**\*\*\*\*\*\*C# fundamentals\*\*\*\*\*\*\*\***

**variable declaration** -camel case

-> ex: int firstNumber;

**class declaration**- Pascal case

->ex: class HelloWorld

**Type Conversion:**

->converting the value of one type (int, float, double, etc.) to another type

**1.Implicit Type Conversions**

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

ex: int numInt = 500;

double numDouble = numInt;

**2. Explicit Type Conversions**

->converts larger types to smaller types

ex:double numDouble = 1.23;

int numInt = (int) numDouble;

**Type Conversion using Parse()**-> conversion between non-compatible types like int and string

ex: string n = "100"; // converting string to int type

int a = int.Parse(n);

string str = "test"; //Error,,,,,We cannot use Parse() to convert a textual string like "test" to an int

int a = int.Parse(str);

**TryParse**-> converts string value to integer value & returns boolean values(True/False)

ex: int result;

bool isConvertible =int.TryParse("abcd", out result);

if (isConvertible)

{

Console.WriteLine("Success");

}

else

{

Console.WriteLine("Failed"); //shows tis output

}

**base**-> used to access fields, constructors and methods of base class.

**what if we want to call the method of the base class as well?**

- refers to parent class

- use the base keyword to call the method of the base class from the derived class.

**Types of Polymorphism:**

**Polymorphism**: ability to take more than 1 form

**1. Compile Time Polymorphism / Static Polymorphism**

-> the compiler identifies which method is being called at the compile time.

-> we achieve compile time polymorphism through 2 ways:

* + **Method overloading**
  + **Operator overloading**

**2. Run-Time Polymorphism / Dynamic Polymorphism**

-> the method that is being called is determined at the runtime.

-> runtime polymorphism is achieved by:

* **Method Overriding**- use **"virtual"** and **"override"**

**Constructor**

-> same name as the class.

->without return type

->used to initialize private field of a class

->It is a method in the class which gets executed when a class object is created.

->when we don’t create constructor for the class, the compiler automatically creates a default constructor for the class.

**Syntax for Constructor:**

[Access Modifier] ClassName([Parameters])

{

}

**Types of Constructors:**

->**Default Constructor**- does not have any parameters

->**Parameterized Constructor**- at least one parameter

->**Static Constructor**

->**Private Constructor**

**Static constructor**

->use it for any work that has to be done before class is used and that has to be done only once.

-> called before the first object is created of the class, there is no way to pass any parameter

-> only one static constructor should be in the class.

->without parameters.

->only access the static members of the class.

->no access modifier

**Private Constructor**

->prevents object creation

->accessible within the class itself

->If you want to create object which has private constructor, then you need to create a public constructor along with private constructor.

**Abstract class**

=> we can't create object

=> acts as a base class for other classes.

=> Abstract class can't be static.

=> Abstract class can't be sealed.

=> abstract is not valid on fields, use property

public abstract int Id; //error : not valid

public abstract int Id { get; set; } // property must be used like this

**Abstract v/s Interface**

=> Use Abstract Class as a base class for big projects where most of functionality is common in all the derived classes.

=>Use an interface where, in every derived class, you want to define all the methods of the interface.

=> A class can inherit only one abstract class. A class can implement several interfaces.

**Auto implemented properties**

=> allows to define a property without explicitly implementing the getter and setter methods.

=> These properties can be accessed and modified just like regular properties, but the getter and setter methods are automatically generated by the compiler

=> if you don't want to implement properties without logic use auto implemented properties

ex: public string Name { get; set; } = "Mahesh Chandra";

If you want to convert a numeric value to a string => use Convert.ToString().

=>**ToString() and Convert.ToString**(variable) both are used for converting data into string

=>But the main difference is that Convert.ToString() handles null, while ToString() doesn't handle null value.

**Enumeration (or enum)**

=>It is a value data type

=>It is mainly used to assign the names or string values to integral constants, that make a program easy to read and maintain.

=>the first member of an enum has the value 0, and the value of each successive enum member is increased by 1

**Exception**

=>It is a run time errors, handled using try,catch & finally

**New Gen Delegates**

* **Action=>** points to function with no return type(void)
* **Func=>** points to function with return type
* **Predicate=>**points to function with bool return type & takes only 1 parameter

=>it's a kind of Func delegate

=>it is a generic delegate

**What is Anonymous Method ?**

=>As the name suggests, an anonymous method in C# is a method without a name.

=>It is defined using the delegate keyword and can be assigned to a variable of the delegate type

ex: public delegate string GreetingsDelegate(string name);

GreetingsDelegate gd = delegate (string name)

{

return "Hello @" + name + " Welcome to Dotnet Tutorials";

};

=>As you can see, the above code block is without a name and it contains only the method body and the method is defined using the delegate keyword.

=> We do not require writing any access modifiers like public, private, protected, etc.

