Angular 14

Session-01

1. TypeScript – Complete
2. Angular 14  
    [Angular 12,13,14]
3. End to End Application  
    MEAN Stack  
    M MongoDB Database

E Expresss MiddleWare  
 A Angular Front End  
 N Node JS Server Side

1. Deployment

2 Month course duration

What is Angular ?

Why Angular ?

What is difference between Angular and React ?

What are the challenges in modern web development ?

* Web introduced in early 1990’s by Tim Berners Lee
* We are developing applications for web from early 1990
* Angular and React were introduced in 2013
* Modern web is used by smart device users – 80%

1. Unified UX – every where should same features experienced like mobile or pc

2. Fluid UX – Continuously content loading while scrolling down like facebook, youtube etc.

3. Loosely coupled and extensible.

4. Simplified deployment

What is solution ?

* Better build SPA[Single page application]
* World first SPA – Twitter[before Angular and React]

How to build SPA ?

* HTML,CSS,JavaScript,jQuery : yes
* What are the issues in JavaScript and jQuery
* Lot of DOM manipulations
* Lot of references
* Heavy
* Explicit Ajax

What is solution ?

* React , Angular, Vue, Knockout, backbone, Ember etc..

What is difference between React and Angular ?

* React is a library of javascript – Building UI
* Angular is developers platform (from building to deploy)

[language, library, framework]

What is difference between Angular JS and Angular?

Session-02

* Early 2010 Google introduced angular JS for public domain
* Angular js have lots of issues

1. Angular JS have lots of GAPS(what cant fullfil by this technology).
2. Angular JS is not built for what you are using
3. Angular JS JavaScript
4. JavaScript is not strictly typed language
5. it is not a strongly typed language
6. it will not support complete oop
7. extensibility issues
8. code security issues
9. Slow in rendering
10. Legacy library

Angular JS 1X version

* In 2013 Google angular JS Team
* Angular team started a new language called AtScript
* Microsoft already have a language called TypeScript

C# -Anders Hejberg - TypeScript

* Google angular team started integrating TypeScript with their new version angular 2
* angular 2 was designed as alternative for angular JS not replacement

What are the features of Angular ?

* Angular is not just a framework as angular JS
* it is a developers platform
* it provides end to end solution for developer
* it provides all library required from building ,debugging, testing to deploying.
* It is modular[only the library that is required will be included and loaded]
* application specific framework
* lightweight
* faster[angular is 10X faster than angular JS]
* supports TypeScript

1. strictly typed
2. strongly typed
3. oop supports

- Asynchronous approach

- latest version of angular is 14

What is deference between React and Angular ?

* React is javascript library
* Angular is framework with developers platform.

Session**-**03

TypeScript:

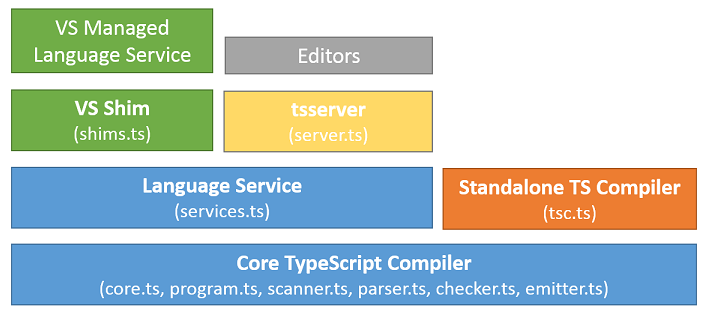
What are issues with JavaScript ?

* JavaScript is not default strictly typed.
* JavaScript is not strongly typed.
* It will not support all oop features.
* Security issues
* Extensibility issues

To overcome the above problem **typescript** comes into the picture.

* introduced in early 2003 by Microsoft Architect “Anders hejlberg”
* Angular,React,Vue and other javascript based technologies are now using typescript.
* Typescript is strictly typed.
* Typescript is strongly typed
* Typescript is an OOP language
* Typescript alone can build large scale applications
* Typescript is build with Typescript , means it does not need support like os to build.

**TypeScript Architecture**



1. TypeScript core compiler

* Identify the error in program
* It translates the program into machine code.

1. core.ts : launch the program
2. program.ts : verify the program structure
3. scanner.ts : responsible for handling input
4. emitter.ts : responsible for handling output
5. parser.ts : converting one type to another
6. chechers.ts : checking data type
7. TypeScript standalone compiler

* It is used to transcompile the typescript program into javascript.

“tsc.ts”

1. Language Service (services.ts)

* Service is predefined business logic
* We can inject into application to implement specific functionality
* Service is a factory of functions and values used by typescript program.

Var categories =[]; Array

Categories.length

Categories.push()

* All the libraries present in language service.

1. TsServer(server.ts)

* Server is used for hosting
* Handling request and response
* Deploying – making ready to use
* TsServer is responsible for hosting and deploying typescript programs.

