**Day 1 : 27-08-2019**

**akash300383@gmail.com**

**Java**

**J2SE J2EE J2ME**

**JavaSE JavaEE JavaME**

**JSE JEE JME**

**Java Standard Edition**

**Java Enterprise Edition**

**Java Micro Edition**

**Basic Java**

**OOPS**

**Exception Handling**

**Multithreading**

**packages**

**lang, io, util, awt, swing**

**rmi, net, sql etc**

**AWT/Swing ------.GUI**

**SQL (JDBC )**

**Oracle /MySQL**

**https://www.google.com ---->URL**

**Http/Https(Req)-------->**

**Client Server**

**<----------Http/Htts(REs)------------------**

**HTML**

**CSS**

**HTML 1,2,3,4,5 JavaScript**

**CSS 1, 2,3**

**Web Application : Servlet /JSP /EJB**

**JEE**

**Servlet/JSP/EJB**

**Asp.net**

**Php**

**Servlet : Servlet is a normal Java Program which help to create dynamic web page on server side.**

**javax.servlet.\*: servlet is package which contains set of classes and interfaces which help to create the dynamic web page.**

**Servlet : Servlet is a interface.**

**5 methods**

**init, service, destroy, getServletInfo, ServletConfig**

**init, service, destroy is known as life cycle methods.**

**public class Demo implements Servlet {**

**5 methods must be override.**

**}**

**GenericServlet : It is type of abstract class which internally implements Servlet interface and override or provided the body for 4 methods except service**

**public class Demo extends GenericServlet {**

**We have to override only service method.**

**}**

**HttpServlet : It is type of abstract class which internally extends GenericServlet and provided the body for service methods. This class provided extra methods in the form of doXXX() like**

**doGet(), doPost(), doPut(), doDelete(), doHead() etc**

**public class Demo extends HttpServlet {**

**public void doGet(req, res){**

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

**PrintWtier pw = res.getWriter();**

**pw.println("Welcome to Servlet");**

**}**

**doPost(res,res) {}**

**}**

**DD File (Deployment Descriptor file )**

**web.xml**

**<web-app>**

**<servlet>**

**<servlet-name>A</servlet-name>**

**<servlet-class>com.Demo</servlet-class>**

**</servlet>**

**<servlet-mapping>**

**<servlet-name>A</servlet-name>**

**<url-pattern>/hi</url-pattern>**

**</servlet-mapping>**

**</web-app>**

**http://localhost:8080/DemoWebApp/hi**

**Container : It is a part of Servlet or also known a engine. Responsible to execute servlet, jsp and ejb programs.**

**Loading class, creating objects, calling life cycle methods etc.**

**2 types**

**Container**

**Web Server : Web Container : Tomcat**

**Execute only Servlet and JSP**

**Application Server : WebLogic, Jboss : Web Container**

**and EBJ Container**

**execute Servlet, JSP and EJB . Provided extra services like connection pooling, thread management, resource management, security.**

**http://localhost:9999/DemoWebApp/**

**http://localhost:9999/DemoWebApp/hi?user=Raj**

**http://localhost:9999/DemoWebApp/**

**Get**

**http://localhost:9999/DemoWebApp/hi?user=Ravi&pass=1234**

**http://localhost:9999/DemoWebApp/**

**Post**

**http://localhost:9999/DemoWebApp/hi**

**RequestDispatcher : It is interface which provided set of methods which help to navigate from one page to another page(Servlet/JSP/HTML)**

**syntax**

**RequestDispatcher rd = request.getRequestDispatcher("path");**

**Target page --> Servlet then path must be URL pattern of that Servlet**

**Target page --> jsp /html then path must be pageName.html/pageName.jsp**

**Limitation of Servlet :**

**1. Servlet is a normal java program if we do any change we have to re-compile and re-deploy the application on Server.**

**2. Servlet is complex, ie If we want to display simple message through Servlet we have make normal Java class extends /implements type of servlet then override life cycle methods, then create PrintWriter class object and give description using xml file.**

**3. If we want to write any presentation login in servlet it must be inside the pw.println("<font color=red>Welcome</font>");**

**JSP : Java Server Page : JSP is a tag based scripting language which help to create the dynamic web page on server side.**

