**Phase 1**

**Day 1**

**02-08-2021**

**Web Technologies**

**https://**[**www.google.com**](http://www.google.com) **– URL : Uniform Resource Locator**

**http : protocol : hypertext transfer protocol.**

**Set of rules**

**https: secure**

**stateless protocol**

[**https://www.google.com**](https://www.google.com) **-🡪 URL**

**req(http/https)---🡪**

**Client Server**

**🡨--res(http/https)--- HTML/HTML5**

**HTML : Hyper text mark up language.**

**Static web page.**

**Dynamic web page.**

**HTML provide lot of pre-defined tags which help to create web page.**

1. **Html**
2. **Head**
3. **Body**
4. **Title**
5. **P**
6. **Heading tag h1 to h6**
7. **A anchor tag : hyper reference**
8. **Images**
9. **List tag**
10. **Table tag**
11. **Forms tag**

**In html 4**

**JSF : Java Server Faces .jsf**

**<!doctype HTML PUBLIC URL=”uRLAddress.dtd”/>**

**Document type definition**

**Root tag name ie html, two child head and body, inside head title, script, style and meta etc.**

**Inside body optional or more than one p, h1 to h6, div, span, table, form tags.**

**In html5 they remove this concept.**

**<!doctype html> : in html5 we are writing this tag giving the instruction to browser we are using html5 features(optional).**

**Before HTML5**

<input type="text/password/radio/checkbox/button/file/submit/reset"/>

**After HTML5**

<input type="number/email/date/url/color/"/> etc

**By default every html form method is consider as GET.**

**If method GET then Information send through URL using query param concept.**

**URL?key=value&key=value&key=value**

**In get method data not secure.**

**Method = post . if method is post data send through body part of request.**

**Div is know as division tag. It is also known as container tags. which can contains more than other tag like p, b, i, h1 to h6 as well as another div tag. Which help to represent the part of the web page.**

**HTML5 provided semantics tags.**

**CSS**

**Class and id selector**

**Span**

**Font and text properties diff**

**Box model 1**

**Flexbox 2**

**Grid model 3**

**Position**

**Float**

**background**

**Absolute and relative**

**@media**

**Frameset : deprecated iframe**

**Types of CSS**

1. **Inline**
2. **Internal css or embedded css**
3. **External**

**class : groups of tags. two tags can contains same class name.**

**id : to make tag unique we have to provide the id. Id must be unique.**

**<div>**

**<p class=”abc” id=”p1”>First Para</p>**

**<p class=”xyz abc” id=”p2”>Second Para</p>**

**<p class=”abc” id=”p3”>Third Para</p>**

**<p class=”xyz” id=”p1”>Fourth Para</p>**

**</div>**

**Box model**

**Border properties**

**Box Model : row or columns**

**Flex model : we can display the tag or dom elements may be row wise columns wise.**

**In CSS we have to write the style rules from a scratch.**

**Bootstrap : bootstrap is a external CSS open source framework. Which provide lot of pre-defined css classes with respective all tags like p, div, button, form, table etc.**

**First framework provide to make responsive web page or application.**

**Phase 1**

**Day 2**

**03-08-2021**

**Position : fixed, static, relative and absolute etc**

**By default position is static.**

**Bootstrap**

**We can add bootstrap features to our web pages**

1. **Using CDN URL**
2. **Download Bootstrap files.**
3. **Using Node JS.**

**Container and container-fluid**

**Container : container is pre-defined bootstrap class which is known as fixed width container.**

**Container –fluid 🡪 It is a pre-defined class which takes full width of the web page.**

**Bootstrap gridlayout**

**Grid layout is use to arrange the dom elements in row and columns format.**

**By default every two divided into 12 columns.**

**Screen size**

**Xs : extra small <576px**

**Sm : small >=576px**

**Md : medium >=768px**

**Lg : large >=992px**

**Xl : extra large >=1200px**

**JavaScript**

**=== vs ==**

**Control flow**

**looping**

**Alert and prompt**

**Anonymous function**

**Module and**

**Generic function**

**Array object**

**JavaScript Object**

**DOM**

**Set and map collection**

**Ajax**

**Promise**

**Asyn and await**

**JavaScript was object based interpreter scripting language Till ES5**

**ECMA : European Computer Manufacture Association**

**ECMA is a concept.**

**One of the implementation of ECMA (ES) is JavaScript.**

**Object based Vs Object oriented : no class keyword it contains only pre-defined object as well as we can create user-defined objects. Even we can call as protocol base.**

**Interpreter Vs Compiler : interpreter check the code line by line. Compile convert all code at time.**

**Using JavaScript we can do programming on web page.**

**HTML -🡪 content display**

**CSS/bootstrap 🡪 presentation on content or look and feel.**

**JavaScript 🡪 action on contents or programming on contents.**

**In JavaScript to declare the variable we use var keyword till ES5.**

**Control flow**

**If statement**

**if(condition) {**

**}**

**If else**

**if(condition){**

**}else {**

**}**

**If else if**

**if(condition){}**

**else if(condition){}**

**else {}**

**Switch statement**

**Looping :**

**While loop**

**Do while loop**

**For loop**

**For in**

**For of**

**forEach : it is a function.**

**Function and events**

**Function : function is use to write set of instruction to perform a specific task.**

**Pre-defined function**

**alert()**

**prompt()**

**parseInt()**

**parseFloat()**

**eval()**

**User-defined function**

**In JavaScript we can declare function in different ways.**

1. **Normal function declaration syntax**

**function functionName() {**

**}**

1. **Expression style function**

**var functionName = function() {**

**}**

1. **Arrow function : arrow function is a short cut of expression style function.**

1. **Callback : passing the function name or function body or function itself to another function is known as callback function.**

**Apply(), bind(), call() : ES5 style object creation.**

**Phase 1**

**Day 3**

**04-08-2021**

**In JavaScript we can create user defined object**

1. **Function style using ES5 style**
2. **Literal style using es5 style**
3. **Class style using ES6 style**

**array : array is use to store more than one value of same or different types.**

**syntax to create the array**

**literal style**

**var obj1 = [];**

**var obj3 = [10,20,30,40,50,60]**

**object creation style**

**var obj2 = new Array();**

**var obj4 = new Array(10,20,30,40,50,60);**

**Set : set is a type of data structure which help to store unique elements values.**

**Map : map is a type data structure which help to store the value in key value pairs.**

**Key is unique and value may be duplicate.**

**DOM : document object model**

**Index.html**

**<html>**

**<head>**

**<title>This is simple web page</title>**

**</head>**

**<body>**

**<p>Welcome to My Web Page</p>**

**</body>**

**</html>**

**Run application in browser internally DOM hierarchy create.**

**Html**

**Head body**

**Title p**

**textNode textNode**

**textData textData**

**If we want to read, write and update HTML contents dynamically. All programming language provide DOM API (Document object model application programming language).**

**Java, Python, C++, .net and JavaScript etc.**

**Ajax**

**Promise**

**Asyn and await**

**Synchronous and asynchronous operation**

**Synchronous statement execution : it execute line by line**

**document.write(“1st statement”);**

**document.write(“2nd statement”);**

**document.write(“3rd statement”);**

**Synchronous statement execution : it execute line by line**

**document.write(“1st statement”);**

**asyn document.write(“2nd statement”); it execute independently**

**document.write(“3rd statement”);**

**synchronous function call**

**fun1();**

**fun2();**

**fun3();**

**asynchronous function call**

**fun1();**

**asyn fun2();**

**fun3();**

**synchronous communication**

**1st req**

**2nd req**

**3rd req**

**Client Server**

**asynchronous communication**

**1st req**

**asy 2nd req**

**asy 3rd req both request work independently.**

**Client Server**

**setTime**

**setInteval**

**clearInterval**

**Ajax**

**Promise**

**Asyn and await**

**AJAX : Asynchronous JavaScript and XML**

**XMLHttpRequest : It is pre-defined object which help to do asynchronous communication.**

**Promise : Promise is a pre-defined object which help to handle asynchronous events of data. Data can be text, number, json or xml.**

**Promise can be resolve(successfully done) or reject(failure).**

**If we want to load the data from promise object**

**We have to use then() and catch()**

**Then will execute when promise resolve**

**Catch will execute when promise rejected.**

**Fetch() : it is pre-defined function which internally use ajax to call backend technologies services.**

**REST API call.**

**Backend technologies in Java, python, .net, node etc**

**When we call they return in text format or json format, xml format.**

**Fetch() function return type is promise. Using then we can load the data if promise resolve or if any error generate we can handle using catch.**