1. VsShim [shims.ts]

* Shim is responsible to translate your unmanaged code into managed code and managed code is platform neutral/
* Shims.ts will make typescript platform neutral.

1. Managed Language Service

* It is a set of factories with function with values that are supported cross platform

1. Editor

* Editor is tool for building,debugging,testing and deploying applications.
* Typescript supports all editors

editorconfig.org

Setup environment for typescript

1. Download and install “Node.js” on your PC

[Package Manager called NPM]

1. Yarn , bower,grunt,rubygems,nuget,NPM

Windows 8+

Node JS version 12 +

NPM version 6+

1. Check the version installed

node -v

npm -v

1. Download and install TypeSript on your PC

C:\> npm install -g typescript

C:\> tsc -v

Session-04

* Install following extensions for VS code

1. Live Server : Required for hosting and executing your web application
2. Angular Language Service : Required for angular support
3. Vs code icon : icon for various types of files
4. Intellisense for CSS class Names in HTML : CSS support

Create a web application with TypeScript

1. Create new folder in your pc
2. Open project folder name in VS code
3. Open terminal and run the command [use command promt]

* npm init -y [This will generate package.json]
* package.json comprises of project meta data.
* {
* "name": "typescript-webapp",
* "version": "1.0.0",
* "description": "",
* "main": "index.js",
* "scripts": {
* "test": "echo \"Error: no test specified\" && exit 1"
* },
* "keywords": [],
* "author": "",
* "license": "ISC"
* }

1. Add following folder into project

* public : it comprises of static resources like html,text,image,pdf,audio and video etc.
* src : It comprises of dynamic resources like css , sass, less , js ,ts etc.

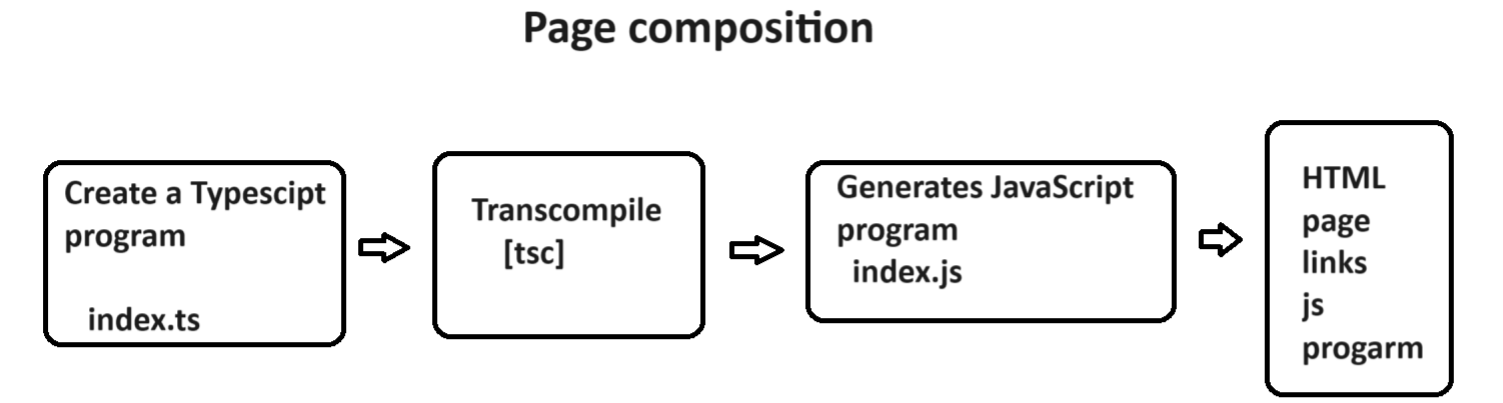
1. Add a startup page for project in “public” folder by name index.html

! [tab key] for basic index page

1. In the status bar of VS code click on “Go Live”

Or

Right click on your index page and select “open with live server”



1. go to SRC folder and add a new typescript file

“index.ts”

function bodyload(){

    var username:string|null=prompt("Enter Name");

    document.querySelector("p").innerHTML = `Hello ! ${username}`

}

1. Open file location in “Integrated Terminal”

Src> tsc index.ts

1. This will generate “index.js”
2. Link the js file to your Index.html page

FAQ : How to decide which version of javascript to use for TypeScript

Ans: by using “tsconfig.json”

* tsc –init

Session-05

Creating a new project

Handling Typescript file

TsConfig > tsc --init

Package.json > npm init -y

TSLnt > Obsolate[No longer in use]

TypeScript Language

1. Language Basics

* Variables
* Data types
* Operators
* Statements
* Functions

Variables::

* Variables declaration, initialization and assignment are all same as java script.

1. var
2. let
3. const

Data Type

* data type determines the data structure.
* Javascript is implicitly types and not strongly typed

Let x=10; //number

x = “john” ; // string

let y : //undefined

* TypeScript is strongly typed.

Let variableName : data type;

* Typescript data type are same as java script.