**1. scriptlet**

**<%**

**Java coding**

**%>**

**2. Declarative tag**

**<%!**

**%>**

**Limitation of JSP**

**1. JSP Not secure.**

**2. Performance wise JSP is slower than Servlet. Because JSP is a type of Servlet Where contains internally convert JSP to Servlet ie Page Translation Phase.**

**3. JSP Code we can't do re-usability.**

**MVC**

**View --->HTML/JSP**

**Controller ---> Servlet /Filter**

**Model ----> JavaBean**

**CRUD Operation (Create/Insert, Read, Update and Delete)**

**show databases; It is use to display all databases**

**create database databaseName;**

**use databaseName;**

**create table tableName(columnName dataType, columnName dataType)**

**create table login(user varchar(10), pass varchar(10));**

**insert into login values('Raj','123');**

**select \* from login;**

**JDBC : Java Database Connectivity :**

**JDBC is a API (Application Programming interfaces) which contains set of classes and interfaces which help to database through Java Application.**

**Steps to connect the database**

**1. import java.sql.\*;**

**2. Write the method main method or user-defined methods with exception handling concepts using try-catch or throws exception. Because JDBC throw checked exception ie SQL Exception.**

**3. Load the driver : Driver is a pre-defined class provided by vendor which help to connect the database.**

**Ex : Class.forName("driverName");**

**com.mysql.jdbc\_5.1.5.jar**

**4. Establish the connection :**

**Login.JSP ----->View**

**LongController.java ----> Controller**

**Model**

**Login.java**

**LoginService.java**

**LoginDao.java**

**EJB : Enterprise Java Bean :EJB is use to create distributed high secure application.**

**Type of EJB**

**2.x 3.x**

**Session Bean Session Bean**

**Entity Bean JPA**

**Message Driven Bean MDB**

**Collection Framework :**

**Framework :Framework contains set of classes and interfaces which internally connected to each other to perform specific task. Framework is not a final product. It is protocol or template. When we develop any application using framework 70 to 80% task taken care by framework. Framework internally follow design pattern or implementation of Design pattern is taken care by All framework.**

**Struts : Internally follow MVC design pattern. It is known as Controller Centric Framework**

**JSF : Internally follow MVC design pattern. It is known as View Centric Framework.**

**Spring MVC: Internally follow MVC design pattern. It is known as Model Centric Framework.**

**View**

**JSF VS HTML5/CSS3/NodeJS/Angular/React**

**EJB Vs Spring MVC**

**Spring Framework**

**Spring is a open source, layer architecture or onion architecture framework.**

**Spring Modules**

**1. Spring core**

**2. Spring Context**

**3. Spring MVC (Model View Controller)**

**4. Spring JDBC : Java Database Connectivity**

**Spring AOP : Aspect Oriented Programming**

**5. Spring ORM (Hibernate / JPA)**

**6. Spring Rest : RestFull Web Service**

**7. Spring Data**

**8. Spring Security**

**9. Spring Cloud**

**10. Spring boot**

**etc**

**Spring Core and Context**

**IOC : Inversion of Control : IOC is design pattern. In place of creating object explicitly or maintaining resource. Allow to container to create or maintain. Whenever required from contains pull it.**

**DI : Dependency Injection :**

**DI is a implementation of IOC.**

**3 ways**

**1. Constructor base**

**2. Setter base**

**3. Interface base**

**but Spring support only Constructor and Setter base**

**DI Using XML or Annotations.**

**Spring Framework**

**Spring Core**

**Spring ApplicationContext**

**Spring MVC**

**Spring JDBC**

**Spring ORM (Hibernate /JPA)**

**Spring Transaction Management**

**Spring Rest**

**Spring Data**

**Spring Security**

**Spring boot with All those modules**

**Day 2 - 28-08-2019**

**IOC : Inversion of Control**

**DI : Dependency Injection**

**Maven : It is build tool. it is use to compile, run, creating jar or war or ear file. Creating the documentation. Maintain the dependencies.**

