Version Java EE 8 (J2EE 1.8) maintained under Oracle / Jakarta EE 8 (maintained by eclipse foundation)

1. What is J2EE ?(Java Enterprise Edition)

Consists of specifications only .

Which specs ? (Rules or contract)
Specifications of primary services required for any enterprise application.

Q . What is enterprise application ?

An enterprise application (EA) is a large software system platform designed to operate in a corporate environment .

It includes online shopping and payment processing, interactive product catalogs, computerized billing systems, security, content management, IT service management, business intelligence, human resource management, manufacturing, process automation, enterprise resource planning ....

These specifications include ---

Servlet API,JSP(Java server page) API,Security,Connection pooling ,EJB (Enterprise Java Bean), JNDI(Naming service -- Java naming & directory i/f),JPA(java persistence API),JMS(java messaging service),Java Mail, Java Server Faces , Java Transaction API, Webservices support(SOAP/REST) etc...

Vendor of J2EE specs -- Oracle / Sun / Eclipse

Implementation -- left to vendors (J2EE server vendors)
J2EE compliant web server --- Apache -- Tomcat (web server)
Services implemented --- servlet API,JSP API,Security,Connection pooling,JNDI(naming service)

J2EE complaint application server --- web container + EJB (enterprise java bean) container
+ ALL J2EE services implementation

J2EE server Vendors & Products
Apache -- tomcat(web server) / Tomee (app server)
Oracle / Sun --- reference implementation --- Glassfish
Red Hat -- JBoss (wild fly)
Oracle / BEA -- weblogic
IBM -- Websphere

2. WHY J2EE

1. Can support different types of clients --- thin client(web client)
thick client --- application client(eg : TCP client)
smart clients -- mobile clients

2. J2EE server independence --- Create & deploy server side appln --on ANY J2ee compliant server --- guaranteed to produce SAME results w/o touching or re-deploying on ANY other J2EE server

3. Ready made implementation of primary services(eg --- security, conn,pooling,email....)--- so that J2EE developer DOESn't have to worry about primary services ---rather can concentrate on actual business logic.

3. Layers involved in HTTP request-response flow (refer to day1-data\day1_help\diags\request-response-flow.png)

Web browser sends the request (URL)
 eg : http://www.abc.com:8080/day1.1
/day1.1 --- root / context path /web app name

Host --Web server--Web Container(server side JVM)--Web application---HTML/JSP/Servlet....

4. What is dynamic web application --- server side application --deployed on web server --- meant for servicing typically web clients(thin) -- using application layer protocol HTTP /HTTPS
(ref : diag request-resp flow)

Read --HTTP basics including request & response structure from day1-data\day1_help\j2ee_prerequisites\HTTP Basics

5. Objective ?: Creating & deploying dyn web appln on Tomcat -- For HTML content

6. IDE automatically creates J2EE compliant web application folder structure .
Its details -- Refer to diag (J2EE compliant web app folder structure)

7. What is Web container --- (WC) & its jobs
1. Server side JVM residing within web server.
Its run-time environment for dynamic web components(Servlet & JSP,Filter) .
Jobs ---
1. Creating Http Request & Http response objects
2. Controlling life-cycle of dyn web comps (manages life cycle of servlet,JSP,Filters)
3. Giving ready-made support for services --- Naming,security,Conn pooling .
4. Handling concurrent request from multiple clients .
5. Managing session tracking...

8. What is web.xml --- Deployment descriptor one per web appln
created by -- developer
who reads it -- WC
when --- @ deployment
what --- deployment instructions --- welcome page, servlet deployment tags, sess config, sec config.....

---

9. Why servlets? --- To add dynamic nature to the web application

What is a servlet ?
-- Java class (with NO main method) -- represents dynamic web component - whose life cycle will be managed by WC(web container : server side JVM)
no main method
life cycle methods --- init,service,destroy

Job list

1. Request processing
2. B.L (Buisness Logic)
3. Dynamic response generation
4. Data access logic(DAO class --managing DAO layer)
5. Page navigation

Servlet API details --refer to diag servlet-api.png

Objective 1: Test basic servlet life cycle -- init , service ,destroy
10. Creating & deploying Hello Servlet.

Deployment of the servlet
1. Via annotation
eg : @WebServlet(value="/validate")
public class LoginServlet extends H.S {....}
Map :
key -- /validate — / is important
value -- F.Q servlet cls name servlet page Name
URL : http://host:port/day1.1/validate?....

2. Using XML tags
How to deploy a servlet w/o annotations --- via XML tags
web.xml

```
<servlet>
 <servlet-name>abc</servlet-name>
<servlet-class>pages.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>abc</servlet-name>
 <url-pattern>/test2</url-pattern>
</servlet-mapping>
```

these two Names should Match

WC : map
key : /test2
value : pages.SecondServlet

eg URL --http://host:port/day1_web/hello

At the time of web app deployment ---WC tries to populate map of url patterns , from XML tags (from web.xml). Later ---it will check for @WebServlet annotation

Objective 2: Test basic servlet life cycle -- init , service ,destroy (deployed via xml)

How to read request params sent from the clnt ?

javax.servlet.ServletRequest i/f methods
1. public String getParameter(String paramName)

2. public String[] getParameterValues(String paramName) Multiple values we use Array

Objective 3 : Accept different type of i/ps from user , in HTML form.Write a servlet to display request parameters.

Why HttpServlet classs is declared as abstract class BUT with 100 % concrete functionality ?

It is abstract because the implementations of key servicing methods have to be provided by (e.g. overridden by) servlet developer. Since it's abstract , it's instance can't be created.

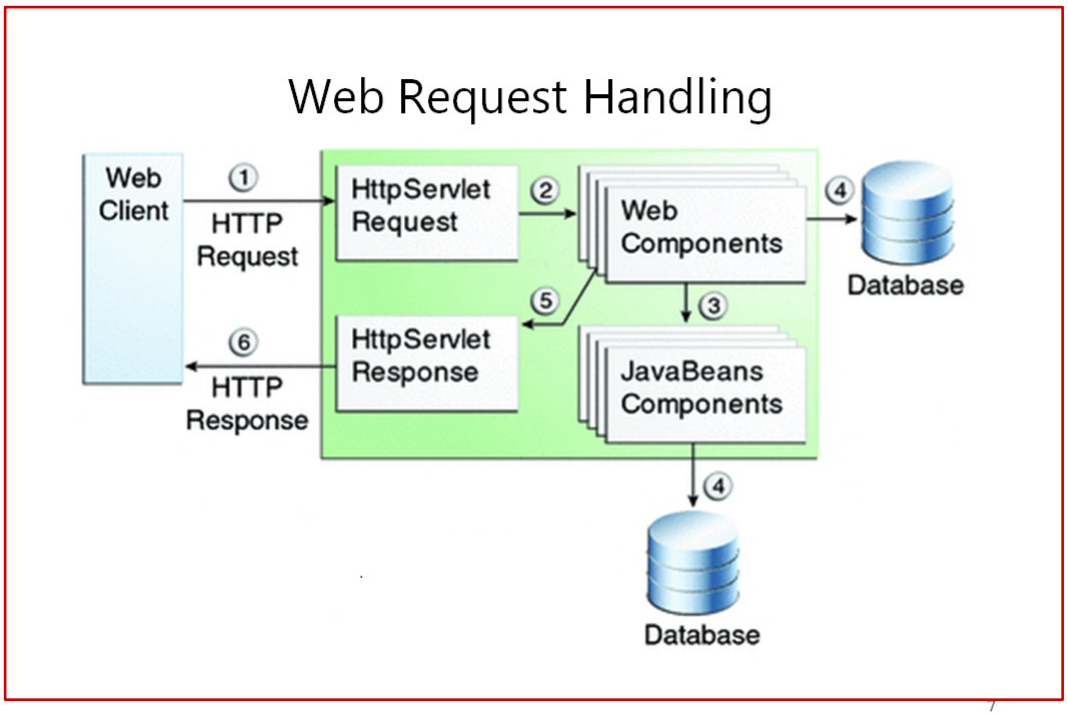
A subclass of HttpServlet must override at least one method, usually one of these:

doGet, if the servlet supports HTTP GET requests
doPost, for HTTP POST requests
doPut, for HTTP PUT requests
doDelete, for HTTP DELETE requests
init and destroy, to manage resources that are held for the life of the servlet

If you extend the class without overriding any methods, you will get a useless servlet; i.e. it will give an error response for all requests.(HTTP 405 : Method not implemented) . So , if the class was not abstract, then any direct instance of HttpServlet would be useless.

So the reason for making the HttpServlet class abstract is to prevent a programming error.

As a servlet developer , you can choose to override the functionality of your requirement (eg : doPost)
& ignore other methods.



Servlet API implementation classes in server provided JARs

Servlet API

javax.servlet.Servlet ---i/f --consists of mainly life cycle methods --init,service,destroy

javax.servlet.GenericServlet --abstract imple class of Servlet i/f
init & destory --concrete , service --abstract
represents pro tocol inde. servlet.
(not reco to be used with HTTP)

javax.servlet.http.HttpServlet -- abstract sub class of GenericServlet.
concrete service method.
public void service(ServletRequest rq,ServletResponse rs) throws ServletException,IOException

protected void service(HttpServletRequest rq,HttpServletResponse rs) throws Servlet Exception , IOException

Checks for HTTP request method

get post put delete

protected void doGet(HttpServletRequest rq,HttpServletResponse rs)
throws SE,IOExc
doPost /doPut/doDelete ---

For creating a servlet ---eg : HelloServlet
public class HelloServlet extends H.S
{
//MUST override at least one of the doXXX methods --eg : doGet
For complete understanding of life cycle
override --init , doGet & destroy.
doGet ---

1. set response content type
API of HttpServletResponse
public void setContentType(String type)
eg : rs.setContentType("text/html");
2. To send response from servlet --- clnt browser , attach data strm.
API of HttpServletResponse
public PrintWriter getWriter()
PW --char , buf , o/p strm conn from server to clnt for sending resp

3. Send dyn contents.
PW API -- print/write
4. close PW

After creating servlet class -- supply deployment instrs to WC
2 ways.
1. Via annotation -- run time anno.
pkg -- javax.servlet.annotation
@WebServlet("/hi")
public class HelloServlet ....{...}
class level anno.

Who -- WC
When -- @ dep time of web app
Meaning ---If clnt send request ending in url-pattern /hi , use HelloServlet class for its servicing.

What is HTTP?

HTTP stands for Hyper Text Transfer Protocol which is an application level protocol for transferring textual data between client and server.

HTTP is stateless protocol. can't maintain the sessions throughout.

Web Client such as web browser sends the request to server, server responds and sends the requested data (such as a HTML document) or returns error code and closes the connection. Once the response is returned, server doesn't remember anything about the client.(even the very next request)

Http request methods and Headers

eg : URL -- http://www.server.com:8080/bank/login.html
Whenever the client sends the request to server for any resource such as a HTML document, it specifies
1. HTTP method (get/post/put/delete....)
2. Request URI (eg : /bank/login.html)
3. Protocol version
4. Optional Header information(eg : accept , cookies)

After the client sends the request, server processes the request and sends the response.

HTTP Response contains

1. Response status information(eg : 404/200)
2. Response headers (eg : cookies/resp cont type
eg : text/html, application/json image/gif.... /cont length)
3. Response data.

Following is an example of HTTP request which uses GET request method to ask for a HTML document named tutorial.html using HTTP protocol version 1.1

```
GET /tutorial.html HTTP/1.1
```

Following is the example of request header, which provides additional information about the request.

```
User-Agent: Mozilla/4.0 (compatible; MSIE 4.0; Windows 95)
Accept: image/gif, image/jpeg, text/*, */*
eg : text/html, text/xml,application/json
```

Above header specifies the information about the client software and what MIME(Multi-purpose Internet Mail Extension) type the client accepts in response, using User-Agent and Accept headers.

Http request methods

The request method indicates to the server what action to perform on the resource identified by the request-URI.

HTTP 1.1 specifies these request methods:
GET, POST , HEAD,OPTIONS, PUT, TRACE, DELETE, CONNECT.
Servlets can process any of the above requests.

But GET and POST methods are most commonly used.

The GET request method -- safe(doesn't change state of the resource) & idempotent (repeated reqs do not have any further side effects after the first request)

The GET request is used typically for getting static information from the server such as HTML document, images, and CSS files.
It can be used to retrieve dynamic information also by appending query string at the end of request URI.

Query String

Query string is used to pass additional request parameters along with the request.

Query string format

URL?name1=value1&name2=value&name3=value3 ….

eg -- http://www.abc.com/test/login.jsp?userid=10&name=abc&age=25

The POST request method

The POST request method is typically used to access OR create new dynamic resources.

POST request is used to
1. send the large amount of data to the server.
2. upload files to servers
3. send serializable Java objects or raw bytes.

GET Vs POST

1.
GET request sends the request parameters as query string appended at the end of the request URL, whereas POST request sends the request parameters as part of the HTTP request body.

2.Unlike GET, POST request sends all the data as part of the HTTP request body, so it doesn't appear in browser address bar and cannot be bookmarked.

3. GET -- non-secure, POST -secure
GET -- idempotent
POST---non-idempotent

Typically use GET -- to retrieve stuff from server.
use POST -- for changing some state on server(something like update)

4. Some web servers limit the length of the URL.
So if in GET request -- too many parameters are appended in query string and URL exceeds this length, some web servers might not accept the request.
For POST -- no such limitations.

Safe HTTP methods
HTTP methods are considered safe if they do not alter the server state.
So safe methods can only be used for read-only operations.

eg : GET, HEAD, OPTIONS and TRACE.

In practice it is often not possible to implement safe methods in a way they do not alter any server state.

eg : a GET request might create log or update statistic values or trigger a cache refresh on the server.

Idempotent HTTP methods

Idempotency means that multiple identical requests will have the same outcome. So it does not matter if a request is sent once or multiple times.

The following HTTP methods are idempotent: GET, HEAD, OPTIONS, TRACE, PUT and DELETE.

All safe HTTP methods are idempotent but PUT and DELETE are idempotent but not safe.

Note that idempotency does not mean that the server has to respond in the same way on each request.

eg : If you delete emp details by an id using DELETE request:
1st time , it will succeed(server will send resp status code 200) then next time onwards ,since emp details are already deleted , server will send resp code 404 (indicating resource not found!)

HTTP method overview

The following table summarizes which HTTP methods are safe and idempotent:

HTTP Method	Safe	Idempotent
GET	Yes	Yes
HEAD	Yes	Yes
OPTIONS		Yes Yes
TRACE	Yes	Yes
PUT	No	Yes
DELETE		No Yes
POST	No	No
PATCH	No	No

Content Type

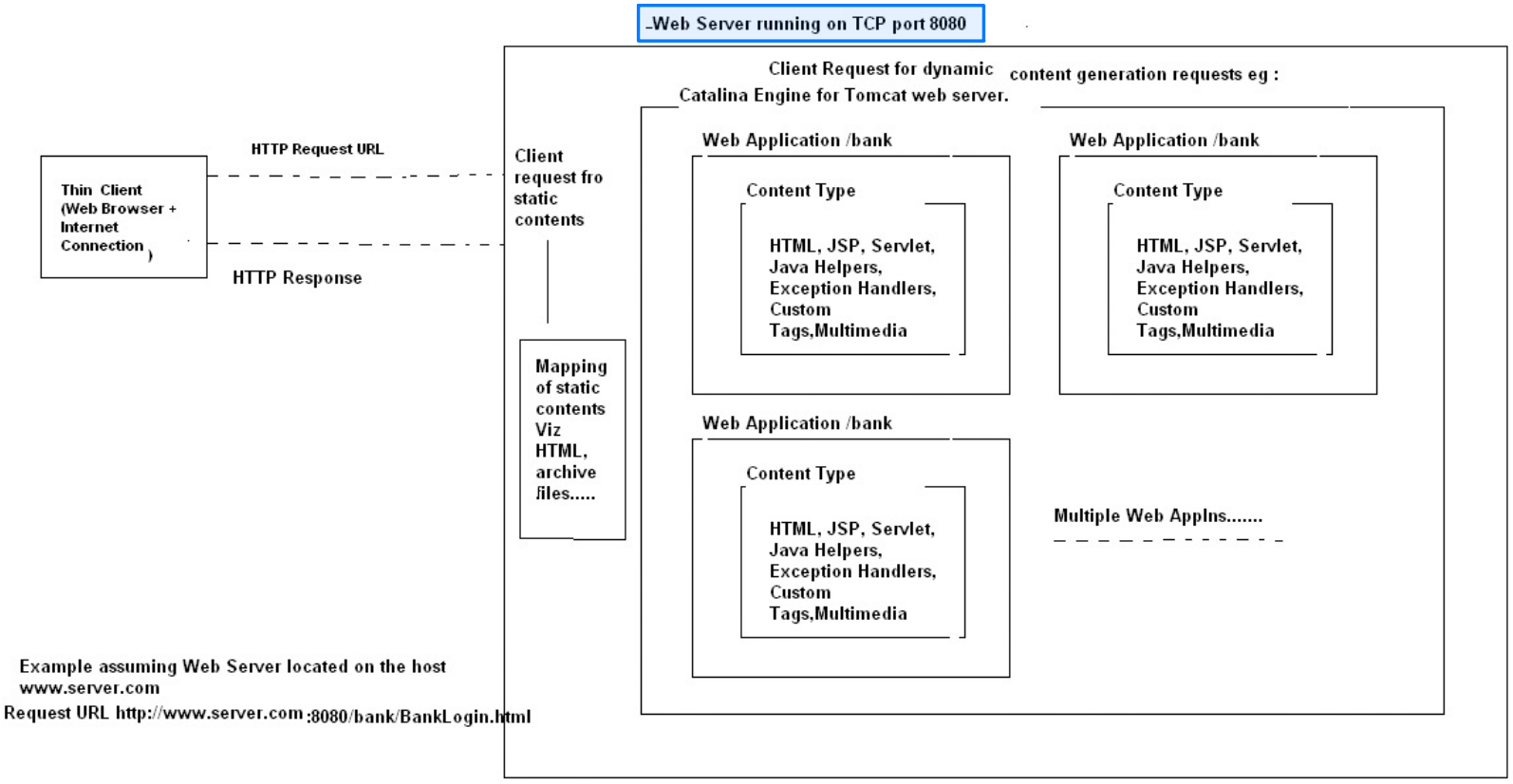
Content Type is also known as MIME (Multipurpose internet Mail Extension) Type. It is a HTTP header that provides the description about what are you sending to the browser.