**Asyn and await**

**ES7 features : It is a replacement of then and catch.**

**User defined object creation**

**Oops**

**Object 🡪 any real world entity**

**Properties -- have – fields / variables**

**Person**

**Behaviour -- do/does – functions / methods**

**Place**

**Bank**

**Animal**

**Car**

**Object is concept.**

**//ES5 style object creation : function or literal style**

**//ES6 style object creation : class**

**var, let and const**

**using var keyword we can re-declare same variable once again. But using let keyword we can’t declare same variable once again.**

**Using var keyword we can create global scope or function scope. Let using let we can create block or local scope.**

**const a=10;**

**//a=20; we can’t change the value**

**Node JS**

**Overview of TypeScript**

**Angular Framework**

**Day 4**

**HTML,CSS,JavaScript and bootstrap**

**Library and framework**

**jQuery , backbone js, coffee js, angular js and angular framework, react js and vue JS etc.**

**Node JS : Node js is not a library or framework. It is run time environment for JavaScript library or framework.**

**Like a jre in java.**

**Before Node js JavaScript is known as client side scripting language. But after node js JavaScript we be use client side as well as serve scripting language.**

**Node js contains lot of pre-defined modules may be local module or external module which help to do server side programming language.**

**Using Node js we can do file handling programming, we can create server side program, we can create REST API, we can connect database like SQL or Non SQL database etc.**

**MEAN Stack : Mongo DB, Express Module, Angular Node JS**

**MERN Stack Mongo DB, Express Module, React Node JS**

**In node js we can’t use window and document objects.**

**BOM and DOM**

**Browser Object Model**

**Document object Model**

**Node js provide**

**npm (node package manager).**

**Npm is use to install external module.**

**Syntax to install external module**

**Npm install –g moduleName globally**

**Or**

**Npm install moduleName locally**

**Typescript : Typescript is a super set of JavaScript which support all ES6 features.**

**Angular JS : HTML,CSS,JavaScript using ES5 or ES6.**

**1.x 1.8 etc**

**MVC base architecture**

**Angular Framework 2 to 12 : technologies require**

**Html, css, Typescript.**

**Component base architecture**

**Browser can’t understand Typescript scripting language.**

**So we have to convert ts to js.**

**Traspiler ( it is type of compile which help to convert ts to js).**

**tsc (typescript compiler).**

**To convert ts to js we require tsc**

**npm install –g typescript**

**Typescript features**

**Typescript support data types.**

**number, Boolean, string, any**

**we can use array with data types.**

**TypeScript function**

**Typescript OOPS concept.**

**Using import and export we can access function, classes, interfaces from more than one file in typescript.**

**Modules : it is collection of function, variable, classes and interfaces.**

**In typescript two type of modules**

**Internal modules**

**External module (angular use).**

**We have to create typescript configuration file**

**tsconfig.json**

**tsc –-init : this command is use to create tsconfig file.**

**Angular project creation**

**First install node js**

**Using node js command install ng**

**npm install –g @angular/cli**

**(next generation)**

**ng new project-name**

**routing 🡪 no**

**styling -> css**

**after project created successfully**

**move inside a project directory**

**cd project-name**

**ng serve : to run the project**

**after compiled successfully**

**open browser** [**http://localhost:4200**](http://localhost:4200)

**decorator**

**property binding and string interpolation**

**forms**

**Observable**

**dependency injection and service**

**routing : linking pages, wild cards.**

**Angular is a open source framework provided by google.**

**Angular is use to create SPA Singe Page application.**

**Multi page application.**

**Index.html welcome.html**

**Hyperlink**

**Button**

**Submit button**

**Etc**

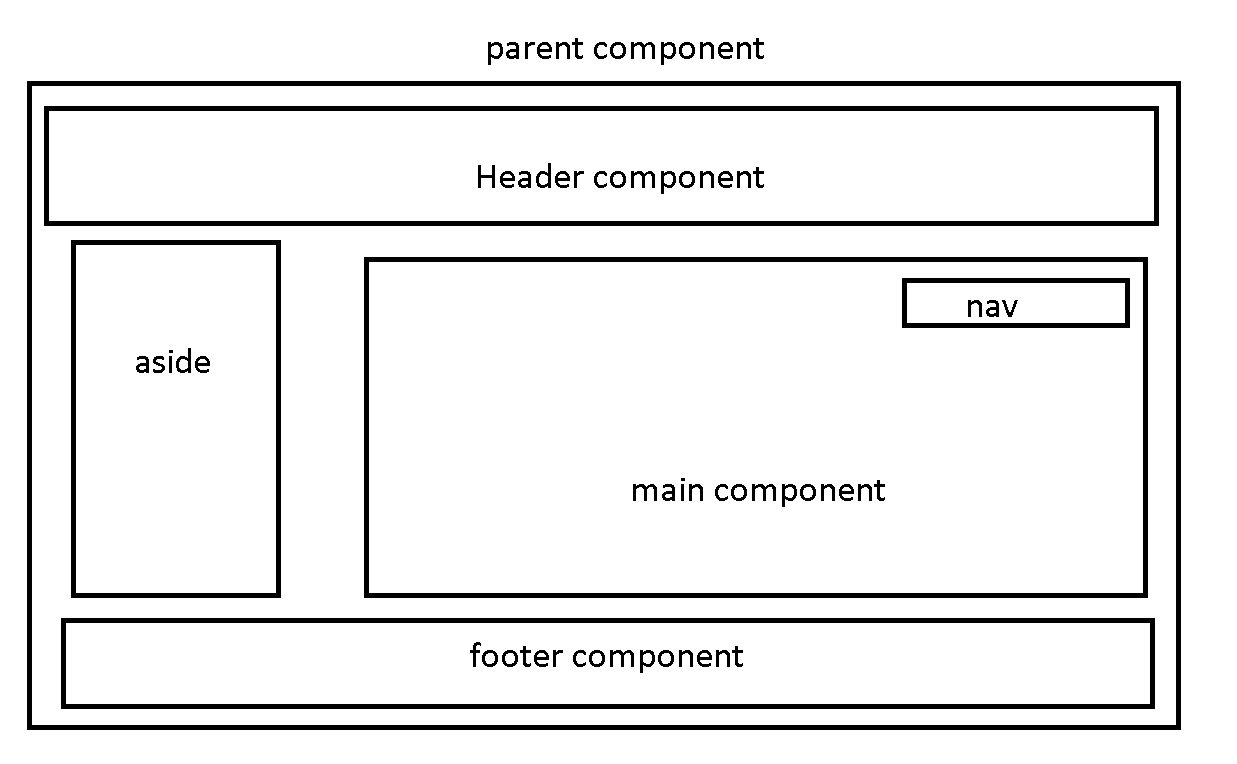
**Single page application :**

**Rather than loading whole page we can’t load the part of the in first page.**

**Component : it control the view or part of the view.**

**Module is a combination of more than one components.**

**Module is a like a package in java.**

****

**Src**

**App**

**Open app.component.html**

**Write your own html code**

**No html, head, and body tags.**

@Component

**Typescript provide one of the great features ie**

**Decorator : decorator is a type of special function which help to make class or property as special type. Decorator is also known as meta data (data about data). Decorator like a annotation in Java.**

**All decorator start with @ symbol and followed by decorator name.**

**@Component**

**@NgModule**

**@Injectable**

**@Pipe**

**@Input**

**@Output**

**@ViewChild**

**Etc**

**@Component decorator contains lot of property**

**selector : unique name for that class**

**templateUrl**

**styleUrl**

**in Angular using component we are creating user-defined tags**

**selector is behave like a user-defined tags.**

**templateUrl : connected to html page**

**stylesUrl : connected to css page**

**app.module.ts**

@NgModule: module is a collection of more than one components.

**declaration : all component declaration details provide in this section.**

**Imports : this property is use to import pre-define or user defined modules.**

BrowserModule: this module help to use to display or render the data in browsing area.