**pom.xml (Project Object Model)**

**Class.forName("driverName");**

**Connection con = DriverManager.getConnection(url,username,password);**

**DataSource**

**javax.sql.\*;**

**DataSource ds =**

**Connection con = ds.getConnection();**

**Spring Boot: Spring boot is a boot strap for the Spring application or Spring Modules.**

**Like Spring core, Context, Spring MVC, Spring Rest etc.**

**Spring boot itself is standalone project which help to create the RAD(Rapid Application Development).**

**Spring boot = All Spring modules - No XML File + Few Annotation + Embedded Http Server (Tomcat/ Jetty etc)**

**1. Spring boot starter : Spring boot starter which help to maintain the common related dependencies.**

**Spring.core.xxx.jar**

**spring.context.xxx.jar**

**spring.mvc.jar file**

**spring.testing.jar file**

**2. Spring Auto Configurator : In Spring module we have to configure spring modules details using xml or annotation.**

**But Spring Auto Configurator Using @SpringBootApplication annotation Configure all details without XML file.**

**@SpringBootApplication = @Configuration + @ComponentScan + @AutoConfiguration.**

**Spring MVC**

**pom.xml**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-webmvc</artifactId>**

**<version>5.0.10.RELEASE</version>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-jdbc</artifactId>**

**<version>5.0.10.RELEASE</version>**

**</dependency>**

**<dependency>**

**<groupId>mysql</groupId>**

**<artifactId>mysql-connector-java</artifactId>**

**<version>5.1.42</version>**

**</dependency>**

**index.jsp**

<html>

<body>

<h2>Hello World!</h2>

<a href=*"demo.spring"*>Call Spring MVC</a>

<br/>

<form action=*"store.spring"* method=*"post"*>

<label>Id</label><input type=*"text"* name=*"id"*><br/>

<label>Name</label><input type=*"text"* name=*"name"*><br/>

<label>Salary</label><input type=*"text"* name=*"salary"*><br/>

<input type=*"submit"* value=*"Store Emp Info"*>

</form>

</body>

</html>

**web.xml file**

<!DOCTYPE web-app PUBLIC

"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"

"http://java.sun.com/dtd/web-app\_2\_3.dtd" >

<web-app>

<display-name>Archetype Created Web Application</display-name>

<servlet>

<servlet-name>Abc</servlet-name>

<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-class>

</servlet>

<servlet-mapping>

<servlet-name>Abc</servlet-name>

<url-pattern>\*.spring</url-pattern>

</servlet-mapping>

</web-app>

**Abc-servlet.xml**

<?xml version=*"1.0"* encoding=*"UTF-8"*?>

<beans xmlns=*"http://www.springframework.org/schema/beans"*

xmlns:xsi=*"http://www.w3.org/2001/XMLSchema-instance"*

xmlns:context=*"http://www.springframework.org/schema/context"*

xsi:schemaLocation=*"http://www.springframework.org/schema/beans http://www.springframework.org/schema/beans/spring-beans.xsd*

*http://www.springframework.org/schema/context http://www.springframework.org/schema/context/spring-context-4.2.xsd"*>

<context:component-scan base-package=*"com"*></context:component-scan>

<bean id =*"ds"*

class=*"org.springframework.jdbc.datasource.DriverManagerDataSource"*

scope=*"singleton"*>

<property name=*"driverClassName"* value=*"com.mysql.jdbc.Driver"*></property>

<property name=*"url"* value=*"jdbc:mysql://localhost:3306/mydb1"*></property>

<property name=*"username"* value=*"root"*></property>

<property name=*"password"* value=*"root"*></property>

</bean>

</beans>

**com.controller**

**EmployeeController.java**

**package com.controller;**

**import javax.servlet.http.HttpServletRequest;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.stereotype.Controller;**