1. Primitive Types

Number

String

Boolean

Null

Undefined

1. Non primitive types

Array

Object

Map

Syntax:

let username: string=”john”|’john’|`john`;

let age:number = 22;

let subscribed : Boolean = true;

* If you don’t defined data type for variable in typescript the it is default any type.

let username; // any type

let username : string // string

* Type script supports “Typescript Inference”

[If you initialize variable with a value then value type is used as data type]

Let username; //any

Username =”john”; //any

Username = 22; //any

Let username = “john”; // string // type inference

Username = 22; //invalid

* TypeScript supports union of types

let variable: datatype|datatype|datatype;

let username:string|number |boolean= "raj";//string

username = "john";

username = 30;

username = true;

Summary Primitive types

number : signed,unsinged,float,double,decimal,exponent

string : “”,’’,``

boolean : true/false [not 1 or 0]

null : null

undefined : undefined

let username : string|null = prompt(“Enter name”)

String Function:

* charAt()
* charCodeAt()
* indexOf()
* lastIndexOf()
* substring()
* substr()
* slice()
* match()

Array Type:::

* JavaScript array allows various types of values
* TypeScript can handle array in 2 ways

1. With similar type of values
2. With various value types

Syntax::

Let categories:string[]=[“Electronics”,”Footwear”];

Let sales : number[]=[1000.34000];

Let values : any[] = [“Electronics”,1300,true];

FAQ : what is difference between Array() and [] ?

Let values : string = new Array();

Let values : string[] = [];

Ans: Both are used for initialization of memory for array.

Array() can define size for array.

However in JS and TS array size can change dynamically.

Array() will allow to initialize only similar type of values even when the data type is defined as “any”.

The data type for values will be determined according to the first value type

Let values : any[] = new Array(20,”A”); // invalid

FAQ : What is Tuple ?

Ans : tuple is collection that allows to initialize any type of values.

Let values : any[] =[] ; //tuple

Let values : any[] = new Array() ; // array not tuple

* Array can handle various types of values

1. Primitive type
2. Not primitive type
3. Functions

* Javascript array from ES5+ supports destructuring.
* let values:any[]=["TV",45000,true,function(){console.log('Hello ! typescript')} ];
* let [productName,price,stock,hello]= values;
* console.log(`Name = ${productName}\n${price}\n${stock}`);
* hello();

>tsc index.ts

>node index.js

- Array Method all are same as java script

toString()

join()

slice()

find()

filter()

map()

pop()

shift()

push()

unshift()

splice()

indexOf()

lastIndexOf()

sort()

reverse()etc..

Object, Map – in next session

Session-06

Object Type::

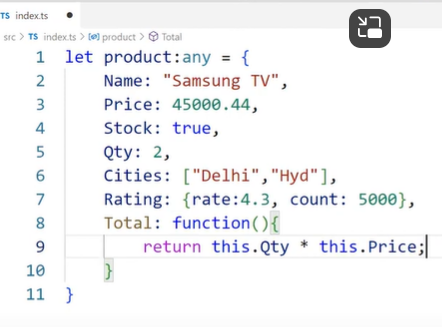
* There is no specific data type for object
* “any” is used for objects

Let product : any = {

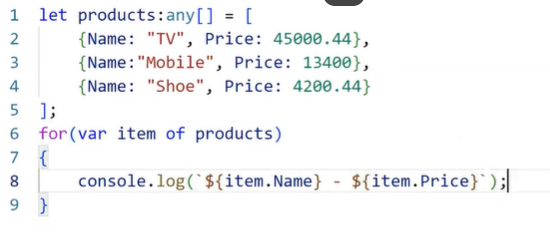
Key : value

}

* Key is always string
* Value is any type

Array of objects ::

* No specific data type for array of objects
* “any ” is used as data type
* 

**Map Type ::**

* ES6 map type
* Collection of key and value.
* FAQ: What is the difference between map and object?

Ans :

Object Map

Key string Keys any

No size for keys can define size for keys

Requires iterators implicit iterators

for reading keys , value

slow in access fast in access

Syntax::

A text on a white background

Description automatically generated

**Date type ::**

* No specific data type for date.
* “any” is used for data.
* “date ” is stored in memory by using “Date()”
* All date and time methods are same as in javascript.
* Syntax

let mfd : any = new Date(“2022-06-24”);

getHours()

getMinutes()

getSeconds()

getMilliseconds(0

getdate()

getDay()

getMonth()

getFullYear()

toLocaleDateString()

toLocalTimeString()

setHours()

setMinutes()

setSeconds()

setDate()

setMonth()

setYear()

**Regular Expression Type ::**

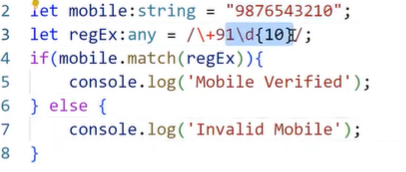
<input type =”text” pattern=”[A-Z]{4,10}”>

* JavaScript and typescript regular expression are defined in “/ /”

Syntax:

Let regExp:any = /[A-Z]{4,10}/;

* Regular expression is verified by using ‘match()’ method.

