j2EE syllabus

1. Java Server Pages,
2. JDBC,
3. JavaBeans,
4. Java Security,
5. Naming Services,
6. Java Annotations,
7. Java Mail,
8.
```
Java Messaging Services,
```
9.
```
Transactions,
```
10. Apache maven,
11. Introduction to hibernate,
12.
```
HQL,
```
13.
```
 Hibernate,
```
14.
```
 Spring Framework,
```
15.
```
 Hands on Web services – JSON/XML/oData (data format conversation)
```

day1

demo

1. add external tomcat server to eclipse java

- for formatting

```
cltr + shift + F
```

2. Servelets in

> javax.servlet

```
javax.servlet
public interface Servlet
```

- Defines methods that all servlets must implement.
- A servlet is a small Java program that runs within a Web server. Servlets receive and respond to requests from Web clients, usually across HTTP, the HyperText Transfer Protocol.
- To implement this interface, you can write a generic servlet that extends javax.servlet.GenericServlet or
- an HTTP servlet that extends javax.servlet.http.HttpServlet

3. Generic Servelet

```
public abstract class GenericServlet
extends Object
implements Servlet, ServletConfig, Serializable
```

- Defines a generic, protocol-independent servlet. To write an HTTP servlet for use on the Web, extend HttpServlet instead.

4. HTTP Servlet

```
public abstract class HttpServlet
extends GenericServlet
```

- Provides an abstract class to be subclassed to create an HTTP servlet suitable for a Web site.
- A subclass of HttpServlet must override at least one method, usually one of these:

1. doGet, if the servlet supports HTTP GET requests
2. doPost, for HTTP POST requests
3. doPut, for HTTP PUT requests
4. doDelete, for HTTP DELETE requests

- init and destroy, to manage resources that are held for the life of the servlet

- getServletInfo, which the servlet uses to provide information about itself

- There's almost no reason to override the service method.

- service handles standard HTTP requests by dispatching them to the handler methods for each HTTP request type (the doXXX methods listed above).

- Likewise, there's almost no reason to override the doOptions and doTrace methods.

demo in eclipse

5. method for deployment of servlet in app

- declare survolate in a dynamic web app
- 1. using xml
- server deployment tags in web.xml file

```
 <!-- server deployment tags -->
  <servlet>
  <servlet-name>abc</servlet-name>
  <servlet-class>pages.HelloServlet2</servlet-class>
  <load-on-startup>2</load-on-startup>

  </servlet>
  <servlet-mapping>
  <servlet-name>abc</servlet-name>
  <url-pattern>/test3</url-pattern>
  <url-pattern>/hello</url-pattern>

  </servlet-mapping>
```

- 2. using annotation

```
    // URL ://host:port/lab_1.1/test
    // URI : context path (lab_1.1)
    // URL Pattern : /test



@WebServlet(value= {"/test","/test2"},loadOnStartup = 1)
//WC processes this annotation , at the deployment time and adds the
mapping between URL pattern and servlet
//WC creates an empty map (Hash Map)
//key : URL pattern (/test)
//value : Fully qualified servlet class name

// this is same in both methods
public class HelloServlet extends HttpServlet {
//todo step 3
}
```

- 3. rest of code , in both method
- need to override method (init,destroy, and service method)

```
// this is same in both methods
public class HelloServlet extends HttpServlet {

    private static final long serialVersionUID = 1L;

    @Override
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        System.out.println("in do-get " + Thread.currentThread());
        // set response content type for the cln broswwer
        resp.setContentType("text/html");
        // open print writer to send response from servlet --> clnt (text)
        try(PrintWriter pw = resp.getWriter())
        {
            pw.print("<h1> hello from HelloServlet @" + LocalDateTime.now()
+ "</h2>");
        }
    }
    @Override
    public void destroy() {

        System.out.println("in destroy  "  + Thread.currentThread());
    }
    @Override
    public void init() throws ServletException {

        System.out.println("in init of " + this.getClass().getName() + " "
+ Thread.currentThread());
    }
}
```

7. having two servlet with same url pattern,or without '/' ,gives exception

```
Caused by: java.lang.IllegalArgumentException: The servlets named
[pages.HelloServlet] and [pages.HelloServlet2] are both mapped to the url-
pattern [/test] which is not permitted
```

8. get info from request object using

- 1. getParameters - single value
- 2. getParametersValues - for array

```
protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
        response.setContentType("text/html");
        try(PrintWriter pw = response.getWriter())
        {
            pw.print("<h2> Email : " + request.getParameter("email"));
            pw.print("<h2> Password : " +
request.getParameter("password"));
        }
    }
```

Day2

to read

1. overriding form of the method cannot add any new or Broader checked exception

notes

demos

1.

-
    1.

```

```

-
    2.

```

```

-
    3. it redirects doesnt print pw given here
- as WC clears discard pw buffer , and send redirect response only

```
try(PrintWriter pw = response.getWriter())
        {
            pw.print("<h5> Successfully Login </h5>");
            // http proto stateless, so new request no data of pre request
            pw.print("Details" + request.getParameter("em"));

        }
    }
```

2.

-
    1.

```
```

-
    2.

```
```

-
    3.

```
```

3.

-
    1.

```
```

-
    2.

```
```

-
    3.

```
```

4.

-
    1.

```
```

-

2.

```
```

-
3.

```
```

Day3

to read

1. read Interface HttpSession

notes

demos

1.

-
1.

```
```

-
2.

```
```

-
3.

```
```

2.

-
1.

```
```

- 2.

```
```

- 3.

```
```

3.

- 1.

```
```

- 2.

```
```

- 3.

```
```

4.

- 1.

```
```

- 2.

```

```

-
    3.

```

```

Day4

to read

Lab sequence

Common instructions after you import web application in your workspace

1. Import project in your workspace
2. Fix build errors (How R click on project --build path --configure build path --Edit)
3. Copy your DBUtils under utils package.(remove mine) OR make the changes in web.xml

- In case of problems : follow clean up instructions , explained in theory session

1. Import day4.1 .

- Follow common instructions
- Demo of servlet config , init-params & writing DB independent web application(using xml config)
- Open : web.xml , DBUtils & LoginServlet

2. Import day4.3 . Follow common instructions

- Revise server pull (Request dispatching : forward scenario)
- OPen LoginServlet & CatalogServlet

3. Import day4.4 . Follow common instructions

- Revise server pull (Request dispatching : include scenario)
- OPen LoginServlet & CatalogServlet

4. Import day4.5 Follow common instructions

- Demo to Make web application DB independent (using ServletContext)
- (i.e if underlying DB changes ---no changes in java code BUT add DB specific details(JDBC drvr class, dbURL,userName , pwd) in xml based config files)
- Open : web.xml, DBConnectionManager(listener), DBUtils , DAOs.

5. Solve assignemnt

DAY5

notes

demos

1.

- 1.

```
```

- 2.

```
```

- 3.

```
```

Day 7

to read Common instructions

1. Create from scratch , hibernate based Java SE application. Test the same.

2. Steps for Hibernate + Java SE

- 1. Change perspective to Java
- 2. Create Java project(Java SE project) , in the same workspace (DON'T change the workspace)
- 3. Create new user lib containing hibernate JARs.+ JDBC JARs
- (window--preferences---user lib --hib_lib --add external jars
- from (hibernate-help/hibernate 5 jars/) -
- go to preference ---> select All (Ctrl A) ---apply n close

4. Add user lib to project's build path

- R click on project ---build path --configure build path --add user lib

5. Create folder --as a new src folder.

- Copy from (day7_data/day7_help/hibernate-help/config-files) -- hibernate.cfg.xml & EDIT it as per your DB settings.
- (r click on --new --src folder ---resources)

6. Copy folder under

- (Contains HibernateUtils)

7. Create a class TestHibernate under package.

- Add following code.

```
import static utils.HibernateUtils.*;
import org.hibernate.*;

public class TestHibernate {

    public static void main(String[] args) {
        try(SessionFactory sf=getSf())
        {
            System.out.println("Hibernate booted.....");
        }catch (Exception e) {
            e.printStackTrace();
        }

    }

}
```

8. Run this as java application ,

- check console to see , sf created & hib booted .
- Above confirms bootstrapping of hibernate framework.

9. Create a POJO n test auto table creation

- 9.1. Create a User POJO
    - Add these Data members
    - userId (PK) ,name,email,password,role(enum),confirmPassword, regAmount; LocalDate/Date regDate; byte[] image;
    - Add JPA annotations
    - Confirm auto table creation.
- 9.2 Add entry per POJO in hibernate.cfg.xml

10. Create Hibernate based DAO layer , to insert a record.

- 10.1 DAO layer i/f
- String registerUser(User user);
- 10.2 Hibenrate based DAO implementation class

- no data mebers, no constr,no clean up
- CRUD method

11. Create a main(...) based tester to test entire application , for user registration.

- Next Objective : time permitting

- 1. Import day7.2 project in your workspace
- 2. Fix build errors (How R click on project --build path --configure build path --Edit)
- 3. Edit web.xml

- In case of problems : follow clean up instructions , explained in theory session

- Revise : Open login.jsp & trace the flow

demos

0. Test URl rewriting,Centralized Error Handling in JSP,include directive

```
<body>

<h5> <a href="test1.jsp?pid=123&price=200&category=book">Test URl
rewriting: client pull I  </a> </h5>

<h5> <a href="test3.jsp?pid=123&price=200&category=book">Test URl
rewriting: client pull II  </a> </h5>


<h5> <a href="test4.jsp?pid=123&price=200&category=book">Test URl
rewriting: client pull I using JSTL   </a> </h5>

<h5> <a href="test5.jsp?pid=123&price=200&category=book">Test URl
rewriting: client pull II using JSTL   </a> </h5>

<h5> <a href="test6.jsp">Test Centralized Error Handling in JSP </a> </h5>

<h5> <a href="test7.jsp"> Testing include directive</a> </h5>
</body>
```

1. Test URl rewriting: client pull I

- using jsp action

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<body>

<%-- save product details under session scope w/o using scriplet --%>

<c:set var="product_dtls" value ="${param.pid} : ${param.price} :
```

```
${param.category}" scope="session" />
<%
// use url rewriting : method of http servlet response : encodeURL
    String encodedURL = response.encodeURL("test2.jsp");
%>
<h4> <a href="<%= encodedURL %>"> Next</a></h4>
</body>

--- test2
<body>
<h5 >Prodct details : ${sessionScope.product_dtls} </h5>
</body>
```

2. Test URl rewriting: client pull II

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<body>
<%-- save product details under session scope w/o using scriplet --%>
<c:set var="product_dtls" value ="${param.pid} : ${param.price} :
${param.category}" scope="session" />
<%
// use url rewriting : method of htpp servlet response : encodeURL
    String encodedURL = response.encodeRedirectURL("test2.jsp");
response.sendRedirect(encodedURL);
%>
</body>
```

3. Test URl rewriting: client pull I using JSTL

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<body>
<%-- save product details under session scope using JSTL --%>
<c:set var="product_dtls" value ="${param.pid} : ${param.price} :
${param.category}" scope="session" />
<%-- <c:url value="next page location (test2.jsp)" />  --%>
<h4> <a href="<c:url value='test2.jsp'/>"> Next</a></h4>
</body>
```

4. Test URl rewriting: client pull II using JSTL

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
<body>
<%-- save product details under session scope--%>
<c:set var="product_dtls" value ="${param.pid} : ${param.price} :
```

```
${param.category}" scope="session" />
<c:redirect url="test2.jsp" />
```

5. Test Centralized Error Handling in JSP

```
--- in web.xml file
  <!--  error page tags used for centralized error handling -->
  <error-page>
  <exception-type> java.lang.Exception</exception-type>
  <location>/err_Handler.jsp</location>
  </error-page>
  <!-- <error-page>
  <error-code>404</error-code>
  <location>/page_not_found.jsp</location>
  </error-page> -->
```

- err_Handling.jsp

```
%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8" isErrorPage="true"%>
<body>

<%-- get error data --%>
<h5> Error Causing page : ${pageContext.errorData.requestURI}  </h5>
<h4 >Error message: ${pageContext.exception.message} </h4>
<h4> Error Status code :${pageContext.errorData.statusCode}</h4>
<h4> Error details: ${pageContext.errorData.throwable} </h4>
<h4 >Via Expression Tag Errro Details : <%= exception %></h4>
</body>
```

-

6. Testing include directive

- both pages get merged, into the page in which include tag is used

```


---
 <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c"%>
</head>
<%! String message="hello..."; // what is it? : instance var available :
private %>
<body>
<%
int data = 1234; // what is it? : method local variable : within
_jspService of test7.jsp
%>
```

```
<%-- page scoped attribute available to current page only  --%>
<c:set var="server_date" value="<%= LocalDateTime.now() %>" />
<%-- use include directive  --%>
<%@ include file="test8.jsp" %>
</body>

--- test8.jsp

<body>

<h5> Instance var:<%= message %></h5>
<h5> Instance var:<%= data%></h5>
<h5> ${pageScope.server_date} </h5>
</body>
```