MIME is an internet standard that is used for extending the limited capabilities of email by allowing the insertion of sounds, images and text in a message.

List of Content Types

There are many content types. The commonly used content types are given below:

```
text/html
text/plain
application/msword
application/json

application/jar
application/pdf
mages/jpeg
images/png
images/gif
audio/mp3
video/mp4
```


Web Server running on TCP port 8080
Client Request for dynamic content generation requests eg :
Catalina Engine for Tomcat web server.
HTTP Request URL
Thin Client (Web Browser + Internet Connection)
HTTP Response
Client request fro static contents
Mapping of static contents Viz HTML, archive files.....
Web Application /bank
Content Type
HTML, JSP, Servlet, Java Helpers, Exception Handlers, Custom Tags,Multimedia
Web Application /bank
Content Type
HTML, JSP, Servlet, Java Helpers, Exception Handlers, Custom Tags,Multimedia
Web Application /bank
Content Type
HTML, JSP, Servlet, Java Helpers, Exception Handlers, Custom Tags,Multimedia
Multiple Web Applns.....
Example assuming Web Server located on the host www.server.com
Request URL http://www.server.com :8080/bank/BankLogin.html

A web server is the combination of computer and the program installed on it.

Web server interacts with the client through a web browser. It delivers the web pages to the client and to an application by using the web browser and the HTTP protocols respectively.

We can also define the web server as the package of large number of programs installed on a computer connected to Internet or intranet for downloading the requested files using File Transfer Protocol, serving e-mail and building and publishing web pages.

A web server works on a client server model. A computer connected to the Internet or intranet must have a server program. While talking about Java language then a web server is a server that is used to support the web component like the Servlet and JSP. Note that the web server does not support to EJB (business logic component) component.

A computer connected to the Internet for providing the services to a small company or a departmental store may contain the HTTP server (to access and store the web pages and files), SMTP server (to support mail services), FTP server (for files downloading) and NNTP server (for newsgroup). The computer containing all the above servers is called the web server.

Internet service providers and large companies may have all the servers like HTTP server, SMTP server, FTP server and many more on separate machines. In case of Java, a web server can be defined as the server that only supports to the web component like servlet and jsp. Notice that it does not support to the business component like EJB.

The term web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver web content that can be accessed through the Internet.

The most common use of web servers is to host websites, but there are other uses such as gaming, data storage or running enterprise applications.
The primary function of a web server is to deliver web pages to clients. The communication between client and server takes place using the Hypertext Transfer Protocol (HTTP). Pages delivered are most frequently HTML documents, which may include images, style sheets and scripts in addition to text content.

A user agent, commonly a web browser or web crawler, initiates communication by making a request for a specific resource using HTTP and the server responds with the content of that resource or an error message if unable to do so. The resource is typically a real file on the server's secondary storage, but this is not necessarily the case and depends on how the web server is implemented.

While the primary function is to serve content, a full implementation of HTTP also includes ways of receiving content from clients. This feature is used for submitting web forms, including uploading of files.

Many generic web servers also support server-side scripting using Active Server Pages (ASP), PHP, or other scripting languages. This means that

the behaviour of the web server can be scripted in separate files, while the actual server software remains unchanged. Usually, this function is used to create HTML documents dynamically ("on-the-fly") as opposed to returning static documents. The former is primarily used for retrieving and/or modifying information from databases. The latter is typically much faster and more easily cached but cannot deliver dynamic content.

DAY 2

Today's Topics
Revision of concepts
Revision of hands on
Continue to : Servlet Programming
Servlet JDBC Integration
Page Navigation
Session Management

Revision

What is Java EE/Jakarta EE ? Specification for prim services , necessary for building enterprise apps.

eg of prim services : Servlet specs , JSP, Conn pooling , security ,JPA , Java Mail , web sevices....

Why J2EE 1.8/Java EE 8 /Jakarata EE 8
1. Java EE server independent
2. Supports variety of clnts -- thin clnts/ thick clnt /smart clnts
3. Prim services are imple in rdy manner : by web / app server --so that prog can focus only B.L

Who provides Specs of prim services in J2EE / Java EE : Oracl/Sun / Eclipse (Jakarta EE)
Implementation : Server vendors

Popular vendors & their products
(Apache : Tomcat / Tomee, Oracle --Glassfish , Jboss , weblogic....)

Request response flow(Layers)
URL --- http://www.abc.com:9090/day2.1

URL : Uniform resource locator

Revise request response flow
http : scheme(protocol : app layer)

www.abc.com : DNS qualified host name (IP adr) ---IP layer resolves port no --req reaches host

9090 : port no --resolved by TCP -- req reaches web server

web server --> chks path(URI) --->if itcontains any dyn web app --delegates it to WC

/day2.1 : ctx path (web app name)

WC --routes request to the specific web app
web.xml --- Dep descriptor meant for WC per web app

what are contents ? -- XML tags

welcome-file : index.html (Location : / : root of web =>webapp) => public

Resp : SC 200 | Header/s | body : index page ---> clnt side

What is WC ? web container
Server side JVM
Manages web apps.
Provides run time env for server side dynamic web comps(servlets, JSP,filter)

1. Server side JVM residing within web server.
Its run-time environment for dynamic web components(Servlet & JSP,Filter) .
Jobs ---
1. Creating Http Request & Http response objects
2. Controlling life-cycle of dyn web comps (manages life cycle of servlet,JSP,Filters)
3. Giving ready-made support for services --- Naming,security,Conn pooling .
4. Handling concurrent request from multiple clients .
5. Managing session tracking...

2. What is web.xml? --- Deployment descriptor one per web appln
popualated by -- prog (with help IDE)
who reads it -- WC
when ---once at the web app deployment time
what does it consist of --- XML tags based instructions meant for WC

Mandatory till Java EE 1.5 , optional later.

J2EE compliant folder structure of dynamic web application
refer to a diagram.("day1_data\day1_help\web programming related diags\j2ee compliant web appln folder structure.png")

3. Why servlets??? Adds dyn nature

Job list
1. Request processing
2. B.L
3. Dynamic response generation
4. Data access logic(DAO class --managing DAO layer)
5. Page navigation

What is a servlet ?
-- Java class (with NO main method) -- represents dynamic web component - whose life cycle will be managed by WC(web container : server side JVM)

life cycle methods --- init , service,destroy

Servlet API details --refer to diag servlet-api.png (from above folder) : IMPORTANT
<Tomcat_Home>\lib --- servlet-api.jar => specs only !
imple left to vendors --catalina.jar => imple classes

API
javax.servlet.Servlet i/f ---- init , service,destroy
imple class : javax.servlet.GenericServlet --abstract
abstract method : service overriden by do GET | POST.
concrete : init ,destroy
sub class -- javax.servlet.http.HttpServlet : abstract class , NO abstract methods

```
public void service(ServletRequest rq ,ServletResponse rs) throws ServletExc, IOExc
--> protected void service(HttpServletRequest rq ,HttpServletResponse rs) throws
ServletExc, IOExc
--> dispatching
get --> doGet
post -->  doPost ...
```

Steps of creating the servlet

```
public class MyServlet extends HttpServlet
{
   @Override
   public void doGet(HttpServletRequest rq ,HttpServletResponse rs) throws
ServletExc, IOExc
{
   rs.setContentType("text/html");
   try(PW pw=rs.getWriter())
   {
     pw.print(.....);
   } //pw.close -->pw.flush--> committing the resp

}
}
```

```
Deployment of the servlet
@WebServlet(value="/hello",loadOnStartup=1)
.....
WC creates a map : Key -- /hello
value : F.Q servlet cls name

OR
XML Tags


For HTTP response status codes :
refer : https://www.restapitutorial.com/httpstatuscodes.html

For  primer in  TCP/IP in web programming
https://www.techtarget.com/searchnetworking/definition/TCP-IP#:~:text=TCP%2FIP
%20stands%20for%20Transmission,(an%20intranet%20or%20extranet).

How does WC store the mapping info of the servlet ???
In the map created @ web app dep time
key :
value :

Trace Servlet Life cycle

URL request after Client clicks on the link : "Invoke A Servlet"
http://localhost:8080/day2.1/hello

WC -- chks for the key(/hello)  ---found --1st request ---WC starts life cycle
Key (URL pattern) not found --- HTTP 404

class loading ---singleton instance ---init => init seq

init --success ---???

fail --ServletExc --- WC ABORTS life cycle
```

At the end :
public void destroy() : inherited from G.S --> servlet instance is marked for GC
Triggers : server shut down / reload (re deploy) /un deploy

Deployment of the servlet
1. Via annotation
eg : @WebServlet(value="/validate")
public class LoginServlet extends H.S {....}
Map :
key -- /validate
value -- F.Q servlet cls name
URL : http://host:port/day1.1/validate?....

OR

2. Using XML tags
How to deploy a servlet w/o annotations --- via XML tags
web.xml

```
<servlet>
 <servlet-name>abc</servlet-name>
<servlet-class>pages.SecondServlet</servlet-class>
</servlet>
<servlet-mapping>
 <servlet-name>abc</servlet-name>
 <url-pattern>/test2</url-pattern>
</servlet-mapping>
```

WC : map
key : /test2
value : pages.SecondServlet

eg URL --http://host:port/test_web/hello

At the time of web app deployment ---WC tries to populate map of url patterns , from XML tags (from web.xml). Later ---it will check for @WebServlet annotation

Objective 3 .
What is default loading policy of WC for servlets ? lazy (i.e WC will start life cycle of the servlet :only after clnt sends the 1st request to the servlet)

Can you change it to eager ? Yes

Use Case : Typically for time consuming(heavy weight) inits.
eg : setting up DB conn, setting up spring frmwork....
HOW ?
@WebServlet (value="/test",loadOnStartup=1)
.....
OR
xml tag :
<load-on-startup>1</....>

How to read request params sent from the clnt in the servlet ?

javax.servlet.ServletRequest i/f methods
1. public String getParameter(String paramName)

2. public String[] getParameterValues(String paramName)

~~Objective 3 : Accept different type of i/ps from user , in HTML form.Write a servlet to display request parameters. (lab work !)~~

~~Objective : Servlet --JDBC Integration~~
~~0. JDBC JAR : <WEB-INF>/lib OR <tomcat> / lib~~
~~Layers involved : Servlet --DAO (Uses DBUtils) --POJO -- DB~~

~~Reference case study : Online Voting~~
~~Design~~
~~1. Table : users~~
~~2. POJO : User~~
~~3. Edit DBUtils :~~
~~3.1 open fixed DB connection ,~~
~~3.2 close~~
~~3.3 get connection~~

~~4. DAO : IUserDao n implementation class~~

~~6. welcome page : login.html(post) --> LoginServlet~~

~~7. --> LoginServlet~~
~~7.1 --init , destroy, doGet/doPost~~

Centralized err handling in Servlets
How ?

```
@Override
public void init() throws ServletException
{
try {
  open connection
  create dao inst
} catch(Exception e)
{
//re throw the exception back to WC , so that WC doesn't continue to service.
  throw new ServletException("err in init" +getClass().getName(),e);
}
}
```

API of javax.servlet.ServletException
Constructor :
public ServletException(String message,Throwable rootCause)

Objective :
Complete login--logout flow in current app
(login.html -- LoginServlet --successful login --check the role
If admin --redirect to admin servlet
If voter , who has not yet voted -- redirect to candidate list page
If voter , who has already voted -- redirect to candidate list page

Page Navigation Techniques

Page Navigation=Taking user from 1 page to another page.

2 Ways

1. Client Pull

Taking the client to the next page in the NEXT request (coming all the way from client)

1.1 User takes some action --eg : clicking on a button or link & then client browser generates new URL to take user to the next page.

1.2 Redirect Scenario

User doesn't take any action. Client browser automatically generates new URL to take user to the next page.(next page can be from same web appln , or diff web appln on same server or any web page on any srvr)

API of HttpServletResponse i/f
public void sendRedirect(String redirectURL) throws IOException
eg : For redirecting client from Servlet1 (/s1) to Servlet2 (/s2) , use response.sendRedirect("s2");

If the response already has been committed(pw flushed or closed) , this method throws(WC) an IllegalStateException.(since WC can't redirect the client after response is already committed)

2. Server Pull.

Taking the client to the next page in the SAME request.
Also known as resource chaining or request dispatching technique.
Client sends the request to the servlet / JSP. Same request can be chained to the next page for further servicing of the request.

Steps
1. Create Request Dispatcher object for wrapping the next page(resource --can be static or dynamic)
API of ServletRequest
javax.servlet.RequestDispatcher getRequestDispatcher(String path)

2.Forward scenario

API of RequestDispatcher
public void forward(ServletRequest rq,ServletResponse rs)

This method allows one servlet to do initial processing of a request and another resource to generate the response. (i.e division of responsibility)

Uncommitted output in the response buffer is automatically cleared before the forward.

If the response already has been committed(pw flushed or closed) , this method throws an IllegalStateException.

Limitation --only last page in the chain can generate dynamic response.

3. Include scenario

API of RequestDispatcher
public void include(ServletRequest rq,ServletResponse rs)

Includes the content of a resource @run time (servlet, JSP page, HTML file) in the response. -- server-side includes.

Limitation -- The included servlet/JSP cannot change the response status code or set headers; any attempt to make a change is ignored.

What is a Session?

Session is a conversional state between client and server and it can consists of multiple request and response between client and server. Since HTTP and Web Server both are stateless, the only way to maintain a session is when some unique information about the session is passed between server and client in every request and response.

HTTP protocol and Web Servers are stateless, what it means is that for web server every request is a new request to process and they cant identify if its coming from client that has been sending request previously.

But sometimes in web applications, we should know who the client is and process the request accordingly. For example, a shopping cart application should know who is sending the request to add an item and in which cart the item has to be added or who is sending checkout request so that it can charge the amount to correct client.

What is the need of session tracking?

1. To identify the clnt among multiple clnts
2. To remember the conversational state of the clnt(eg : list of the purchased books/ shopping cart/bank acct details/stocks) throughout current session

session = Represents duration or time interval
default session timeout for Tomcat =30minutes

Session Consists of all requests/resps coming from/ sent to SAME clnt from login to logout or till session expiration tmout.

There are several techniques for session tracking.
J2EE specific techniques :
1. Plain Cookie based scenario
2. HttpSession interface
3. HttpSession + URL rewriting

Techniques

1. Plain Cookie based scenario

What is a cookie?

Cookie is small amount of text data.
Created by -- server (servlet or JSP prog or WC) & downloaded (sent) to clnt browser---within response header
Cookie represents data shared across multiple dyn pages from the SAME web appln. (meant for the same client)

Steps :

1. Create cookie/s instance/s
javax.servlet.http.Cookie(String cName,String cVal)

2.Add the cookie/s to the resp hdr.
HttpServletResponse API :
void addCookie(Cookie c)

3. To retrieve the cookies :
HttpServletRequest :
Cookie[] getCookies()

4.Cookie class methods :
String getName()
String getValue()
void setMaxAge(int ageInSeconds)
def age =-1 ---> browser stores cookie in cache
=0 ---> clnt browser should delete cookie
>0 --- persistent cookie --to be stored on clnt's hard disk.

int getMaxAge()

Disadvantages of pure cookie based scenario
0. Web developer (servlet prog) has to manage cookies.
1. Cookies can handle only text data : storing Java obj or bin data difficult.
2. As no of cookies inc., it will result into increased net traffic.
3. In cookie based approach : entire state of the clnt is saved on the clnt side. If the clnt browser rejects the cookies: state will be lost : session tracking fails.

How to redirect client automatically to next page ? (in the NEXT request)
API of HttpServletResponse
public void sendRedirect(String redirectLoc)
eg : resp.sendRedirect("s2");

IMPORTANT :
WC -- throws
java.lang.IllegalStateException: Cannot call sendRedirect() after the response has been committed(eg : pw.flush(),pw.close()...)

Technique # 2 : Session tracking based on HttpSession API
In this technique :
Entire state of the client is not saved on client side , instead saved on the server side data structure (Http Sesion object) BUT the key to this Http Session object is STILL sent to client in form of a cookie.(cookie management is done by WC)

Servlet programmer can store/restore java objects directly under the session scope(API : setAttribute/getAttribute)

Above mentioned , disadvantages ---0, 1 & 2 are reomved.
BUT entire session tracking again fails , if cookies are disabled.

Steps for javax.servlet.http.HttpSession i/f based session tracking.

1. Get Http Session object from WC Web Container

API of HttpServletRequest ---
HttpSession getSession()
Meaning --- Servlet requests WC to either create n return a NEW HttpSession object(for new clnt) or ret the existing one from WC's heap for existing client.