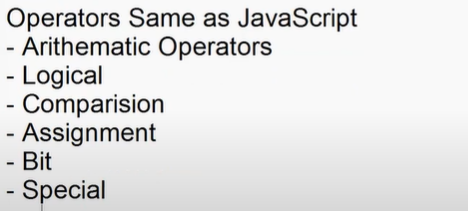


Summary : How to store various types of data in variable

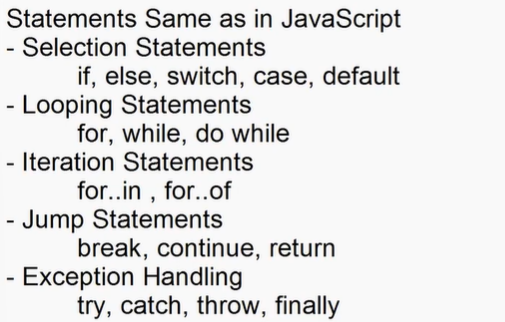
A white text with black text

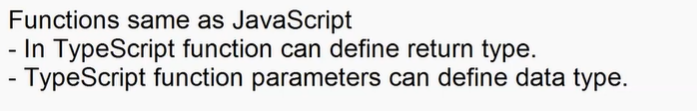
Description automatically generated

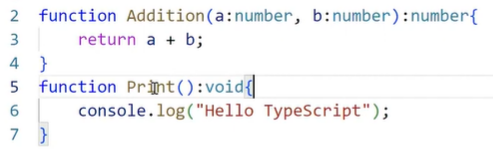
* Operators same as javascript



new , void,yield,in,of,typeof,delete,instanceof,etc.







* Typescript function can defined optional parameters.
* Optional parameters is defined by using “null” reference characters ‘**?:**’.



Note: Typescript function cant have required parameters following the optional parameters.



FAQ : What is Rest parameters?

Ans: Single parameter can handle multiple arguments

function(…params)

{

}

* Even function can have only one rest parameter
* Rest parameter must be the last parameter in formal list

Arrow function same as in javascript

Anonymous function

Callback function

Function recursion

Function closure etc..

**Session-07**

**TypeScript OOP :**

1. Contracts in OOP :

* Every component in OOP is designed as per contract.
* Contracts defines rules for designing a component.
* In OOP contracts are designed using “Interface”.
* Contracts can contain only rules no values.

**Syntax:**

**interface IName{**

**//rules…**

**}**

EX:

interface IProduct{

    name:string;

    price:number;

    stock:boolean;

}

let product:IProduct ={

    name:"Samsung Tv",

    price:44000.44,

    stock:true

}

console.log(`Name=${product.name}/nPrice=${product.price}/nStock=${product.stock}`);

for(var p in product){

    console.log(`${p}`)

}

* Contract can contain optional rules

interface IProduct{

    name:string;

    price:number;

    stock:boolean;

    rating?:number; // optional rules

}

* Optional rules in contract can be defined using “null reference character - ?”

Syntax:

Interface IName{

Property ?:Datatype;

}

* Contracts can be defined read-only rules.
* Every rule defined in contract can be re-assigned with a new value.
* If you want any rule to define property that cant be changed after initialization then configure it as “readonly”.

Syntax:

Interface IName{

Readonly property:datatype;

}

EX:

interface IProduct{

    readonly name:string;

    price:number;

    stock:boolean;

    rating?:number;

}

let product:IProduct ={

    name:"Samsung Tv",

    price:44000.44,

    stock:true,

    rating:4.4

}

product.name="Nike running shoes"; //Invalid

product.price = 6000.34;

product.stock=false;

product.rating=4.5;

* Contract can define rules for methods.
* Rule of method comprises of

1. Return type
2. Parameter type

A white background with black text

Description automatically generated

* Rule is for signature of method not for definition

Syntax:

Interface IName{

Method():Datatype;

}

* Methods also can be optional.

Syntax:

Interface IName{

Method()?:Datatype;

}

EX:

interface IProduct{

    readonly name:string;

    price:number;

    stock:boolean;

    rating?:number;

    qty:number;

    total():number;

    print?():void;

}

let product:IProduct ={

    name:"Samsung Tv",

    price:44000.44,

    stock:true,

    rating:4.4,

    qty:2,

    total:function(){

        return this.qty \* this.price;

    },

    print:function(){

        console.log(`

          Name : ${this.name}\n

          Price : ${this.price}\n

          Stock : ${this.stock}\n

          Rating : ${this.rating}\n

          Qty : ${this.qty}\n

          Total : ${this.total()}

        `)

    }

}

product.print();

* A contract can extends another contract.
* When you design an app with multiple versions it will have contracts extended.