7. Hibernate app to for user table

1. hibernate.cfg.xml

```
<hibernate-configuration>

    <session-factory>
        <property name="hibernate.connection.autocommit">false</property>
        <property
name="hibernate.connection.driver_class">com.mysql.cj.jdbc.Driver</property
>
        <property name="hibernate.connection.password">password</property>
        <property
name="hibernate.connection.url">jdbc:mysql://localhost:3306/day2?
useSSL=false</property>
        <property name="hibernate.connection.username">dac</property>
        <property
name="hibernate.current_session_context_class">thread</property>
        <property name="hibernate.connection.pool_size">2</property>
        <!-- <property
name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property> -->
        <property name="hibernate.show_sql">true</property>
        <property name="hibernate.format_sql">true</property>
        <property name="hibernate.hbm2ddl.auto">update</property>
        <mapping class="pojos.User"/>

    </session-factory>
</hibernate-configuration>
```

2. hb utils

```
import org.hibernate.SessionFactory;
import org.hibernate.boot.MetadataSources;
import org.hibernate.boot.registry.StandardServiceRegistry;
import org.hibernate.boot.registry.StandardServiceRegistryBuilder;

public class HibernateUtils {
    private static SessionFactory sf;
    static {
        System.out.println("in static init block");
        try {
            // create std service reg instance from its builder
            StandardServiceRegistry reg =
                    new StandardServiceRegistryBuilder().
                    configure().build();
            // build session factory from Metadata
            sf = new MetadataSources(reg).
                    buildMetadata().buildSessionFactory();
            System.out.println("sf created....");
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
    public static SessionFactory getSf() {
        return sf;
    }
}
```

3. hb pojo with annotation for ORM

```
import javax.persistence.*; // import all JPA compliant annotations

@Entity // mandetory: to inform hibernate whatever follows is pojo/entity :
whose life cycle has to be
// managed by hibernate framework
@Table(name = "users") // optional anno
public class User {
    private Integer userId; // hibernate mandates to add unique ID property
:
    // Serializable (e.g Integer,Long,int,long,String ..)
    private String name,email,password,confirmPassword;
    private Role role;
    private double regAmount;
    private Date regDate;
    private byte[] image;

    // mandetory  :provide argumentless constructor

    public User() {
        System.out.println("in user const");
    }
    // optional : can add parameterized ctor

    // mandetory : all setters and getters

    @Id // mandetory : unique ID property : constraint : PK
    @GeneratedValue(strategy = GenerationType.IDENTITY) // to tell hb for
auto ID generation
    // constraint : auto increment : for oracle : sequence gen
    @Column(name = "user_id")
    public Integer getUserId() {
        return userId;
    }

    @Column(length = 20) // varchar size 20
    public String getName() {
        return name;
    }

    @Column(length = 20,unique = true) // unique constraint
    public String getEmail() {
        return email;
    }

    @Column(length = 20)
    public String getPassword() {
        return password;
    }

    @Transient // to tell HB to skip this from persistence (no column
created)
    public String getConfirmPassword() {
        return confirmPassword;
    }

    @Enumerated(EnumType.STRING) // to generate column as per enum name :
varchar
    @Column(length = 20)
    public Role getRole() {
        return role;
    }

    @Column(name = "reg_amount")
    public double getRegAmount() {
        return regAmount;
    }

    @Temporal(TemporalType.DATE)
    @Column(name = "reg_date")
    public Date getRegDate() {
        return regDate;
    }

    @Lob // large binary object(BLOB) : property type is byte[] : long BLOB
in db
```

```
    // if char[] : CLOB datatype created in DB
    public byte[] getImage() {
        return image;
    }
}
```

4. dao

```
public interface IUserDao {
//add a method for user registeration
    String registerUser(User user);
}

--- implementation

import static utils.HibernateUtils.getSf;
import org.hibernate.*;
// below is native hibernate based DAO layer (completely hibernate specific
)
public class UserDaoImpl implements IUserDao {
    @Override
    public String registerUser(User user) {
    String message = "User registration failed...";
    // 1. get hibernate session from session Factory: opensession /
getCurrentSession()
        Session session = getSf().openSession();

    // 2. begin a transaction : as recommended as any operation in hb
should be in transaction
    Transaction tx = session.beginTransaction(); // pools out DB
connection,wraps db conn in Session
   // L1 cache associated with session : created in empty manner
    try {
            // operation save : insert
            session.save(user);
            // success : commit transaction
            tx.commit(); // automatic dirty checking
            message = "user registeration  successfully with ID" +
user.getUserId();
        } catch (HibernateException e) {
            if(tx != null)
            tx.rollback();
            // inform /alert the caller about exception  :rethrow the same
exception to caller(main() tester)
             throw e;
        }finally {
        if(session != null)
        session.close(); // L1 cache destroyed and n pooled out DB conn
returnds to conn pool
        // so that same DB connection can be reused for another request
        }
        return message;
    }
}
```

5. test/ main method to call

```
import static utils.HibernateUtils.getSf;
import org.hibernate.*;
public class RegisterNewUser {
    public static void main(String[] args) {

    // Testing bootstrapping of hibernate configuration (creating singleton
n
    // immutable instance of SessionFactory(sf)

    // to parse string to --> Date
    SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

    try(org.hibernate.SessionFactory sf = getSf() ; Scanner sc = new
Scanner(System.in);)
    {
    System.out.println("Enter user details :
name,email,password,confirmPassword,role,regAmount,regdate");

    // create a transient POJO (not yet persistent )
    User u1 = new User(sc.next(), sc.next(), sc.next(), sc.next(),
Role.valueOf(sc.next().toUpperCase()), sc.nextDouble(),sdf.parse(sc.next())
);

    // create dao instance n invoke method
    UserDaoImpl dao = new UserDaoImpl();

    System.out.println("Reg status " + dao.registerUser(u1));
        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }
    }
}
```

day 8

to read

Steps

1. Details of HibernateUtils

- Need -- to supply singleton , immutable instance of SF
- How ? --static init block.
- 1. Create Service(JPA) registry instance,using its builder class.
    - How ?

> ServiceRegistry reg=new StandardServiceRegistryBuilder().configure().build();

- configure() --- hib will read hibernate.cfg.xml & process its instrs.

- 2. Build SF from Metadata. For MetaData , create MetaDataSources instance. SessionFactory sf=new MetadataSources(reg).buildMetaData().buildSessionFactory();

2. Create User POJO to represent a user in an application

- Data members
- userId (PK) ,name,email,password,role(enum),confirmPassword, regAmount; LocalDate/Date regDate;byte[] image;
- Rules on Hibernate managed POJO / Entity
- Is it mandatory for POJO class to imple Serializable ? NO
- POJO's unique ID property should be Serializable : eg : int , long ,Integer,Long, String, Date...

3. Prog has to supply mapping (ORM) instructions to Hibernate

- 2 ways

- 1. XML tags
    - Per POJO : supply pojo.hbm.xml : mapping instructions
- 2. annotations : preferred approach
    - JPA compliant : javax.persistence : prefer this
    - hibernate annotations : org.hibernate.annotations

- Annotate it.

- Package : javax.persistence

- @Entity : Mandatory : cls level

- @Id : Mandatory : field level or property (getter) : PK

- @Table(name="tbl_name) : to specify table name n more

- @GeneratedValue : to tell hib to auto generate ids

- auto / identity(auto incr : Mysql) / table / sequence(oracle)

- eg : @Id => PK

- @GenertedValue(strategy=GenarationType.IDENTITY) => auto increment

- @Column(name,unique,nullable,insertable,updatable,length,columnDefinition="double(8,2)") : for specifying col details

- @Transient : Skipped from persistence(no col will be generated in DB table)

- @Temporal : java.util.Date , Calendar , GregorianCalendar LocalDate(date) ,LocalTime(time) , LocalDateTime (timestamp/datetime) : no temporal anno.
- @Lob : BLOB(byte[]) n CLOB(char[]) : saving / restoring large bin /char data to/from DB
- @Enumerated (EnumType.STRING): enum (def : ordinal : int)

4. Add in hibernate.cfg.xml

5. Create DAO i/f & write its implementation class

- Hib based DAO impl class

1. No data members ,constructor , cleanup
2. Directly add CRUD methods. 3 Steps

- 1. Get hib session from SF
- API of org.hibernate.SessionFactory
- public Session openSession() throws HibernateException OR
- public Session getCurrentSession() throws HibernateException
- 2. Begin a Transaction
- API of Session
- public Transaction beginTransaction()throws HibernateException
- 3.

```
 try {
  perform CRUD using Session API (eg : save)
  commit the tx.
   } catch(HibernateException e)
   {
      roll back tx.
      re throw the exc to caller
   } finally {
      close session --destroys L1 cache , pooled out db cn rets to the
pool.
   }
```

- 4. Refer to Hibernate Session API
- (hibernate api-docs & readme : hibernate session api)
    - 1. Create main(..) based Tester & test the application.
- 5. Add a breakpoint before commit , observe n conclude.

7. Replace openSession by getCurrentSession

8. Objective : Get user details I/P : user id API : session.get

9. Confirm L1 cache by invoking session.get(...) multiple times.

10. Hibernate POJO states : transient , persistent , detached.

11. Objective : Display all user details

- 1. Solve it using HQL(Hibernate query language)/JPQL (Java Persistence Query Language) Object oriented query language, where table names are replaced by POJO class names & column names are replaced by POJO property names.

- eg :

- sql -- select * from users

- hql -- from User

- jpql -- select u from User u

- u -- alias (POJO ref)

- 2. Create Query Object --- from Session i/f

> org.hibernate.query.Query createQuery(String jpql,Class resultType)

- T --result type.
- 3. To execute query
- Query i/f method

> public List getResultList() throws HibernateException

- --Rets list of PERSISTENT entities.

- eg :

- List users=session.createQuery("select u1 from User u1",User.class).getResultList();

12. Objective :

- Display all users registered between strt date n end date & under a specific role
- eg : sql = select * from users where reg_dt between ? and ? and user_role=?

1. jpql ="select u from User u where u.regDate between :begin and :end and u.role=:rl"

- Passing named IN params to the query
- Query i/f method

> public Query setParameter(String paramName,Object value) throws HibernateException.

- eg : List users=session.createQuery(jpql,User.class).setParameter("begin",beginDate).setParameter("end",endDate).setParameter("rl",role).getResultList();

15. Objective :

- 1. Display all user names registered between strt date n end date & under a specific role
- 2. Display all user names,reg amount,reg date registered between strt date n end date & under a specific role

1. Objective : User login

- API : getSingleResult (to be done in lab)

- 3. Update Objective :

1. Change password

- i/p --user id , new pass
- eg : get

2. Apply discount to reg amount , for all users , reged before a specific date.

- i/p -- discount amt, reg date

- String jpql="update User u set u.regAmount=u.regAmount-:disc where u.regDate < :dt";

- 1. Query API

> public int executeUpdate() throws HibernateException

- -use case --DML
- 2. Session API

> public Query createQuery(String jpql) throws HibernateException

- jpql -- DML

18. Un subscribe user

- i/p user id
- o/p user details removed from DB

19. H.W

- Objective --delete vendor details for those vendors reg date > dt.
- via Bulk delete
- String jpql="delete from Vendor v where v.regDate > :dt";

20. Save n restore images to / from DB

demo on JPA

1. structure of transaction in java bean / main() method

```
main()
{
        // getting session from session factory
        Session session = getSf().getCurrentSession();

        // begin trans
        Transaction tx = session.beginTransaction();

        try {


            tx.commit();
        } catch (Exception e) {
            // TODO: handle exception

            if(tx != null)
                tx.rollback();
        }

        return null;
    }
```

- main method call

```


    public static void main(String[] args) {

        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory(sf)

        try(org.hibernate.SessionFactory sf = getSf()) {

            System.out.println("hibernate up and running ");

        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    }
```