**Provider : inside this property we will provide the angular service class details.**

**Bootstrap : this property is use to load the parent components.**

**nain.ts**

**This file load the main or parent modules.**

**appModule LoginModule CustomerModule OrderModule**

**appComponent createLoginComponent 🡪 one html page.**

**displayLoginComponent**

**deleteLogincomponent**

**angular.json file**

"index": "src/index.html",

"main": "src/main.ts",

**Open index.html**

**<app-root></app-root>**

**package.json file**

**this file contains angular version and other dependencies details.**

**Index.html welcome.html**

**<html>**

**<body>**

**<a href=”welcome.html”>click</a>**

**</body>**

**</html>**

**Whole dom once again loaded in browser.**

**<html>**

**<body>**

**<h2>WElcome to Home Page </h2>**

**</body>**

**</html>**

**index.jsp**

**<html>**

**<body>**

**Header jsp footer jsp nav jsp**

**<h2>WElcome to Home Page </h2>**

**</body>**

**</html>**

**Header.jsp : it own dom**

**Footer.jsp : it own dom**

**Nav.jsp : it own dom**

**Index.html**

**Header html include**

**<footer>**

**Coding….**

**</footer>**

**Dashboard.html**

**Include header html**

**<footer>**

**Coding….**

**</footer>**

**Header.html**

**<header>**

**Coding…..**

**</header>**

**3 to 4 project**

**ng new angular-data-binding**

**ng new angular-forms**

**ng new angular-service**

**ng new angular-routing (routing yes)**

**we can create the component using ng command**

**ng generate component componentname**

**or**

**ng g c componentname**

**ng serve –o : after compiled program automatically open in default browser.**

**data binding : data binding provide the bridge between html (template) and component. Data binding help use to share the data between template to component.**

1. **One way data binding**
2. **String interpolation : component ----🡪 view**

**{{variableName}}**

**{{expression}} {{5+6}}**

**{{functionName()}}**

1. **Property binding : component ----🡪 View**

**[]**

1. **Event binding : view --🡪 Component**

**() with on pre-fix. Only event name must wrap with ().**

**Angular using same dom events**

**DOM JavaScript event Angular Event**

**onClick (click)**

**onDblClick (dblclick)**

**onMouseOver**

**onMouseOut**

**using event binding and string interpolation /property binding we can do 2 way data binding**

**passing the value from component to view.**

**Template Reference**

1. **Two way data binding**

**Angular support 2 way data binding**

**[()]**

**To achieve the two way data binding angular provide ngModel pre-defined attribute. ngModel pre-defined attribute part of FormsModule. So we have to import this module in app.module.ts file.**

**Angular Forms**

**Angular support two type of forms**

1. **Template Driven Form : view 🡪 component**

**Easy to develop. Good for develop small application**

1. **Model Driven form or reactive forms: component 🡪 view**

**Complex we have to write more code in ts file. Good for complex forms.**

**Template Driven Form**

**In template driven form we have to create the form reference**

**<form #loginRef=”ngForm”>**

**</form>**

**ngForm is a pre-defined attribute part of FormsModules so we have to import FormsModule in app.module.ts in import section.**

**Name :<input type=”text” #nameRef/>**

**Password :<input type=”password” #passRef/>**

**Model Driven Form**

**According to model driven form or reactive forms**

**Every html form component is known as FormControl**

**Like textField, passwordField, radiobutton, checkbox, dropdown etc.**

**FormControl must be wrap inside a FormGroup.**

**Login Page -🡪 FormGroup**

**TextField🡪FormControl**

**Password 🡪FormControl**

**FormGroup contains one or more form control as well as another form group.**

**Angular provide FormGroup and FormControl pre-defined API to bind with Reactive forms.**

**In component we have to create FormGroup reference and which can contains more than one FormControl reference as well as FromControl.**

**Those reference we have to bind with in html using formGroup and formControlName attribute.**

**formGroup and formControlName pre-defined attribute part of ReactiveFormsMdodule. So we have to import ReactiveFormsModule in app.module.ts file in import section.**

**If we write the business logic in component it may be simple or complex that logic become local to that component.**

**But if we want to some business logic globally accessible in all component then we can use service class.**

**Service**

**Component1 fun1()**

**Component2 fun2() fun()**

**Component3 fun3();**

**Angular Service divided into two types**

1. **User-defined service**
2. **Creating service class object using new keyword**
3. **Creating service class object using DI**
4. **Pre-defined service**

**IOC : Inversion of control : rather than creating or maintain any resource explicitly allow to maintain and create by container. If container create it maintain properly. Design patter or programming pattern.**

**IOC is concept.**

**DI : Dependency Injection**

**Implementation of IOC is DI.**

**Constructor base**

**Setter base**

**Interface base**

**But angular support only one type of DI is constructor base.**

**If we want to make DI for service class we have to create normal type script class with @Injectable decorator.**

**Then provide the details about service class in app.module.js file in provide section.**

**Angular provide pre-defined HttpClient API which help to call REST API or load json Data.**

**So in service class we have to do the DI for HttpClient.**

**HttpClient is a predefined API part of HttpClientModule so we have to import HttpClientModule in app.module.ts file.**

**Day 6**

**09-08-2021**

**Directives**

**HttpClient repeat**

**Routing**

**Node JS provide pre-defined module ie**

**json-server**

**which help to make static json file as a server.**

**npm install –g json-server**

**sudo npm install –g json-server**

**json-server filename.json**

**component creation**

**ng g c componentname**

**creating service using ng command**

**ng g s servicename**

**directives is use to add extra behaviour to DOM.**

**3 types of directives**

1. **Component directive : component is a type of directive which help to create user-defined tags.**

**Component connected to html ie template. Template contains static as well as dynamic data. We share the data between component to template using data binding.**

1. **Structure directive : it is use to add or remove dom elements.**

**\*ngIf and \*ngFor**

**Using this attribute we can achieve if and looping statement in html page.**

1. **Attribute directive**

**ngStyle and ngClass like a inline css and class selector.**

**json-server questions.json --port 3001**

**angular routing : using Angular routing we can navigate from one component’s template to another component’s template using path provided in routing file.**

**ng new angular-login-routing**

**Routing option – yes**

**Styling – CSS**

**ng g c about-us**

**ng g c contact-us**

**ng g c login**

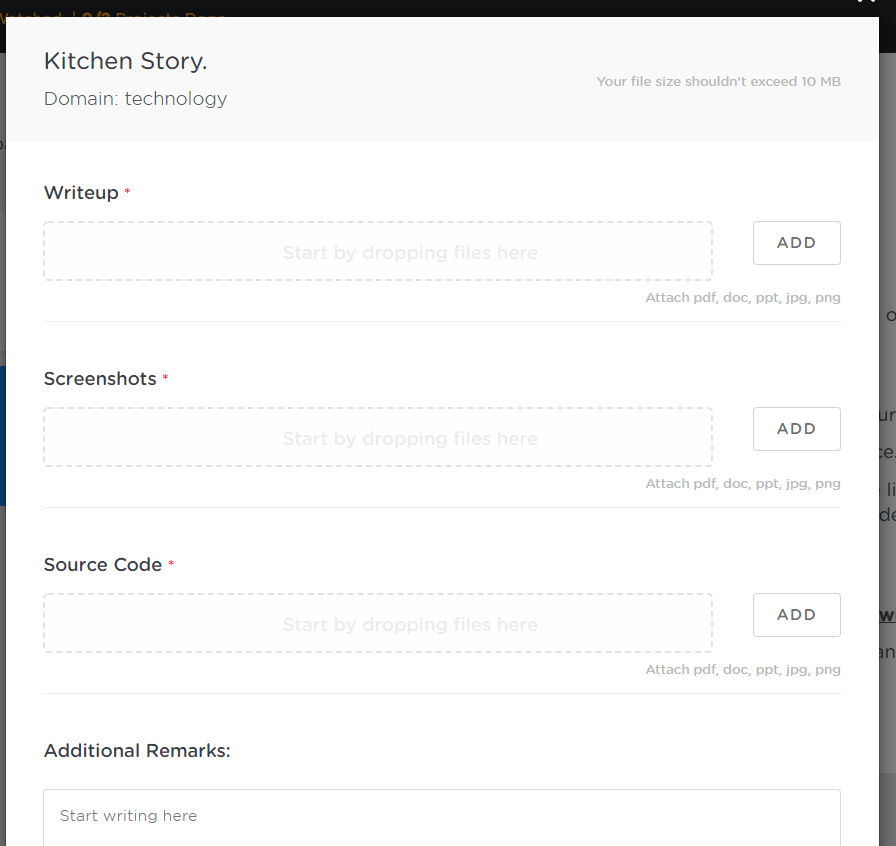
**ng g c register**

**ng g s login**

**Angular provided pre-defined tag ie**

**<router-outlet></router-outlet>**

**This tag is behave like a place holder which help to load the component contents base upon the path provided in url or using hyperlink or button.**

****

**You have to create the project and push the project in your git account.**

**You have to create the document**

**Inside document you have to provide the GIT URL.**

**Technologies. Angular and JSON Server**

**HTM/CSS/JavaScript and JSONServer**

**Project structure description**

**Steps to run the application.**

**Screen shot**

**Source code**

**Index.html**

**Sample.js**

**App.component.html**

**App.component.ts**

**App.module.js**

**Login.serve.ts**

import { HttpClient } from '@angular/common/http';

import { Injectable } from '@angular/core';

import { Observable } from 'rxjs';

import { Login } from './login.model';

@Injectable({

  providedIn: 'root'

})

export class LoginService {

  //DI for HttpClient

  constructor(public http:HttpClient) { }

  // we are converted all json data to login array object.

  getLoginDetails():Observable<Login[]> {

    return this.http.get<Login[]>("http://localhost:3000/login")