EX:

interface IBank\_Version1{

    personal:string;

    nri:string;

}

interface IBank\_version2 extends IBank\_Version1{

    agri:string;

}

let LatestHDFC : IBank\_version2 = {

    personal:"Personal Banking",

    nri: "NRI Banking",

    agri:"Govt. Schemes"

}

let LegacyHDFC:IBank\_Version1 = {

    personal:"Personal Banking",

    nri:"NRI Banking"

}

**Session-08**

Typescript OOP :

* Contracts -> Interface
* Rules for properties
* Rules for method
* Optional rules
* Readonly rule
* Extending rules

1. Simple

one contracts extending another

1. Multi level

Extended contract is again extended

1. Multiple

Contracts will support multiple inheritance

EX:

interface IProduct{

    name:string;

    price:number;

    stock:boolean;

}

interface ICategory{

    categoryId:number;

    categoryName:string;

}

interface ProductDetails extends IProduct,ICategory{

    print():void;

}

let TV:ProductDetails ={

    name:"TV",

    price:56000.00,

    stock:true,

    categoryId:1,

    categoryName:"Electronics",

    print:function(){

        console.log(`Name:${this.name}\nPrice:${this.price}`)

    }

}

**Class in OOP :**

* Class is a program template.
* A template comprises of pre-defined logic and data , which you can implement and customize according to requirements.
* Class is referred as entity when it is mapping to client requirements.
* Class is referred as modal when it is mapping to data requirements.
* A class in typescript can contain only 4 members

1. Properties
2. Methods
3. Accessors
4. Constructors

FAQ: Can we declare a variable directly in class ?

Ans : No.

FAQ: Why variable is not allowed in class ?

Ans: variables are Immutable types.

EX:

class Demo{

    var x =10;//Invalid

}

FAQ: Can we declare function directly in class?

Ans: No> Funtion are Immutable.

* Data is stored in properties
* Logic is defined in methods.

FAQ: What is difference between Method and function?

Ans: Function is Immutable | Function intended to return a value

Method is mutable | Methods may or may not return a value.

* Typescript class can have static and non-static members.

Static:

- static refers to continuous memory.

- Memory allocated for first object will continue for second object.

- developer uses static members for continuous operations.

- It will be fromm applications start to application end.

No-static

- It refers to discreet memory.

- Memory will rest for every obect

- developer uses non-static members for discreet operations.

- it will be from session start to session end.

EX:

class Demo{

    static s:number =0;

    n:number=0;

    constructor(){

        Demo.s = Demo.s+1;

        this.n=this.n+1;

    }

    print():void{

        console.log(`S=${Demo.s} N=${this.n}`);

    }

}

let obj1 = new Demo();

obj1.print();

let obj2 = new Demo();

obj2.print();

let obj3 = new Demo();

obj3.print();

* Typescript classes will support access modifiers.

1. Public

* Public is accessible within the class or outside the class.
* You can access by usong class object or derived class object
* You can access in derived class or outside class.

1. Private

* It is accessible only within the class

1. Protected

* It is accessible within the class
* It is accessible outside the class but,only with in derived class and only by using derived class object.

EX:

class Base{

    public name:string='';

    private price:number=0;

    protected stock:boolean=false;

    print(){

        console.log(`Name=${this.name}\nPrice=${this.price}\nStock=${this.stock}`)

    }

}

class Derived extends Base{

    print(){

        let obj = new Derived();

        obj.stock;

        obj.name;

    }

}

**Session-13 (from 9 to 12 same as previous)**

**Class Members :**

1. Property:

* It is used to store data
* It is defined with access modifier and data type.

Syntax:

Public propertyName:datatype=value;

FAQ: What is the difference between a variable and property?

Ans : Variables are Immutable[structure cant change]

Properties are Mutable [structure can change]

1. Accessors:

It provides a fine grained control over property.

You can control the property and its structure using accessors.

Accessors are of two types

1. Getter – used to read the value
2. Setter – used to set the value

Syntax:

get propertyName()

{

Return value;

}

set propertyName(new Value){

Value = newValue

}

**Ex: A**ccessing property without accessor

class Product{

    public name:string="TV";

}

let tv= new Product();

tv.name="Samsung TV"; //set

console.log(`TV=${tv.name}`) //get

FAQ: How to restrict a property from storing new values?

Ans: by marking it as ‘readonly’.

EX:

public readonly name:string="TV";

FAQ: How to configure a property that can change its behaviour according to state and situation?

Ans: By using Accessors.

EX

var userName:string|null = prompt("Enter user Name");

var role:string|null= prompt("Enter your role");

var productName:any= prompt("Enter product name");

class Product{

    public \_productName:any; // \_ requires further implementation, just a tradition

    get ProductName(){

        return this.\_productName;

    }

    set ProductName(newName:string){

        if(role=="admin"){

            this.\_productName=newName;

        }else{

            document.write(`Hello ! ${userName} - You are not authorised to product name`)

        }

    }

}

let obj = new Product();

obj.ProductName = productName;

if(obj.ProductName){

    document.write("Product Name = "+obj.ProductName);

}

Note : link js file to html to run the dom manipulation.