1. CRUD operation :save

```
@Override
    public String registerUser(User user) {

        String msg = "User reg failed";

        // instance of  user  :TRANSIENT (not in L1 cache and not in DB)
:EXISTS  only in java heap
        //1 .get session from sf  :openSession
        Session session = getSf().getCurrentSession();   // new session ,
empty cache
        Session session2 = getSf().getCurrentSession();

        System.out.println(session == session2); // true
        //2. start/begin  transaction  , managed by programmer

      Transaction tx = session.beginTransaction();
      // db conn is pooled out and wrapped in session n returned to the
caller
      // Empty L1 cache is created, empty
 System.out.println("after begin tx : session open "+session.isOpen()+"
conn "+session.isConnected());//true true

      // 3.try catch block for trans

      try {
          // insert new users info

        Integer id =  (Integer) session.save(user); // user : PERSISTENT
(only added in L1 cache  : not yet part of DB
          System.out.println("generated id " + id);

          tx.commit(); //at the time of commit, Hibernate performs :
automatic dirty checking(check state of L1, DB)
          //after check as user ref not in db: insert query fired: to synch
state of L1 cache with DB
          // session is implicitely closed here i.e db conn returns to pool
and L1 cache is destroyed
        msg = "User registered with ID" + id;
    System.out.println("after commit tx : session open "+session.isOpen()+"
conn "+session.isConnected());// false false
    } catch (HibernateException e) {
        // TODO: handle exception
        // roolback trx n re throw the exc to the caller


        if(tx != null)
        {
            tx.rollback();
            // session is implicitely closed here i.e db conn returns to
pool and L1 cache is destroyed
        }
        throw e;
    }
      System.out.println(" before returning from dao : session open
"+session.isOpen()+" conn "+session.isConnected());//false false


        return msg; // user : DETACHED : here L1 cache user is destroyed ,
but it exists in DB
    }
```

- main

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

        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory(sf)

        // to parse string to --> Date
        SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

        try(org.hibernate.SessionFactory sf = getSf() ; Scanner sc = new
Scanner(System.in);)
        {
            System.out.println("Enter user details :
name,email,password,confirmPassword,role,regAmount,regdate");

            // create a transient POJO (not yet persistent )
            User u1 = new User(sc.next(), sc.next(), sc.next(), sc.next(),
Role.valueOf(sc.next().toUpperCase()), sc.nextDouble(),sdf.parse(sc.next())
);
            // u1 : exists in java Heap : TRANSIENT
            // create dao instance n invoke method

            UserDaoImpl dao = new UserDaoImpl();

            System.out.println("Reg status " + dao.registerUser(u1));

        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    }
```

3. get

- find by primary key so use this, or jpql

```
@Override
    public User fetchUserDetails(int userId) {
        User u = null; // u  :Not applicable
        // get session

        Session session = getSf().getCurrentSession();

         Transaction tx = session.beginTransaction();

         try {
             // Session API : Tget(Class<T> class,Serializable id)
             u = session.get(User.class,userId); // int datatype of userId
: --> Integer(auto boxing) ---> Serializable (up casting)
             // u: in case of valid id, u : PERSISTENT

             u = session.get(User.class,userId);

             u = session.get(User.class,userId);

             tx.commit();
            // perform auto dirty checking, and L! and DB same : no queries
fired , db conn returns the pool iand L1 cache is destroyed
        } catch (HibernateException e) {
            // TODO: handle exception

            if(tx != null)
            tx.rollback();// db conn returns to the pool and L1 cache is
destroyed
            throw e;
        }

        return u; // u  :DETACHED
    }

```

- main

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

        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory(sf)

        try(org.hibernate.SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {

            // dao instance

            UserDaoImpl dao = new UserDaoImpl();


            System.out.println("Enter User ID");

            System.out.println(dao.fetchUserDetails(sc.nextInt()));



        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }
```

```
    }
```

4. using jpql(java persistence query language) ,

- session createQuery() method
- setParameter() method in jpql statment

```
@Override
    public List<User> fetchAllUserDetails() {

        // getting session from session factory

        String jpql = "select u from User u ";
        List<User> users = null; // ussers  :null
        Session session = getSf().getCurrentSession();

        // begin trans
        Transaction tx = session.beginTransaction();

        try {
            // create query from session and execute the same
            users = session.createQuery(jpql, User.class).getResultList();
            // users : list of Persistent pojo
            /*
             * users = session.createQuery(jpql,
User.class).getResultList(); users =
             * session.createQuery(jpql, User.class).getResultList();
             */
            // here 3 times select is called , not take it from L1 cache
            tx.commit(); //

        } catch (Exception e) {
            // TODO: handle exception

            if(tx != null)
                tx.rollback(); // l1 cache destroed and db conn returned to
pool
        }

        return users;// users  : list of Detached pojos  : i.e detached
from l1 cache that is destroyed on commit
    }
```

- main

```
    public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory(sf)

        try(org.hibernate.SessionFactory sf = getSf()) {


            UserDaoImpl dao = new UserDaoImpl();
            System.out.println("Existing user details ");
    dao.fetchAllUserDetails().forEach(System.out::println);


        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    }

```

5. demo on jpql

- dao implementaiton

```
// Display all users registered between strt date n end date & under a
specific role
        @Override
        public List<User> fetchSelectedUserDetails(Date strtDate, Date
endDate, Role userRole) {
            List<User> users = null;
            String jpql = "select u from User u where u.regDate between
:start and :end and u.role=:rl";
            // get session from SF
            Session session = getSf().getCurrentSession();
            // begin tx
            Transaction tx = session.beginTransaction();
            try {
                users = session.createQuery(jpql,
User.class).setParameter("start", strtDate).
                        setParameter("end", endDate)
                        .setParameter("rl", userRole).getResultList();
                // users : list of persistent pojos/entities
                tx.commit();
            } catch (HibernateException e) {
                if (tx != null)
                    tx.rollback();// db cn rets to the pool n L1 cache is
destroyed.
                throw e;
            }
            return users;
        }
```

- main method call

```
public static void main(String[] args) {
        // for parsing string ---Date : use SimpleDateFormat
        SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory (SF)
        try (SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {
            System.out.println("Enter begin date end date n role");
            // dao instance
            UserDaoImpl dao = new UserDaoImpl();
            System.out.println("Selected  Users : ");
            dao.fetchSelectedUserDetails(sdf.parse(sc.next()),
sdf.parse(sc.next()),

Role.valueOf(sc.next().toUpperCase())).forEach(System.out::println);
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

6.

```
    @Override
    public List<String> fetchSelectedUserName(Date strtDate, Date endDate,
Role userRole) {

        // getting session from session factory
        List<String> users = null;
        /*
         * this.name = name; this.email = email; this.password = password;
         * this.confirmPassword = confirmPassword; this.role = role;
this.regAmount =
         * regAmount; this.regDate = regDate;
         */

        String jpql = "select u.name from User u where u.regDate between
:start and :end and u.role=:rl";
                Session session = getSf().getCurrentSession();

                // begin trans
                Transaction tx = session.beginTransaction();
                try {
                    users = session.createQuery(jpql,
String.class).setParameter("start", strtDate).setParameter("end",
endDate).setParameter("rl", userRole).getResultList();

                    // user : list of persistent pojos /entites


                    tx.commit();
                } catch (Exception e) {
                    // TODO: handle exception

                    if(tx != null)
                        tx.rollback();
                }

                return users;
    }
```

- same main as 5

7. using constructor

- dao

```
@Override
    public List<User> fetchSelectedDetails(Date strtDate, Date endDate,
Role userRole) {
        List<User> users = null;
        String jpql = "select new pojos.User(name,regAmount,regDate) from
User u where u.regDate between :start and :end and u.role=:rl";
        // get session from SF
        Session session = getSf().getCurrentSession();
        // begin tx
        Transaction tx = session.beginTransaction();
        try {
            users = session.createQuery(jpql,
User.class).setParameter("start", strtDate).
                    setParameter("end", endDate)
                    .setParameter("rl", userRole).getResultList();
            // users : list of persistent pojos/entities
            tx.commit();
        } catch (HibernateException e) {
            if (tx != null)
                tx.rollback();// db cn rets to the pool n L1 cache is
destroyed.
            throw e;
        }
        return users;//users : list of detached pojos/entities
    }
```

- main

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

            // for parsing string ---Date : use SimpleDateFormat
            SimpleDateFormat sdf = new SimpleDateFormat("dd-MM-yyyy");
            // Testing bootstrapping of hibernate configuration (creating
singleton n
            // immutable instance of SessionFactory (SF)
            try (SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {
                System.out.println("Enter begin date end date n role");
                // dao instance
                UserDaoImpl dao = new UserDaoImpl();
                System.out.println("Selected  User Details : ");
                dao.fetchSelectedDetails(sdf.parse(sc.next()),
sdf.parse(sc.next()),
                        Role.valueOf(sc.next().toUpperCase())).forEach(u ->
                        System.out.println(u.getName()+"
"+u.getRegAmount()+" reged on "+u.getRegDate()));
            } catch (Exception e) {
                e.printStackTrace();
            }

        }
```

8. using PERSISTENCE object to update DB, at commit

- dao

```
   @Override
    public String changePassword(String email, String oldPwd, String
newPwd) {
    String msg = "password change failed";
    String jpql = "select u from User u where u.email=:em and
u.password=:pass";
    // session
    Session session = getSf().getCurrentSession();
    // begin  trans
    User u =  null;
    Transaction tx = session.beginTransaction();
    try {
        // create session query  ,set IN params ,get single result
         u = session.createQuery(jpql, User.class)
                    .setParameter("em", email)
                    .setParameter("pass", oldPwd)
                    .getSingleResult();

            // in case of valid credentials method returns : PERSISTENT
POJO reference
        // no null checking is required : since methods throws exception in
case no reuslt found
        // u : PERSISTENT : exist in L1 cache , exist in DB
        u.setPassword(newPwd); // abcd : modifying state of Persistent POJo

            //session.evict(u);
            // clears u from L1 cache , u : DETACHED

        //  session.clear();
        // clear entire L1 cache (i.e all persistent entities are unbounded
from L1 cache
         System.out.println("L1 cache contains" + session.contains(u));




        tx.commit(); // hb  perform auto dirty checking : detects change :
update,session close
        // db conn returnds to the pool, L1 cache is destroyed
        msg = "password changed successfully";
        } catch (Exception e) {
            if(tx != null)
            tx.rollback();
        }
        // u: DETACHED : hibernate does not propogate the changes done to
state of detached pojo
        u.setPassword(newPwd.toUpperCase()); // ABCD
        return msg;
    }
```

- main

```
  public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory(sf)
    try(org.hibernate.SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {
        // dao instance
            UserDaoImpl dao = new UserDaoImpl();
            System.out.println("Enter User email , old password , new
password");
            System.out.println(dao.changePassword(sc.next(), sc.next(),
sc.next()));
        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    }
```

done on day 9

9. Unsubscibe user i/p : email n password

- dao

```
  @Override
    public String unsubscribeUser(String email, String password) {
        String mesg="User un subscription failed...";
        // add jpql : to authenticate user
        String jpql = "select u from User u where u.email=:em and
u.password=:pass";
        // session
        Session session = getSf().getCurrentSession();
        // tx
        Transaction tx = session.beginTransaction();
        try {
            // validate user
            User u = session.createQuery(jpql,
User.class).setParameter("em", email).
                    setParameter("pass", password)
                    .getSingleResult();
            //u : PERSISTENT
            //Session API : public void delete(Object o)
            session.delete(u);//u : is marked for removal, neither  gone
from L1 cache nor DB : REMOVED
            tx.commit();//dirty chking : delete query , session is closed :
db cn rets to the pool , L1 cache is destroyed
            //entity is removed from cache
            mesg="User "+u.getName()+" un subscribed...";
        } catch (RuntimeException e) {
            if (tx != null)
                tx.rollback();// session closed
            // --db cn rets to the pool , L1 cache is destroyed
            throw e;
        }
        // u : TRANSIENT : exists only in java heap
        return mesg;
    }// user object is marked for garbage collection
```

- main

```
    public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable singleton instance of SessionFactory (SF)
        try(SessionFactory sf=getSf();Scanner sc=new Scanner(System.in))
        {
            //dao instance
            UserDaoImpl dao=new UserDaoImpl();
            System.out.println("Enter User email n pwd");
            System.out.println(dao.unsubscribeUser(sc.next(), sc.next()));
            System.out.println("cntd....");
        }catch (Exception e) {
            e.printStackTrace();
        }

    }
```

10. save vs persist saveOrUpdate merge

- dao

```

    @Override
     public String testSessionApi(User user) {
String msg = "User reg failed";

        // instance of  user  :TRANSIENT (not in L1 cache and not in DB)
:EXISTS  only in java heap


        //1 .get session from sf  :openSession
        Session session = getSf().getCurrentSession();   // new session ,
empty cache

      Transaction tx = session.beginTransaction();

      // 3.try catch block for trans

      try {
          // insert new users info

        //session.persist(user);; // user : PERSISTENT (only added in L1
cache  : not yet part of DB
          session.saveOrUpdate(user);

          System.out.println("generated id " + user.getUserId());

          tx.commit();

        msg = "User registered with ID" + user.getUserId();

    } catch (RuntimeException e) {
        // TODO: handle exception
        // roolback trx n re throw the exc to the caller


        if(tx != null)
        {
            tx.rollback();
            // session is implicitely closed here i.e db conn returns to
pool and L1 cache is destroyed
        }
        throw e;
    }



        return msg; // user : DETACHED : here L1 cache user is destroyed ,
but it exists in DB
    }

