DIP)

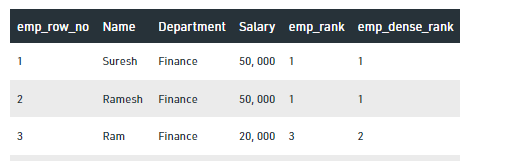
**Types Of Contructor**

* Default(without any parameter)
* Parameterized(with parameters)
* Copy(construcor in which it copies data of another object)
* Static(Invoked only once for all instances, it will be created when the first instace is created)
* Private(class cant be derived by other classess and also object cannot be created)

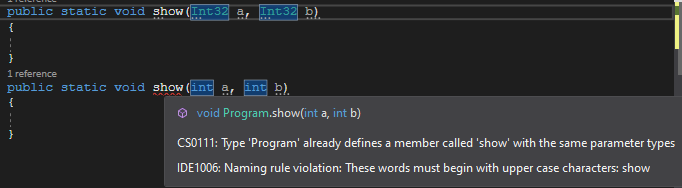
Second highest salary

Select EmpName from (Select \* Dense\_Rank() over (order by salary desc) from Employee as Rnk) a where a.Rnk=2

Explicit type casting



GetType- method to get the type of variable



**Since by default int is 32 bit singned integer thats why above error incase if it was Int64 there will not be any error and it will run int method upon call.**

**Similarly long is Int64.**

v.GetType().Name.Equals("Int32")

Boxing means converting data from primitive type to Object type

Unboxing is Object type to primitive.

Int a=10;

Object b=a;//boxing

Int b=(int)b;//unboxing

IactionResult maps or converts data to view

Dotnet ef migrations add InitialDb

Dotnet ef database update

Cte in sql

Not distinct

Types of access modifiers

Public

Private

Protected

Internal-can be accessed only inside the assembly(project). If we are using in other project the same class object then it cannot use that.

When to use arraylist and when to use linked list?

* Linked list addition of element after a given value is simple while same in arraylist is difficult.
* Arraylist is usefull when we need to find the element at a given index while in linked list it is complex.

Singleton class

private static readonly Lazy<Singleton> s = new Lazy<Singleton>(()=>new Singleton());

***Constructor chaining***

Parent class constructor always runs first then child

Below can also be done to create child Object

Parent obj=new Child();

If parent and child both have same method Show then obj.Show() will run Parent method(Note override case is different if method is overriden then in that case child class object will run since parent got overriden) Destructor of Child class will run since object comes from child.

If Child obj=new Child() then obj.Show() will run child method

In case of static constructors

Class A{ A() {Console.WiteLine(“A”); } }

Class B:A{ static B() {Console.WiteLine(“B”); } }

Class C:B{ static C() {Console.WiteLine(“C”); } }

C obj=new C();

This will print C B A as output since constructors are static if they were not static then the output would have been A B C.

Stack

Value Types(System.ValueType)

Bool,byte,char,int,double,decimal,struct,etc

Fast access

Heap

Reference Types(System.

Class,interface,string,object,delegate

Slow access

Garbage collector automatically manages memory. When enough space is not there it will automatically collects and clears garbage memory.

//Case 1

MyClass Obj=new MyClass();

Try{

Obj.DoSomething();

}

Finally{

Obj.Dispose();

}

//Case 2

Using(MyClass obj=new MyClass()){

Obj.DoSomething();

}

1NF

2NF

3NF

**IEnumerable(loads data in-memory and then filters), Ienumerator, Iqueryable(filters data on db itself and then loads data)**

1NF

2NF

3NF

Handler of Route

Lifecycle of API Request

ref and out keyword

dyanamic and var keyword

Request a webApi

var myvalue = 10; // statement 1  
myvalue = “GeeksforGeeks”; // statement 2  
Here the compiler will throw an error because the compiler has already decided the type of the myvalue variable using statement 1

dynamic myvalue = 10; // statement 1  
myvalue = “GeeksforGeeks”; // statement 2  
Here, the compiler will not throw an error though the type of the myvalue is an integer.

Request a webApi

Create an angular project

* ng new AngularProjectName
* Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser
* ng g component AngularComponentName
* ng g service ServiceName
* ng g class ModelName
* npm start

**Application Lifecycle**

App Start => *Request Life Cycles* => App End

**Request Life Cycle**

Events that occur whenever an HTTP request is handled.

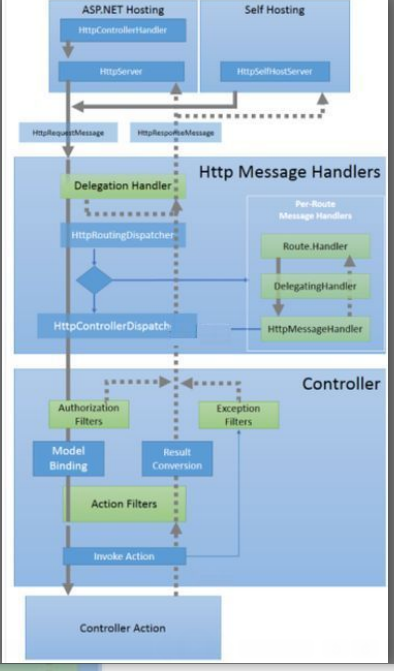
Request comes=>Routing(By URL routing module it is decided how request should be handled or which controller it should be assigned)=>Controller initialized(now action invoker finds and selects appropriate action method which should be invoked)=>Action Execution(action result is prepared)=>Result Execution(if result is view type then View Engine will be called which will find and render the view, if not view type then action result will do the job on its own. Result Execution generates the response of HTTP request)

DispatcherServerlet is the class which maps incoming request to most appropriate Model, View, Controller

**HTTP Message Lifecycle**

**(**[**https://www.c-sharpcorner.com/UploadFile/2b481f/lifecycle-of-Asp-Net-web-api-message/**](https://www.c-sharpcorner.com/UploadFile/2b481f/lifecycle-of-Asp-Net-web-api-message/)

**)**

****

1. IIS hosting or OWIN self hosting receives HTTP request
2. HttpServer converts HTTP request to HttpRequestMessage object(strongly typed access to the request) and sends it to HttpRoutingDispatcher.
3. HttpRoutingDispatcher which chooses which routing handler to call based on matching route and it will send the HttpRequestMessage to that routing handler.
4. Route Handler sends therequest to HttpMessageHandler which can return the request to the main path(HttpControllerDispatcher) or to custom endpoint.
5. Request is received by HttpControllerDispatcher which will route the request to appropriate controller and action to route based on the URL.
6. Authorization filters will run
7. Model Binding will be done
8. Action invoked
9. Result conversion is done and result is send back via the same path.

Model Binding:-

URI-ModelBinderParameterBinding Simple Type

Entity-Body-FormatterParameterBinding Complex Type

Entire reques-HttpParameterBinding Any type

Result Conversion:-

Media Type Formatter converts request to HttpResponseMessage.

Self Join

Cte

Optional parameter

***Switch statements require break it is mandatory else compile error***

Ref-It will pass the data as parameter and any change made to that data will change the original value

Out- We are not passing any value whatever is output of the method will come in that keyword

ObjectPool

Functions in sql and their types