  }

  //post is use to store the records

  //it takes 2 parameter 1st parameter url

  // 2nd parameter is data in json format.

  createLoginAccount(login:Login){

    this.http.post("http://localhost:3000/login",login).

    subscribe(result=>console.log(result),error=>console.log(error));

  }

}

**Mongo DB : No SQL Databases**

**Mongo DB is open source No SQL Databases.**

**Mongo DB is schema less no sql database.**

**RDBMS : Oracle, MySQL, Db2, SQL server 2020 etc.**

**Table 🡪Emp**

**Id Name Age PhNumber**

**1 Raj 34 null**

**2 Seeta 23 null**

**3 Meeta 33 99000**

**4 Veeta null 9911**

**REST API**

**Java(Spring boot), Asp.net, Python, Node JS**

**We can consume and produce the data in json format.**

**No SQL Databases.**

**CRUD Operation**

**Mongo DB store the data using document format with the help of JSON**

**In window OS**

**First install mongo DB**

**C:\Program Files\MongoDB\Server\5.0\bin**

**By default mongo all database path consider as**

**In c drive data/db**

**From this path**

**First run mongod**

**Then run mongo command to run the application.**

**Mongo db commands**

**Show dbs**

**Show databases**

**If you want to create the database**

**use databaseName; : it will create and switch to that database.**

**In Mongo DB table is known as Collection**

**db.createCollection(“CollectionName”);**

**show collections**

**show tables**

**In collection we can store more than documents.**

**In Mongo DB records is known as documents.**

**In mongo DB every document store independently.**

**To view the document from the collection**

**db.CollectionName.find();**

**to retrieve the records using index position.**

**db.CollectionName.find()[index];**

**to view particular fields value using index position.**

**To retrieve more than one fields values.**

**db.CollectionName().find({},{propertyName:1,propertyName:1});**

**retrieve the document base upon the conditions**

**db.collectionName.find({propertyName:value})**

**Trainer Student**

**TId TName Tech Sid Sname Age**

**100 Raj Java 1 Seeta 21**

**100 Raj Java 2 Meeta 22**

**100 Raj Java 3 Reeta 23**

**Normalization is use to break the table base upon the type of NF.**

**Join is use to do De-normalization.**

**Trainer**

**PK**

**TId TName Tech**

**100 Raj Java**

**Student**

**PK FK**

**Sid SName Age TSId**

**1 Seeta 21 100**

**2 Meeta 22 100**

**3 Reeta 23 103**

**Mongo DB we can achieve relationship using two ways**

1. **Embedded style**
2. **Linking style**

**Employee**

**{“empId”:100,name:”Ravi”,age:21}**

**Address**

**{“city”:”Bangalore”,”state”:”Kar”}**

**{\_id:100,name:”Ravi”,age:21,**

**address:**

**{city:”Bangalore”,state:”Kar”},**

**projects:[{pid:1122,tech:”Java”},**

**{pid:1123,tech:”Python”},**

**{pid:1124,tech:”Angular”}]**

**}**

**Phase 2**

**Day 1**

**12-08-2021**

**Java**

**DevOps**

**Agile :**

**Git :**

**Core Java :**

**Basic Programming**

**Oops Concept**

**Object, class, Encapsulation,**

**Abstraction, Inheritance, Polymorphism**

**Static, abstract and final.**

**Interface.**

**Package**

**User-defined**

**Built in packages**

**Lang**

**Exception Handling**

**IO package**

**Util package**

**Collection framework and data structure**

**Generic**

**JDBC**

**Java 8 Features**

**Default interfaces and static interfaces**

**Functional interfaces**

**Lambda expression**

**Stream API.**

**Date API.**

**Unit testing.**

**Data structure**

**Internalization, Regular Expressions and reflection**

**Maven**

**1: Display all files in ascending order.**

**2: Sub Option**

1. **Add new file (take the file name through keyboards).**
2. **Delete the file may be present or not**
3. **Search file**
4. **Exit sub option.**

**3: Exit**

**Git : git is known as sub version control system.**

**Sub Version control system is use to record the application or project flow.**

**Online Shopping**

**Git is type of distributed sub version control system.**

**git –version**

**To make the folder as local repositories**

**git init**

**This command is use to check the current status of local repositories.**

**git status**

**To add the file from file system to staging area.**

**git add filename**

**To move the file from staging area to local repositories.**

**git commit –m “commit message”**

**Remote repositories :**

**Git hub**

**Git lab**

**Aws : code commit**

**Azure :**

**Cloud machine which provide the remote repositories.**

**To link local repositories to remote repositories we have to run the command ad**

**Git remote add origin URL**

git remote add origin https://github.com/Kaleakash/test\_java\_app.git

**git push**

**git push –u origin branchName**

**git push –u origin HEAD HEAD consider as last commit.**

**Git branch :**

**Manager : project skeleton**

**Developer1 Developer2**

**To create local repositories from existing remote repositories**

**git clone URL**

**git branch**

**branch is like a pointer which hold more than one commit details.**

**This command is use to display all branches present in local repositories.**

**git branch :**

**to create new branch command is**

**git branch branchName**

**git checkout branchName**

**This command is use to merge other branch code in current branch.**

**git merge branchName**

**To delete the branch we have to use the command as**

**git branch –D branchName**

**Java**

**Collection Framework.**

**Exception**

**Interface**

**Multithreading**

**Java 8**

**Polymorphism**

**File handling**

**What is Java :**

**Java is platform independent and pure object oriented programing language.**

**70 era --- C**

**80 Era --- C++**

**90 -- Java (Oak)**

**Nov 1995 rename Java.**

**2000 .net**

**2010 JavaScript and Big data and Hadoop**

**2020 AI and Python**

**Features of Java**

1. **Simple :C Pointer, destructor, friend keyword, virtual keyword, operator overloading, pointer object.**

**In Java they remove this concept. In Java Pointer is there.**

**But we are not working on pointer. Memory task taken care by JVM.**

1. **Compiler and interpreter:**
2. **Pure object oriented programming language.**
3. **Portable : machine must be change not os**
4. **Platform independent: machine with os must be change.**
5. **Exception handling**
6. **Multi threading**
7. **Auto GC**
8. **Robust**
9. **Secure**
10. **Distributed using RMI and EJB**

**RMI-🡪EJB🡪 Web Service (SOAP)--🡪REST Web Service**

**Spring Framework -🡪 Spring boot**

**Spring REST-🡪 Spring micro services**

**Object :**

**Class :**

**Class syntax**

**class className {**

**variables;**

**methods;**

**}**

**Pascal naming rules :**

**One world first letter must be upper case**

**If more than one world each world first letter must be upper case.**

**Demo**

**Employee**

**EmployeeDetails**

**Simple Java Programs**

**class Demo {**

**public static void main(String args[]) {**

**System.out.println("Welcome to Java");**

**System.out.print("Welcome to Java");**

**System.out.printf("Welcome to Java");**

**}**

**}**

**Data types**

**2 types**

1. **Primitive data types : it is use to store only value**

**8 types**

1. **byte 1**
2. **short 2**
3. **int 4**
4. **long : without decimal 8**
5. **float 2**
6. **double : with decimal 8**
7. **char : single character 2**
8. **boolean : true or false 1 bit**

**Data types example**

**class Demo {**

**public static void main(String args[]) {**

**int a=10;**

**System.out.println("The value of a is "+a);**

**System.out.printf("The value of a is %d\n",a);**

**}**

**}**

**Type casting : converting from one data type to another data types is known as type casting**

1. **implicit**
2. **explicit**

**int family**

**-------🡪 Implicit -----------🡪**

**byte short int long**

**🡨---------Explicit ----------------**

**Syntax of type casting**

**datatype variableName = (type)variableName**

**Type casting Example**

**class Demo {**

**public static void main(String args[]) {**

**byte a=10;**

**short b=a; // implicit**

**short c =20;**

**byte d=(byte)c; // explicit**

**System.out.println(a);**

**System.out.println(b);**

**System.out.println(c);**

**System.out.println(d);**

**}**

**}**

**Implicit**

**Int ----🡪 float**

**🡨----**

**In Java every decimal number consider as double. Double data type size id 8 byte.**

**class Demo {**

**public static void main(String args[]) {**

**int a=10;**

**float b=a; // implicit**

**System.out.println(a);**

**System.out.println(b);**

**//float c=(float)20.0;**

**float c = 20.0f;**

**int d = (int)c; // explicit**

**System.out.println(c);**

**System.out.println(d);**

**}**

**}**

**class Demo {**

**public static void main(String args[]) {**

**byte a = 127; // range of byte data is -128 to 127**

**}**

**}**

**Operators**

**Arithmetic operator +, -, \*, /, %**

**Conditional operator >, >=, <, <=, ==, !=**

**Assignment operator =**

**Increment and decrement ++, --**

**Increment and decrement operator**

**Increment ++ it increment the value by 1**

**Pre increment ++a**

**Post increment a++**

**Both are increment the value by one**

**But if we use pre and post increment inside some expression and assign to another variable then pre and post behave differently.**