HttpSession --- i/f from javax.servlet.http
In case of new client :
 HttpSession<String,Object> --empty map
String,Object ---- (entry)= attribute

OR
HttpSession getSession(boolean create)

2. : How to save data in HttpSession?(scope=entire session)
API of HttpSession i/f
public void setAttribute(String attrName,Object attrVal)
eg : hs.setAttribute("clnt_info",validatedCustomer);//no javac err
 attribute : server side object ---server side entry (key n value pair) --map

equivalent to map.put(k,v)
eg : hs.setAttribute("cart",l1);

3. For retrieving session data(getting attributes)
public Object getAttribute(String attrName) //key
eg : Customer cust=(Customer) hs.getAttribute("clnt_info");

4. To get session ID (value of the cookie whose name is jsessionid -- unique per client by WC)
String getId()

4.5 How to remove attribute from the session scope?
public void removeAttribute(String attrName)
eg : hs.removeAttribute("clnt_info");

5. How to invalidate session?
HttpSession API
public void invalidate()
(WC marks HS object on the server side for GC ---BUT cookie is NOT deleted from clnt browser)

6. HttpSession API
public boolean isNew()
Rets true for new client & false for existing client.

7.How to find all attr names from the session ?
public Enumeration<String> getAttributeNames()
--rets java.util.Enumeration of attr names.

8. Default session timeout value for Tomcat = 30 mins
How to change session tmout ?
HttpSession i/f method
public void setMaxInactiveInterval(int secs)
eg : hs.setMaxInactiveInterval(300); --for 5 mins .

OR via xml tags in web.xml
<session-config>
  <session-timeout>5</session-timeout> : unit : min
</session-config>

NOTE :
What is an attribute ?
attribute = server side object(entry/mapping=key value pair)
who creates server side attrs ? -- web developer (servlet or JSP prog)
Each attribute has --- attr name(String) & attr value (java.lang.Object)
Attributes can exist in one of 3 scopes --- req. scope,session scope or application scope
1. Meaning of req scoped attr = attribute is visible for current req.
2. Meaning of session scoped attr = attribute is visible for current session.
(shared across multiple reqs coming from SAME clnt)
3. Meaning of application scoped attr = attribute is visible for current web appln.
(shared across multiple reqs from ANY clnt BUT for the SAME web application)

Regarding javax.servlet.ServletContext (i/f)

1. Defined in javax.servlet package.
2. Who creates its instance -- WC
3. When -- @ Web application (=context) deployment time
NOTE : The ServletContext object is contained within the ServletConfig object, which the WC provides the servlet when the servlet is initialized.

4. How many instances ? --one per web application

5. Usages
5.1 Server side logging
API public void log(String mesg)
5.2 To create context(=application) scoped attributes
API public void setAttribute(String nm,Object val)
NOTE : Access them always in thread safe manner (using synchronized blocks)

5.3 To access global(scope=entire web application) parameters
How to add context scoped parameters ?

In web.xml
```
<context-param>
  <param-name>name</param-name>
      <param-value>value</param-value>
</context-param>
```
How to access these params in a Servlet ?
(can be accessed from init method onwards)

1. Get ServletContext
API of GenericServlet
ServletContext getServletContext() --method inherited from GenericServlet

2. ServletContext API
String getInitparameter(String paramName) : rets the param value.
eg : ctx param name : user_name value : abc
In the Servlet : getServletContext().getInitparameter("user_name") ---abc

5.4 Creating request dispatcher
H.W

Page Navigation Techniques (Taking thin clnt (user) from one page to other page.)

Client Pull — Min scope reqd=session scope.

In this tech. user is navigated to next page in NEW request

Clnt Pull I

BOTH (client --person & clnt Browser) is involved.

eg -- Button click,link .

Clnt Pull II

Client (as a person) is NOT involved BUT clnt browser is involved.
Re-direct scenarion.
API of HttpServletResponse
void sendRedirect(String redirectURL)

Server Pull

In this tech. user is navigated to next page in same request.

Resource chaining or request dispatching technique. --- NO round trip delay is involved since entire navigation takes place in server space.
API of ServletRequest
1. Get Request dispatcher object to chain resources(JSP,Servlet)
RequestDispatcher getRequestDispatcher(String dispatcherURL) 2. Use either forward or include scenario as appropriate.

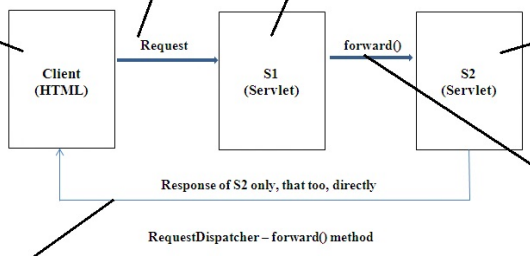
---

http://host:port/day4.1/validate

login form

rq1

LoginServlet : success :
req.setAttribute(nm,val); RD rd=req.getRD("category");
rd.forward(req,resp);

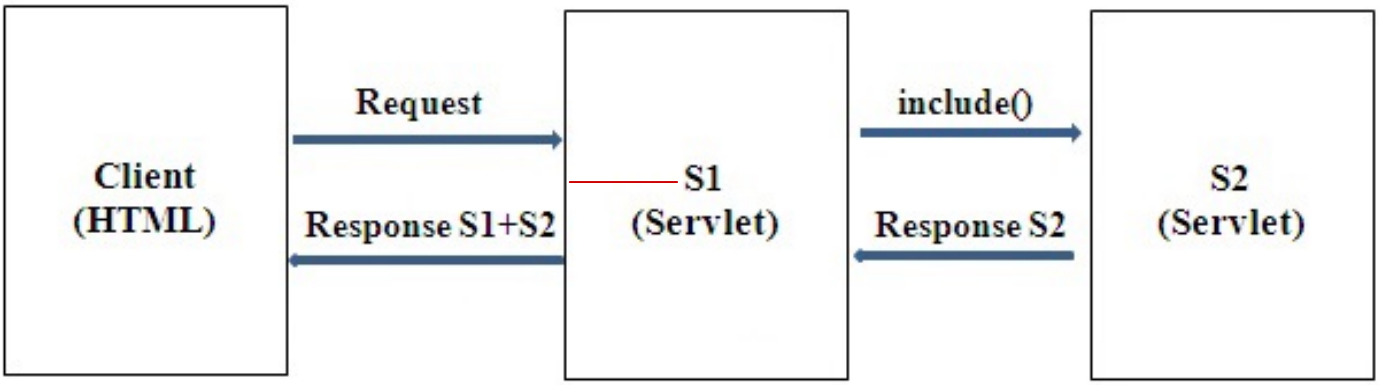


PW
req.getAttribute(nm) : val (obj)
pw.print(val);
pw.close();

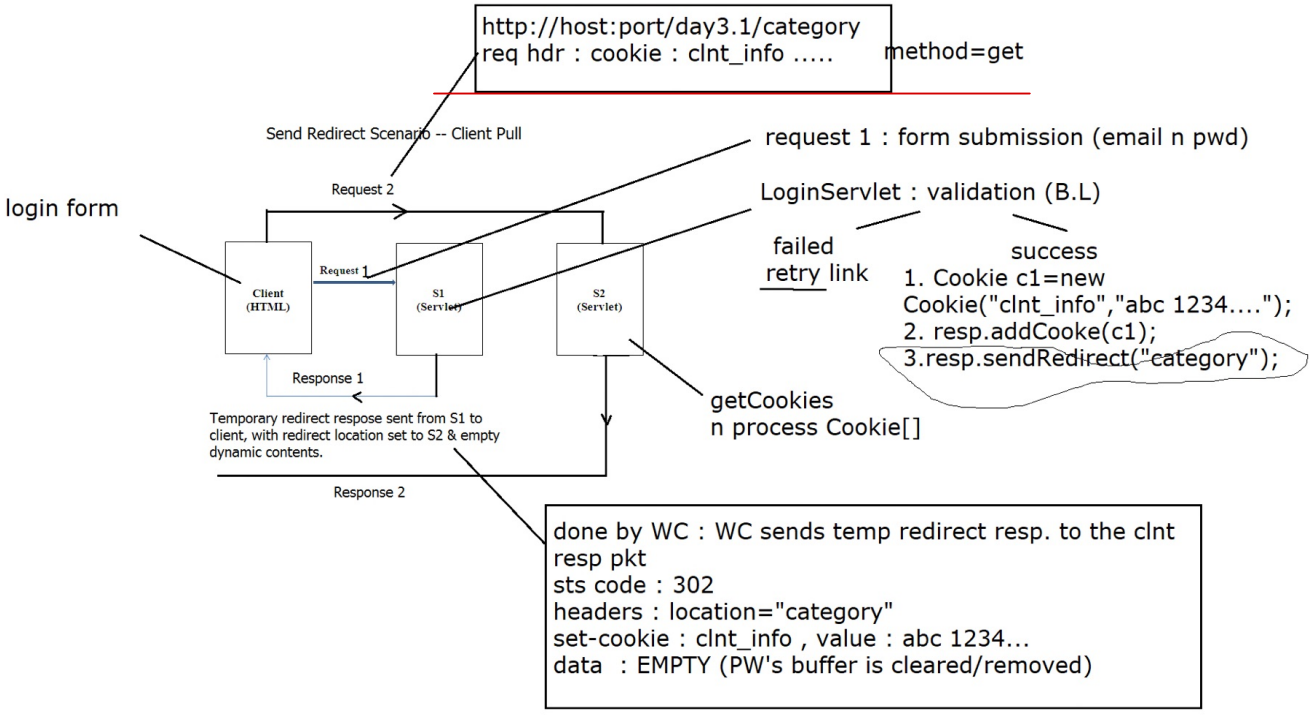
rq1 , rs1

rs1

what is min scope of attributes to be used in server pull : request
what is min scope of attributes to be used in client pull : session



RequestDispatcher – include() method



DAY 3

Revise Page Navigation
(Send redirect)
Revise Cookie based Session Tracking
Continuation of the case study
ServletConfig
Server Pull
Scopes of attributes in web programming

Revise
1. Which are the page navigation techniques
Clnt pull n server pull

What is common between them ? In either of the technique , clnt is taken to the next page

What is the difference ? In clnt pull --clnt is navigated to the next page in the NEXT req coming from the clnt example – web based email client
In server pull --clnt is navigated to the next page in the SAME req coming from the clnt Example – stock Market

2. What will WC do if you write

In Servlet1 : doPost method
response.sendRedirect("servlet2");
where /servlet2 is the url-pattern of the Servlet2 ? : eagerly inited
WC --discards resp buffer , sends temp redirect resp
SC 302 | Location : servlet2 | body : empty ---> clnt browswer
clnt browser send NEXT req ---> URL : http://host:port/day3/servlet2 ,method =GET
WC --> chks the map --> if the entry exists --> HttpServlet's public service -->
protected service -->doGet : servicing logic

3. What will happen ?
In Servlet1 : doPost method
pw.print("testing 12345");
response.sendRedirect("servlet2");

Which method has to be overridden in Servlet2 ? : doGet
clnt side : testing 12345 : DOES NOT appear !

4. What will happen ?
In Servlet1 : doPost method
pw.print("testing 12345");
pw.flush();//sending the resp to the clnt
response.sendRedirect("servlet2");
clnt browser : testing 12345
server console : java.lang.IllegalStateExc : Can not call sendRedirect after committing the resp.

5. Given
In Servlet1 : doPost method
String email=request.getParamter("em");// abc@gmail.com
pw.print("from 1st page "+email);
response.sendRedirect("servlet2");

In Servlet2 , doGet method
pw.print("from 2nd page "+request.getParamter("em"));//null
What will be the resp seen on clnt browser ? from 2nd page null
Why ? : HTTP stateless inherently

Session Tracking

```
# Enter Session Tracking
Refer : day2_data\day2_help\session tracking\regarding session tracking.txt"
1. Cookie based session Tracking
Steps
1.1Create a cookie
Cookie c1=new Cookie("user_info",user.toString());
1.2Add a cookie in resp header
resp.addCookie(c1);
1.3 resp.sendRedirect("admin");
WC --> discard buffer , send temp redirect resp
SC 302 | Location=admin , Set-cookie: user_info: user.toString() |body empty
---> clnt browser --chk privcay settings --if accepted
age (expiry) -- -1 => clnt browser will add cookie in cache.

1.4 Clnt browser --> request --header : Cookie : user_info: user.toString()

1.5 How to get cookies ?
Cooke[] cookies=request.getCookies();
if(cookies != null) => cookies are present in hdr
for (Cookie c : cookies)
 if(c.getName().equals("user_info"))
  pw.print("Clnt details "+c.getValue());


# 2. HttpSession based session tracking
  steps
2.1 get session from WC
2.2 save details under session scope
2.3 gget details
2.4 Discard session : invalidate
```

HttpSession Internals
Prog invokes --- HttpSession hs=request.getSession();

WC checks -- for old user (alrdy exising --participated in session tracking) or new user. (A)

old user

rets alrdy existing HS obj ref to the caller.

New User

1. Empty HttpSession Object is created on heap.
eg -- HttpSession hs1=new HttpSession();
2. WC creates 1 cookie.
Cookie c1=new Cookie("JSESSIONID","abc123456"); --- value is generated uniquely per clnt.
3. WC adds cookie to resp hdr
resp.addCookie(c1);
4. Rets hs1 to the caller(P)

Prog invokes --- hs.setAttribute("user_info",ref);

WC simply adds entry(key -- attr name ---user_info, value --attr value --user pojo ref. ---clnt's state. --- into HttpSession object(hm.put(k,v))

(A)

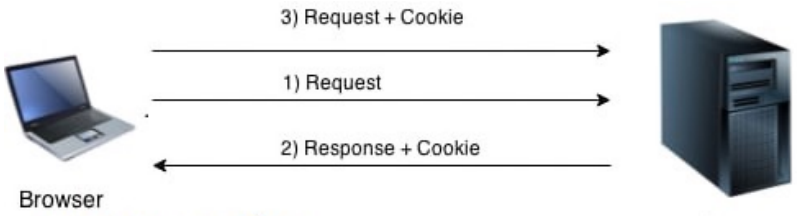
How does WC know ---old user or new user.

WC invokes
Cookie[] cookies=req.getCookies();

null ---NEW

WC invokes
for(Cookie c : cookies)
if (c.getName().equals("JSESSIONID")) ---
c.getValue() --- "abc123456" --
From this value --- WC retrieves HS object associated with cookie value --- HS1

OLD user -- alrdy visited clnt.



1. Create a Cookie
API --javax.servlet.http.Cookie(String nm,String val)
eg : Cookie c1=new Cookie("user_dtls",u.toString());
2. Add cookie/s to resp header
API HttpServletResponse
public void addCookie(Cookie c1);

3. Client browser chks privacy settings
Cookies disabled -- can't remember clnt
Cookies enabled --- browser chks cookie age (expiry)
-1 -- default -- cookie is stored in browser cache.
0 -- delete the cookie
> 0 --save it on clnt's hard disk (persistent)
4. When clnt send a request to same web appln
Clnt browser --sends --req +cookies(hdr)

5. How to read cookie data ?
API of HttpServletRequest
Cookie[] getCookies()

Http Session

Cookie

atributes --- server side objs (nm(string), value(Object)) --- can be added to different scopes. --- page|request|session|application (typically created by servlet or JSP programmer)

Invoker	Common Block API	significance
ServletRequest	void setAttribute(String nm,Object val) Object getAttribute(String nm) void removeAttribute(String nm) Enumeration<String> getAtrributeNames()	scope=current request only, GCed after current resp is comitted.
HttpSession		scope=cuurent session;shared across all web pages from SAME web appln for the same clnt.
ServletContext		scope=entire web appln;shared across all web pages from SAME web appln for any clnt. inherently thrd un-safe MUST be accessed in thrd safe (synch block) manner

4. Is current web app DB independent or DB specific : DB specific
4.1 DBUtils
openConnection(url,userName,pwd)
4.2 LoginServlet --> DBUtils : openConnection
modify LoginServlet : init

How to make it DB independent ?
By parameterizing the servlet(using ServletConfig)

Steps
4.1 web.xml
add 3 init params in a servlet
(db config : db URL, user name ,password)
URL : jdbc:mysql://localhost:3306/advjava?
createDatabaseIfNotExist=true&useSSL=false&allowPublicKeyRetrieval=true";

4.2 In Login Servlet
Read init params : from ServletConfig

4.3 pass these params to DBUtils
openConnection : to actually open a new connection.

5. Page Navigation with server pull.
New Scenario
Student Admission Form (Accept student details : first name , last name , entrance test score,course,admission status)
--> submit this info to Student Admission Servlet. Validate if student has min score for a specific course .
Accordingly give admission or deny admission.
Navigate the client to result page n display suitable message.
(refer to flow diagram)

==========
Regarding SERVLET CONFIG

A servlet specific configuration object created by a servlet container to pass

information to a servlet during initialization.

1. Represents Servlet specific configuration.
Defined in javax.servlet.ServletConfig -- interface.

2. Who creates its instance ?
Web container(WC)
3. When ?
After WC creates servlet instance(via def constr), ServletConfig instance is created & then it invokes init() method of the servlet.
4. Usage
To store servlet specific init parameters.
(i.e the init-param is accessible to one servlet only or you can say that the init-param data is private for a particular servlet.)

5. Where to add servlet specific init parameters?
Can be added either in web.xml or @WebServlet annotation.

XML Tags

```
<servlet>
    <servlet-name>init</servlet-name>
    <servlet-class>ex.TestInitParam</servlet-class>
    <init-param>
      <param-name>name</param-name>
      <param-value>value</param-value>
    </init-param>
</servlet>
<servlet-mapping>
<servlet-name>init</servlet-name>
<url-pattern>/test_init</url-pattern>
</servlet-mapping>
```

6. How to access servlet specific init params from a servlet ?
6.1 Override init() method
6.2 Get ServletConfig
Method of Servlet i/f
public ServletConfig getServletConfig()
6.3 Get the init params from ServletConfig
Method of ServletConfig i/f
String getInitparameter(String paramName) : rets the param value.

DAY 04

Assignment status

Today's topics
Revision of HttpSession with API steps n diagram
Server pull
Scopes of attributes in web programming
Complete Servlet Life cycle with ServletConfig n Concurrency
Web app listeners

How to complete Admin Flow : Lab work

Enter JSP : if time permits !