**import org.springframework.web.bind.annotation.RequestMapping;**

**import org.springframework.web.bind.annotation.RequestMethod;**

**import org.springframework.web.servlet.ModelAndView;**

**import com.bean.Employee;**

**import com.service.EmployeeService;**

**@Controller**

**public class EmployeeController {**

**@RequestMapping(value = "demo",method = RequestMethod.GET)**

**public ModelAndView sayHello() {**

**System.out.println("I Came here");**

**ModelAndView mav = new ModelAndView();**

**mav.setViewName("display.jsp");**

**return mav;**

**}**

**@Autowired**

**EmployeeService employeeService;**

**@RequestMapping(value = "store",method = RequestMethod.POST)**

**public ModelAndView storeInfo(HttpServletRequest req,Employee emp) {**

**System.out.println("I Came here");**

**int id = Integer.parseInt(req.getParameter("id"));**

**String name = req.getParameter("name");**

**float salary = Float.parseFloat(req.getParameter("salary"));**

**emp.setId(id);**

**emp.setName(name);**

**emp.setSalary(salary);**

**String res = employeeService.storeEmployeeInfo(emp);**

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

**ModelAndView mav = new ModelAndView();**

**mav.setViewName("index.jsp");**

**return mav;**

**}**

**}**

**EmployeeService.java**

**package com.service;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.stereotype.Service;**

**import com.bean.Employee;**

**import com.dao.EmployeeDao;**

**@Service**

**public class EmployeeService {**

**@Autowired**

**EmployeeDao employeeDao;**

**public String storeEmployeeInfo(Employee emp) {**

**if(emp.getSalary()>12000) {**

**if(employeeDao.storeEmployeeInfo(emp)>0) {**

**return "Record stored";**

**}else {**

**return "Record not store";**

**}**

**}else {**

**return "Salary must be > 12000";**

**}**

**}**

**}**

**EmployeeDao.java**

**package com.dao;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import javax.sql.DataSource;**

**import org.springframework.beans.factory.annotation.Autowired;**

**import org.springframework.stereotype.Repository;**

**import com.bean.Employee;**

**@Repository**

**public class EmployeeDao {**

**@Autowired**

**DataSource ds;**

**public int storeEmployeeInfo(Employee emp) {**

**try {**

**Connection con = ds.getConnection();**

**PreparedStatement pstmt = con.prepareStatement("insert into employee values(?,?,?)");**

**pstmt.setInt(1, emp.getId()); pstmt.setString(2, emp.getName());**

**pstmt.setFloat(3, emp.getSalary());**

**int res = pstmt.executeUpdate();**

**return res;**

**} catch (Exception e) {**

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

**}**

**return 0;**

**}**

**}**

**Web Service : Giving the Service for Web Application when both application running using same technologies or different technologies on same os or different os.**

**2 types**

**1. SOAP Web Service**

**Base upon SOA (Service Oriented Architecture).**

**Data must be only XML format.**

**JAX\_WS (Java API for XML Web Service)**

**2. RestFull Web Service**

**Architecture style. Exposing resource (Servlet/JSP) as a web Service in the form of XML or Non XML (ie JSON).**

**1. JAX\_RS**

**2. Spring MVC**

**29-08-2019 Day 3**

**Spring Boot with Rest Full Service**

**With GET,POST, PUT and DELETE methods**

**GET : Get Resource : Employee details all or specific employee**

**POST : Store Resources : Store Employee Details.**

**PUT :Update Resources : Update Employee Salary by ID**

**Delete : Delete Resources : Delete Employee Info By ID**

**@Controller**

**public class Demo {**

**@RequestMapping(value="say")**

**@ResponseBody**

**public String sayHello() {**

**return "Welcome";**

**}**

**}**

**@RestController = @Controller + @ResponseBody**

**{"id":102,"name":"Ajay","salary":34000}**

**GET and POST**

**View : browser plugin**

**Java Rest Client Using JAX\_RS and Spring MVC with boot, Asp.net, Php, Angular and React JS, Vue.js**

**Controller / RestController :**

**Service Layer : Business Logic**

**Dao Layer : JDBC**

**DAO Layer : ORM :**

**Object Relation Mapping :**

**Limitation of JDBC :**

**1. Java is object oriented programming language and backend is RDBMS(Oracle/mySQL).**

**But when storing record or retrieving the records from database we are converting java object to string and vice-versa.**