**Pre means first increment and then use or assign.**

**Post means first use or assign and then increment.**

**Decrement – it decrement the value by 1**

**Increment and decrement example**

**class Demo {**

**public static void main(String args[]) {**

**int a=10;**

**int b=a;**

**System.out.println(b);**

**b = a++; //increment and store**

**System.out.println(b);**

**b = ++a; // store and increment**

**System.out.println(b);**

**}**

**}**

**If statement**

**Simple if**

**If else**

**Nested if**

**Switch statement**

**Switch statement**

**Taking the value through keyboards in Java**

**Using Scanner class**

**Using DataInputStream**

**Using BufferedReader**

**Command line arguments**

**Scanner is a pre-defined class part of util package. Package is a collection of classes and interfaces.**

**Syntax to create the Scanner class object.**

**Scanner sc = new Scanner(System.in);**

**So when we crate the Scanner class object we will get the error. We have to import util package**

**import java.util.Scanner;**

**nextByte()**

**nextShort()**

**nextInt();**

**nextLong();**

**nextFloat();**

**nextDouble();**

**nextBoolean();**

**String : In Java String is a pre-defined class or also known as reference data types.**

**import java.util.Scanner;**

**class Demo {**

**public static void main(String args[]) {**

**Scanner sc = new Scanner(System.in);**

**System.out.println("enter the id");**

**int id = sc.nextInt(); // scan integer value;**

**sc.nextLine(); // to hold enter key**

**System.out.println("enter the name");**

**//String name = sc.next(); // it takes only one word**

**String name = sc.nextLine(); // it takes more than one word but default terminator for**

**// this method is enter key**

**System.out.println("The value is "+id);**

**System.out.println("The value is "+name);**

**}**

**}**

**Switch example**

**import java.util.Scanner;**

**class Demo {**

**public static void main(String args[]) {**

**Scanner sc = new Scanner(System.in);**

**System.out.println("Enter the value of ch");**

**int ch = sc.nextInt();**

**switch(ch){ // ch can be int family,char or string.**

**case 1 : System.out.println("block 1");**

**break;**

**case 2 : System.out.println("block 2");**

**break;**

**case 3 : System.out.println("block 3");**

**break;**

**default:System.out.println("wrong block");**

**break;**

**}**

**System.out.println("finish");**

**}**

**}**

**Looping**

**While loop**

**Do while loop**

**For loop**

**Assignment 1**

**do {**

**1: English 2: Math 3: GK 4: Wrong option**

**switch()**

**{**

**case 1: welcome English exam**

**break**

**case 2: welcome Math exam**

**break**

**}**

**Do you want to continue. Yes/no or 1/0**

**}while(); condition.**

**Each block mandatory and only one time.**

1. **Non primitive data types or reference data types : it is use to store value as well as reference of another data types.**

**array**

**class : pre-defined or user-defined**

**interface : pre-defined or user-defined**

**enum**

**array : array is a type of reference data type which is use to store more than one value of same types.**

**syntax**

**int abc[10]; in C/C++**

**int abc[]; in Java**

**array example**

**class Demo {**

**public static void main(String args[]) {**

**int abc1[];**

**int []abc2;**

**int [] abc3;**

**int[] abc4;**

**}**

**}**

**In Java introduce one of the type of loop ie**

**Enhanced loop**

**for(datatype variableName : arrayName) {**

**}**

**Array with looping**

**class Demo {**

**public static void main(String args[]) {**

**int []abc={10,20,30,40,50,60,150,45,78,23,45,67,89,100};**

**System.out.println(abc[0]);**

**System.out.println(abc[1]);**

**System.out.println("Using for loop");**

**for(int i=4;i<abc.length;i=i+2) { // for loop we can customize**

**System.out.println(abc[i]);**

**}**

**System.out.println("Using for enhaced loop");**

**for(int n : abc) { // retrieve element one by one from begining to end**

**System.out.println(n);**

**}**

**}**

**}**

**Creating memory for array**

**datatype []arrayName=new datatype[size];**

**int abc[]=new int[10];**

**float xyz[]=new float[20];**

**String []name= new String[10];**

**Taking more than one ids and names through keyboards using array concept.**

**import java.util.Scanner;**

**class Demo {**

**public static void main(String args[]) {**

**Scanner obj = new Scanner(System.in);**

**System.out.println("how many records do you want to store");**

**int n = obj.nextInt();**

**int id[]=new int[n];**

**String names[]=new String[n];**

**System.out.println("Plz enter the records one by one");**

**for(int i=0;i<n;i++) {**

**System.out.println("Enter the id");**

**id[i]=obj.nextInt();**

**System.out.println("Enter the name");**

**names[i]=obj.next();**

**}**

**System.out.println("All ids and names are ");**

**for(int i=0;i<n;i++) {**

**System.out.println("id is "+id[i]+" Name is "+names[i]);**

**}**

**}**

**}**

**Oops : Object oriented programming**

**object : object is any real world entity.**

**Properties or state ---have – variables/fields**

**person**

**behaviour --- do/does --- functions / methods**

**bank**

**animal**

**car**

**Account**

**Customer**

**Employee**

**class : blue print of object or template of object or collection object which have same properties and behaviour or class is user/ reference data type which use to describe the objects.**

**Syntax to create the memory of user-defined class**

**Classname referenceName = new ClassName();**

**Calling method belong to class**

**referenceName.methodName();**

**assessing value to instance variable**

**referenceName.variableName = value;**

**class Car {**

**int wheel;**

**float price;**

**String color;**

**void start() {**

**System.out.println("Car Start");**

**}**

**void appliedGear() {**

**System.out.println("Applied Gear");**

**}**

**void moving() {**

**System.out.println("Car is moving");**

**}**

**void stop() {**

**System.out.println("Car Stop");**

**}**

**}**

**class CarTest {**

**public static void main(String args[]) {**

**//System.out.println("Main method");**

**Car innova = new Car(); // one heap memory created..**

**innova.start();**

**innova.stop();**

**}**

**}**

**Types of variable/fields**

**In Java variable are divided 3 types**

1. **Instance variable**
   1. **The variable which declare outside method but inside a class is known as instance variable.**
   2. **Instance variable hold default value according to their data types. like**

**int family 0**

**float family 0.0**

**char 🡪 space (white)**

**boolean 🡪 false**

**String 🡪 null**

* 1. **Instance variable we can access directly in all methods but method must part of same class and it must be non-static methods.**

1. **Local variable** 
   1. **The variable which declare inside a method is known as local variable.**
   2. **c**
   3. **The scope of the variable within that method or block where it declare.**
2. **Static variable**

**Create two classes Employee and EmployeeTest**

**Employee class contains three instance variable id,name,salary**

**Create Scanner class object in employee class**

**Three function**

**read(), calSalary(), display()**

**read to receive the value through keyboards**

**calSalary()**

**three local hra, da, and pf**

**hra = 10% on salary**

**da = 5% on salary**

**pf = 7% on salary**

**salary + hra + da – pf**

**you have to find gross salary but not no gross salary variable has to declare.**

**Display**

**Id,name and salary(salary must be gross salary)**

**Employee Test main method and create the object of the class.**

**And class read, calSalary and display**

**Phase 2**

**Day 3**

**16-08-2021**

**Encapsulation : Binding or wrapping data(variables) and code(methods) in a single unit is known as encapsulation.**

**Class is a good example for Encapsulation.**

**Simple example**

**class Employee {**

**String name;**

**float salary;**

**void dis() {**

**System.out.println("name is "+name);**

**System.out.println("salary is "+salary);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Employee emp1 = new Employee();**

**emp.name = "Raj";**

**emp.salary = 12000;**

**emp.dis();**

**}**

**}**

**According to encapsulation we have to make instance variable as private. If variable is private we can’t assign the value for those variable directly as well as through object also.**

**If local variable or parameter variable and instance variable have same name then local variable hide the visibility of instance variable.**

**If you want to refer the instance variable we have to use this.instanceVariblename**

**class Employee {**

**private String name;**

**private float salary;**

**public void setValue(String name, float salary) {**

**this.name = name; //both are local varaible**

**if(salary<0) {**

**this.salary = 8000;**

**}else {**

**this.salary = salary;**

**}**

**}**

**void dis() {**

**System.out.println("name is "+name);**

**System.out.println("salary is "+salary);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Employee emp1 = new Employee();**