Solve
eg : remote web server IP address is --ip1
In web app(/day4.1) --- /s1(is a servlet) ---
Creates a cookie --- name --"clnt_info" , value --"details1234" & sends it to a clnt.

clnt IP adr-- ip2

Q Will Clnt browser of ip2 send the cookie in request header ?

1. clnt sends the URL --- http://ip3:8080/day4.1 : NO : since it's different IP addr
2. clnt sends the URL --- http://ip1:8080/day2/....NO : since it's different web app
3. clnt sends the URL --- http://ip1:8080/day4.1/s2 : YES
4. clnt sends the URL --- http://ip1:8080/day4.1/s10 : YES

What is the Default behaviour : cookies are by default sent to the SAME web app of origin

Can it be modified by Cookie class methods??? YES

Which ones ?

setPath : allows the cookie/s to be shared across multiple web apps hosted on the same server.
setDomain : allows the cookie/s to be shared across multiple hosts from the same domain

Answer this
What is ServletConfig ? i/f : javax.servlet.ServletConfig
Who creates it's instance ? WC
When ? After instance creation(via def ctor) n before init
How many instances ? 1 per servlet
Use case ? for parameterizing the servlet
eg : passing DB config settings

How did we make the web app DB independent ?
1. DB props : url , user name , pwd
Add them in web.xml
<servlet>
 servlet-name , servlet-class
<init-param>
param-name: url , param-value

```
</init-param>
<load-on-startup>....

2. LoginServlet
init
ServletConfig config=getServletConfig();
String dbURL=config.getInitParameter("url");
....
3. openConnection(url,name,pwd);

Revise :
HttpSession i/f based session management
(Refer to code , API n draw the diagram)

1. HttpServletRequest getSession methods (refer to a diag : "day4-data\\day4_help\
diagrams\getSession details.png")
1.1 HttpSession getSession() : WC will create a new Session if it doesn't exist OR
rets the existing one
1.2 HttpSession getSession(boolean create)


2. Session Tracking with HttpSession i/f
2.1 Get HttpSession from WC
HttpSession session=request.getSession();
What will WC do internally ?

2.2 How to save validated user details under session scope?
HttpSession API
public void setAttribute(String attrName,Object attrValue)

2.3 How to retrieve ?
public Object getAttribute(String attrName)

2.4 How to remove attribute from the session scope ?
public void removeAttribute(String attrName)

2.5 How to get all attr names bound to HttpSession ?
Enumeration<String> getAttributeNames()

2.6 To send to clnt : session creation date/time
pw.print("Session Created on "+new Date(session.getCreationTime()));

2.7 How to invalidate session ?
session.invalidate(); => WC marks HS object for GC , unbinds all session scoped
attributes.


3. Page Navigation with server pull.
New Scenario
Student Admission Form (Accept student details : first name , last name , entrance
test score,course,admission status)
--> submit this info to Student Admission Servlet. Validate if student has min
score for a specific course .
Accordingly give admission or deny admission.
Navigate the client to result page n display suitable message.
(refer to flow diagram)
```

Request Dispatching technique (Server Pull)

refer : readme n diagrams

Student Admission
1. Student POJO
firstName,lastName, --string
course : enum
score : int
admissionStatus : boolean

3.1RD's forward scenario

3.2 RD's include scenario

4. Complete Servlet Life cycle (including thread pool)
Executor Framework (used by WC to support concurrent handling of multiple client requests)
CGI Vs Servlets :(reading H.W)

5. Web app listeners
Add a listener in the existing web app n test it.
What is a Servlet Listener(or web application listener)?

During the lifetime of a typical web application, a number of events take place.
eg : requests are created or destroyed.
sessions are created & destroyed
Contexts(web apps) are created & destroyed.
request or session or context attributes are added, removed, or modified etc.

The Servlet API provides a number of listener interfaces that one can implement in order to react to these events.

eg : Event Listener i/f
1. ServletRequestListener
2. HttpSessionListener
3. ServletContextListener
....
Event Handling Steps
1. Create a class , implementing from Listener i/f.
2. Register it with WC
2.1 @WebListener annotation(class level)
OR
2.2 XML tags in web.xml
<listener>
 <listener-class>F.Q cls name of listener</listener-class>
</listener>

Servlet Life cycle in detail (IMPORTANT) managed by WC

@ Web server start up : WC starts up -- creates thread pool (based upon exec frmwork) core size n max size of the thrd pool -- will be as per the default OR can be configured from xml config files(eg : server.xml)

@ web app dep time :
1. WC creates the instance of ServletContext per web app
ServletContext : (i/f) --> imple class instance (i/f ---> servlet-api.jar) n imple classes : Tomcat --catalina.jar
Represents : ENTIRE web app
2. Prepares servlet url map
Key : url-pattern
Value : F.Q servlet cls name
eg : @WebServlet("/test") public class MyServlet extends HttpServlet {....}
Entry : key : /test n value : pages.MyServlet

WC checks for load-on-startup

load-on-startup > 0 => Eager init => starts servlet life cycle @ web app dep time : init seq begins
1. Loads servlet class (server side method area)
2. Creates servlet instance : server side heap using def ctor.
3. WC creates ServletConfig instance , populates it with init-params .
4. Invokes public void init() throws ServletException (WC implicitly passes ServletConfig obj to the init)

load-on-startup : not specified(=-1) => lazy init.
WC waits for 1st request from any clnt --then start the life cycle(exec SAME init seq)

Failure (i.e throws ServleExc)
WC aborts servlet life cycle

success :
continues with the life cycle

req processing or service phase

clnts sending requests --->

WC simply pools out existing idle thread ---> run --> invokes public service(rq,rs)---> proted service(HttpServletReq,HttpServletResp) --> dispatches req --> doXXX --> {servicing logic}
doXXX rets --> service rets --> run ---> pooled out thrd simply rets to the pool (so that SAME thread can service further reqs)

WC invokes public void destroy() (mainly used for cleaning up of resources)
Then WC marks servlet instance for GC --ending servlet life cycle

Triggers for destroy
1. server shut down
OR
2. web app re loaded
OR
3. web app un deployed

sendRedirect Vs Request Dispatcher forward

common --both of above are page navigation techniques.

1. Navigates the user to next page in NEXT request

2.Client browser is involved (round trip delay involved)

3. Min scope required for sharing attributes = session scope

4. URL changes to the next(redirected) page.

5. Usage --
To navigate client outside current web application.
Double submit guard.

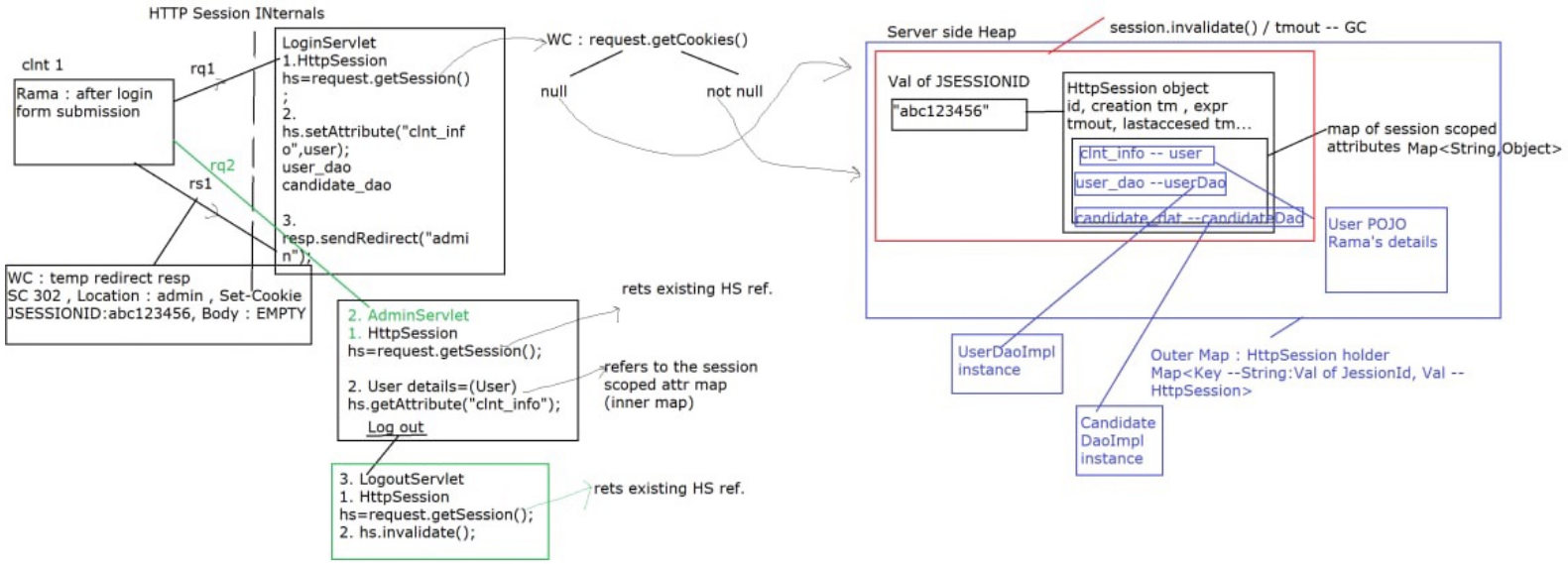
1. Navigates the user to next page in SAME request.

2. Client browser is NOT involved (no round trip delay)

3. Min scope required for sharing attributes =request scope

4.URL seen by the client is that of 1st page(in the chain of resources)

5. Usage --
In MVC applications, to navigate the client from controller to the view layer.
(Used as default navigation by all major MVC frmworks)



JSP Directives

commands/messages for JSP Engine(=JSP container=WC) -- to be used @Translation time.

Syntax ---
<%@ Directive name attrList %>

1. page directive
--- all commands applicable to current page only.
Syntax
<%@ page import="comma separated list of pkgs" contentType="text/html" %>
eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>
Imp page directive attributes
1. import --- comma separated list of pkgs
2. session --- boolean attribute. default=true.
   To disable session tracking, spectify session="false"
3. errorPage="URI of err handling page" --- tells WC to forward user to err handler page.
4. isErrorPage="true|false" def = false
If you enable this to true--- one can access 'exception' implicit object from this page.

This exception obj is stored under current page ---i.e under pageContext (type=javax.servlet.jsp.PageContext -- class which represents curnt JSP)
EL expresssion to display error mesg
${pageContext.exception.message}
-- evals to pageContext.getException().getMessage()

Additional EL syntax

EL syntax to be used in error handling pages

```
ERR causing URI :  ${pageContext.errorData.requestURI }<br/>
 ERR code :  ${pageContext.errorData.statusCode}<br/>
 ERR Mesg :  ${pageContext.exception.message} <br/>
 Throwable : ${pageContext.errorData.throwable}<br/>
 Throwable Root cause: ${pageContext.errorData.throwable.cause}
```

5. isThreadSafe="true|false" default=true. "true" is recommended
true=>informing WC--- JSP is already written in thrd -safe manner ---- DON'T apply thrd safety.

false=>informing WC --- apply thrd safety.

(NOT recommended) ---WC typically marks entire service(servlet scenario) or _jspService in JSP scenarion --- synchronized. --- this removes concurrent handling of multiple client request --so not recommended.
What is recommended? --- isThreadSafe=true(def.) --- identify critical section(i.e code prone to race condition among threads)--guard it in synchronized block.
eg ---Context scoped attrs are inherently thrd -un safe. So access them always from within synched block.

Equivalent step in Servlet
Servlet class can imple. tag i/f -- javax.servlet.SingleThreadModel(DEPRECATED) --
WC ensures only 1thread (representing clnt request) can invoke service method. --

NOT recommended.

2. include directive
<%@ include file="URI of the page to be included" %>
Via include directive ---- contents are included @ Translation time.--- indicates page scope(continuation of the same page).
Typically used -- for including static content (can be used to include dyn conts)
eg ---one.jsp
....<%@ include file="two.jsp" %>
two.jsp.....

JSP actions ---- commands/messages meant for WC
to be interpreted @ translation time & applied @ req. processing time.(run time)

Syntax ---standard actions --specifications are present in jsp-api.jar.
(implementations in jasper jar)

<jsp:actionName attribute list>Body of the tag/action
</jsp:actionName>

OR

<jsp:actionName attr list />

JSP Using Java beans(JB)
Why Java Beans
1. allows prog to seperate B.L in Javabeans(Req processing logic, Page navigation & resp generation will be still part of JSP)

Javabeans can store conversational state of clnt(Javabeans 's properties will reflect clnt state) + supplies Business logic methods.

2. simple sharing of JBS across multiple web pages---gives rise to re-usability.

3. Automatic translation between req. params & JB props(string--->primitive data types automatically done by WC)

What is JB?
1. packaged public Java class
It's actually an attribute automatically created by WC.(trigger : jsp:useBean) & WC will automatically store it under the specified scope
2. Must have def constr.(MUST in JSP using JB scenario)
3. Properties of JBs --- private, non-static , non-transient Data members --- equivalent to request params sent by clnt.(Prop names MUST match with req params for easy usage)
In proper words --- Java bean properties reflect the conversational state of the clnt.
4. per property -- if RW
naming conventions of JB
supply getter & setter.
Rules for setter (Java Bean Naming convention) : strict
public void setPropertyName(Type val)
Type -- prop type.
eg -- private double regAmount;

```
public void setRegAmount(double val)
{...}
Rules for getter
public Type getPropertyName()
Type -- prop type.
eg -- public double getRegAmount(){...}

5. Business Logic --- methods
public methods --- no other restrictions
-------------------------------------------------------------------------------
-----------------------------------------------

# Using Java Beans from JSP Via standard actions

1. <jsp:useBean id="BeanRef name" class="F.Q. Bean class name" scope="page|request|
session|application/>

default = page scope.

pre-requisite --- JB class exists under <WEB-INF>/classes.
 JB = server side obj (attribute), attr name --- bean id,attr val -- bean inst.,can
be added to any scope using scope atribute.

eg :
eg --- beans.Userbean
props --- email,pass
setters/getters
B.L mehod -- for validation

Usage ---
<jsp:useBean id="user" class="beans.UserBean" scope="session"/>

W.C invokes JB life-cycle
1. WC chks if specified Bean inst alrdy exists in specified scope
java api --- request.getAttribute("user")
---null=>JB doesn't exist
---loc/load/inst JB class
UserBean u1=new UserBean();
--add JB inst to the specified scope
java api -- request.setAttribute("user",u1);
--- not-null  -- WC continues....

2. JSP using JB action
2.1 <jsp:setProperty name="Bean ref Name" property="propName" value="propVal---
static/dyn" />
Usage--
<jsp:setProperty name="user" property="email"
value="a@b"/>
WC invokes --- session.getAttribute("user").setEmail("a@b");

<jsp:setProperty name="user" property="email"
value="<%= request.getParameter("f1") %>"/>

OR via EL
<jsp:setProperty name="user" property="email"
value="${param.f1}"/>

WC invokes ---
session.getAttribute("user").setEmail(request.getParameter("f1"));
```

2.2
<jsp:setProperty name="Bean ref Name" property="propName" param="rq. param name"/>

Usage eg --
<jsp:setProperty name="user" property="email" param="f1"/>

WC invokes ---
((Userbean)request.getAttribute("user")).setEmail(request.getParameter("f1"));

2.3
<jsp:setProperty name="Bean ref Name" property="*"/>

usage

<jsp:setProperty name="user" property="*"/>

eg -- If rq. param names are email & password(i.e matching with JB prop names) then
---matching setters(2) will get called

3. <jsp:getProperty name="Bean ref name" property="propName"/>
Usage --
<jsp:getProperty name="user" property="email"/>
WC ---
session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.

Better equivalent -- EL syntax
${sessionScope.user.email} ---
session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.

${requestScope.user.validUser.email}
request.getAttribute("user").getValidUser().getEmail()

${pageContext.exception.message}

4.JSP standard actions related to Request Dispatcher
RD's forward scenario
<jsp:forward page="dispatcher URI" />
eg : In one.jsp
<jsp:forward page="two.jsp"/>
WC invokes ---RequestDispatcher rd=reuqest.getRequestDispatcher("two.jsp");
rd.forward(request,response);

RD's include scenario
<jsp:include page="dispatcher URI" />

eg : In one.jsp
<jsp:include page="two.jsp"/>
WC invokes ---RD rd=reuqest.getRD("two.jsp");
rd.include(request,response);

Why JSTL ? JSP standard tag library

When JSP standard actions are in-sufficient to solve requirements ,
w/o writing scriptlets --- use additional standard actions --- supplied as JSTL
actions

JSP standard Tag Library
Has become standard part of J2EE specs from version 1.5 onwards.

---It's support exists in form of a JAR ---
1. jstl-1.2.jar
For using JSTL steps
1.Copy above JAR into your run-time classpath(copy jars either in <tomcat_home>/lib
OR <web-inf>/lib
2. Use taglib directive to import JSTL tag library into JSP pages.
tag=action
tag library=collection of tags
supplier = JSTL vendor(specification vendor=Sun, JAR vendor=Sun/any J2EE compliant
web/app server)
jstl.jar --- consists of Tag implementation classes
Tag libr-   TLD -- Tag library descriptor -- desc of tags -- how to use tags
<%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>

eg --- To import JSTL core lib
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

3. Invoke JSTL tag/action
3.1 eg
<c:set var="abc" value="${param.f1}" />
WC : pageContext.setAttribute("abc",request.getParameter("f1"))

WC invokes --- session.setAttribute("abc",request.getparameter("f1"));

menaing of <c:set> sets the specified attr to specified scope.

<c:set var="details" value="${sessionScope.abc}" />
WC : pageContext.setAttribute("details",session.getAttribute("abc"));

2. <c:remove var="abc" scope="request"/>
WC ---request.removeAttribute("abc") ---removes the attr from req scope.