```

- main

```

public static void main(String[] args) {
SimpleDateFormat sdf = new SimpleDateFormat("yyyy-MM-dd");

try(org.hibernate.SessionFactory sf = getSf() ; Scanner sc = new
Scanner(System.in);)
{
System.out.println("Enter user details :
name,email,password,confirmPassword,role,regAmount,regdate");

// create a transient POJO (not yet persistent )
User u1 = new User(sc.next(), sc.next(), sc.next(), sc.next(),
Role.valueOf(sc.next().toUpperCase()), sc.nextDouble(),sdf.parse(sc.next())
);
// u1 : exists in java Heap : TRANSIENT
// create dao instance n invoke method

    // u1 id : null
    u1.setUserId(222);

//u1.setUserId(1234); // not existing in DB
System.out.println("user id " + u1.getUserId()); // null
    UserDaoImpl dao = new UserDaoImpl();

System.out.println("session api status " + dao.testSessionApi(u1));

} catch (Exception e) {
// TODO: handle exception
    e.printStackTrace();
        }

    }

```

11. BulkUpdate (refer to hibernate session api readme)

- as here 1 select query + 10 update query are fired
- so use executeUpdate() method for bulk update for single (unrelated) table (no joins).
- it returns update count
- not recommended ,
  - as bypass L1 cache
  - cascade not supported
  - does not support optimistic locking
- dao

```
@Override
    public String bulkUpdateUsers(Date date, double discount) {
        String mesg="bulk updation failed...";
        //1 : update jpql
        String jpql="update User u set u.regAmount=u.regAmount-:disc where 
u.regDate < :dt";
        //session
        Session session=getSf().getCurrentSession();
        //tx
        Transaction tx=session.beginTransaction();
        try {
            int updateCount=session.createQuery(jpql).
                    setParameter("disc", discount).setParameter("dt", 
date).executeUpdate();
            tx.commit();//update query , empty L1 cache is destroyed , cn 
rets to the pool.
            mesg=updateCount+" users updated...";
        }catch (RuntimeException e) {
            if (tx != null)
                tx.rollback();// session closed
            // --db cn rets to the pool , L1 cache is destroyed
            throw e;
        }
        return mesg;
    }
```

- main

```
    public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating 
singleton n
        // immutable singleton instance of SessionFactory (SF)
        SimpleDateFormat sdf=new SimpleDateFormat("yyyy-MM-dd");
        try(SessionFactory sf=getSf();Scanner sc=new Scanner(System.in))
        {
            //dao instance
            UserDaoImpl dao=new UserDaoImpl();
            System.out.println("Enter reg date n discount");
            System.out.println(dao.bulkUpdateUsers(sdf.parse(sc.next()), 
sc.nextDouble()));
```

```
                System.out.println("cntd....");
            }catch (Exception e) {
                e.printStackTrace();
            }

        }
```

12. saving image to DB

- dao

```


import org.apache.commons.io.FileUtils;
import org.hibernate.*;
    @Override
    public String saveImage(int userId, String fileName) throws Exception {
        String msg="Saving image failed....";
        //validate file : check if its readable existing data file
        // create instance of java.io.File

        File file = new File(fileName);

        if(file.exists() && file.isFile() && file.canRead())
        {

            //session
            Session session=getSf().getCurrentSession();
            //tx
            Transaction tx=session.beginTransaction();
            try {
                User user = session.get(User.class,userId);

                if(user != null)
                {
                    // user : Pers
                    // method : read binary file and return s content it
byte[] n closes file
                    user.setImage(FileUtils.readFileToByteArray(file));//
modifying state of persistent pojo

                    msg = "image saved to db ...";
                }

                tx.commit(); // hibernate perform  auto dirty check :
update query  :close session --> conn returnds to pool
            }catch (RuntimeException e) {
                if (tx != null)
                    tx.rollback();// session closed
                // --db cn rets to the pool , L1 cache is destroyed
                throw e;
            }
        }


        return msg;
    }
```

- main

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


        try(org.hibernate.SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {

            // dao instance

            UserDaoImpl dao = new UserDaoImpl();


            System.out.println("Enter User id and File Name along with
path");

            System.out.println(dao.saveImage(sc.nextInt(), sc.next()));

            System.out.println("cntd");


        } catch (Exception e) {
            // TODO: handle exception
            e.printStackTrace();
        }

    }
```

13. retrieve image from DB

- dao

```
@Override
    public String restoreImage(int userId, String fileName) throws
Exception {
        String mesg = "Restoring image failed....";
        //Session
        Session session=getSf().getCurrentSession();
        //tx
        Transaction tx=session.beginTransaction();
        try {
            //get user details from id
            User user=session.get(User.class, userId);
            if(user != null) {
                //user : PESRSISTENT
                //gets image from user POJO in byte[] form n creates a new
file if none exists n write bin data to it.
                FileUtils.writeByteArrayToFile(new File(fileName),
user.getImage());
                mesg="Restored image succefully....";
            }
            tx.commit();

        } catch (Exception e) {
            if(tx != null)
                tx.rollback();
            throw e;
        }
        return mesg;
    }
```

- main

```


    public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable singleton instance of SessionFactory (SF)
        try(SessionFactory sf=getSf();Scanner sc=new Scanner(System.in))
        {
            //dao instance
            UserDaoImpl dao=new UserDaoImpl();
            System.out.println("Enter User id n image file name along with
path , to restore image from DB");
            System.out.println(dao.restoreImage(sc.nextInt(), sc.next()));
            System.out.println("cntd....");
        }catch (Exception e) {
            e.printStackTrace();
        }

    }
```

day 9

demo

1. demo on joins

- pojo for course

```
@Entity
@Table(name = "courses_tbl")
public class Course {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "cid")
    private Integer courseId;
    @Column(length = 20,unique = true)
    private String name;
    private int capacity;
    @Column(name="start_date")
    private LocalDate startDate;
    @Column(name="end_date")
    private LocalDate endDate;
    private double fees;

    // one to many : , bi directional association  between two entities :
onse sid eof asso :
    // parent : and non -owning (inverse ) side of association
    @OneToMany(mappedBy ="selectedCourse", cascade =
CascadeType.ALL,orphanRemoval = true)
    private List<Student> students=new ArrayList<>();

    //def constr
    public Course() {
        System.out.println("in course cnstr");
    }

    //add all s/g


    // add helper method : for two reasons :
    //1. to support adding (student details )
    // 2. to remove (student details)
    // Optional : Recommended

    // add student detial to a course
    public void addStudent(Student s)
    {
        students.add(s); // adding parent ---> child

        s.setSelectedCourse(this); // child ---> parent

    }

    // remove student details
    public void removeStudent(Student s)
    {
        students.remove(s); // removing  parent ---> child

        s.setSelectedCourse(null);// removing child ---> parent

    }
    @Override
    public String toString() {
        return "Course [courseId=" + courseId + ", name=" + name + ",
capacity=" + capacity + ", startDate=" + startDate
                + ", endDate=" + endDate + ", fees=" + fees + "]";
    }






}
```

- pojo for Student

```

@Entity
@Table(name="students_tbl")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "student_id")
    private Integer studentId;
    @Column(length = 20,unique = true)
    private String email;
    @Column(length = 20)
    private String name;

    // bi - directional association between entities
    // many side of association  and owning side(since it has FK column)
    @ManyToOne
    @JoinColumn(name = "c_id",nullable = false) // constraint : Not null :
optional but recommended
    private Course selectedCourse;
    public Student() {
        System.out.println("in student cnstr");
    }
    public Student(String email, String name) {
        super();
        this.email = email;
        this.name = name;
    }
    //all s/g

    @Override
    public String toString() {
        return "Student [studentId=" + studentId + ", email=" + email + ",
name=" + name + "]";
    }




}

```

2. for course related

- 1. for launchCourse
- dao

```

@Override
    public String launchCourse(Course c) {


        // session

           Session session = getSf().getCurrentSession();

           String msg = "Launching course failed";
        // begin trans

             Transaction tx = session.beginTransaction();

             try {
                 // c : Transient
                 session.persist(c); // Persistent


                 tx.commit(); // dirty checking : check : insert query
fired , session closed

                 msg = " Launched course with course id " +
c.getCourseId();

            } catch (RuntimeException e) {

                if(tx != null)
                tx.rollback();
             throw e;
            }
        return msg;
    }
```

- tester

```
public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory (SF)
        try (SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {
            CourseDaoImpl courseDao = new CourseDaoImpl();
            System.out
                    .println("Enter course details :
name,capacity,strt_date,end_date(yr-mon-day with 0 prefix),fees");
            // create transient pojo n pass it to dao layer for auto
persistence
            Course c1 = new Course(sc.next(), sc.nextInt(),
parse(sc.next()),
                    parse(sc.next()), sc.nextDouble());

            // accept 3 students details , who want to enroll in this
course


            for (int i = 0; i < 3; i++) {

                System.out.println("Enter student detials : email and name
");

                Student s = new Student(sc.next(), sc.next());

                // add student  reference in arraylist
                /*
                 * c1.getStudents().add(s); // course ---> student
                 *
                 * s.setSelectedCourse(c1); // student ---> course
                 */

                c1.addStudent(s); // invoking helper /convinience method



            }

            System.out.println("status " + courseDao.launchCourse(c1));
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

- 2. for
- dao

```
@Override
    public String cancelCourse(int courseId) {

        String msg = "Course cancellation failed";
           Session session = getSf().getCurrentSession();

        // begin trans

         Transaction tx = session.beginTransaction();

         try {
             // get course detail from course id

        Course c = session.get(Course.class, courseId);

        if(c != null)
        {
            // delete course details
            session.delete(c); // REMOVED (not yet gone from L1 cache or DB
) : simply marked for removal

        }


             tx.commit(); // delete query

            msg = "Course with name " + c.getName() + "cancelled ... ";

            } catch (RuntimeException e) {

                if(tx != null)
                tx.rollback();
             throw e;
            }




        return msg;
    }
```

- tester

```
public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory (SF)
        try (SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {

            StudentDaoImpl dao = new StudentDaoImpl();
            System.out
                    .println("Enter students email and course name , to
cancel admission ");
            // create transient pojo n pass it to dao layer for auto
persistence

            System.out.println("status " +
dao.cancelStudentAdmission(sc.next(), sc.next()));
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

3. for student related

- dao

```
    @Override
    public String cancelStudentAdmission(String studentEmail, String
courseName) {
        String msg = "Cancelling admission failed";

        String jpqlStudent = "select s from Student s where
s.email=:email";

        String jpqlCourse = "select c from Course c where c.name=:nm";

        Session session = getSf().getCurrentSession();

        // begin trans

        Transaction tx = session.beginTransaction();

        try {

            // get student details from its email

            Student s = session.createQuery(jpqlStudent,
Student.class).setParameter("email", studentEmail)
                    .getSingleResult();

            // s: persistent

            // get course details from its name

            Course c = session.createQuery(jpqlCourse,
Course.class).setParameter("nm", courseName).getSingleResult();

            // c : PERSISTENT
            c.removeStudent(s);// helper method to delink bi dir
association between course and student

            tx.commit();
            msg = s.getName() + "'s admission cancelled ... ";

        } catch (RuntimeException e) {

            if (tx != null)
                tx.rollback();
            throw e;
        }

        return msg;
    }
```

- tester

```
public class CancelStudentAdmission {

    public static void main(String[] args) {
        // Testing bootstrapping of hibernate configuration (creating
singleton n
        // immutable instance of SessionFactory (SF)
        try (SessionFactory sf = getSf(); Scanner sc = new
Scanner(System.in)) {

            StudentDaoImpl dao = new StudentDaoImpl();
            System.out
                    .println("Enter students email and course name , to
cancel admission ");
            // create transient pojo n pass it to dao layer for auto
persistence

            System.out.println("status " +
dao.cancelStudentAdmission(sc.next(), sc.next()));
        } catch (Exception e) {
            e.printStackTrace();
        }

    }
```

day 10

sequence

demo

1. demo on solution to LazyInitiatization Exception

- 1. using fetch annotation attribute
- it is not recommended

```
// one to many : , bi directional association  between two entities : onse sid eof asso :
    // parent : and non -owning (inverse ) side of association
    @OneToMany(mappedBy ="selectedCourse", cascade = CascadeType.ALL,orphanRemoval = true, fetch = FetchType.EAGER)
    private List<Student> students=new ArrayList<>();
```

- 2. call size method on Collection , to hint hibernate to call select on many side table