**//emp1.name = "Raj";**

**//emp1.salary = -12000;**

**emp1.setValue("Raj",12000);**

**emp1.dis();**

**}**

**}**

**Inheritance :**

**Inheritance is use to inherits the properties and behaviour of old class to new class.**

**class OldClass { // super class, base class or parent //class**

**properties**

**behaviour**

**}**

**class NewClass exends OldClass{**

**properties // sub class or child class or**

**behaviour // derived class**

**}**

**So in Java to achieve inheritance we have to use the keyword as extends.**

**With help of sub class object we can access its own as well as super class properties and behaviour. But with the help of super class object we can access only its properties and behaviour.**

**class A {**

**void dis1() {**

**System.out.println("A class method");**

**}**

**}**

**class B extends A{**

**void dis2() {**

**System.out.println("B class method");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**A obj1 = new A();**

**obj1.dis1();**

**B obj2 = new B();**

**obj2.dis2();**

**obj2.dis1();**

**}**

**}**

**Types of Inheritance :**

1. **Single inheritance : one super class and one sub class**

**Class A {}**

**Class B extends A {}**

1. **Multilevel inheritance : one super class and n number of sub class one by one.**

**Class A {}**

**Class B extends A {}**

**Class C extends B{}**

**Class D extends C{}**

1. **Hierarchical inheritance : one super class and n number of sub classes connected directly to super class.**

**Class A {}**

**Class B extends A{}**

**Class C extends A{}**

**Class D extends A{}**

1. **Multiple inheritance : more than one super class and one sub class**

**Class A {}**

**Class B {}**

**Class C extends A,B {} //Error in this like**

**So java doesn’t support this type of inheritance directly it support in directly using interface.**

**OOPs Relationship**

1. **is a relationship : Inheritance using class or interface.**
2. **Has a relationship :**

**class Employee {**

**id,name,salary**

**}**

**class Address**

**city, state, pincode**

**}**

**class Manager extends Employee{**

**Address add = new Address();**

**numberOfEmp;**

**}**

**class Programmer extends Employee{**

**projectName;**

**}**

**Super class must be generic and sub class must be specific.**

**Manager is a Employee**

**Programmer is a Employee**

**Inside one class we are creating the object of other class.**

**Manager Has a Address**

**Has a relationship**

**Association**

**Aggregation**

**Composition**

**class A {**

**B class zero, one object or many object.**

**}**

**class B {**

**A class zero, one object or many object.**

**}**

**Association either any side we have to create one or many object of opposite class. Then class relationship is known as association.**

**class Employee {**

**Address object may one or many.**

**}**

**class Address {**

**}**

**It is also known as association but weak association the weak association is known aggregation.**

**class Student {**

**StudentHistory object**

**}**

**class StudentHistroy {**

**}**

**It is also known as association but it strong association. Without student class StudentHistory class cant’ present.**

**This association is known composition**

**Polymorphism : one name many forms.**

**2 types**

**Compile time run time**

**Static binding dynamic binding**

**Early binding late binding**

**Method overloading Method overriding**

**The method have same name and different parameter list is known as methods overloading.**

**10+20**

**Raj + Deep**

**10 + ”Raj”**

**Raj + 20**

**Method overloading**

**class Abc {**

**void add(int a, int b) {**

**System.out.println(a+b);**

**}**

**void add(int a, int b, int c) {**

**System.out.println(a+b+c);**

**}**

**void add(float a, float b) {**

**System.out.println(a+b);**

**}**

**void add(String a, String b) {**

**System.out.println(a+b);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Abc obj = new Abc();**

**obj.add(10,20,30);**

**obj.add(10,20);**

**obj.add(10.10f,20.20f);**

**obj.add("10","20");**

**}**

**}**

**Method Overriding**

**The method have same name and same signature(number of parameter, type of parameter and return type) must be same.**

**To achieve method overriding it require inheritance concept.**

**Java provide annotation concept.**

**Annotation means meta-data. Data about data.**

**All annotation start with @ followed by annotation name.**

**@Override**

**Method overriding example**

**class Bike {**

**void speed() {**

**System.out.println("60km/hr");**

**}**

**}**

**class Tvs extends Bike {**

**void color() {**

**System.out.println("Black");**

**}**

**@Override**

**void speed() { // merge both method logic as one.**

**System.out.println("50km/hr");**

**super.speed(); // calling super class speed method code.**

**}**

**}**

**class Honda extends Bike { // re-usablility for speed() method**

**void color() {**

**System.out.println("Black");**

**}**

**}**

**class Pulsar extends Bike {**

**@Override**

**void speed() { // don't like override**

**System.out.println("90km/hr");**

**}**

**void color() {**

**System.out.println("Red");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Honda hh = new Honda(); hh.color(); hh.speed();**

**Pulsar pu = new Pulsar(); pu.color(); pu.speed();**

**Tvs tv = new Tvs(); tv.color(); tv.speed();**

**}**

**}**

**Abstract :**

1. **abstract is a keyword we can use with method and class but not with variable.**
2. **Abstract method : the method without body or without curly braces or incomplete method is known as abstract method.**

**abstract returnType methodName(parameterList);**

1. **If class contains one or more abstract method we have to declare class as abstract class.**

**abstract class className {**

**}**

1. **Whichever class extends abstract class that class must be provide the body for all abstract method belong to that class. that class can ignore if that class itself is a abstract class.**
2. **Abstract class can contains normal as well as abstract methods. It can contains zero or 1 or many abstract methods.**
3. **Abstract class can contains default as well as parameterized constructor.**

**final**

**final keyword we can use with variable, method and class.**

**final variable : to declare constant value we use final keyword with variable.**

**final int A=10;**

**A=20; Error**

**final method : if method is final we can’t override that method. final method can use it but can’t override.**

**final class : if class is final we can’t inherit or extends.**

**final class Bike {**

**final void speed(){**

**System.out.println("60km/hr");**

**}**

**}**

**class Honda extends Bike {**

**/\*@Override**

**void speed() { // don't like override**

**System.out.println("40km/hr");**

**}\*/**

**void color() {**

**System.out.println("Black");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**final int A=10;**

**//A=20;**

**Honda hh = new Honda();**

**hh.speed();**

**}**

**}**

**We can’t use final and abstract together.**

**static : static keyword we can use with variable and method but not with class(if class in inner class we can use static keyword).**

1. **If variable is static we can access the variable without creating the object of that class with help of class name. We can assign the value for static variable with the help of object also.**
2. **If method is static we can call that method with the help of class name without object. Static method can call through object also.**
3. **Inside non static method we can access static as well as non static variable.**
4. **Inside static method we can access only static variable directly. We can’t access non static variable.**