3.2 JB --- ShopBean -- property --
private AL<Category> categories; --g & s

<c:forEach var="cat" items="${sessionScope.shop.listCategories()}">
${cat}
</c:forEach>

WC invokes ---

for(Category cat : session.getAttribute("shop").listCategories())
  out.print(cat);

eg :
<c:forEach var="acct" items="${sessionScope.my_bank.acctSummary}">
${acct.acctID} ${acct.type} ${acct.balance}
</c:forEach>

http://localhost:8080/day6_web/close_acct.jsp?acId=101

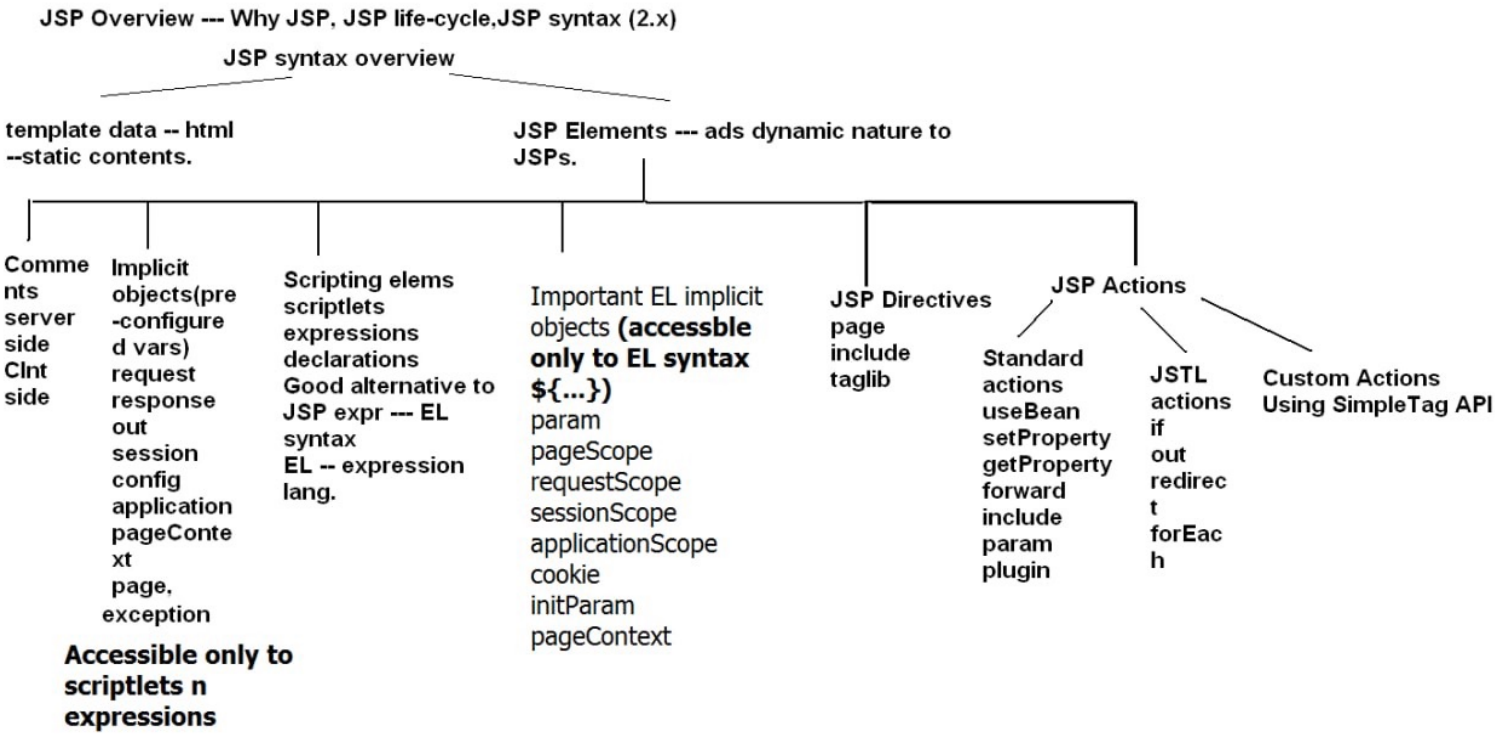
<input type="submit" name="btn" value="Withdraw"
                              formaction="transactions.jsp" /></td>
                        <td><input type="submit" name="btn" value="Deposit"
                              formaction="transactions.jsp" /></td>

<%
   request.getPrameter("btn").equals("Deposit") ---

```
%>
<c:if test="boolean val">
....
</c:if>

<c:if test="${param.btn eq 'Deposit'}">
  in deposit
</c:if>
<c:if test="${param.btn eq 'Withdraw'}">
  in withdraw
</c:if>

http://localhost:8080/day6_web/transactions.jsp?acId=102&amount=500&btn=Deposit


<c:redirect url="${sessionScope.my_bank.closeAccount()}"/>
WC --- response.sendRedirect(session.getAttribute("my_bank").closeAccount());

3.3 JSTL action --- for URL rewriting
<c:url var="attr Name" value="URL to be encoded" scope="page|request|session|
application"/>

eg : <c:url var="abc" value="next.jsp" />
WC invokes --- pageContext.setAttribute("abc",resp.encodeURL("next.jsp"));

<a href="${abc}">Next</a>


# How to set session tm out ?
1. programmatically --- using Java API
    From HttpSession --- setMaxInactiveInterval(int secs)
2. declaratively -- either using Java annotations OR using XML config files
(web.xml)




Session Tracking technique :
 HttpSession + URL rewriting

Why ????
To develop a web app , independent of cookies , for session tracking.

For tracking the clnt (clnt's session) : the only information,  WC needs from the
clnt browser is JSessionID value. If clnt browser is not sending it using cookie :
Servlet/JSP prog can embed the JSessionID info in each outgoing URL .(response:
location / href /form action)


What is URL Rewriting : Encoding the URL to contain the JSessionID info.

W.C always 1st chks if JsessionID is coming from cookie, if not ---> then it will
chk in URL : if it finds JsessionID from the encoded URL : extracts its value &
proceeds in the same manner as earlier.

How to ?
```

API :
For URLs generated by clicking link/buttons(clnt pull I) use
HttpServletResponse method
public String encodeURL(String origURL)
Rets : origURL;JSESSIONID=12345

For URLs generated by sendRedirect : clnt pull II : use
HttpServletResponse method
public String encodeRedirectURL(String redirectURL)
Rets : redirectURL;JSESSIONID=12345



JSP life cycle

test.jsp

any clnt sending rq1 to JSP page

Translation phase --- o/p will transalate servlet page -- performed by WC. called once per jsp life-cycle

o/p --- container specific name (Tomcat test_jsp.java , Its typically stored under work.) Its implementatation class from Servlet i/f + vendor specific APIs.

Compilation phase --- o/p --page translation class (.class) called by WC --- once per life-cycle

locate .class (from server specific folder) , loaded using Class.forName(...) & instantiated using def. constructor.

WC invokes (once per life-cycle) ---public void jspInit --- declared within javax.servlet.jsp.JspPage i/f

WC pools out a thrd -- run() --- public void _jspService(HttpServletRequest rq,HttpServletResponse rs) throws SE,IOExc --- should not be overridden by JSP prog. ---- called per each request. (code from scriptlets,exprs & templates appear within _jspService) --- when _jspService rets --run() rets --- thrd simply rets to pool. ---- Run time or req. processing phase. declared in javax.servlet.jsp.HttpJspPage i/f.

public void jspDestroy() --- called by WC at end of life-cycle before GCing jsp's transalted page cls inst. Triggers -- server shut down , un-deploy or re-deplyment of web appln.

JSP

What is JSP? (Java server pages)
Dynamic Web page (having typically HTML 5 markup) , can embed Java code directly.

Dynamic web component , whose life-cycle is managed by WC(JSP container/Servlet container/Servlet engine)

WHY JSP?

1. JSP allows developer to separate presentation logic(dyn resp generation) from Business logic or data manipulation logic.
Typically JSPs -- used for P.L(presentation logic)

Java Beans or Custom Tags(actions) --- will contain Business logic.

2. Ease of development --- JSP pages are auto. translated by W.C in to servlet & compiled & deployed.

3. Can use web design tools -- for faster development (RAD --rapid application development) tools.

JSP API : is a part of Java EE specs
jsp-api.jar --- <tomcat>/lib : specs

Contains JSP API implementation classses. : jasper.jar

0. javax.servlet.Servlet -- super i/f

1. javax.servlet.jsp.JspPage -- extends Servlet i/f

1.1 public void jspInit()
1.2 public void jspDestroy()
[Can be overridden by JSP page author]

2. Further extended by javax.servlet.jsp.HttpJspPage

2.1 public void _jspService(HttpServletRequest rq,HttpServletResponse rs) throws ServletExc,IOExc.
[Never override _jspService ---JSP container auto translates JSP tags (body) into _jspService.]

JSP life-cycle

1. Clnt sends the 1st request to the JSP (test.jsp)
2. Web-container invokes the life cycle for JSP
3. Translation Phase : handled by the JSP container.
   I/p : test.jsp O/p : test_jsp.java (name : specific to the Tomcat container)
   Meaning : .jsp is translated into corresponding servlet page(.java)
   Translation time errs : syntactical errs in using JSP syntax.
   In case of errs : life-cycle is aborted.
4. Compilation Phase : handled by the JSP container.
   I/p : Translated servlet page(.java) O/p : Page Translation class(.class)
   Meaning : servlet page auto. compiled into .class file
   Compilation time errs: syntacticle errs in generated Java syntax.
5. Request processing phase / Run time phase. : typically handled by the Servlet Container.
6. S.C : will try to locate,load,instantiate the generated servlet class.
7. The 1st it calls : public void jspInit() : one time inits can be performed.

(jspInit availble from javax.servlet.jsp.JspPage)
8. Then it will call follwing method using thrd created per clnt request :
   public void _jspService(HttpServlet Rq,HttpServletResponse) throws ServletException,IOException(API avlble from javax.servlet.jsp.HttpJspPage)
   When _jspService rets , thread's run method is over & thrd rets to the pool, where it can be used for servicing some other or same clnt's req.
9. At the end ...(server shutting down or re-deployment of the context) : the S.C calls
   public void jspDestroy()
   After this : translated servlet page class inst. will be GCEd....
10. For 2nd req onwards ..... : SC will invoke step 8 onwards.

JSP 2.0/2.1/2.2/2.3 syntax

1. JSP comments

`1.1 server side comment
syntax : <%-- comment text --%>`
significance : JSP translator & compiler ignores the commented text.

1.2 clnt side comment
syntax : <!-- comment text -->
significance : JSP translator & compiler does not ignore the commented text BUT clnt browser will ignore it.

2. JSP's implicit objects (available only to _jspService) -- available to scriptlets,exprs
2.1 out - javax.servlet.jsp.JspWriter : represents the buffered writer stream connected to the clnt via HttpServletResponse(similar to your PrintWriter in servlets)
Has the same API as PW(except printf)
usage eg : out.print("some text sent to clnt");

2.2 request : HttpServletRequest (same API)

2.3 response : HttpServletResponse

2.4 config : ServletConfig (used for passing init params)

2.4 session : HttpSession (By def. all JSPs participate in session tracking i.e session obj is created)

2.5 exception : java.lang.Throwable (available only to err handling pages)

2.6 pageContext : current page environment : javax.servlet.jsp.PageContext(this class stores references to page specific objects viz -- exception,out,config,session)

2.7 application : ServletContext(used for Request dispatching, server side logging, for creating context listeners,to avail context params, to add/get context scoped attrs)
2.8 page --- current translated page class instance created for 'this' JSP

9. Servlet Config

3. Scripting elements : To include the java content within JSP : to make it dynamic.

3.1 Scriptlets : can add the java code directly . AVOID scriptlets . (Use only till you learn Javabeans & custom tags or JSTL,). we will use use the scriptlets to add : Req. processing logic, B.L & P.L)
syntax : <% java code..... %> : within <body> tag.
location inside the translated page : within _jspService
usage : till Java beans / JSTL or cust. tags are introduced : scriptlets used for control flow/B.L/req. proc. logic

3.2 JSP expressions :
`syntax : <%= expr to evaluate %>`
--Evaluates an expression --converts it to string --send it to clnt browser.
eg : <%= new Date() %>

expr to evaluate : java method invocation which rets a value OR
const expr or attributes(getAttribute) or variables(instance vars or method local)
location inside the translated page : within _jspService
significance : the expr gets evaluated---> to string -> automatically sent to clnt browser.

eg <%= new Date() %>
eg <%= request.getAttribute("user_dtls") %>
<%= 12*34*456 %>
<%= session.getAttribute("user_dtls") %>
<%= session.setAttribute("nm",1234) %> -- compiler error
<%= session.getId() %>

Better alternative to JSP Expressions : EL syntax (Expression Language : avlble from JSP 1.2 onwards)
syntax : ${expr to evaluate} (to be added directly in ,<body> tag)
EL syntax will evaluate the expr ---to String --sends it clnt browser.

JSP implicit object --- request,response,session.....---accessible from scriptlets & JSP exprs. ---

8 EL implicit objects --- can be accessible only via EL syntax
param =Name of the map , created by WC : containing request parameters
pageScope=Name of the map , created by WC : containing page scoped attrs
requestScope=map of request scoped attrs
sessionScope=map of session scoped attrs
applicationScope=map of application(=context) scoped attrs
pageContext --- instance of PageContext's sub class
cookie -- map of cookies(cookie objects)
initParam -- map of context params.

--avlable ONLY to EL syntax ${...}
--to be added directly within <body> ...</body>

eg : ${param.user_nm} ---param.get("user_nm") --value --to string ---> clnt request.getParameter("user_nm") --value --to string ---> clnt

${requestScope.abc} ---request.getAttribute("abc") ---to string --sent to clnt browser.

eg : suppose ctx scoped attr --- loan_scheme
${applicationScope.loan_scheme} ---

↓ EL Syntax change.

getServletContext().getAttribute("loan_scheme") ---to string --sent to clnt

${abc} ---
pageContext.getAttribute("abc") ---not null -- to string -clnt
 null
--request.getAttribute("abc") -- not null -- to string -clnt
null
session.getAttribute("abc") ---
null
getServletContext().getAttirbute("abc") --not null -- to string -clnt
null ---BLANK to clnt browser.

eg : ${sessionScope.nm} OR ${nm}

${pageContext.session.id}
--pageContext.getSession().getId() --- val of JessionId cookie w/o java code.

${pageContext.request.contextPath} ---/day5_web

${pageContext.session.maxInactiveInterval}

${param}
{user_nm=asdf, user_pass=123456}

eg : ${param.f1} ---> request.getParameter("f1").toString()---> sent to browser

param ----map of req parameters.

param : req. param map

${requestScope.abc} ----- out.print(request.getAttribute("abc").toString())

${abc} -----pageContext.getAttribute("abc")----null ---request ---session---
application ---null ---EL prints blank.

3.3 JSP declarations (private members of the translated servlet class)
syntax : <%! JSP declaration block %> (outside <body>)
Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)
2. Also can be used for overriding life cycle methods (jspInit,jspDestroy)

location inside the translated page : outside of _jspService (directly within JSP's translated class)

DAY 6

Have you downloaded hibernate dependencies ???????????

Today's Topics

Revise JSP Basics
URL rewriting
JSP Directives : page
JSP include Directive
JSP Actions
JSP using JavaBean
JSTL

Revise
0. Why JSP , life cycle , scripting elements , EL syntax

Name JSP impl objects available from scriptlets n exprs
out, request,response,session, config,application,exception,page,pageContext

Name EL impl objects available from EL syntax(${...})
param,pageScope,requestScope,sessionScope,applicationScope,cookie,initParam,pageContext

Solve :

change to EL Syntax

${sessionScope.user_details}
=> session.getAttribute("user_details") ---to string --sends to clnt
${pageContext.session.id}
=> pageContext.getSession().getId() --to string --sends to clnt
${param.email}
=> request.getParameter("email") -to string --sends to clnt

${pageContext.errorData.requestURI} : later

Solve with EL syntax
How to print session expiration tmout on clnt browser ?
HttpSession API : public int getMaxInactiveInterval()
${pageContext.session.maxInactiveInterval}

How to display context path to clnt ?
Method of HttpServletRequest : public String getContextPath()
JSP Expr : <%= request.getContextPath() %>
OR
EL Syntax : ${pageContext.request.contextPath}

How to invalidate session ?
1. Scriptlet :
```
<%
 session.invalidate();
%>
```

OR
2. JSP expr : <%= session.invalidate() %> => compilation err

OR

3. EL syntax : ${pageContext.session.invalidate()}

2. (Are there any problems observed in client pull , if cookies are disabled?) :
YES
Session Tracking failure!!!!!!!!!!!!!!

Any solution ? URL Rewriting

Any problems in URL rewriting tech :

Vulnerable (open to security threats) :

Security attack : Session Fixation or man in middle / session stealing

Soln : Do not use URL rewriting approach n force the clients to accept the cookies
(Cookies also can be stored in a secure manner , Cookie class API : public void
setSecure(boolean flag))
OR use https : for end to end encryption

3. JSP Directives : page , include

4. JSP Actions : request dispatcher related

Enter JSP Using Java Beans
1. What is a java bean ?
2. Why Java Beans ?