```
       // c : persistent
    // Hint : access the size of collection witn in session scope
    c.getStudents().size();
```

- 3.
    - Using "join fetch" keyword in JPQL

  > String jpql = "select c from Course c left outer join fetch c.students where c.title=:ti";

    - most recommended this approach

   ```
    String jpql = "select c from Course c join fetch c.students where c.name=:nm";
   ```

2. Joining Tables

- 0. mapping in hibernate.cfg.xml file

```
<hibernate-configuration>

    <session-factory>
        <mapping class="pojos.Course" />
        <mapping class="pojos.Student" />
        <mapping class="pojos.Address" />
        </session-factory>
</hibernate-configuration>
```

- 1. Address pojo
    - creating FK to connect to Student pojo
    - on using annotation on parent pojo object

```
@Entity
@Table(name="address_tbl")
public class Address {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "address_id")
    private Integer addressId;

    @Column(length = 20)
    private String city;
    @Column(length = 20)
    private String state ;
    @Column(length = 20)
    private String country;
    @Column(length = 20,unique = true)
    private String phoneNo;

    // bi directional association between entities  :owning side

    @OneToOne
    @JoinColumn(name = "stud_id", nullable = false)
    private Student stud ;

    public Address() {
        // TODO Auto-generated constructor stub
        System.out.println("in address constr");
    }
```

- 2. Student pojo
  - mapping PK of Student i.e FK of Address pojo using mappedBy
  - its automatically created, on using annotation on child table pojo object

```
@Entity
@Table(name="students_tbl")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "student_id")
    private Integer studentId;
    @Column(length = 20,unique = true)
    private String email;
    @Column(length = 20)
    private String name;

    // bi - directional association between entities
    // many side of association  and owning side(since it has FK column)
    @ManyToOne(fetch = FetchType.LAZY) /* (fetch = FetchType.LAZY) */ //
fetch policy:  eager by default
```

```
    @JoinColumn(name = "c_id",nullable = false) // constraint : Not null : 
optional but recommended
    private Course selectedCourse;

    @OneToOne(mappedBy = "stud", cascade = CascadeType.ALL ) // Eager
    private Address studentAdr;

    // one to one association between entity and value type

    @Embedded //OPTIONAL added only for understanding it is embedded
    private AdharCard card;

    public Student() {
        System.out.println("in student cnstr");
    }
    public Student(String email, String name) {
        super();
        this.email = email;
        this.name = name;
    }
    // one to many asso between entity and collection of value types : it 
is Uni directional

    @ElementCollection // mandetory  : if not : mapping exception
    @CollectionTable(name = "hobbies_tbl",joinColumns = @JoinColumn(name = 
"s_id") ) // optional but recommended
    @Column(name = "hobby",length = 20)
    private List<String> hobbies = new ArrayList<>();
    // one to many asocation between entity n value type : uni 
directional(entity--> value type)

    @ElementCollection
    @CollectionTable(name = "edu_qualifications",joinColumns = 
@JoinColumn(name = "s_id"))
    private List<EducationalQualifications> qualifications= new ArrayList<>
();
```

3. Embedding
4. value type table into Entity type table

- in case of one to one relation
- value type are added as column to parent table
- 1. Aadhar table : value type

```
@Embeddable // mandetory  : required to tel HB : that whatever follows is 
value type ,
// whic h does not have a standAlone existence and its detials must be 
embedded in owning entity
public class AdharCard {
```

```
@Column(name = "card_number",length = 20,unique = true)
    private String cardNumber ;
@Column(length = 50)
    private String location;
@Column(name = "created_on")
    private LocalDate createdOn;

    public AdharCard() {
        // TODO Auto-generated constructor stub
        System.out.println("in aadhar contr");
    }

```

- 2. in parent table

```
@Entity
@Table(name="students_tbl")
public class Student {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "student_id")

    // bi - directional association between entities
    // many side of association  and owning side(since it has FK column)
    @ManyToOne(fetch = FetchType.LAZY) /* (fetch = FetchType.LAZY) */ //
fetch policy:  eager by default
    @JoinColumn(name = "c_id",nullable = false) // constraint : Not null :
optional but recommended
    private Course selectedCourse;

    @OneToOne(mappedBy = "stud", cascade = CascadeType.ALL ) // Eager
    private Address studentAdr;

    // one to one association between entity and value type

    @Embedded //OPTIONAL added only for understanding it is embedded
    private AdharCard card;

    public Student() {
        System.out.println("in student cnstr");
    }
    public Student(String email, String name) {
        super();
        this.email = email;
        this.name = name;
    }
```

2. in case of one to many , relation a separate table is created of Value types , for maintaining normalization

- 1. for hobby value type table

  ```
  // one to many asso between entity and collection of value types : it
  is Uni directional

  @ElementCollection // mandetory  : if not : mapping exception
  @CollectionTable(name = "hobbies_tbl",joinColumns = @JoinColumn(name =
  "s_id") ) // optional but recommended
  @Column(name = "hobby",length = 20)
  private List<String> hobbies = new ArrayList<>();
  ```

- 2. edu qualification table created

```
    // one to many asocation between entity n value type : uni
directional(entity--> value type)

    @ElementCollection
    @CollectionTable(name = "edu_qualifications",joinColumns =
@JoinColumn(name = "s_id"))
    private List<EducationalQualifications> qualifications= new ArrayList<>
();
```

4. Case Study - wen app using hibernate

1. web listener for ini and destroy -- resources

- session factory

```
package listerners;

import javax.servlet.ServletContextEvent;
import javax.servlet.ServletContextListener;
import javax.servlet.annotation.WebListener;

import utils.HibernateUtils;

@WebListener
public class SessionFactoryManager implements ServletContextListener {
public void contextDestroye(ServletContextEvent sce) {
        System.out.println("context destroyed");
        HibernateUtils.getSf().close();
    }

public void contextInitialized(ServletContextEvent sce)  {

        System.out.println("contenxt inited");

        HibernateUtils.getSf();
```

```
    }

}

```

2. pojo class

- 1. vendor class

```


@Entity
@Table(name = "vendors_tbl")
public class Vendor {
    @Id //PK
    @GeneratedValue(strategy = GenerationType.IDENTITY) //strategy = AUTO
will be replaced : auto_increment
    @Column(name = "vendor_id")
    private Integer vendorId;
    @Column(length = 30)
    private String name;
    @Column(length = 30,unique = true)
    private String email;
    @Column(length = 30)
    private String password;
    @Column(name="reg_amount")
    private double regAmount;
    @Column(name = "reg_date")
    private LocalDate regDate;//col type=date
    @Enumerated(EnumType.STRING)
    @Column(name="user_role",length = 20)
    private Role userRole;

    // one to many : bi directional asso between entities : here parent ,
one sode , inverse side

    @OneToMany(mappedBy = "accountOwner",cascade =
CascadeType.ALL,orphanRemoval = true)
    private List<BankAccount> bankAccounts = new ArrayList<BankAccount>();

    //def ctor : mandatory
    public Vendor() {
        System.out.println("in vendor ctor");
    }
    //add parametrized constr
    //add all getters n setters
    // add helper method
    public void addAccount(BankAccount b)
    {bankAccounts.add(b);
    b.setAccountOwner(this);
    }
    public void removeAccount(BankAccount b)
    {   bankAccounts.remove(b);
```

```
        b.setAccountOwner(null);
    }
}
```

- 2. Bank Account pojo

```
import javax.persistence.*;

@Entity
@Table(name = "accts_tbl")
public class BankAccount {

    @Id // PK
    @GeneratedValue(strategy = GenerationType.IDENTITY)
   @Column(name = "acct_id")
    private Integer acctNo;

    @Enumerated(EnumType.STRING)
    @Column(name = "ac_type", length = 20)
    private AcType acType;

    private double balance;

    @Column(name = "creation_date")
    private LocalDate creationDate;

    // many to one association between two entities : owning side

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "vendor_id",nullable = false)
    private Vendor accountOwner;

    public BankAccount() {
    System.out.println("in  contsr of" + getClass().getName());
        // TODO Auto-generated constructor stub
    }
}
```

3. bean class

- 1. vendor bean

```
package beans;
import dao.VendorDaoImpl;
import pojos.Role;
import pojos.Vendor;
```

```
public class VendorBean {
    private String email;
    private String password;
    // manage dao
    private VendorDaoImpl vendorDao;
    // add a property to stored validated user details
    private Vendor validatedDetails;
    // add a property to incdicate status
    private String message;
    // default constructor
    public VendorBean() {
    vendorDao = new VendorDaoImpl();
    }
    // setter and getter

    // Add B.L method  :to authenticate user and return dynamic
navigational outcome

    public String validateUser() {

System.out.println("in validate user " + email + password);

// invode dao method : check for runtime exception
try {
validatedDetails =  vendorDao.authenticateUser(email, password);
// valid Login : check role
message = "Login successful";
if(validatedDetails.getUserRole().equals(Role.ADMIN))
return "admin" ;

return "vendor_details";
} catch (RuntimeException e) {
System.out.println("error in bean " + e);
// => implies invalid login
message = "Invalid Login,Please retry ... ";
return "login";
        }
    }
}
```

- 2. bank account bean

```
package beans;
import java.util.List;
import dao.BankAccountImpl;
import pojos.BankAccount;

public class BankAccountBean {
// dao // add a property to represent vendor Id
private int vendorId;
private BankAccountImpl acctDao;
public void setVendorId(int vendorId) {
this.vendorId = vendorId;}

public BankAccountBean() {
// create dao instance
acctDao = new BankAccountImpl();}
// add B.L method to fetch acct for logged in vendor
// method in dao layer
public List<BankAccount> fetchAccounts(){
        // invode account dao method
return acctDao.getAllAccountsByVendorId(vendorId);
    }
}
```

4. JSP

- 1. login.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<%-- create JB instance and add it under sssion scope --%>
<%-- session.addAttribute("vendor_bean", new VendorBean()) --%>
<jsp:useBean id="vendor_bean" class="beans.VendorBean" scope = "session" />

<jsp:useBean id="acct_bean" class="beans.BankAccountBean" scope="session"/>

<body>
<form action="validate.jsp" method="post">
<table style="background-color: cyan; margin: auto;" border="1">
<tr>
    <td>Enter User Email</td>
    <td><input type="text" name="email" /></td>
    </tr>
        <tr>
        <td>Enter Password</td>
<td><input type="password" name="password" /></td>
    </tr>
<tr>
<td><input type="submit" value="Login" /></td>
    </tr>
    </table>
</form>
</body>
</html>
```

```
```

- 2. validate.jsp

```
   <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

<%--transfer conversational state of client to java Bean : setter  --%>
<jsp:setProperty property="*" name="vendor_bean"/>

<%-- redirect client to next page in the Next request --%>
<%--  response.sendRedirect(response.encodeRedirectURL(
session.getAttribute("vendor_bean").valudateUser().concat(".jsp"))) --%>

<c:redirect url="${sessionScope.vendor_bean.validateUser()}.jsp"/>
```

- 3. vendor_details.jsp

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>

<%--
acct_bean.setVenorId(session.getAttribute("vendor_bean").getValidatedDetail
s().setVendorId()) --%>
<jsp:setProperty property="vendorId"
value="${sessionScope.vendor_bean.validatedDetails.vendorId}"
name="acct_bean"/>

<body>

<%-- display login successful message --%>

<h5> ${sessionScope.vendor_bean.message} </h5>

<h4>Vendor details :  ${sessionScope.vendor_bean.validatedDetails}  </h4>


<h5  align="center"> A/C Summary  </h5>
<%-- <h5> ${sessionScope.vendor_bean.validatedDetails.bankAccounts } </h5>
--%>
```

```
<h3>${sessionScope.acct_bean.fetchAccounts()} </h3>

</body>
</html>
```

- 4. admin.jsp

```
<%-- display login successful message --%>

<h5> ${sessionScope.vendor_bean.message} </h5>
<h4>Admin details :  ${sessionScope.vendor_bean.validatedDetails}  </h4>
</body>
</html>
```

Day11

sequence

- Open spring api docs in your web browser (from spring-help/javadocs)

2. Create from scratch spring based Java SE application.
3. In eclipse , change perspective to Java (if not already in Java)
4. Create Java project
5. Create User lib --containing spring/hibernate/jdbc drvr/REST.... JARs.

DON'T use earlier created hibernate lib.

3. Add user lib in build path.(R click --build path --confgure build path--add user lib --only spring_all)

3.5 Copy dependent & dependency packages (containing spring beans) from day11_help/spring-help/rdy code.

4. Create new src folder -- & create spring bean config xml file. R click on src --new src folder --resources R click on resource --new --spring bean configuration file --spring-config.xml
5. Choose namespace beans
6. Configure dependency n dependent beans (as discussed) For setter based D.I
7. Create a tester application , to start Spring container & run this as java application , to confirm spring in core java. Confirm spring bean life cycle (along with scopes, lazy-init,init n destroy methods)

demo on Dependency Injection configuration Explicitely - manual wiring

1. demo on singleton and prototype scope on bean

- in create: resource ---> config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<!--  configure dependency beans  -->
<!--  default scope = Singleton  default loading policy for singleton beans
:eager -->
<bean id="test" class="dependency.TestTransport" lazy-init="true"/>
<bean id="http" class="dependency.HttpTransport" scope="prototype"/>
<bean id="soap" class="dependency.SoapTransport"/>
<!--  configure dependent bean  -->
<!-- default : scope -  singleton : load policy- eager -->
<!-- scope : prototype :only applicable  load policy : lazy (upon demand) :
one per demand-->
<bean id="my_atm" class="dependent.ATMImpl" scope="singleton" lazy-
init="false"
 init-method="myInit" destroy-method="myDestroy" >
<!-- setter based D.I -->
<property name="myTransport" ref="http"/>
</bean>
</beans>
```