**class Abc {**

**int a;**

**static int b;**

**void dis1() {**

**System.out.println("Non static method");**

**System.out.println("a "+a);**

**System.out.println("b "+b);**

**}**

**static void dis2() {**

**System.out.println("Static method");**

**System.out.println("a "+a);**

**System.out.println("b "+b);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Abc obj1 = new Abc();**

**obj1.dis1();**

**Abc.dis2();**

**obj1.a=100;**

**Abc.b=200;**

**obj1.b=300;**

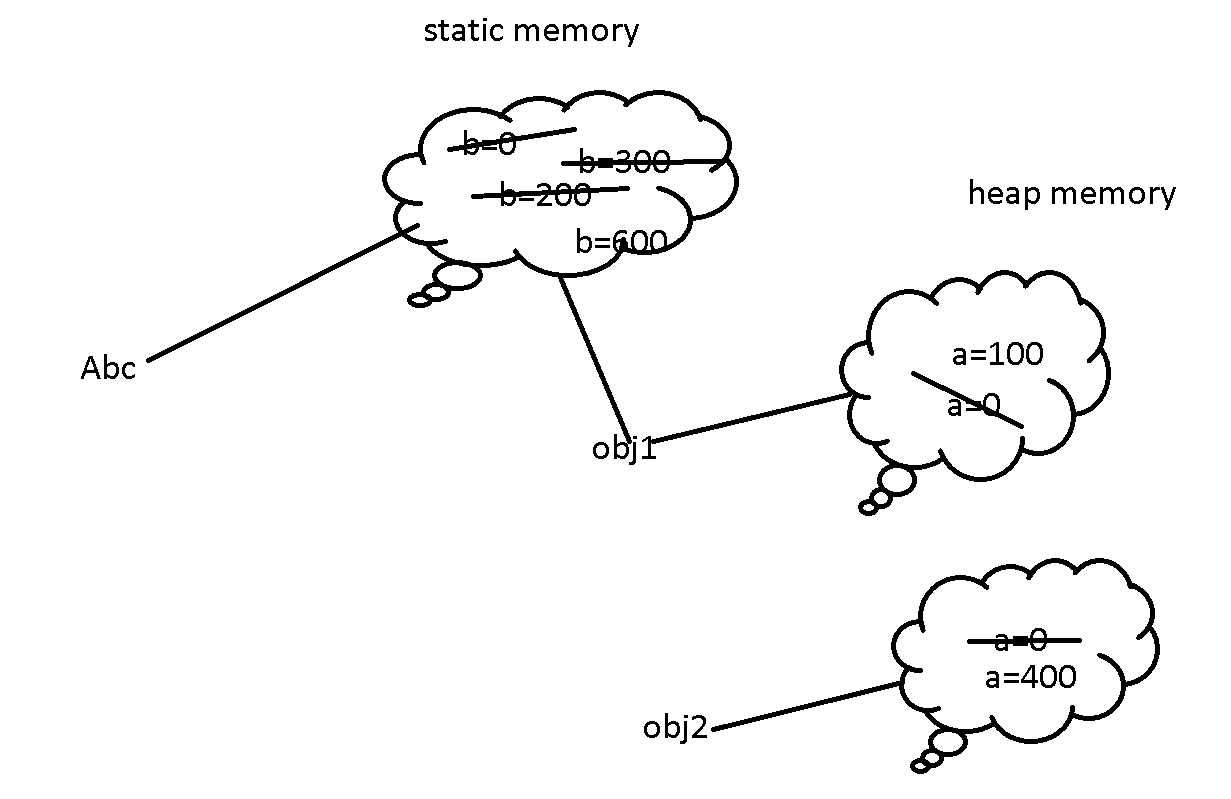
**}**

**}**

**Every class contains only one static memory**

**Every class can contains more than one heap memory base upon the number of object created.**

**Static is global to all object.**

****

**class Abc {**

**int a;**

**static int b;**

**void dis() {**

**System.out.println("Non static method");**

**System.out.println("a "+a);**

**System.out.println("b "+b);**

**}**

**static void dis1() {**

**Abc obj1 = new Abc();**

**System.out.println(" a "+obj1.a);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Abc.dis1();**

**Abc obj1 = new Abc();**

**Abc obj2 = new Abc();**

**obj1.a=100;**

**Abc.b=200;**

**obj1.b=300;**

**obj2.a=400;**

**Abc.b=500;**

**obj2.b=600;**

**obj1.dis(); // a= 100 b=600**

**obj2.dis(); //a=400 b=600**

**}**

**}**

**Interface : interface is type of reference data type. Which is also known as 100% pure abstract class till Java7.**

**Interface contains final variable and abstract methods.**

**Syntax to declare the interface**

**interface interfaceName {**

**fields;**

**methods;**

**}**

**By default all fields in interface are public static and final.**

**By default all methods are public and abstract.**

**Like a class one interface can extends another interface. Interface can extends More than one interface.**

**Class always implements interface and class can implements more than one interface.**

**Which ever class implements any interface that class must be provide the body for all abstract method mandatory.**

**interface Abc {**

**int A=10;**

**void dis1();**

**}**

**interface Xyz {**

**int B=20;**

**}**

**interface Mno extends Abc,Xyz{**

**int C=30;**

**}**

**class Demo implements Abc,Xyz {**

**}**

**Access specifiers for overriding rules**

**Super class / interface sub class**

**public public**

**protected public**

**protected**

**default(nothing) public**

**protected**

**default (nothing)**

**private can’t override.**

**interface Abc {**

**int A=10;**

**void dis1();**

**}**

**interface Xyz {**

**int B=20;**

**void dis2();**

**}**

**interface Mno extends Abc,Xyz{**

**int C=30;**

**void dis3();**

**}**

**class Demo implements Abc,Xyz {**

**public void dis1() {**

**System.out.println("Abc interface method");**

**}**

**public void dis2() {**

**System.out.println("Xyz interface method");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Demo obj1 = new Demo();**

**obj1.dis1();**

**obj1.dis2();**

**}**

**}**

**super, this, super(), this()**

**class A {**

**int a=10;**

**}**

**class B extends A {**

**int a=20;**

**void dis1() {**

**int a=30;**

**System.out.println("a "+a); // local variable**

**System.out.println("instance a "+this.a); // instance variable**

**System.out.println("super a "+super.a);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**B obj1 = new B();**

**obj1.dis1();**

**}**

**}**

**this() : it is use to do constructor chaining for the same class. it must be first statement inside a constructor.**

**class A {**

**A() {**

**this(10);**

**System.out.println("A()");**

**}**

**A(int x) {**

**this(10,20);**

**System.out.println("A(int)");**

**}**

**A(int x, int y) {**

**System.out.println("A(int,int)");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**A obj1 = new A();**

**//A obj2 = new A(10);**

**//A obj3 = new A(10,20);**

**}**

**}**

**this()**

**class Employee {**

**int id;**

**String name;**

**float salary;**

**Employee() {**

**this.id = 123;**

**this.name = "Unknown";**

**this.salary = 8000;**

**}**

**Employee(int id) {**

**this();**

**this.id = id;**

**}**

**Employee(int id, String name) {**

**this(id);**

**this.name = name;**

**}**

**void dis() {**

**System.out.println("Id is "+id);**

**System.out.println("Name is "+name);**

**System.out.println("Salary is "+salary);**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Employee emp1 = new Employee(); emp1.dis();**

**Employee emp2 = new Employee(100); emp2.dis();**

**Employee emp3 = new Employee(101,"Raju"); emp3.dis();**

**//Employee emp4 = new Employee(102,"Ramesh",14000);**

**}**

**}**

**super() : super() parameter is use to do constructor chaining from sub class to super class. it must be first statement inside a sub class constructor. Every sub class constructor contains default super(). It always call super class empty constructor.**

**class Employee {**

**int empId;**

**String name;**

**Employee() {**

**System.out.println("Employee class object");**

**}**

**Employee(int empId, String name){**

**this();**

**this.empId = empId;**

**this.name = name;**

**System.out.println("Employee class object with id");**

**}**

**}**

**class Manager extends Employee {**

**int numberOfEmp;**

**Manager(int id, String name, int numberOfEmp) {**

**super(id,name);**

**this.numberOfEmp = numberOfEmp;**

**System.out.println("Manager class object");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Manager mgr = new Manager(100,"Raj",10);**

**}**

**}**

**Run type polymorphism using up casting and down casting.**

**In java we can create sub class object and super class reference. With the help of that reference we can call only those methods which belong to super class. if class override the method then we will get the sub class method output(method overriding).**

**SuperClass referenceName = new SubClass()**

**Run time polymorphism.**

**Super class normal class**

**class A {**

**void dis1() {**

**System.out.println("A class method");**

**}**

**}**

**class B extends A{**

**void dis1() {**

**System.out.println("A class method override by B class");**

**}**

**void dis2() {**

**System.out.println("B class method");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**A obj1 = new A(); obj1.dis1();**

**B obj2 = new B();**

**obj2.dis1(); obj2.dis2();**

**//B obj3 = new A();**

**A obj4 = new B(); // creating sub class object and super class reference possible up casting**

**obj4.dis1(); //obj4.dis2();**

**B obj5 = (B)obj4; // down casting**

**obj5.dis1(); obj5.dis2();**

**}**

**}**

**Super abstract class**

**abstract class A {**

**abstract void dis1();**

**}**

**class B extends A{**

**void dis1() {**

**System.out.println("A class method override by B class");**

**}**

**void dis2() {**

**System.out.println("B class method");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**A obj4 = new B(); // creating sub class object and super class reference**

**obj4.dis1(); // super class may be abstract class.**

**}**

**}**

**Super class is a type of interface.**

**interface A {**

**void dis1();**

**}**

**class B implements A{**

**public void dis1() {**

**System.out.println("A interface method override by B class");**

**}**

**void dis2() {**

**System.out.println("B class method");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**A obj4 = new B(); // creating sub class object and interface reference**

**obj4.dis1(); // super class may be abstract class.**

**}**

**}**

**Run time polymorphism example**

**interface A {**

**int add(int a, int b);**

**}**

**interface B {**

**int sub(int a, int b);**

**}**

**class Server implements A,B{**

**public int add(int a, int b) {**

**return a+b;**

**}**

**public int sub(int a, int b) {**

**return a-b;**

**}**

**void ownMethod() {**

**System.out.println("Own Method");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**Server s1 = new Server();**

**System.out.println(s1.add(1,2));**

**System.out.println(s1.sub(10,2));**

**s1.ownMethod();**

**A obj1 = new Server();**

**System.out.println(obj1.add(1,2));**

**//System.out.println(obj1.sub(10,2));**

**B obj2 = new Server();**

**//System.out.println(obj2.add(1,2));**

**System.out.println(obj2.sub(10,2));**

**}**

**}**

**Java Oops Concept.**

**Exception Handling**

**Exception is a object which occurs when unexpected things happen during the execution of a programs.**

**Java program**

**Compile program run program**

**javac java**

**javac java**

**compile interpreter**

**compile time error run time error**

**syntax error**

**or typo error**

**command line argument**

**class Test {**

**public static void main(String args[]) {**

**for(String str:args){**

**System.out.println(str);**

**}**

**}**

**}**

**java Test.java**

**java Test.java v1 v2 v3**

**run time error**

**Error Exception**

**Error and Exception both are pre-defined classes part of lang package. By default every java program import lang package.**