Steps
1. UserBean : for authentication of user

3. JSP standard Actions Using java beans
3.1
<jsp:useBean id="bean id" class="Fully qualified bean class name" scope="page|
request|session|application"/>
default scope : page
eg : <jsp:useBean id="user" class="beans.UserBean" scope="session"/>

3.2 <jsp:setProperty name="user" property="email" value="abc@gmail.com"/>

```
<jsp:setProperty name="user" property="email" value="${param.em}"/>


<jsp:setProperty name="user" property="email" param="em"/>


<jsp:setProperty name="user" property="*"/>


WC : Tries to call ALL MATCHING setters
Request param names MUST MATCH with JB property setters

eg : http://host:port/day6/validate.jsp?em=abc&pass=1234
validate.jsp
<jsp:useBean id="user" class="beans.UserBean" scope="session"/>
<jsp:setProperty name="user" property="*"/>

UserBean : setEmail n setPassword
How many setters ? 0

4. How to invoke B.L method of a java bean  w/o java code(i.e scriptlet)
Via EL syntax.
${sessionScope.user.validateUser()}


5. How to invoke getters of JB ?
5.1 <jsp:getProperty name="user" property="validatedUserDetails"/>


OR
EL syntax :
${sessionScope.user.validatedUserDetails}
```

Maven

What is Maven ?

Build automation tool for overall project management.

It helps in
1. checking a build status
2. generating reports (basically javadocs)
3. setting up the automated build process and monitors the same.

Why Maven ?

It eases out source code compilation, distribution, documentation, collaboration with different teams .

Maven tries 2 describe

1. How a software is built.
2. The dependencies, plug-ins & profiles that the project is associated in a standalone or a distributed environment.

Vendor -- Apache

Earlier build tool -- Ant
Vendor -- Apache.

Ant disadvantages
1. While using ant , project structure had to be defined in build.xml. Maven has a convention to place source code, compiled code etc. So no need to provide information about the project structure in pom.xml file.

2.
Maven is declarative, everything you define in the pom.xml file.
No such support in ant.

3.
There is no life cycle in Ant, where as life cycle exists in Maven.

Maven advantages

4. Managing dependencies
5. Uses Convention over configuration - configuration is very minimal
6. Multiple/Repeated builds can be achieved.

7. Plugin management.
8. Testing - ability to run JUnit and other integration test suites.

What is POM? (Project Object Model)

It is the core element of any maven project.
Any maven project consists of one configuration file called pom.xml.
Location --In the root directory of any maven project.

It contains the details of the build life cycle of a project.

Contents
Dependencies used in the projects (Jar files)
Plugins used

Project version
Developers involved in the project
Build profiles etc.

Maven reads the pom.xml file, then executes the goal.

Elements of maven pom.xml file

1. project It is the root element of pom.xml file.
2. modelVersion It is the sub element of project. It specifies the modelVersion.
3. groupId It is the sub element of project. It specifies the id for the project group.(typically organization name)
4. artifactId It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, WARs.
5. version It is the sub element of project. It specifies the version of the artifact under given group.
6.
packaging - defines packaging type such as jar, war etc.
7. name -- defines name of the maven project.
8. plugins ---compiler plugins , eclipse plugins
9. dependencies -- collection of dependencies for this project.
Within that --
dependency -- defines a specific dependency.(eg : hibernate dependency,spring web)
9. scope -- defines scope for this maven project. It can be compile, provided, runtime, test and system.

Goals in Maven
Goal in maven is nothing but a particular task which leads to the compiling, building and managing of a project. A goal in maven can be associated to zero or more build phases. Only thing that matters is the order of the goals defined for a given project in pom.xml. Because, the order of execution is completely dependent on the order of the goals defined.
eg : clean , build ,install ,test

What is a Maven Repository

A maven repository is a directory of packaged JAR file with pom.xml file. Maven searches for dependencies(JARs) in the repositories. There are 3 types of maven repository:

- Local Repository
- Central Repository
- Remote Repository

Maven searches for the dependencies in the following order:

Local repository then Central repository then Remote repository.
maven repositories

If dependency is not found in these repositories, maven stops processing and throws an error.

1. Maven Local Repository

Maven local repository is located in the file local system. It is created by the maven when you run any maven command.

By default, maven local repository is HOME / .m2 directory.
(Can be updated by changing the MAVEN_HOME/conf/settings.xml)

2) Maven Central Repository

Maven central repository is located on the web(Created by the apache maven community)

The path of central repository is: https://mvnrepository.com/repos/central

3) Maven Remote Repository

Maven remote repository is also located on the web. Some of libraries that are missing from the central repository eg JBoss library , Oracle driver etc, can be located from remote repository.

Maven Build Life Cycle

What is it ?
The sequence of steps which is defined in order to execute the tasks and goals of any maven project is known as build life cycle in maven.

Maven comes with 3 built-in build life cycles

Clean - this phase involves cleaning of the project (for a fresh build & deployment)
Default - this phase handles the complete deployment of the project
Site - this phase handles the generating the java documentation of the project.

Build Profiles in Maven

It is a subset of elements which allows to customize builds for particular environment. Profiles are also portable for different build environments.

Build environment basically means a specific environment set for production and development instances. When developers work on development phase, they use test database from the production instance and for the production phase, the live database will be used.

So, in order to configure these instances maven provides the feature of build profiles. Any no. of build profiles can be configured and also can override any other settings in the pom.xml

eg : profiles can be set for dev, test and production phases.

Installation (w/o IDE)
1. Download Maven from Apache (version 3.x)
2. Add MAVEN_HOME as environment variable
3. Add maven/bin under path (for easy accessibility)
4.Verify maven
  mvn -- version

  OR use m2e plug-in (a standard part of Eclipse for J2EE)

Hibernate → Session factory

Hibernate API

0. org.hibernate.SessionFactory API
getCurrentSession vs openSession

public Session openSession() throws HibernateException
opens NEW session from SF,which has to be explicitely closed by programmer.

OR
public Session getCurrentSession() throws HibernateException
Opens new session , if one doesn't exist , otherwise continues with the exisitng one.
Gets automatically closed upon Tx boundary or thread over(since current session is bound to current thread --mentioned in hibernate.cfg.xml property --- current_session_context_class ---thread)

1. CRUD logic (save method)
API (method) of org.hibernate.Session
public Serializable save(Object o) throws HibernateException

I/P ---transient POJO ref.
save() method auto persists transient POJO on the DB(upon committing tx) & returns unique serializable ID generated by hib frmwork, if @GeneratedValue is used for id.

2. Hibernate session API -- for data retrieval by PK
API (method) of org.hibernate.Session
public <T> T get(Class<T> c,Serializable id) throws HibernateException
T -- type of POJO
Returns --- null -- if id is not found.
returns PERSISTENT pojo ref if id is found.

Usage of Hibernate Session API's get()
int id=101;
BookPOJO b1=hibSession.get(BookPOJO.class,id);//int -----Integer(auto boxing) ---- upcasting------Serializable
OR
BookPOJO b1=hibSession.get(Class.forName("pojos.BookPOJO"),id);

2. Display all books info :

using HQL -- Hibernate Query Language --- Objectified version of SQL --- where table names will be replaced by POJO class names & table col names will replaced by POJO property names.
(JPA--- Java Persistence API compliant syntax --- JPQL)
eg --- SQL -- select * from books
HQL --- "from BookPOJO"
eg JPQL -- "select b from BookPOJO b"

2.1 Create Query Object --- from Session i/f
<T> org.hibernate.query.Query<T> createQuery(String hql/jpql,Class<T> resultType)

eg : Query<Book> q=hs.createQuery(hql/jpql,Book.class);

2.2. Execute query to get List of selected PERSISTENT POJOs
API of org.hibernate.query.Query i/f
(Taken from javax.persistence.TypedQuery<T>)

List<T> getResultList()

Execute a SELECT query and return the query results as a generic List<T>.
T -- type of POJO / Result

eg : hs,tx
String jpql="select b from Book b";
try {
    List<Book> l1=hs.createQuery(jpql,Book.class).getResultList();
}

 Usage ---
String hql="select b from BookPOJO b";
List<BookPOJO> l1=hibSession.createQuery(hql).getResultList();

3. Passing IN params to query. & execute it.
Objective : Display all books from specified author , with price < specified price.
API from org.hibernate.query.Query i/f

Query<R> setParameter(String name,Object value)

Bind a named query parameter using its inferred Type.
name -- query param name
value -- param value.I

String hql="select b from BookPOJO b where b.price < :sp_price and b.author
= :sp_auth";

How to set IN params ?
org.hibernate.query.Query<T> API
public Query<T> setPrameter(String pName,Object val)

List<Book> l1 =
hibSession.createQuery(hql,Book.class).setParameter("sp_price",user_price).setParam
eter("sp_auth",user_auth).getResultList();

Objective --Offer discount on all old books
i/p -- date , disc amt

4. Updating POJOs --- Can be done either with select followed by update or ONLY
with update queries(following is eg of 2nd option
commonly known as bulk update)
Objective : Reduce price of all books with author=specified author , published
before a sepecific date
String jpql = "update BookPOJO b set b.price = b.price - :disc where b.author = :au
and b.publishDate < :dt ";
set named In params
exec it (executeUpdate) ---
int updateCount= hs.createQuery(jpql).setParameter("disc", disc).setParameter("dt",
d1).executeUpdate();

---This approach is typically NOT recommended often, since it bypasses L1 cache .
Cascading is not supported. Doesn't support optismistic locking directly.
Use case -- for bulk updations to be performed on a standalone (un related) table)

5. Delete operations.
API of org.hibernate.Session

--void delete(Object o) throws HibernateException
i/p --persistent POJO ref

---POJO is marked for removal , corresponding row from DB will be deleted after comitting tx & will be removed form entity cache(l1 cache) closing of session.

OR
5.5
One can use directly "delete HQL" & perform deletions.
eg
int deletedRows = hibSession.createQuery ("delete Subscription s where s.subscriptionDate < :today").setParameter ("today", new Date ()).executeUpdate ();

Detailed API
0. SessionFactory API
getCurrentSession vs openSession

public Session openSession() throws HibernateExc
opens new session from SF,which has to be explicitely closed by prog.

public Session getCurrentSession() throws HibernateExc
Opens new session , if one doesn't exist , otherwise continues with the exisitng one.
Gets automatically closed upon Tx boundary or thread over(since current session is bound to current thread --mentioned in hibernate.cfg.xml property --- current_session_context_class ---thread)

1. Testing core api
persist ---
public void persist(Object transientRef)

if u give some non-null id (existing or non-existing) while calling persist(ref) -- gives exc
org.hibernate.PersistentObjectException: detached entity passed to persist:
why its taken as detached ? ---non null id.

2.
public Serializable save(Object ref)
save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exc.
Ignores ur passed id & creates its own id & inserts a row.

3. saveOrUpdate
public void saveOrUpdate(Object ref)
--either inserts/updates or throws exc.
null id -- fires insert (works as save)
non-null BUT existing id -- fires update (works as update)
non-null BUT non existing id -- throws StaleStateException --to indicate that we are trying to delete or update a row that does not exist.

3.5
merge
public Object merge(Object ref)
I/P -- either transient or detached POJO ref.
O/P --Rets PERSISTENT POJO ref.

null id -- fires insert (works as save)

non-null BUT existing id -- fires update (select , update)
non-null BUT non existing id -- no exc thrown --Ignores ur passed id & creates its own id & inserts a row.(select,insert)

4. get vs load
& LazyInitilalizationException.

5. update
Session API
public void update(Object object)
Update the persistent instance with the identifier of the given detached instance.
I/P --detached POJO containing updated state.
Same POJO becomes persistent.

Exception associated :
1. org.hibernate.TransientObjectException: The given object has a null identifier:
i.e while calling update if u give null id. (transient ----X ---persistent via update)

2. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.
3.
org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session

6. public Object merge(Object ref)
Can Transition from transient -->persistent & detached --->persistent.
Regarding Hibernate merge
1. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().
2. API of Session
Object merge(Object object)
3.
Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
3.If there is no persistent instance currently associated with the session, it will be loaded.
4.Return the persistent instance. If the given instance is unsaved, save a copy of and return it as a newly persistent instance. The given instance does not become associated with the session.
5. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

7.public void evict(Object persistentPojoRef)

It detaches a particular persistent object
from the session level cache(L1 cache)
(Remove this instance from the session cache. Changes to the instance will not be synchronized with the database.)

8.
void clear()

When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.

But Databse Connection is not returned to connection pool.
(Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)

9. void close()

When close() is called on session object all
the persistent objects associated with the session object become detached(l1 cache is cleared) and also closes the Database Connection.

10. void flush()

When the object is in persistent state , whatever changes we made to the object state will be reflected in the databse only at the end of transaction.

BUT If we want to reflect the changes before the end of transaction
(i.e before commiting the transaction)
call the flush method.
(Flushing is the process of synchronizing the underlying DB state with persistable state of session cache)

11. boolean contains(Object ref)

The method indicates whether the object is
associated with session or not.(i.e is it a part of l1 cache ?)

12.
void refresh(Object ref) -- ref --persistent or detached

This method is used to get the latest data from database and make
corresponding modifications to the persistent object state.
(Re-reads the state of the given instance from the underlying database

API of org.hibernate.query.Query<T>

1. Iterator iterate() throws HibernateException

Return the query results as an Iterator. If the query contains multiple results per row, the results are returned Object[].

Entities returned --- in lazy manner

Pagination

2. Query setMaxResults(int maxResults)

Set the maximum number of rows to retrieve. If not set, there is no limit to the number of rows retrieved.

3. Query setFirstResult(int firstResult)

Set the first row to retrieve. If not set, rows will be retrieved beginnning from row 0. (NOTE row num starts from 0)

eg --- List<CustomerPOJO> l1=sess.createQuery("select c from CustomerPOJO c").setFirstResult(30).setMaxResults(10).list();

4.How to count rows & use it in pagination techniques?

```
int pageSize = 10;
String countQ = "Select count (f.id) from Foo f";
Query countQuery = session.createQuery(countQ);
Long countResults = (Long) countQuery.uniqueResult();
int lastPageNumber = (int) ((countResults / pageSize) + 1);

Query selectQuery = session.createQuery("From Foo");
selectQuery.setFirstResult((lastPageNumber - 1) * pageSize);
selectQuery.setMaxResults(pageSize);
List<Foo> lastPage = selectQuery.list();
```

5. org.hibernate.query.Query API

<T> T getSingleResult()

Executes a SELECT query that returns a single result.

Returns: Returns a single instance(persistent) that matches the query.

Throws:
NoResultException - if there is no result
NonUniqueResultException - if more than one result
IllegalStateException - if called for a JPQL UPDATE or DELETE statement

6. How to get Scrollable Result from Query?

ScrollableResults scroll(ScrollMode scrollMode) throws HibernateException

Return the query results as ScrollableResults. The scrollability of the returned results depends upon JDBC driver support for scrollable ResultSets.

Then can use methods of ScrollableResults ---first,next,last,scroll(n) .

7. How to create Named query from Session i/f?

What is a named query ?
Its a technique to group the HQL statements in single location(typically in POJOS) and lately refer them by some name whenever need to use them. It helps largely in code cleanup because these HQL statements are no longer scattered in whole code.

Fail fast: Their syntax is checked when the session factory is created, making the application fail fast in case of an error.
Reusable: They can be accessed and used from several places which increase re-usability.

eg : In POJO class, at class level , one can declare Named Queries

```
@Entity
@NamedQueries
({@NamedQuery(name=DepartmentEntity.GET_DEPARTMENT_BY_ID, query="select d from DepartmentEntity d where d.id = :id")}
```

```
public class Department{....}

Usgae
Department d1 = (Department)
session.getNamedQuery(DepartmentEntity.GET_DEPARTMENT_BY_ID).setInteger("id", 1);
```

8. How to invoke native sql from hibernate?

```
Query q=hs.createSQLQuery("select * from books").addEntity(BookPOJO.class);
 l1 = q.list();
```

9. Hibernate Criteria API

A powerful and elegent alternative to HQL

Well adapted for dynamic search functionalities where complex Hibernate queries have to be generated 'on-the-fly'.

Typical steps are -- Create a criteria for POJO, add restrictions , projections ,add order & then fire query(via list() or uniqueResult())

10. For composite primary key

Rules on prim key class

Annotation -- @Embeddable (& NOT @Entity)

Must be Serializable.

Must implement hashCode & equals as per general contract.

In Owning Entity class

Add usual annotation -- @Id.

1.1 Testing core api

persist ---

```
public void persist(Object transientRef)
```

if u give some non-null id (existing or non-existing) while calling persist(ref) -- gives exc

org.hibernate.PersistentObjectException: detached entity passed to persist:

why its taken as detached ? ---non null id.

2.

```
public Serializable save(Object ref)
```

save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exception

Ignores your passed id & creates its own id & inserts a row.

3. saveOrUpdate

```
public void saveOrUpdate(Object ref)
```

--either inserts/updates or throws exc.

null id -- fires insert (works as save)

non-null BUT existing id -- fires update (works as update)

non-null BUT non existing id -- throws StaleStateException --to indicate that we are trying to delete or update a row that does not exist.

3.5

merge

```
public Object merge(Object ref)
```

I/P -- either transient or detached POJO ref.

O/P --Rets PERSISTENT POJO ref.

null id -- fires insert (works as save)

parameters vs attributes

name(String) & value(String)

1. Request params --api --- getParameter(nm) --val
<%= request.getParameter("nm") %>
${param.nm}

Readble --thrd safe.