- 2. dependency
- to make loose coupling
- on lhs of Transporter

```
public class ATMImpl implements ATM {
    /* private TestTransport myTransport = new TestTransport(); */
    /* private Transport myTransport = new HttpTransport(); */
    private Transport myTransport;
    public ATMImpl() {  }
    @Override
    public void deposit(double amt) {
        System.out.println("depositing "+amt);
        byte[] data=("depositing "+amt).getBytes();
        myTransport.informBank(data);
    }
    @Override
    public void withdraw(double amt) {
        System.out.println("withdrawing "+amt);
        byte[] data=("withdrawing "+amt).getBytes();
        myTransport.informBank(data);
    }
   // setter DI
    public void setMyTransport(Transport myTransport) {
        System.out.println("in set transport setter ");
        this.myTransport = myTransport;
    }
    // init stype method
        public void myInit() {
            System.out.println("in my init of " + getClass().getName() +
"dependency " + myTransport);
```

```
        }
        // destory style method
    public void myDestroy() {
        System.out.println("in my destroy of " + getClass().getName() +
"dependency " + myTransport);
    }
}
```

- 3. dependent

```
public class TestTransport implements Transport {
    public TestTransport() {
        System.out.println("in cnstr of " +getClass().getName());
    }

    @Override
    public void informBank(byte[] data) {
        System.out.println("informing bank using " + getClass().getName() +
" layer");
    }
}
```

- 4. tester

```
public class TestSpring {

    public static void main(String[] args) {
    // start spring container : using xml based metadata instructions ,
placed in run time classpath
        // class :
o.s.c.s(.context.support).ClassPathXmlApplicationContext(String configFIle)
throws BeansException
        try (ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("config.xml")){
            System.out.println("SC started ...");
            // get readymade springbean instance from SC , for invoking B.L
            System.out.println("making first demand");
            ATMImpl atmBean = ctx.getBean("my_atm", ATMImpl.class);

            // B.L
            atmBean.deposit(1000);
            System.out.println("making second demand");
            ATMImpl atmBean2 = ctx.getBean("my_atm", ATMImpl.class);

        System.out.println(atmBean == atmBean2);
        // System.out.println(atmBean..equals(atmBean2));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

2. demo on DI using setter with init and destroy method

- in create: resource ---> config.xml

```
<beans>
<!-- dependency bean config  -->
<bean id="test" class="dependency.TestTransport" scope="singleton" lazy-init="false"/>
<bean  id="http" class="dependency.HttpTransport" scope="prototype"/>
<bean id="soap" class="dependency.SoapTransport" scope="singleton" lazy-init="true"/>
<bean id="email" class="dependency.EmailNotification"/>
<bean id="sms" class="dependency.SMSNotification" />
<!--  dependent bean config  -->
<bean id="atm_bean" class="dependent.ATMImpl"
   init-method="myInit" destroy-method="myDestroy" >
<!-- for setter based D.I -->
<property name="myTransport" ref="soap" />
<property name="customerNotification" ref="email" />
</bean>

</beans>
```

- 2. dependency

```
public class SMSNotification implements NotificationService {

    public SMSNotification() {
    System.out.println("in contr of" + getClass().getName());
    }
    @Override
    public void notifyCustomer(String txType, double amount) {
System.out.println(" NOTIFYING CUSTOMER : Tx type "+ txType + " for amount " + amount + " @ " + LocalDateTime.now()  + "VIA SMS");
    }
}
```

- 3. dependent

```
public class ATMImpl implements ATM {
    private Transport myTransport;
    private NotificationService[] customerNotification;
    public ATMImpl() {
        System.out.println("in cnstr of " +getClass().getName()+" "+
myTransport);
    }
    @Override
    public void deposit(double amt) {
        System.out.println("depositing "+amt);
        byte[] data=("depositing "+amt).getBytes();
        myTransport.informBank(data); // dependent object calling method of
dependency
        // ATM---> NotificationService for alerting customer
        for(NotificationService service : customerNotification)
              service.notifyCustomer("deposit", amt);
    }

    @Override
    public void withdraw(double amt) {
        System.out.println("withdrawing "+amt);
        byte[] data=("withdrawing "+amt).getBytes();
        myTransport.informBank(data);  // dependent object calling method
of dependency
        for(NotificationService service : customerNotification)
          service.notifyCustomer("withdraw", amt);
    }
    public void setMyTransport(Transport myTransport) {
        this.myTransport = myTransport;

    }

    // init stype method : mandetory public ,void
    //  called in both singleton and prototype scope
    // for clean init and destroy code
        public void myInit() {
            System.out.println("in my init of " + getClass().getName() +
"dependency " + myTransport);
        }
        // destory style method  : mandetory public ,void
        // called for only prototype method

    public void myDestroy() {

        System.out.println("in my destroy of " + getClass().getName() +
"dependency " + myTransport
    }

    public void setCustomerNotification(NotificationService[]
customerNotification) {
    System.out.println("in set Notification setter");
        this.customerNotification = customerNotification;
    }
}
```

- 4. tester

```
public class TestSpring {

    public static void main(String[] args) {
        //start spring container in java app: using xml based instruction
stored under runtime class path
        // o.s.c.s.ClassPathXmlApplicationContext : clas
        // BeanFactory <== ApplicationContext <===
ClassPathXMlApplicationContext
        try(ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("config.xml")) {
            System.out.println("SC booted");
            // 1st demand : tell SC to supply located--loaded---
instantiated (defauld constr) -- D.I--bean Instance
            // API : o.s.b.BeanFactory: T getBean(String beanId,Class<T>
beanClass) throws BeanException
        ATMImpl atm1 = ctx.getBean("atm_bean", ATMImpl.class);
        // B.L
        atm1.deposit(1000);
        ATMImpl atm2 = ctx.getBean("atm_bean", ATMImpl.class);
        System.out.println(atm1 == atm2);

        } catch (Exception e) {
            e.printStackTrace();
            // TODO: handle exception
        }}}
```

3. constructor DI

- in create: resource ---> config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<!-- dependency bean config  -->
<bean id="test" class="dependency.TestTransport" scope="singleton" lazy-
init="false"/>
<bean  id="http" class="dependency.HttpTransport" scope="prototype"/>
<bean id="soap" class="dependency.SoapTransport" scope="singleton" lazy-
init="true"/>
<bean id="email" class="dependency.EmailNotification"/>
<bean id="sms" class="dependency.SMSNotification" />
<!--  dependent bean config  -->
<bean id="atm_bean" class="dependent.ATMImpl"  init-method="myInit"
destroy-method="myDestroy" >
```

```
<!-- for constructor based D.I -->
<constructor-arg name="t" ref="test" />
<constructor-arg name="services" ref="sms" />
</bean>
</beans>
```

- 2. dependency

```

```

- 3. dependent

```
package dependent;
public class ATMImpl implements ATM {
    /* private TestTransport myTransport = new TestTransport(); */
    /* private Transport myTransport = new HttpTransport(); */
    private Transport myTransport;
    private NotificationService[] customerNotification;
    public ATMImpl(Transport t,NotificationService[] services) {
         myTransport = t;
        customerNotification = services;
        System.out.println("in cnstr of " +getClass().getName()+" "+
myTransport + services);
    }

    @Override
    public void deposit(double amt) {
        System.out.println("depositing "+amt);
        byte[] data=("depositing "+amt).getBytes();
        myTransport.informBank(data); // dependent object calling method of
dependency
        // ATM---> NotificationService for alerting customer
        for(NotificationService service : customerNotification)
              service.notifyCustomer("deposit", amt);
    }

    @Override
    public void withdraw(double amt) {
        System.out.println("withdrawing "+amt);
        byte[] data=("withdrawing "+amt).getBytes();
        myTransport.informBank(data);  // dependent object calling method
of dependency
        for(NotificationService service : customerNotification)
          service.notifyCustomer("withdraw", amt);
    }
    // init stype method : mandetory public ,void
    //  called in both singleton and prototype scope
    // for clean init and destroy code
        public void myInit() {}
        // destory style method  : mandetory public ,void
        // called for only prototype method
    public void myDestroy() {}

}
```

- 4. tester

```
public class TestSpring {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        //start spring container in java app: using xml based instruction
stored under runtime class path
        // o.s.c.s.ClassPathXmlApplicationContext : clas
        // BeanFactory <== ApplicationContext <===
ClassPathXMlApplicationContext

        try(ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("config.xml")) {

            System.out.println("SC booted");
            // 1st demand : tell SC to supply located--loaded---
instantiated (defauld constr) -- D.I--bean Instance
            // API : o.s.b.BeanFactory: T getBean(String beanId,Class<T>
beanClass) throws BeanException
        ATMImpl atm1 = ctx.getBean("atm_bean", ATMImpl.class);
        // B.L
        atm1.deposit(1000);
        } catch (Exception e) {
            e.printStackTrace();
            // TODO: handle exception
        }


    }

}
```

4. DI with property and constructor

- in create: resource ---> config.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <!-- dependency bean config. -->
    <!-- singleton n eager -->
    <bean id="test" class="dependency.TestTransport" />
    <!-- scope=prototype -->
    <bean id="http" class="dependency.HttpTransport" scope="prototype" />
    <!-- singleton n lazy -->
    <bean id="soap" class="dependency.SoapTransport" lazy-init="true" />
    <!-- add dependency beans for cust notification -->
    <bean id="email" class="dependency.EmailNotification" />
    <bean id="sms" class="dependency.SMSNotification" />

    <!-- dependent bean : atm_bean(id) :singleton n eager , dep : setter
based D.I
        : soap -->
    <bean id="atm_bean" class="dependent.ATMImpl"
        init-method="init123" destroy-method="destroy123">
        <!-- constructor based D.I -->
        <constructor-arg name="cash123" value="12345678"/>
        <!-- setter based D.I -->
        <property name="myTransport" ref="soap"/>
        <property name="customerNotification" ref="email"/>
    </bean>
</beans>
```

- 2. dependency

```
public class ATMImpl implements ATM {
    private Transport myTransport;
    private NotificationService[] customerNotification;
    private double cash;

    public ATMImpl(double cash123) {
        cash=cash123;
        System.out.println("in cnstr of " + getClass().getName() + " " +
myTransport+" "+customerNotification+" "+cash);
    }
    @Override
    public void deposit(double amt) {
        System.out.println("depositing " + amt);
        byte[] data = ("depositing " + amt).getBytes();
        myTransport.informBank(data);// dependent obj(ATM) is calling
method of dependency(Tranport) : for informing
                                        // underlying bank
        // ATM ---> NoticationService for alerting the customer
        for (NotificationService service : customerNotification)
            service.notifyCustomer("Withdraw", amt);
    }

    @Override
    public void withdraw(double amt) {
        System.out.println("withdrawing " + amt);
        byte[] data = ("withdrawing " + amt).getBytes();
        myTransport.informBank(data);// dependent obj is calling method of
dependency
        // ATM ---> NoticationService for alerting the customer
        for (NotificationService service : customerNotification)
            service.notifyCustomer("Withdraw", amt);

    }
    // add init n destroy style methods
    public void init123() {
        System.out.println("in init " + myTransport+"
"+customerNotification+" "+cash);
    }
    //add 2 setters for setter based D.I
    public void setMyTransport(Transport myTransport) {
        System.out.println("in set transport");
        this.myTransport = myTransport;
    }

    public void setCustomerNotification(NotificationService[]
customerNotification) {
        System.out.println("in set cust notification");
        this.customerNotification = customerNotification;
    }
    public void destroy123() {
        System.out.println("in destroy " + myTransport);
    }