**Error : The error which generate at the run time which we can’t handle it. Example JVM crash, out of memory, software or hardware issue.**

**Exception : The error which generate at the run time which we can handle it. ArithmeticException, NumberFormatException etc.**

**Object**

**Throwable**

**Exception**

**Checked exception unchecked exception**

**RuntimeException**

**IOException ArithmeticException**

**FileNotFoundException ArrayIndexOutOfBoundsException**

**ClassNotFoundException NumberFormatException**

**SQLException NullPointerException**

**Etc etc**

**javap java.packageName.className/interface**

**To handle both the type of exception java provide 5 keyword**

**try**

**catch**

**finally**

**throw**

**throws**

**try catch block**

**ArithmeticException**

**class Test {**

**public static void main(String args[]) {**

**System.out.println("Hi");**

**int a =10;**

**int b=0;**

**try{**

**int res = a/b; // new ArithmeticException();**

**System.out.println("res "+res);**

**}catch(Exception e) {**

**//System.out.println("I Take Care!");**

**System.out.println(e.toString());**

**System.out.println(e);**

**}**

**System.out.println("Bye..");**

**System.out.println("Bye..");**

**System.out.println("Bye..");**

**}**

**}**

**Array index out of bounds exception**

**class Test {**

**public static void main(String args[]) {**

**System.out.println("Hi");**

**int a =10;**

**int b=1;**

**int abc[]={10,20,30,40};**

**try{**

**int res = a/b;**

**int res1 = abc[5]; // ArrayIndexOutOfBoundsException**

**System.out.println("res "+res);**

**System.out.println("res1 "+res1);**

**}catch(Exception e) {**

**//System.out.println("I Take Care!");**

**System.out.println(e.toString());**

**System.out.println(e);**

**}**

**System.out.println("Bye..");**

**System.out.println("Bye..");**

**System.out.println("Bye..");**

**}**

**}**

**Try with multiple catch block**

**If any exception generate you want execute generic solution then you can write try with single catch block.**

**But if you want to execute specific task depends upon the type of exception then we have to write with multiple catch block.**

**In multiple try and catch block sub classes must be first and super classes must be bottom. In catch block hierarchy.**

**Try with multiple catch block code**

**import java.util.\*;**

**class Test {**

**public static void main(String args[]) {**

**System.out.println("Hi");**

**int a =10;**

**int b=1;**

**Scanner obj = new Scanner(System.in);**

**int abc[]={10,20,30,40};**

**try{**

**int res = a/b;**

**int res1 = abc[2];**

**System.out.println("Enter the id");**

**int id = obj.nextInt();**

**System.out.println("res "+res);**

**System.out.println("res1 "+res1);**

**System.out.println("No Exception");**

**}catch(ArithmeticException e) {**

**System.out.println("Divided by Zero "+e);**

**}catch(ArrayIndexOutOfBoundsException | InputMismatchException e) {**

**System.out.println("Array index and Input Mish Match"+e);**

**}**

**catch(Exception e) {**

**System.out.println("Generic "+e);**

**}**

**System.out.println("Bye..");**

**System.out.println("Bye..");**

**System.out.println("Bye..");**

**}**

**}**

**Finally block**

**This block will execute if any exception generate or not generate.**

**Try block : keep one line code or multiple line code inside a try block if you thing this code can generate exception.**

**Catch block : This block execute only if any exception generate.**

**Finally block : 100% sure if any exception generate or not this block execute.**

**Try**

**Catch catch catch catch finally**

**Catch finally catch**

**Finally**

**import java.util.\*;**

**class Test {**

**public static void main(String args[]) {**

**try {**

**int a =10/1;**

**System.out.println(a);**

**System.out.println("No Exception");**

**}catch(Exception e) {**

**System.out.println("Catch block");**

**}finally {**

**System.out.println("Finally block"); // to close the resources properly.**

**}**

**System.out.println("Normal Statement");**

**}**

**}**

**throw and throws**

**throw keyword is use to generate or raise pre-defined or user-defined(custom) exception depending upon the conditions.**

**Syntax**

**throw new Exception();**

**or**

**throw new ExceptionSubClass();**

**throw keyword example**

**class CantWithdraw extends Exception {**

**CantWithdraw(String msg) {**

**super(msg); // passing the message to super class constructor**

**}**

**CantWithdraw() {**

**super("Default message");**

**}**

**}**

**class Test {**

**public static void main(String args[]) {**

**int amount =500;**

**try {**

**if(amount<=500) {**

**//throw new Exception();**

**//throw new ArithmeticException();**

**//throw new ArithmeticException("amount must be < 500");**

**throw new CantWithdraw();**

**//throw new CantWithdraw("amont must be < 500");**

**}else {**

**System.out.println("No");**

**}**

**}catch(Exception e) {**

**System.out.println(e.toString());**

**}**

**}**

**}**

**throws : throws keyword is use to throw the exception to caller methods.**

**Throws keywords we use with method signature.**

**void dis() throws Exception, ExceptionSubClass{**

**}**

**Throws Exception example**

**class Test {**

**static void dis1() throws NumberFormatException{**

**try{**

**int a=10/0;**

**}catch(ArimeticException e){}**

**System.out.println("dis1() method");**

**}**

**static void dis2() throws Exception{**

**try{**

**dis1();**

**}catch(Exception e){**

**System.out.println(e);**

**}**

**System.out.println("dis2() method");**

**}**

**public static void main(String args[]) throws Exception{**

**//try{**

**dis2();**

**//}catch(Exception e){}**

**System.out.println("main() method");**

**}**

**}**

**Checked exception**

**class Test {**

**public static void main(String args[]) throws Exception {**

**System.out.println("Hi");**

**//try{**

**Thread.sleep(3000);**

**//}catch(Exception e) {}**

**System.out.println("Hello");**

**}**

**}**

**Checked exception check at compile time as well as run time.**

**We have to handle checked exception mandatory with try and catch or throws.**

**Multithreading**

**Program : set of instruction to perform a specific task.**

**Processor : responsible to execute the code.**

**Process : program in execution or time taken to execute the code.**

**Thread : small execution of a code within a process.**

**By default java a thread base programming language.**

**To check default thread details in java**

**We have to use pre-defined methods**

**currentThread(). It is a static method part of Thread class. Thread is a pre-defined class part of lang package.**

**Multi tasking**

**Process base**

**Thread base**

**Thread is light weighted. They execute independently.**

**They shared same memory space.**

**We can create thread using two ways**

1. **Extends Thread**
2. **Implements Runnable interface.**

**Extends Thread class example**

1. **Create user defined class and extends Thread class.**
2. **Create thread class reference.**
3. **Using reference class start() method part of thread. Here thread ready to run. Ie runnable state.**
4. **Start() method internally call run() part of thread class.**
5. **That run method contains empty logic. So if you want to run custom logic we have to override run method.**

**class A extends Thread{**

**@Override**

**public void run() {**

**for(int i=0;i<10;i++) {**

**System.out.println("i "+i);**

**}**

**}**

**}**

**class B extends Thread{**

**@Override**

**public void run() {**

**for(int j=0;j<10;j++) {**

**System.out.println("j "+j);**

**}**

**}**

**}**

**class Test {**

**public static void main(String args[]) throws Exception {**

**Thread obj1 = new A();**

**Thread obj2 = new B();**

**obj1.start();**

**obj2.start();**

**}**

**}**

**Implements Runnable interface**

1. **Create user defined class and implements runnable interface.**
2. **Run() method is a part of Runnable interface. So when class implements runnable interface that class must be provide the body for run() methods.**
3. **Thread class internally implements runnable interface only.**
4. **Create the reference of Runnable interface.**
5. **So using this reference we can’t call start() methods. But to start thread we required start() method.**
6. **So create Thread class reference and pass the Runnable interface reference pass a parameter in constructor to access start methods.**

**class A implements Runnable{**

**@Override**

**public void run() {**

**for(int i=0;i<10;i++) {**

**System.out.println("i "+i);**

**}**

**}**

**}**

**class B implements Runnable{**

**@Override**

**public void run() {**

**for(int j=0;j<10;j++) {**

**System.out.println("j "+j);**

**}**

**}**

**}**

**class Test {**

**public static void main(String args[]) throws Exception {**

**Runnable r1 = new A();**

**Runnable r2 = new B();**

**Thread t1 = new Thread(r1);**

**Thread t2 = new Thread(r2);**

**t1.start();**

**t2.start();**

**}**

**}**