2. init params -- servlet/JSP specific init params
web.xml /anno
To access --- getServletConfig().getInitParameter(nm)

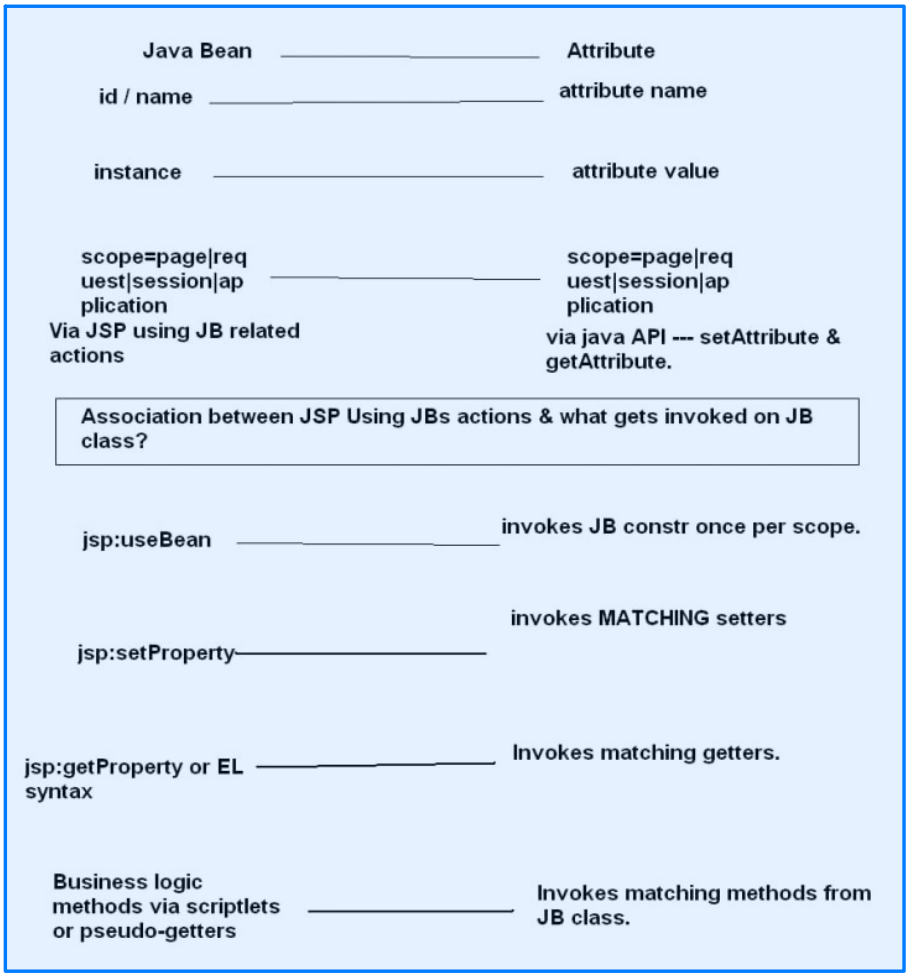
<%= config.getInitParameter(nm) %>

3. Ctx parameters
--scope entire web appln
web.xml
<context-param>....
How to access
getServletContext().getInitParamer(nm)

server side object--name (string) , value(Object)
set/get/remove/getAttrNames
JSP
page --current page only
api -- scriptlets
To add -- pageContext.setAttribute(nm,val)
To get ---pageContext.getAttribute(nm)

or <%= pageContext.getAttribute(nm) %>
${pageScope.nm}
or ${nm}

request --- <% request.setAttr....
%>
${requestScope.nm}



Layers involved in JSP -- JavaBean --DAO --POJO --DB

clnt submits login form

authenticate.jsp
1. For creating Javabean instance(=server side attribute , added in the required scope) :
<jsp:useBean id="bean id" class="F.Q bean class name"
scope="page|request|session|application"/>
def scope=page
eg : <jsp:useBean id="user" class="beans.UserBean" scope="session"/>

WC : session.getAttribute("user")

null

session.setAttribute("user",new UserBean());

not null

2. <jsp:setProperty name="user" property="*"/>
WC tries to call ALL MATCHING setters (data binding : req params --> JB props)

3. Invoke B.L method of Java bean
Using EL syntax

eg : ${sessionScope.user.validateUser()}
WC : session.getAttribute("user").validateUser()

JSP actions for managin Javabean
1. <jsp:useBean id="user" class="beans.BankUserBean"
scope="session" />

WC checks if specified bean exists under specified scope?
session.getAttribute("user")

null

1. locate(WEB-INF/classes)
loads (Class.forName)
inst : def constr
eg : BankUserBean ref=new BankUserBean();

2. WC adds this bean instance under specified scope
session.setAttribute("user",ref);

not null

bean instance is alredy created by WC
(in earlier req coming from SAME clnt)

If there are 10 clnts sending 5 requests each , how many times WC will call Bean's constr : 10

<jsp:useBean id="test" class="beans.TestBean" />
how many times WC will call Bean's constr : once per every page

<jsp:useBean id="test" class="beans.TestBean" scope="request"/>
how many times WC will call Bean's constr : 50

<jsp:useBean id="test" class="beans.TestBean" scope="application"/>
how many times WC will call Bean's constr : 1

DAY 7 HIBERNATE CONT..

Today's topics

Complete web app : refer to JSP sequence

Enter Hibernate
Hibernate Architecture
CRUD operations in Hibernate
Persistence Context (L1 cache) working

1. What is Hibernate?

Complete solution to manage automatic persistence in DB in Java.
ORM tool
JPA implementor
JPA : Java Persistence API --- Java EE / Jakarata EE specs (javax.persistence)
Hibernate : JPA implementor
(DB Journey in Java ---1. JDBC 2. Hibernate (native hibernate) 3. JPA 4. Spring Data JPA

Hibernate :auto persistence provider

Other persistence providers : iBatis,Kodo, EclipseLink,TopLink,JDO

Spring Boot frmwork : default persistence provider = Hibernate

Open source framework : founded by Gavin King

Intermediate layer between Java app(standalone desk top based / web app) n DB

Why Hibernate ? (refer to readme)
1. open source n light weight
2. supports cache (L1 , L2 , query cache) : faster performance
3. auto table creation.
4. simplifies join queries
5. 100 % DB independent (HQL/JPQL ---Hibernate : DB dialect -- converts DB independent queries in DB specific syntax)
Hibernate 5.x onwards : no need to specifiy DB dialect property in config file (hibernate.cfg.xml : run time classpath)

6. Hibernate developer doesn't have
 to go to DB level , DB ,table ,cols , rows
 sql
 set up the db conn , prepare stmts (st/pst/cst)
 exec queries : process RST : convert it into pojo / collection of POJOs
All of above will be automated by Hibernate

7. JDBC : fixed db conn.(new separate conn per call to DriverManager.getConnection)
Hibernate creates :internal connection pool => collection of DB connections
when : hibernate framework booting time
at the time of creation of singleton SessionFactory(SF)
at the time confgure() -- hibernate.cfg.xml(location : by default run time class path) is parsed :
DB config details -- drvr class , db url , user name , pwd
hibernate.connection.pool_size= 10 (max size)

In DAO layer : When you invoke , open session n begin tx : db conn is pooled out -- wrapped in Session instance n reted to caller.
try

CRUD work (save/get/JPQL/update/delete...)
end of try --commit
catch --RunTimeExc --rollback
finally : session .close ---pooled out db cn simply rets to the pool : so that the same conn can be REUSED for some other request.

8.Solves the important issue of Impedance mismatch in DBMS
Object world (java objs in heap , inheritance , association , polymorphism) -----
RDBMS (table , row cols ,E-R,FKs,join tables...)

9. Exception translation mechanism
Hibernate translates checked SQL excs --un checked hibernate excs
(org.hibernate.HibernateException) : so that prog is not forced to handle the same.

& many more advantages...

Hibernate architecture
refer to a diagram

Important Blocks
1.org.hibernate.Session : interface (imple classes : hibernate core jar)
Represents : Main runtime i/f for prog interaction with hibernate
Supplies CRUD APIs(eg : save, persist, get,load, createQuery,update,delete....)
Represents : wrapper around pooled out db connection.
It has INHERENTLY L1 cache(persistence ctx) associated with it.
DAO layer creates session instance as per demand(one per request for CRUD operation)
light weight , thread un safe object
NO NEED for accessing the session in synchronized manner : since different thrd representing different clnt requests , will have their own session object

2. org.hibernate.SessionFactory : interface (imple classes : hibernate core jar)
JOB : to provide session objects(openSession / getCurrentSession)
singleton instance per DB / application
immutable, inherently thread safe
Represents : DB sepecific config , including connection pool.
It has L2 cache associated with it : MUST be configured explcitly

3. Configuration : org.hibernate.cfg.Configuration class.
Provider of SF

4. Additional APIs
, Transaction,Query,CriteriaQuery ...

5. hibernate.cfg.xml : centralized configuration file , to create SessionFactory(i.e bootstrapping hibernate framework)

Important property :
hibernate.hbm2ddl.auto=update
Chks if table is not yet created for a POJO : create a new table.
BUT if table alrdy exists : continues with the existing table.

5.5 HibernateUtils --- to create singleton immutable SF instance
6.

POJO Annotations

Package : javax.persistence
@Entity : Mandatory : cls level
@Id : Mandatory : field level or property (getter) : PK

Optional annotation for further customization :

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

@GeneratedValue : to tell hibernate to auto generate ids
auto / identity(auto incr : Mysql) / table / sequence(oracle)
eg : @Id => PK
@GeneratedValue(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)

Add <mapping class="F.Q POJO class name"/> in hibernate.cfg.xml

7. Create DAO i/f & write its implementation class
Hibernate based DAO impl class

7.1 No data members ,constructor , cleanup
7.2 Directly add CRUD methods.

Steps in CRUD methods
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/get/persist/update/delete/JPQL...)
commit the tx.
} catch(RuntimeException 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)

5. Create main(..) based Tester & test the application.

Which of the following layers are currently hibernate specific(native hibernate) ?

DAO : org.hibernate.SF , Session, Transaction,Query... : hibernate specific
POJO : javax.persistence : annotations => hibernate inde. (JPA compliant)
Utils : Configuration , org.hibernate.SF => hibernate specific

6. Add a breakpoint before commit , observe n conclude.

7. Replace openSession by getCurrentSession

8. Objective : Get user details
I/P : user id
O/P : User details or error
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
Can you solve it using session.get ? :
sql : "select * from users_tbl"
hql : "from User"
jpql : "select u from User u"
u : POJO alias

11.1 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, in case sensitive manner.

11.2. Create Query Object --- from Session i/f
<T> org.hibernate.query.Query<T> createQuery(String jpql/hql,Class<T> resultType)
T --result type.

11.3. To execute query
Query i/f method
public List<T> getResultList() throws HibernateException
--Rets list of PERSISTENT entities.

12. Objective : Display all users registered between strt date n end date & under a specific role
I/P : begin dt , end date , role

eg : sql = select * from users where reg_date between ? and ? and user_role=?
 jpql =

Passing named IN params to the query
Query i/f method
public Query<T> setParameter(String paramName,Object value) throws
HibernateException.

13. User Login (Lab work)
i/p : email n password
o/p User details with success mesg or invalid login mesg

14. Objective : Display all user names registered between strt date n end date &
under a specific role
jpql =

15 Objective : Display all user name,reg amount,reg date registered between strt
date n end date & under a specific role

String jpql ="select u.name,u.regAmount,u.regDate from User u where u.regDate
between :strt and :end and userRole=:rl"

List<Object[]> list=session.createQuery(jpql,Object[].class).setParameter("start",
startDate).setParameter("end", endDate).setParameter("rl",
userRole).getResultList();
In Tester :
list.forEach(o -> sop(o[0]+" "+o[1]+" "+o[2]));

INSTEAD use a constructor expression
eg :
jpql = "select new pojos.User(name,regAmount,regDate) from User u where u.regDate
between :strt and :end and userRole=:rl";

Pre requisite : MATCHING constr in POJO class

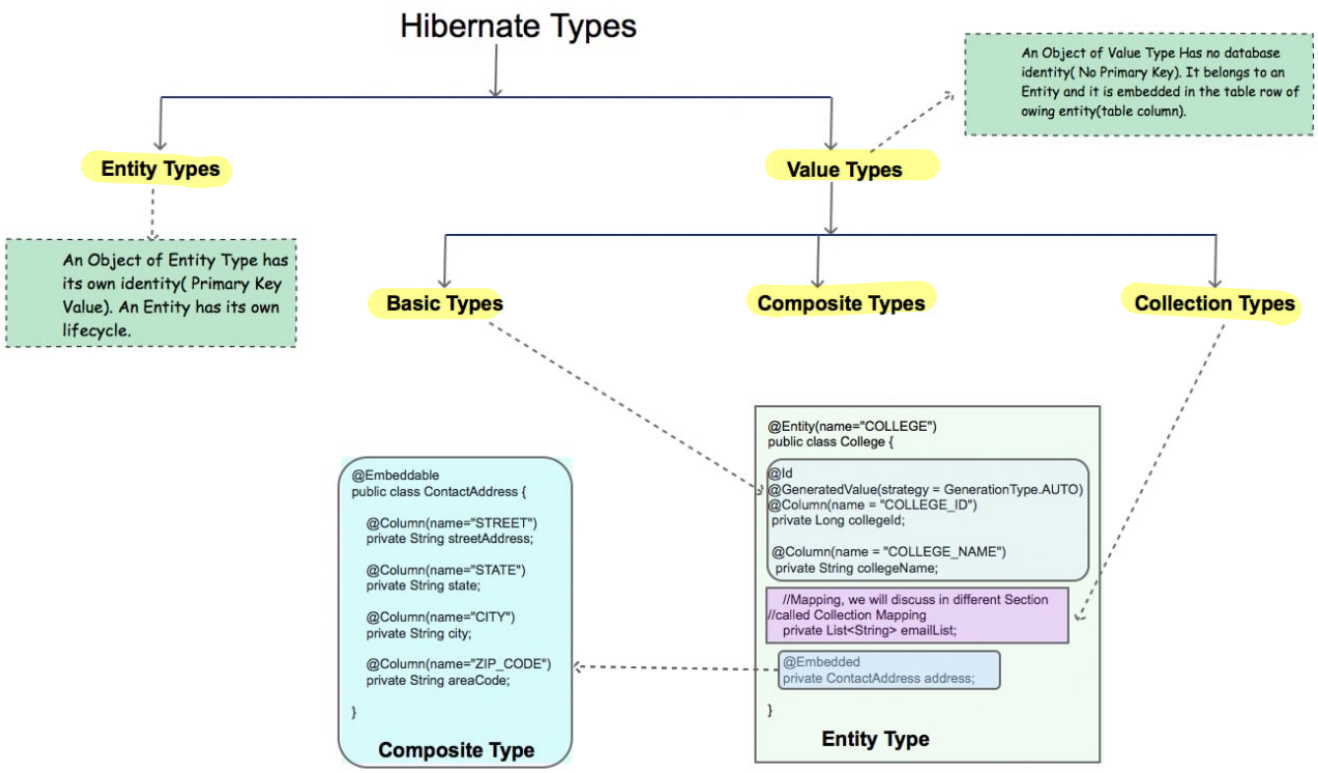
17. Update
Objective :
1. Change password
i/p --email , old password , new pass
o/p : mesg indicating success or a failure

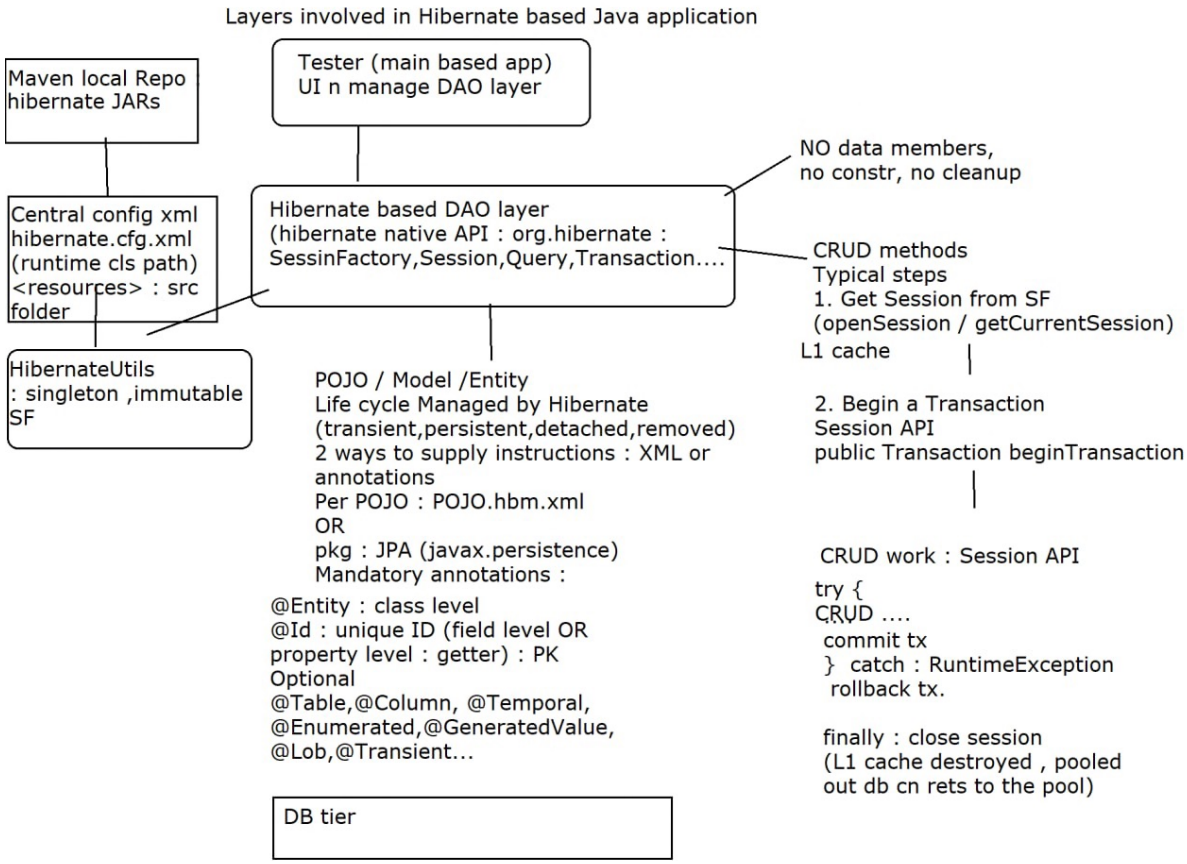
2. Apply discount to reg amount , for all users , reged before a specific date.
(Bulk update)
i/p -- discount amt, reg date
String jpql="update User u set u.regAmount=u.regAmount-:disc where u.regDate
< :dt";

2.1 Query API
public int executeUpdate() throws HibernateException
--use case --DML
2.2 Session API
public Query<T> createQuery(String jpql) throws HibernateException
jpql -- DML

18. Un subscribe user

```
i/p user id
o/p user details removed from DB
Hint : Session API :

19. Lab work
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
FileUtils from Apache commons-io
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
            <dependency>
                  <groupId>commons-io</groupId>
                  <artifactId>commons-io</artifactId>
                  <version>2.11.0</version>
            </dependency>
Methods

1. public static byte[] readFileToByteArray(File file)
                                          throws IOException
 Reads the contents of a file into a byte array. The file is always closed.

2. public static void writeByteArrayToFile(File file,
                              byte[] data)
                                          throws IOException
 Writes a byte array to a file creating the file if it does not exist.
```





SF API openSession vs getCurrentSession
(rets Session object)

openSession

1. Irrespective of the fact that session exists or doesn't exist, a NEW session object is returned to the caller.
2. Prog MUST explicitely close the session -- session.close() , using finally block.

getCurrentSession

1. If session obj exists , then existing session object is reted , ow. new session is created n reted.

2. Session is auto closed --upon tx boundary.

Different type in Hibernate --- Entity Type or a Value Type

1. Has its own DB identity -- Primary key (mapped by @Id)

2.An entity has its own lifecycle -- it may exist independently of any other entity.