* You can also use accessors to access any node from multilevel hierarchy.

Ex:

class Product{

    public name:string="";

    public price:number=0;

    public rating:any={

      RegisterUserRating:{

        rate:4.2,

        count:56000

      },

      GuestUserRating:{

         rate:3.2,

         count:4600      }

    }

    get GuestRating(){

        return this.rating.GuestUserRating.rate as Number;

    }

}

let obj = new Product();

console.log(obj.GuestRating);

Summary:

* Property
* Accessor

**Methods :**

* Methods and function are used for “Refactoring”.
* Refactoring is a technique of extracting a function or method for set of statements , so that you can reuse.

FAQ: what is differenece between method , function,procedure?

Ans:

**Session-14**

Class in Typescript

* Properties
* Accessors

**Methods :**

* Typescript methods are same as javascript methods
* Typescript method can configure data type parameters and return type

EX:

class Demo{

    public name:string="";

    public price:number=0;

    public qty:number=0;

    public Total():number{

        return this.qty \*this.price;

    }

    public print():void{

        console.log(`Name=${this.name}\nPrice={this.price}\nQty=${this.qty}\nTotal=${this.Total()}}`);

    }

}

let obj = new Demo();

obj.print();

* Method can pe parameterized and parameterless.
* Typescript support optional parameters [“?”]
* Optional parameter must be last parameter.
* Require parameters can occur after optional parameter.
* Method supports rest parameter “…param”.

**Constructor :**

* It is used for instantiation
* Javascript and typescript constructor both are same.
* Constructor must be anonymous [no name].

EX:

(function(){

        document.write('hello !');

    })(); //anonymous function

FAQ: What is purpose of anonymous function ?

Ans : Anonymous specifies a function without name. They are used in callback and promise. Callback and promise means function will execute according to situation.

* Constructor is a software design pattern used to create an instance for class.
* Every class implicitly have a constructor.
* Explicitly we can add a constructor in class to configure the actionws to perform at the time of instantiation.
* Constructor is a special type of sub routine(methods) that executes automatically.

EX:

class Database{

    constructor(){

        console.log("Connected..");

    }

    public insert():void{

        console.log("Record Inserted..");

    }

    public delete():void{

        console.log("Record Inserted..");

    }

}

let oracle = new Database();

oracle.insert();

* Constructor can be parameterized or parameter less.

FAQ: if constructor is parameterized then when the parameter are passed?

Ans: At the time of allocating memory for object.

* var obj = new ClassName(p1,p2,..);

EX:

class Database{

    constructor(dbName:string){

        console.log("Connected.."+dbName);

    }

    public insert():void{

        console.log("Record Inserted..");

    }

    public delete():void{

        console.log("Record Inserted..");

    }

}

let oracle = new Database("oracle");

oracle.insert();

* All constructor parameter rules are same as for methods and function.
* Typescript constructor cant be private.
* Typescript cant overload constructor[only one constructor allow in class].
* If derived class is having a constructor then it must have **super call**;
* Super class constructor must be called in derived class constructor.

EX:

class Base{

    constructor(){

        console.log("Base class constructor..");

    }

}

class Derived extends Base{

    constructor(){

        super();

        console.log("Derived class contrustor..");

    }

}

Summary:

* Properties
* Accessors
* Methods
* Constructor

**Templates** :

* Template provides pre-defined logic and data , which you can implement and customize according to your requirements.

EX:

A black text on a white background

Description automatically generated

* Templates are designed by using “Abstract ” class.

**Session-15**

* Template are used in rollout and end to end.
* Templates are designed by using “Abstract ” class.

Interface 🡺 contracts

Abstracts -> Templates

* Abstract class comprises of members classified into two types

1. Implemented.
2. To be Implemented.

* Any member to be implemented is marked as abstract.
* If a class contain at least one abstract member then the class also must be marked as abstract.
* You cant create an object for abstract class. As it is incomplete class.
* abstract class is extended by another class.

Syntax:

abstract class className{

public method(){};

public abstract method();

}

class DerivedClass extends className{

method(){};

}

* The process of hiding structure of a component and providing only functionality is know as “Abstraction”.

EX:

interface ProductContarct{

    name:string;

    price:number;

    qty:number;

    Total():number;

    Print():void;

}

abstract class ProductTemplate implements ProductContarct{

    public name:string="";

    public price:number=0;

    public qty:number=0;

    public abstract Total():number;

    public abstract Print():void;

}

class ProductComponent extends ProductTemplate{

    name = "Samsung TV";

    price = 45000.44;

    qty = 3;

    Total(){

        return this.qty \* this.price;

    }

    Print(){

        console.log(`Name=${this.name}\nPrice=${this.price}\nQty=${this.qty}\nTotal=${this.Total()}`);

    }

}

let obj = new ProductComponent();

obj.Print();

Output:

A black screen with white text

Description automatically generated

Note: Rules are implemented and Templates are extended.