}
```

5. factory based DI

- 1. xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <!-- dependency bean config. -->
    <!-- singleton n eager -->
    <bean id="test" class="dependency.TestTransport" />
    <!-- scope=prototype -->
    <bean id="http" class="dependency.HttpTransport" scope="prototype" />
    <!-- singleton n lazy -->
    <bean id="soap" class="dependency.SoapTransport" lazy-init="true" />
    <!-- add dependency beans for cust notification -->
    <bean id="email" class="dependency.EmailNotification" />
    <bean id="sms" class="dependency.SMSNotification" />
<!-- dependent bean : atm_bean(id) :singleton n eager , dep : setter based
D.I
        : soap -->
    <bean id="atm_bean" class="dependent.ATMImpl"
```

```
init-method="init123" destroy-method="destroy123" factory-
method="myFactory">
        <!-- constructor based D.I -->
        <constructor-arg name="cash123" value="456" />
        <constructor-arg name="t" ref="test" />
        <constructor-arg name="service" ref="sms" />
    </bean>
</beans>
```

- 2. dependent

```
public class ATMImpl implements ATM {
    private Transport myTransport;
    private NotificationService[] customerNotification;
    private double cash;
    private ATMImpl(double cash123,Transport t,NotificationService[]
service) {
        cash=cash123;
        myTransport = t ;
        customerNotification = service;
        System.out.println("in cnstr of " + getClass().getName() + " " +
myTransport+" "+customerNotification+" "+cash);
    }
    @Override
    public void deposit(double amt) {}
    @Override
    public void withdraw(double amt) {}
    // add init n destroy style methods
    public void init123() {
    System.out.println("in init " + myTransport+" "+customerNotification+"
"+cash);
    }
    // no  setters required
    // add a factory method : For demo of Factory based D.I
    public static ATMImpl myFactory(double cash123,Transport
t,NotificationService[] service) {
    System.out.println("in factory method ");
     // invoke private constr   :to create the bean instance and return it
to the caller
    return new ATMImpl(cash123, t, service);}
      public void destroy123() {System.out.println("in destroy " +
myTransport);}
}
```

Day12

demo on Spring Auto wiring

1. demo on configuring spring bean implicitely (auto wiring)

- 1. auto wiring by name (setter based D.I)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<!--  configure dependent bean  -->
<!-- default : scope -  singleton : load policy- eager -->
<!-- scope : prototype :only applicable  load policy : lazy (upon demand) :
one per demand-->
<bean id="my_atm" class="dependent.ATMImpl" scope="singleton" lazy-
init="false"
init-method="myInit" destroy-method="myDestroy" autowire="byName" />
<!--  configure dependency beans  -->
<!--  default scope = Singleton  default loading policy for singleton beans
:eager -->
<bean id="test" class="dependency.TestTransport" lazy-init="true"/>
<bean id="mtTransport" class="dependency.HttpTransport" scope="prototype"/>
<bean id="soap" class="dependency.SoapTransport"/>
</beans>
```

- 2. setter based
- auto wiring by type : setter based D.I (un comment from xml other bean tags n understand exception)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<!--  configure dependent bean  -->
<!-- default : scope -  singleton : load policy- eager -->
<!-- scope : prototype :only applicable  load policy : lazy (upon demand) :
one per demand-->
<bean id="my_atm" class="dependent.ATMImpl" scope="singleton" lazy-
init="false"
init-method="myInit" destroy-method="myDestroy" autowire="byType" />
<!--  configure dependency beans  -->
<!--  default scope = Singleton  default loading policy for singleton beans
:eager -->
<!-- <bean id="test" class="dependency.TestTransport" lazy-init="true"/>
<bean id="mtTransport" class="dependency.HttpTransport" scope="prototype"/>
-->
<bean id="soap" class="dependency.SoapTransport"/>
</beans>
```

- 3. setter based :
- auto wiring by type : setter based D.I (array injected)

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

```
<!--  configure dependent bean  -->
<!-- default : scope -  singleton : load policy- eager -->
<!-- scope : prototype :only applicable  load policy : lazy (upon demand) :
one per demand-->
<bean id="my_atm" class="dependent.ATMImpl" scope="singleton" lazy-
init="false"
init-method="myInit" destroy-method="myDestroy" autowire="byType" />
<!--  configure dependency beans  -->
<!--  default scope = Singleton  default loading policy for singleton beans
:eager -->
<bean id="test" class="dependency.TestTransport" lazy-init="true"/>
<bean id="mtTransport" class="dependency.HttpTransport" scope="prototype"/>
<bean id="soap" class="dependency.SoapTransport"/>
</beans>
<!--NoUniqueBeanDefinitionException: No qualifying bean of type
'dependency.Transport'
available: expected single matching bean but found 3: test,mtTransport,soap
-->
```

- 4. constructor based
- auto wiring using constructor (constr based D.I)

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
<!--  configure dependent bean  -->
<!-- default : scope -  singleton : load policy- eager -->
<!-- scope : prototype :only applicable  load policy : lazy (upon demand) :
one per demand-->
<bean id="my_atm" class="dependent.ATMImpl" scope="singleton" lazy-
init="false"
init-method="myInit" destroy-method="myDestroy" autowire="constructor" />
<!--  configure dependency beans  -->
<!--  default scope = Singleton  default loading policy for singleton beans
:eager -->
<bean id="test" class="dependency.TestTransport" lazy-init="true"/>
<bean id="mtTransport" class="dependency.HttpTransport" scope="prototype"/>
<bean id="soap" class="dependency.SoapTransport"/>
</beans>
<!--NoUniqueBeanDefinitionException: No qualifying bean of type
'dependency.Transport'
available: expected single matching bean but found 3: test,mtTransport,soap
-->
```

- dependent class

```

    public class ATMImpl implements ATM {
    /* private TestTransport myTransport = new TestTransport(); */
    /* private Transport myTransport = new HttpTransport(); */
    private Transport[] myTransport;
    public ATMImpl(Transport[] transports) {

        myTransport = transports;
        System.out.println("in cnstr of " +getClass().getName()+" "+
myTransport);
        } }
```

- 5. hybrid Approach
- ybrid approach --reduced xml n majority of annotations
    - config file

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns=
   xmlns:xsi=
   xmlns:context=
   xsi:schemaLocation=>

<!--  Enable class internal annotation support eg autowired
/requestMapping/postcCnstruct -->
<context:annotation-config />
<!-- To tell SC about the location of base package of the spring beans -->
<context:component-scan base-package="dependency,dependent"/>
</beans>

<!--NoUniqueBeanDefinitionException: No qualifying bean of type
'dependency.Transport'
available: expected single matching bean but found 3: test,mtTransport,soap
-->
```

- dependent class

```
@Component("my_atm") // to tell SC whatever folows is a spring bean : Which
life cycle has to be managed by SC
public class ATMImpl implements ATM {
   @Autowired //(required = false) // by default : required = true =>
mandetory: :
   // :autowire = byType : tries to match data type of property with data
type of dependency bean
   @Qualifier("httpTransport") // @AutoWired + Qualifier  = auto wired : by
Name
   private Transport myTransport;
   public ATMImpl() {
   System.out.println("in cnstr of " +getClass().getName()+" "+
myTransport);
```

```
    }
    @Override
    public void deposit(double amt) {
    }
    @Override
    public void withdraw(double amt) {
    }

    // init stype method
       @PostConstruct
     public void myInit() {
    System.out.println("in my init of " + getClass().getName() + "dependency
" + myTransport);
     }
     // destory style method

    @PreDestroy
    public void myDestroy() {

    System.out.println("in my destroy of " + getClass().getName() +
"dependency " + myTransport);
    }
}
```

- dependency class

```
// singleton and eager
@Component("test")
public class TestTransport implements Transport {
    public TestTransport() {
        System.out.println("in cnstr of " +getClass().getName());
    }
    @Override
    public void informBank(byte[] data) {
        System.out.println("informing bank using " + getClass().getName() +
" layer");

    }

}
--------------------------

@Component
@Scope(value = "prototype")
// derived bean id = "httpTransport"
public class HttpTransport implements Transport {
    public HttpTransport() {
        System.out.println("in cnstr of " +getClass().getName());
    }

    @Override
    public void informBank(byte[] data) {
        System.out.println("informing bank using " + getClass().getName() +
" layer");

    }

}
-------------------------------------------
// singleton and lazy
@Component("soap")
@Lazy(value = true)
public class SoapTransport implements Transport {
    public SoapTransport() {
        System.out.println("in cnstr of " +getClass().getName());
    }

    @Override
    public void informBank(byte[] data) {
        System.out.println("informing bank using " + getClass().getName() +
" layer");

    }

}
```

- tester

```
public static void main(String[] args) {
// start spring container : using xml based metadata instructions , placed
in run time classpath
// class : o.s.c.s(.context.support).ClassPathXmlApplicationContext(String
configFIle) throws BeansException
try (ClassPathXmlApplicationContext ctx = new
ClassPathXmlApplicationContext("config.xml")){

System.out.println("SC started ...");
// get readymade springbean instance from SC , for invoking B.L
System.out.println("making first demand");
ATMImpl atmBean = ctx.getBean("my_atm", ATMImpl.class);
// B.L
    atmBean.deposit(1000);
System.out.println("making second demand");
ATMImpl atmBean2 = ctx.getBean("my_atm", ATMImpl.class);
System.out.println(atmBean == atmBean2);
// System.out.println(atmBean..equals(atmBean2));
} catch (Exception e) {
        e.printStackTrace();
        }
    }
```

demo on Spring MVC :Create spring MVC based web application from scratch

0. create user library for srpring files

- add to deployment path : /WED-INF/spring
- add to libraries

1. config file

- servlet deployment tags to configure Dep.Serv : FC for spring MVC

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://xmlns.jcp.org/xml/ns/javaee"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd" id="WebApp_ID"
version="3.1">
  <display-name>Lab12.6_spring-mvc</display-name>
  <welcome-file-list>
    <welcome-file>index.jsp</welcome-file>

  </welcome-file-list>

  <!-- servlet deployment tags to configure Dep.Serv : FC for spring MVC-->

  <servlet>
  <servlet-name>spring</servlet-name>
  <servlet-
class>org.springframework.web.servlet.DispatcherServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
  <servlet-name>spring</servlet-name>
  <url-pattern>/</url-pattern>

  </servlet-mapping>

</web-app>
```

2. add servlet.config file in WEB_INF

- include 4 namespaces : bean,context,p,mvc
- for enabling class internal annotations
  - to inform SC about the location of base package
- to enable annotations based MVC support
- configure view resolver bean for auto translation

```
<?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.orgschema/context"
xmlns:mvc="http://www.springframework.org/schema/mvc"
xmlns:p="http://www.springframework.org/schema/p"
    xsi:schemaLocation="">

<!-- for enabling class internal annotations  -->
<context:annotation-config/>

<!-- to inform SC about the location of base package -->
<context:component-scan base-package="com.app"/>

<!-- to enable annotations based MVC support -->

<mvc:annotation-driven/>
<!-- configure view resolver bean for auto translation  -->
<bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
 p:prefix="/WEB-INF/views" p:suffix=".jsp"
p:viewClass="org.springframework.web.servlet.view.JstlView"
 />
<!-- import hibernate config xml file -->
<!--<import resource="classpath:/hibernate-persistence.xml"/>-->

</beans>
```

3. add controller in src folder under one packge

- 1. use @Controller
  - mandetory to tell SC:
  - whatever follows is a req handling controller bean
  - spring bean : singleton and eager
- 2. @RequestMapping("")

- to tell SC about request handling method :

  - entry in Handler Mapping bean
  - key = /hello

- value = com.app.controller.HelloController:sayHello()

- 3. o.s.w.s.Model and View :
  - holder for holding ModelAttribute + logical view name :class
  - consructor

> ModelAndView(String logicalViewName,String modelAttrName,Object modelAttrValue)

  - def scope model attr : current request only

- e.g

> return new ModelAndView("/welcome", "time",LocalDateTime.now());

- 4. add request handling methods to Test Map
  - o.s.ui.Model: i/f
  - => holder of Model attributes
  - How to add attributes ?
  - > Model addAttribute(String modelAttrName,Object modelAttrVal)
  - IOC : simply tell SC : to inject EMPTY model map in the request handling method :
  - D.I by adding an argument to req handling method

```
package com.app.controller;

import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.servlet.ModelAndView;

@Controller // mandetory to tell SC: whatever follows is a req handling
controller bean
//spring bean : singleton and eager
public class HelloController {

    public HelloController() {

        System.out.println("in constr of "  + getClass().getName());

        // TODO Auto-generated constructor stub
    }

    // to tell SC about request handling method :
        // entry in Handler Mapping bean
        // key = /hello
        //value = com.app.controller.HelloController:sayHello()
    @RequestMapping("/test")
    public String sayHello() {

        System.out.println(" hello , its test 1");

        return "/welcome";
        // req handling controller returns
        //: logical view name (forward view ) to D.S(Dispatcher Servlet)

    }
    // add request handling method to test o.s.w.s.ModelAndView
    @RequestMapping("/test2")
    public ModelAndView sayhello2() {
        System.out.println("in test 2 ");
        //o.s.w.s.Model and View  :holder for holding ModelAttribute  +
logical view name  :class
        // consructor ModelAndView(String logicalViewName,String
modelAttrName,Object modelAttrValue)
        // def scope model attr : current request only

        return new ModelAndView("/welcome", "time", LocalDateTime.now());
        // request handling controller returning logical view name + 1
model Attr ---> DS
    }
    // add request handling methods to Test Map
        // o.s.ui.Model: i/f => holder of Model attributes
        // How to add attributes  ? Model addAttribute(String
modelAttrName,Object modelAttrVal)
        // IOC : simply tell SC : to inject EMPTY model map in the request
handling method : D.I by
        // : by adding an argument to req handling method
    @RequestMapping("/test3")
    public String sayhello3(Model map) {
        System.out.println("in test 3");

        map.addAttribute("date", LocalDate.now())
        .addAttribute("list", Arrays.asList(10,20,30,40,50));

        return "/welcome";


    }

}
```

1. forward request to welcome.jsp from Controller

- access them using EL synttax from request scope

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
    pageEncoding="UTF-8"%>

<h4> Welocme to Spring MVC by request mapping</h4>

<h4> Test 2: Spring server Time : ${requestScope.time} </h4>

<h4> Test 3 : Spring server date :${requestScope.date}</h4>
<h3>Test 3: Spring serverlist: ${requestScope.list}</h3>
</body>
</html>
```

Lab sequence

0. Open J2EE perspective

1. Create dynamic web project
2. Add spring-all user lib in 2 places

- 2.1 Under build path (R click on project --build path --configure build path --Add --user lib --spring-all) Apply.