=> We also do not require writing any return type like the void, int, double, etc.

=>They are always going to be void type and moreover you cannot reuse the anonymous method.

=>Where you define the anonymous method, at that place only it is going to be used.

=> we can access the variable defined outside the anonymous method

**Advantages** => code volume is very less and if it is one-time use only.

**limitations:**

=>cannot contain any jump statement like goto, break or continue.

=> cannot access the ref or out parameter of an outer method.

**Array:**

=>Fixed Length

=>Cannot insert it into the middle

=>Cannot delete from middle

=>It is type-safe, so we can store only similar types of data based on the data type.

**ArrayList:**

=>Variable Length

=>Can insert an element into the middle of the collection

=>Can delete elements from the middle of the collection

=>It is not type-safe, so we can store any type of data.

**Class can have only 2 types=>public & internal**

**private**-private to class

**internal**- internal to the project-> object must be created

**protected**- is a kind of public to inherited class & main class ->no need of object creation

protected internal-accessing another project via inheritance

**Sorting complex types**->we need to implement IComparable interface & needs to provide the implementation for the CompareTo() method

== ->compares the references

Equals ->compares the content

**Extension Method**

-> It is a static method of a static class, where the "this" modifier is applied to the first parameter. The type of the first parameter will be the type that is extended.

->Extension methods are only in scope when you explicitly import the namespace into your source code with a using

**var**

->must be initialized at the time of declaration

->Use of var when you’re not sure what type of data will be stored in a variable.

->cannot be initialized to null

**dynamic**

->It's not mandatory to initialise at declaration time.

**String Methods**

* replace
* substring
* concat
* format

**List**

CRUD operations

**\*\*\*\*\*\*\*Sql\*\*\*\*\*\*\*\***

**ALTER TABLE Syntax:**

ALTER TABLE table\_name CLAUSE supporting\_codes; //CLAUSE tells how the table is to be altered like ADD, RENAME COLUMN, etc.

**ALTER TABLE Operations:**

=>**Add a column** - ADD

=>**Rename a column** -RENAME COLUMN

=>**Modify a column** -ALTER COLUMN

=>**Delete a column** -DROP COLUMN

=>**Rename a table** - RENAME

IS NULL and IS NOT NULL are used to check if a column in a table contains a NULL value or not.

=>Empty values are considered NULL. However, space and 0 are not considered NULL.

**isnull** is used for replacing null from some user-friendly value

**coalesce** is used for return first non nullable value from the column or list.

**self join**- is a regular join but the table is joined with itself

**views** are read-only by default and cannot be directly updated. However, you can update a view indirectly by modifying the underlying tables and then refreshing the view using the CREATE OR REPLACE VIEW statement.

**Hoisting:** Behaviour of moving variables or function declaration to the top of the environment

var,let & const is not hoisted

fuction expression is not hoisted

**Arguments** is a object, reserved keyword

**this**=> is the object that the function is a property of.

**\*\*\*\*WebAPI\*\*\*\***

Entity Framework Core

ORM->Object Relational Mapper

Database First Approach

Code First Approach

Packages

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.SqlServer

Microsoft.EntityFrameworkCore.Tools

Create an Entity

Create a class which inherits the DbContext

Create a property of type DbSet<YourModel>

Call the constructor of DbContext class with parameter DbContextOptions

Inside appsettings.json add

"ConnectionStrings": {

"ProductConnectionString": "server=(localdb)\\MSSQLLocalDB;database=ProductsDB;Trusted\_Connection=true"

}

Tools menu->Nuget Package Manager-> Package manager Console

add-migration InitialMigration -it will creates a c# file

update-database ->it will checks the migration file & it creates a database

**\*\*\*\*\*Angular\*\*\*\*\***

**npm start**: to start application

**binding** {{title}}

**node\_modules**: all packages will be installed here

**npm install bootstrap**: for bootstrap installation:

**angular.json**: project setting file

**[value]="email"** binding javascript to html

**(click)="save()"** binding html to javascript

**Angular shortcuts**

a-component

Angular cli

**To create component**

npm run ng g c componentname

**To create core module**

ng g m core

**To create a component inside a core module**

npm run ng g c Header --path=src/app/core --module --export

**To install bootstrap**

npm run ng add @ng-bootstrap/ng-bootstrap@13.1.1

**create a component folder**

create a teamlist.component.ts

inside this file

create a class

import the component

decorate the component @Component

in app.module.ts

import the new component

add the newly created component in @NgModule

in app.component.html

add the selector

create teamlist.component.html

add the html part

**open new terminal in vscode**

**npm init -y** ->creates package.json

**npm install bootstrap** ->installs bootstrap inside node\_modules folder

create .gitignore file

inside this file type node\_modules

in index.html add bootstrap link like

<link rel="stylesheet" href="node\_modules/bootstrap/dist/css/bootstrap.css" />