FAQ: What is Abstraction?

Ans : It is the process of hiding the structure of component and providing only functionality.

Summary

1. Contracts [Interface]
2. Templates [Abstract class]
3. Components [class]

**Polymorphism :**

* **WHATWG – Build html**
* **Javascript – MDN Brendan Eich**
* **Typescript – Microsoft MSDN**

EX:

class Employee{

    public firstName:string="";

    public lastName:string="";

    public designation:string="";

    public print():void{

       document.write(`${this.firstName} ${this.lastName}  ${this.designation}`)

    }

}

class Developer extends Employee{

    public firstName: string="Sanjay";

    public lastName: string="kumar";

    public designation: string="Developer";

    role = "Developer Role : Build,Debug ,Test";

    print(){

        super.print();

        document.write(this.role);

    }

}

class Admin extends Employee{

    public firstName: string="Raj";

    public lastName: string="kumar";

    public designation: string="Admin";

    role = "Admin Role :Authorization";

    print(){

        super.print();

        document.write(this.role);

    }

}

class Manager extends Employee{

    public firstName: string="Pinsu";

    public lastName: string="Singh";

    public designation: string="Manager";

    role = "Developer Role :Approvals";

    print(){

        super.print();

        document.write(this.role);

    }

}

let employees = new Array(new Developer(),new Admin(),new Manager());

FAQ: What is difference between implements and extends ?

Ans : Contracts are implemented and classed are extended.

FAQ: Can we extend contract ?

Ans: yes.

FAQ : Can we create object for template ?

Ans: NO.

FAQ: Why multiple inheritance is not supported for classes but supported for interface ?

Ans: Classws leads to constructor Deadlock.

Interface will not have constructor.

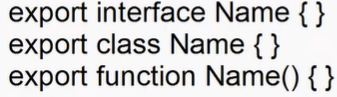
**Modules :**

* Module is collection of variables , functions , contracts , templates and components.
* Modules are used to build a library for applications.
* Library allows to re use the resources.
* Modules require a module system.
* Modules systems are various types

1. Common JS
2. Require JS
3. UMD (Universal module distribution)
4. AMD (Asynchronous Module Distribution)

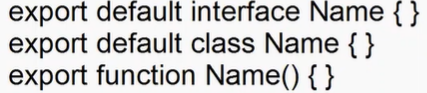
* Typescript by default uses a module system called “Common JS”.
* Any member define in a module is accessible only to the current module.
* If you want any member of a module to be accessed from another module then you have to mark it as “export”.

Syntax:



* Every module can have one default export.Default means loaded in memory automatically. It is not mandatory.

Syntax:



* The members of any module can be imported into the current module.

Syntax:

A close up of a white background

Description automatically generated

* Module system use DI mechanism. [Dependency Injection].
* DI is a design pattern that specifies how an object get holds of its dependencies.
* DI uses 2 major component

1. Provider
2. Injector

* Provider locates the dependencies the loads into memory.
* Injector is responsible for injecting the dependencies.

EX:



EX:

//module

export default interface ProductContract{

    name:string;

    price:number;

    qty:number;

    total():number;

    print():void;

}

export interface CategoryContract{

    categoryName:string;

}

A close up of a sign

Description automatically generated

Ex:

import ProductContract ,{CategoryContract} from "../contracts/ProductContracts"

export abstract class ProductTemplate implements ProductContract,CategoryContract{

    public name:string="";

    public price:number=0;

    public qty:number=0;

    public categoryName: string="";

    public abstract total():number;

    public abstract print():void;

}

A close up of a sign

Description automatically generated

EX:

import { ProductTemplate } from "../templates/ProductTemplates";

export class ProductComponent extends ProductTemplate{

    name = "Samsung TV";

    price = 34300.33;

    qty = 2;

    categoryName = "Electronics";

    total(){

         return this.qty \* this.price;

    }

    print(){

        console.log(`Name=${this,this.name}\nPrice=${this.price}\nqty=${this.qty}\nCatName=${this.categoryName}\nTotal=${this.total()}`)

    }

}

A close up of words

Description automatically generated

EX:

import {ProductComponent} from '../Library/components/ProductComponent'

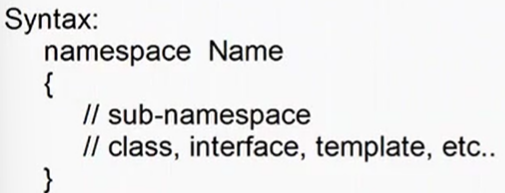
let tv = new ProductComponent();

tv.print();