- 2.2 Under deployment assembly(WEB-INF/lib) Choose deployment assembly option --add --build path entries --select spring-all --apply n close.

- All XML templates are present in : -

> day12-data\day12_help\spring-hibernate-templates folder

3. Copy welcome-file-list , servlet & servlet-mapping tags from the web.xml under templates folder

- Meaning :

- To Configure spring supplied Front controller(o.s.w.s.DispatcherServlet) to intercept any request from any client, in web.xml.

- Detailed explanation of tags

```
<url-pattern>/</url-pattern> => any request received from any client
<servlet-name>spring</servlet-name> => can be replaced by any name
<servlet-class>org.springframework.web.servlet.DispatcherServlet</servlet-
class> => spring supplied Front Controller
<load-on-startup>1</load-on-startup> => Web container (WC) will start the
life cycle of DispatcherServlet , at the web app deployment time.
```

- 4. Job of D.S (DispatcherServlet)
- To start the SC (spring container) in web app.

4. Copy spring-servlet.xml from templates folder , under of your web app.

- Initially comment or remove this line.

- 1. Meaning

    - spring-servlet.xml

    - => master configuration xml file for starting SC.

    - D.S(DispatcherServlet) reads this config file ,

    - @ web app deployment time , to start SC , within a web app.

    - (represented by i/f WebApplicationContext --sub i/f of ApplicationContext)

    - Default location of this master configuration xml file--

    - Default name -- servletName-servlet.xml

    - Since we have supplied , servlet-name , in web.xml as spring , in our case it's spring-servlet.xml

5. Detailed explanation of tags

- 1. <context:annotation-config />
- => Enables class internal annotations.
- 2. <context:component-scan base-package="com.app"/>
- => SC will scan only com.app & its sub packges for spring beans.
- 3. mvc:annotation-driven/
- => Enables annotation based MVC support (
- i.e enables automatic population of HandlerMapping bean using @RequestMapping annotation in Controller beans
- 4. Regarding HandlerMapping bean

> o.s.w.s.HandlerMapping --i/f

- Implementation class -- >RequestMappingHandlerMapping
- Consists of a map , populated by SC @ web app deployment time.
- 1. Key --value of @RequestMapping annotation, in controller bean
- 2. Value --F.Q controller cls name + method name.
-

  5.

```
 <bean id="viewResolver"
class="org.springframework.web.servlet.view.InternalResourceViewResolver"
p:prefix="/WEB-INF/views" p:suffix=".jsp"
p:viewClass="org.springframework.web.servlet.view.JstlView"
/>
```

- 1. Meaning : Configure , spring supplied ViewResolver bean , for translation from logical view name to actual view name (by wrapping it in prefix n suffix)
- 2. More Details about ViewResolver Bean
- o.s.w.s.ViewResolver --i/f
- Implementation class :
- > o.s.w.s.view.InternalResourceViewResolver
- It uses setter based D.I .
- It has 3 properties
- 1. prefix = /WEB-INF/views
- 2. suffix = .jsp

- 3. viewClass = JstlView
- eg : If logical view name is "/welcome"
- Actual view name will be : /WEB-INF/views/welome.jsp

This completes configuration steps .

6. Create request handling controller, to test MVC flow.(under com.app.controller pkg)

- eg : HelloController
- Mandatory Annotations used --
- 1. @Controller --class level.
- 2. @RequestMapping --method level annotation
  - (To map clnt requests onto specific controller class's specific method) eg :

```
 @Controller
public class HelloController {

@RequestMapping("/hello1")
    public String sayHello1()
    {
      return "/welcome";
    }
}
```

- 1. Explanation :
  - HelloController bean is singleton n eager (i.e it's single instance will be created at web app deployment time)
- So for this controller , what will be the entry in HandlerMapping bean, created at web app deployment time ?
  - Key -- /hello1
  - Value -- com.app.controller.HelloController.sayHello1

7. To Test Spring MVC flow --

- 1. Add index.jsp in WebContent , with a link. Test Spring MVC Flow Note : href of the anchor tag MUST match will the value of the @RequestMapping annotation.
- 2. Add a welcome.jsp under a folder /views/ & add a welcome message.

8. Run web application

```
R click --run on server
Troubleshooting tips : check on server console for the following :
7.1 Initializing Spring DispatcherServlet 'spring'
7.2 org.springframework.web.servlet.DispatcherServlet - Initializing
Servlet 'spring'
```

```
7.3 in constr of com.app.controller.HelloController
in init
This indicates that spring based web app is up n running.
Go to client browser & test it!
```

dAY 13

sequence (spring hibernate integration steps)

1. Create dynamic web project

2. Create User library --spring-hibernate-rest jars DO NOT add any other library. Add user lib under 2 places : build path n deployment assembly.

3. Add DispatcherServlet entry in web.xml -- to ensure all request pass through central dispatcher servlet.

4. Create spring-servlet.xml under -- To allow D.S to create Web application context using master config xml file.

- 4.1 Copy earlier entries.(ctx,mvc & view resolver)

5. Create & copy database.properties & hibernate-persistence.xml from What it contains ---

- 5.1 DataSource bean --- Apache (Connection pool)
- 5.2 SF bean -- Spring
- 5.3 Tx Mgr --- Spring
- 5.4 enabled anno support for Txs(@Transactional)

6. import hibernate-persistence.xml into spring-servlet.xml

What is o.s.orm.hibernate5.LocalSessionFactoryBean? A class that creates a Hibernate SessionFactory. This is the usual way to set up a shared Hibernate SessionFactory in a Spring application context; the SessionFactory can then be passed to data access objects via dependency injection.

Configuration steps over....

7. Identify persistence requirements & create POJO/Model/DTO. POJO properites --- represent 1. DB cols 2.Request params --i.e clnt's conversational state.

- P.L validation rules --anno. class level --@Entity,@Table Anno -- field level --- @NotEmpty,@NotNull,@Email.... Annotation -- prop level(getter) --@Id,@Column....

8. Create DAO layer

I/F -- Dao i/f --- validateCustomer Implementation class --- dependency --- SessionFactory -- @AutoWired No need to manage Txs --directly get session from SF & perform CRUD operation.

9. Create Service Layer --i/f & then implementation class @Service & @Transactional --- annotations. Inject dependency of Dao Layer.

10. Create or copy existing controllers & test the flow.

day 14

For Maven web app with hibernate

- (POM.xml support for ---Spring MVC/ AOP/Core/REST/Hibernate)

1. Create Maven Project New -- Maven Project --- Check Create Simple Project (skip archetype selection) Choose WAR Check Use default workspace location Next Group ID -- reversed domain name (eg : if domain name is www.serverside.com , then it can be com.serverside) Artifact ID -- Name of the WAR File Name -- Testing web app with hibernate Finish

2. Modify pom.xml , to add 2.1 Properties -- 2.2 Build plugins -- 2 plugins (maven-compiler-plugin & maven-eclipse-plugin) 2.3 Dependencies for --mysql , JUnit , spring , hibernate & json.

3. Project -- R Click -- Maven --Update Project

4. Check project structure for simple Java web application project. src/main/java -- Java sources src/test/java -- JUnit Test Cases src/main/resources --configuration files src/test/resources --configuration files for testing src/main/webapp --root of web application(equivalent to WebContent)

5. Choose Java EE perspective Project -- R Click --Java EE Tools --Generate dep desc stub This will create src/main/webapp/WEB-INF/web.xml

Add welcome page as "index.jsp" in web.xml

6. Project -- R Click --Properties --Targeted Runtimes -- Choose Tomcat 8

7. Create packaged classes under src/main/java. utils,pojos,dao,beans,listeners

7.5 Copy log4j.properties,database.properties,hibernate-ersistence.xml(i.e all configuration files) under src/main/resources

8. Project -- R Click -- Maven Build --goals --clean install

8.5 In case of any errors(red cross!) Project -- R Click -- Maven --Update Project

9. Project -- R Click --- Run on server ---run

Steps in Spring Boot

If you are using Spring Boot for creating spring MVC web app (view layer)

1. pom.xml : Add 2 dependencies for : tomcat-embed-jasper & JSTL

org.apache.tomcat.embed tomcat-embed-jasper javax.servlet jstl

For Oracle DB com.oracle.database.jdbc ojdbc8 19.6.0.0

2. application.properties spring.mvc.view.prefix=/WEB-INF/views spring.mvc.view.suffix=.jsp

3. Add folders : below src/main webapp/WEB-INF : to add view layer.

4. Replace SF.getCurrentSession : EntityManager (@PersistenceContext)

5. Simple case study : Product based.

Complete admin flow : register new vendor Update vendor details

Tx management internals

PRG pattern(Post-redirect-get pattern) --- to avoid multiple submission issue in a web app. Replace forward view(server pull) by redirect view (clnt pull) --a.k.a double submit guard.

How to replace default forward view by redirect view in spring MVC ? Ans -- use redirect keyword. eg : return "redirect:/vendor/details"; D.S invokes response.sendRedirect(response.encodeRedirectURL("/vendor/details")); Next request from clnt --- ..../vendor/details

How to remember user details till logout? Ans : add them in session scope. How to access HttpSession in Spring? Using D.I How -- Simply add HttpSession as method argument of request handling method.

How to remember the details(attributes) till the next request (typically required in PRG --redirect view) Ans -- Add the attributes under flash scope. (They will be visible till the next request from the same clnt) How to add ? Use i/f -- o.s.w.s.mvc.support.RedirectAttributes Method public RedirectAttributes addFlashAttribute(String attrName,Object value)

How to access them in view layer in the next request? via request scope attributes.

eg : In case of successful login --save user details under session scope(till user log out) & retain status mesg only till the next request. In case of invalid login --save status under request scope.

How to take care of links(href)/form actions + add URL rewriting support ?

1. Import spring supplied JSP tag lib. (via taglib directive) prefix ="spring"

2. Use the tag. Log Out / --- root of curnt web app.

What will be the URL if cookies are enabled ? http://host:port/spring_mvc/user/logout

What will be the URL if cookies are disabled ? http://host:port/spring_mvc/user/logout;jsessionid=egD5462754

OR form action example eg :

.....

From Logout

1. Discard session
2. Forward the client to logout.jsp

How to auto navigate the clnt to home page after logging out after some dly ? Ans : By setting refresh header of HTTP response.

API of HttpServletResponse public void setHeader(String name,String value)

name --- refresh value --- 10;url=home page url (root of web app)

How to get the root of curnt web app ? API of HttpServletRequest String getContextPath()

What will hapeen if any controller returns redirect view name to D.S ? eg : UserController -- return "redirect:/admin/list" D.S skips the V.R & sends temp redirect response to the clnt browser. How ? D.S invokes --- response.sendRedirec(response.encodeRedirectURL(".../admin/list"); So clnt browser will send a next request ---with method=get URL -- http://host:port/spring_mvc/admin/list

Complete Admin Flow

1. List Vendors
2. vendor deletion