**2. JDBC use SQL language to connect the database. SQL is database dependent.**

**create database mydb;**

**use mydb;**

**show tables;**

**show databases;**

**3. JDBC through SQL checked exception and exception hierarchical classes and interfaces are database dependent.**

**4. JDBC doesn't support is a and has relationship.**

**class Employee {**

**}**

**class Manager extends Employee {**

**Address add;**

**}**

**Manager is a Employee**

**Manager has a Address**

**class Address {**

**}**

**has a relationship**

**Association**

**Aggregation**

**Composition**

**class A {**

**B obj1;**

**}**

**class B {**

**A obj2;**

**}**

**0-1**

**1-1**

**1-many**

**many-many**

**class Employee {**

**Address add;**

**}**

**class Address {**

**}**

**class Student {**

**StudentHistory sh;**

**}**

**class StudentHistory {**

**}**

**ORM : Object Relation Mapping**

**class Employee {**

**id,name,salary : properties**

**}**

**Employee emp = new Employee(); objects**

**mapping**

**employee ---> Tables (Relation)**

**Employee(className)< ------- > EMPLOYEE(table)**

**id (primarykey) ID**

**name NAME**

**salary SALARY**

**properties**

**Hibernate /JPA (Java Persistence API)**

**JPA is a technologies or type of EJB. JPA is a specification even though JPA provided the implementation**

**Where Hibernate is framework part of JBoss. It is implementation of JPA.**

**class Employee EMPLOYEE- table**

**mapping details**

**using XML**

**using Annotation**

**configuration details.**

**Hibernate JPA**

**hibernate.cfg.xml persistence.xml**

**This file contains Database details. DriverName, URL, username, password, Dialet class, mapping file formation or annotation details.**

**Table**

**EmployeeDetails**

**empId->pk**

**name**

**salary**

**@Entity**

**@Table(name="EmployeeDetails")**

**class Employee { Entity class**

**@Id**

**@Column(name="empId")**

**private int id;**

**private String name;**

**private float salary;**

**}**

**HQLQuery SQL Query**

**select \* from employee;**

**select e from Employee e**

**JPA : Java Persistence API :**

**JPA is a technologies or types of EJB. JPA a Specification for ORM as well as Implementation**

**EJB Vs Spring MVC**

**EJB + JPA =**

**Spring MVC + Hibernate =**

**Spring MVC + JPA**

**Spring Boot with Hibernate = Spring boot 1.5.x**

**Spring Boot with JPA = 1.x, 2.x**

**Spring boot with Spring Data**

**HQL JPQL**

**select e from Employee e select e from Employee e**

**from Employee e**

**from Employee**

**SQL JPQL**

**select \* from employee where salary > 12000**

**select e from Employee e where e.salary > 12000**

**select name from employee**

**select e.name from Employee e;**

**Day 4 - 30-08-2019**

**JPA Relationship**

**one to many relationship**

**Trainer**

**PK**

**TID TName Tech**

**1 Raj Java**

**2 Ram net**

**Student**

**PK FK**

**SId SName Age TSID**

**100 Seeta 24 1**

**101 Meeta 25 1**

**102 Veeta 26 2**

**103 Keeta 27 2**

**Trainer Table**

**create table trainer(tid int primary key,name varchar(10), tech varchar(10));**

**Student Table**

**create table student(sid int primary key,sname varchar(10),age int, tsid int, constraint ts\_fk foreign key(tsid) references trainer(tid));**

**Has a relationship**

**@Entity**

**class Trainer {**

**@Id**

**int tid;**

**String tname;**

**String tech;**

**@OneToMany(mappedBy="tsid") FK**

**List<Student> std;**

**}**

**@Entity**

**class Student {**

**@Id**

**int sid;**

**String sname;**

**int age;**

**int tsid;**

**}**

**Spring boot with JPA**

**Spring boot with JPA**

**parent starter**

**web starter, jdbc starter, jpa starter and my sql connector.**

**https://github.com/Kaleakash/secundarabadspringtraining.git**

**Spring boot data**

**Spring Reactive Programming**

**Spring boot Flux**

**Spring Microservices and Cloud**

**Spring security**

[**https://start.spring.io/**](https://start.spring.io/)

**Day 5 - 31-08-2019**

**Spring MVC with JPA**

**Spring boot with Rest and JPA**

**1. Local Transaction :**

**In DAO Layer do the @AutWired for the EntityManagerFactory interface.**

**@Autowired**

**EntityManagerFactory emf;**

**And each DAO method maintain the Transaction using EntityTransaction tran = manager.getTranction();**

**tran.begin()**

**DML Operation**

**tran.commit();**

**When we want to do Global Transaction Spring MVC or boot provided @Transactional annotation This annotation we have to use in Service layer. In DAO layer in place of doing @Autowired for EntityManagerFactory we have to do for the EntityManager interface reference.**

**@Autowired**

**EntityManager manger;**

[**https://tcheck.co/ntsNJb**](https://tcheck.co/ntsNJb)

**Reactive Programming :**

**Spring Flux : Spring Flux is base upon the Reactive concept which is use to create asynchronouse, non blocking RestFull Web Services or application.**