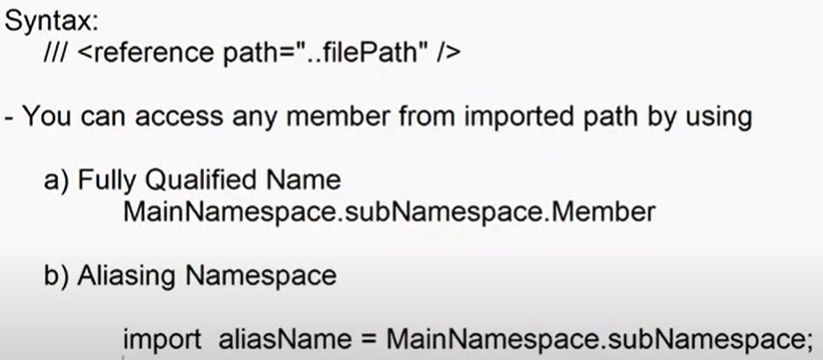
**Session-18**

**Namespace :**

* Namespace is a collection of sub-namespace and modules.
* You can build and import library using namespace.
* Members are configured in a file within the namespace.



* If you want to access namespace outside the declared scope then mark it as “export”.
* if you want to import library from namespace then you have to use “///” directive.



Note : Understand the concept later..

**Genrics** :

* Generic is used to configure type safe members.
* Initially it is open to handle any type of value, once it knows the data type then it becomes strongly typed.
* You can configure generic for

1. Property
2. Class
3. Method

EX:

var Demo = /\*\* @class \*/ (function () {

    function Demo() {

    }

    Demo.prototype.Add = function (a, b) {

        console.log("".concat(a, "-[").concat(typeof a, "] - ").concat(b, " - [").concat(typeof b, "]"));

    };

    return Demo;

}());

var obj = new Demo();

obj.Add("john", "David");

obj.Add(30, 40);

**Session-19**

**Generic Class And Properties** :

* Generic type can handle operators directly.
* You have to use function to handle any specific operation.

EX:

function sum(a,b){

    return a+b;

}

class Demo{

    public Add<T>(a:T,b:T){

        return sum(a,b);

    }

}

let obj = new Demo();

console.log(obj.Add<number>(10,30));

console.log(obj.Add<string>("Raj","Kumar"));

Class as Generic Type:

EX:

interface IOracle{

    username:string;

    password:string;

    database:string;

}

class Database<T>{

    public connectionString:T;

    public connect(){

        return this.connectionString;

    }

}

let oracle = new Database<IOracle>();

oracle.connectionString = {

    username:"scott",

    password:"tiger",

    database:"empdb"

}

for(var property in oracle.connect()){

    console.log(`${property} : ${oracle.connect()[property]}`)

}

FAQ: What is the purpose of making a Generic class ?

Ans : Creating Generic Entity. All members are type safe.

* Generic can be any type

1. Primitive
2. Non-Primitive
3. Custom Type

EX:

interface IProduct{

    name:string;

    price:number;

    stock:boolean;

}

interface ICustomer{

    firstname:string;

    email:string;

}

class DataService{

    public getAllDetails<T>(data:T){

        return data;

    }

}

let product = new DataService();

let products:IProduct[] = product.getAllDetails<IProduct[]>([{name:"TV",price:34500,stock:true}]);

let customer = new DataService();

let customers:ICustomer[]=customer.getAllDetails<ICustomer[]>([{firstname:"Pinsu",email:"pins@gmail.com"}]);

console.log(`-------Product List------`);

for(var item of products){

    console.log(item.name);

}

console.log(`-------Customer List------`);

for(var c of customers){

    console.log(c.firstname);

}

**Enums-Enumeration :**

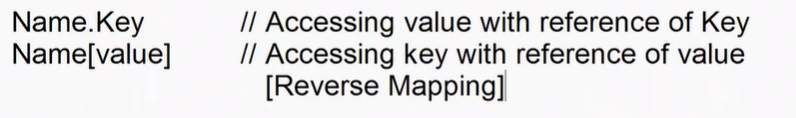
* It is a collection of constants.
* It can be string,number,or expression.
* It can have expression that returns only string or number.
* Enums are defined using “enum”.

Syntax:

Enum Name{

Key = value;

}



Ex:

enum WeekDays{

    sunday = 0,

    mon = 1,

    tue =2,

    wed = 3,

    thr = 4,

    fri = 5,

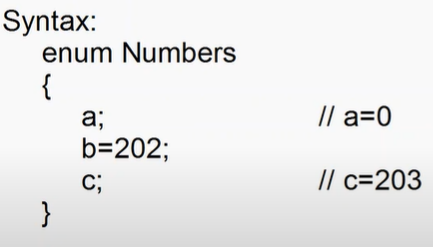
    sat = 6

}

let manufactured : any = new Date("2022-02-18 08:20");

console.log(`Manufactured Week Day : ${WeekDays[manufactured.getDay()]}`)

* Enum numeric constant can implement the value according to previous value.

****

A white background with black text

Description automatically generated



* Enum can also have expression, but its must return string or number.



>,<,+,-,\*,/

EX:

enum Total{

    qty = 2,

    price = 4500,

    result = qty \* price

}

console.log(`Total = ${Total.result}`)