3. Supports shared references.
eg Item & category. 2 Items can share the same category.

4. eg Item,Category,User,Order.

1. Doesn't have any DB identity (no @Id annotation)

2. Owned by Entity, no independent life-cycle. Life cycle bounded by the life cycle of owning entity.
3. Doesn't support shared references.

4. eg all Java types are stored as value type.
UDTs also can be mapped as value types(a.k.a components in hibernate jargon)
eg Person has hobbies . Or Person has Address.

Different Types in Hibernate

Entity Type --
Mandatory annotations
@Entity & @Id

Entities are typically stores in their own tables & have independet life cycle (transient,persistent....)

eg : Student & Address to be stored in separate tables.

The anno used -- @OneToOne.
In the owning side optionally can add @JoinColumn .
MUST add in the inverse side @OneToOne (mappedBy="name of prop as it appears in

the owning side.
cascade options are optional.

Value Types

Won't have @Entity & @Id. Will not have standalone existence. Will be bound to the life cycle of owning entity

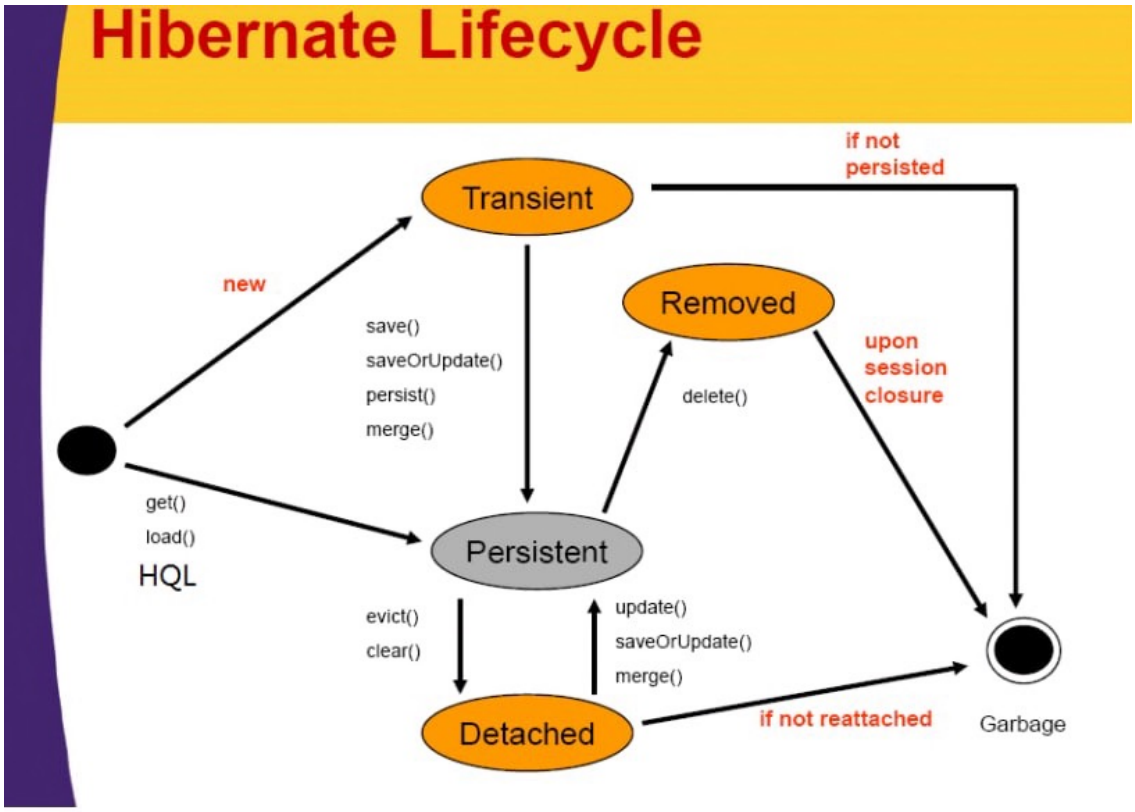
Basic type
single -- no anno required.
For multi valued type mandatory annotation
@ElementCollection
optional anno
@CollectionTable

separate table is creeaed for collection.

Component type
eg : Address.
manadatory anno
@Embeddable
If single valued component --same table as entity table is used.

For multi valued type mandatory annotation
@ElementCollection
optional anno
@CollectionTable

separate table is creeaed for collection.



What is Session? (org.hibernate.Sesion : i/f)

Session object is persistance manager for the hibernate application

Session object is the abstraction of hibernate engine for the Hibernate application

Session object provides methods to perform CRUD operations

Session just represents a thin wrapper around pooled out DB connection.

Session is associated implicitely with L1 cache (having same scope as the session lifetime) , referred as Persistence context.

Example of CRUD

```
  save()                -    Inserting the record
  get() / load()        -    Retrieveing the record
  update()              -    Updating the record
  delete()              -    Deleting the record
public void delete(Object ref) throws HibernateExc
ref -- either persistent or detached pojo ref.
```

What is SessionFactory?(org.hibernate.SessionFactory : i/f)

It is a factory of session objects.(provider of session objects)

we use sessionfactory object to create session object(via openSession or getCurrentSession)
singleton(1 instance per DB / application) ,immutable,inherently thrd safe.
It is a heavy weight object, therefore it has to be created only once for an application & that too at the very beginning.

What is Configuration Object ?(org.hibernate.cfg.Configuration)

Configuration object is used to create the SessionFactory object.

Object Oriented Representation of Hibernate configuration file and mapping file is nothing but Configuration object.

When we call configure() method on configuration object ,hibernate configuration file(hibernate.cfg.xml placed in run time classpath) and mapping files are loaded in the memory.

POJO/Entity Life cycle

1.Transient State

An object is said to be in transient state if it

is not associated
with the session,and has no matching record
in the database table.

For example

```
Account account=new Account();

account.setAccno(101);
account.setName("Amol");
account.setBalance(12000);
```

2.Persistent State

An object is said to be in persistent state if
it is associated with session
object (L1 cache) and will result into a matching record in
the databse table.(i.e upon commit)

```
session.save(account);tx.commit();
```

or
```
Account account=session.get(Account.class,102);
```
OR via HQL/JPQL

Note

When the POJO is in persistent state it
will be in synchronization with the matching
record in DB i.e if we make any changes to the state of persistent POJO it will be
reflected in the database.(after commiting tx) -- i.e automatic dirty checking will be performed(resulting in insert/update/delete)

3.Detached state

Object is not associated with session but
has matching record in the database table.
If we make any changes to the state of
detached object it will NOT be
reflected in the database.

```
session.clear();
session.evict(Object);
session.close();
```

Note :

By calling update method on session object it
will go from detached
state to persistent state.

By calling delete method on session object it will go
from persistenet state to
transient state.

Explain the following methods of Session API

public void persist(Object ref) -- Persists specified transient POJO on underlying DB , upon comitting the transaction.

---
void clear()

When clear() is called on session object all the objects associated
with the session object become detached.
 But Databse Connection is not closed.
(Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)

void close()

When close() is called on session object all
the objects associated with the session object become detached and
also closes the Database Connection.

public void evict(Object ref)

It detaches a particular persistent object
detached or disassociates from the session.
(Remove this instance from the session cache. Changes to the instance will not be synchronized with the database.)

void flush()

When the object is in persistent state ,
whatever changes we made to the object
state will be reflected in the databse only
 at the end of transaction.

If we want to reflect the changes before the end of transaction
(i.e before commiting the transaction)
 call the flush method.
(Flushing is the process of synchronizing the underlying DB state with persistable state of session cache)

boolean contains(Object ref)

The method indicates whethere the object is
associated with session or not.

void refresh(Object ref) -- ref --persistent or detached

This method is used to get the latest data from database and make
corresponding modifications to the persistent object state.
(Re-read the state of the given instance from the underlying database)
---
public void update(Object ref)
Note :-

If object is in persistent state no
need of calling the update method .
As the object is in sync with the
database whatever changes made to the object
will be reflect to database at the
end of transaction.

```
eg --- updateAccount(Account a,double amt)
{
    sess, tx
    sop(a);set amt
    sess.update(a);
    sop(a);
}
```

When the object is in detached state record
 is present in the table
but object is not in sync with database,
therefore update() method can be called
to update the record in the table

Which exceptions update method can raise?
1. StaleStateException -- If u are trying to update a record (using
session.update(ref)), whose id doesn't exist.
i.e update can't transition from transient --->persistent
It can only transition from detached --->persistent.

eg -- update_book.jsp -- supply updated details + id which doesn't exists on db.

2. NonUniqueObjectException -- If there is already persistence instance with same
id in session.
eg -- UpdateContactAddress.java

---
public Object merge(Object ref)
Can Transition from transient -->persistent & detached --->persistent.
Regarding Hibernate merge
1. The state of a transient or detached instance may also be made persistent as a
new persistent instance by calling merge().
2. API of Session
Object merge(Object object)
3.
Copies the state of the given object(can be passed as transient or detached) onto
the persistent object with the same identifier.
3.If there is no persistent instance currently associated with the session, it will
be loaded.
4.Return the persistent instance. If the given instance is unsaved, save a copy of
and return it as a newly persistent instance. The given instance does not become
associated with the session.
5. will not throw NonUniqueObjectException --Even If there is already persistence
instance with same id in session.

---

public void saveOrUpdate(Object ref)

The method persists the object (insert) if matching record is not found (& id inited to default value) or fires update query
If u supply Object , with non-existing ID -- Fires StaleStateException.

lock()
---
when lock() method is called on the
session object for a persistent object ,
untill the transaction is commited in
the hibernate application , externally the matching record in the table cannot be modified.

session.lock(object,LockMode);

eg - session.lock(account,LockMode.UPGRADE);

==========

Hibernate's Automatic Dirty checking

The process of automatically updating the database with the changes to the persistent object when the session is flushed(@ commit) is known as automatic dirty checking.

An object(POJO) enters persistent state when any one of the following happens:

When the code invokes session.save, session.persist or session.saveorUpdate or session.merge
OR
When the code invokes session.load or session.get
OR
Result of JPQL

Any changes to a persistent object are automatically saved to the database when the session in flushed.

Flushing is the process of synchronizing the underlying database with the objects in the session's L1 cache.

Even though there is a session.flush method available but you generally don’t need to invoke it explicitly.

A session gets flushed when the transaction is commited.

---

public void saveOrUpdate(Object ref)

The method persists the object (insert) if matching record is not found (& id inited to default value) or fires update query
If u supply Object , with non-existing ID -- Fires StaleStateException.

lock()
---
when lock() method is called on the
session object for a persistent object ,
untill the transaction is commited in
the hibernate application , externally the matching record in the table cannot be modified.

session.lock(object,LockMode);

eg - session.lock(account,LockMode.UPGRADE);

DAY 8

Revise hibernate architecture
Revise layers in hibernate (building blocks)
Revise flow
Hibernate based CRUD application , searching API
Hibernate managed entity/pojo life cycle

Integration with web app
(user registration n login OR products scenario)

Dis Advantages of Hibernate over JDBC:
1. Learning curve !
2. Booting time (loading hib frmwork) : time consuming
3. If hibernate is not tuned properly , JDBC offers better performance than hibernate .

Refer :
(hibernate api-docs & readme : hibernate session api)

Which of the following layers are currently hibernate specific(native hibernate) ?

DAO : Yes
POJO : NO (JPA compliant !)
Utils : Yes

1. Add a breakpoint before commit , observe n conclude.

2. Replace openSession by getCurrentSession

3. Objective : Get user details
I/P : user id
O/P : User details or error
API : session.get

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

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

6. Objective : Display all user details
Can you solve it using session.get ? :
sql : "select * from users_tbl"

6.1 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, in case sensitive manner.

sql : "select * from users_tbl"
hql : from User
jpql : select u from User u
u : POJO alias (ref)

6.2. Create Query Object --- from Session i/f
<T> org.hibernate.query.Query<T> createQuery(String jpql/hql,Class<T> resultType)
T --result type.

6.3. To execute query
Query i/f method
public List<T> getResultList() throws HibernateException
--Rets list of PERSISTENT entities.

7. Objective : Display all users registered between strt date n end date & under a specific role
I/P : begin dt , end date , role
eg : sql = select * from users where reg_date between ? and ? and user_role=?
 jpql : select u form User u where u.regDate between :strt_date and :end_date and u.userRole=:rl

Passing named IN params to the query
Query i/f method
public Query<T> setParameter(String paramName,Object value) throws HibernateException.

8 . User Login (Lab work)
i/p : email n password
o/p User details with success mesg or invalid login mesg
Hint : Query API : public <T> T getSingleResult() throws HibernateException

9. Objective : Display all user names registered under a specific role
jpql = select u.firstName from User u where u.userRole=:rl

10. Objective : Display all user's last name,reg amount,reg date registered after a particular n under a specific role
String jpql="select u.lastName,u.regAmount,u.regDate from User u....";
List<Object[]> list=session.createQuery(jpql,Object[].class).setParameter("start", startDate).setParameter("end", endDate).setParameter("rl", userRole).getResultList();
In Tester :
list.forEach(o -> sop(o[0]+" "+o[1]+" "+o[2]));

INSTEAD use a constructor expression
eg :
jpql = "select new pojos.User(lastName,regAmount,regDate) from User u where u.regDate between :strt and :end and userRole=:rl";

Pre requisite : MATCHING constr in POJO class

11. Update
Objective :
1. Change password
i/p --email , old password , new pass
o/p : mesg indicating success or a failure

12. Apply discount to reg amount , for all users , reged before a specific date. (Bulk update)
i/p -- discount amt, reg date

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

12.1 Query API
public int executeUpdate() throws HibernateException
--use case --DML
12.2 Session API
public Query<T> createQuery(String jpql) throws HibernateException
jpql -- DML

-------------------PENDING-----------------------------

13. Un subscribe user
i/p user id
o/p user details removed from DB
Hint : Session API : get --> null chekcking --> not null ---> delete

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


15. Save n restore images to / from DB
FileUtils from Apache commons-io
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
            <dependency>
                  <groupId>commons-io</groupId>
                  <artifactId>commons-io</artifactId>
                  <version>2.11.0</version>
            </dependency>
Methods

1. public static byte[] readFileToByteArray(File file)
                                    throws IOException
Reads the contents of a file into a byte array. The file is always closed.

2. public static void writeByteArrayToFile(File file,
                        byte[] data)
                                    throws IOException
Writes a byte array to a file creating the file if it does not exist.
```

Hibernate API

0. org.hibernate.SessionFactory API
getCurrentSession vs openSession

public Session openSession() throws HibernateException
opens NEW session from SF,which has to be explicitely closed by programmer.

OR
public Session getCurrentSession() throws HibernateException
Opens new session , if one doesn't exist , otherwise continues with the exisitng one.
Gets automatically closed upon Tx boundary or thread over(since current session is bound to current thread --mentioned in hibernate.cfg.xml property --- current_session_context_class ---thread)

1. CRUD logic (save method)
API (method) of org.hibernate.Session
public Serializable save(Object o) throws HibernateException

I/P ---transient POJO ref.
save() method auto persists transient POJO on the DB(upon committing tx) & returns unique serializable ID generated by hib frmwork, if @GeneratedValue is used for id.

2. Hibernate session API -- for data retrieval by PK
API (method) of org.hibernate.Session
public <T> T get(Class<T> c,Serializable id) throws HibernateException
T -- type of POJO
Returns --- null -- if id is not found.
returns PERSISTENT pojo ref if id is found.

Usage of Hibernate Session API's get()
int id=101;
BookPOJO b1=hibSession.get(BookPOJO.class,id);//int -----Integer(auto boxing) ---- upcasting------Serializable
OR
BookPOJO b1=hibSession.get(Class.forName("pojos.BookPOJO"),id);

2. Display all books info :

using HQL -- Hibernate Query Language --- Objectified version of SQL --- where table names will be replaced by POJO class names & table col names will replaced by POJO property names.
(JPA--- Java Persistence API compliant syntax --- JPQL)
eg --- SQL -- select * from books
HQL --- "from BookPOJO"
eg JPQL -- "select b from BookPOJO b"

2.1 Create Query Object --- from Session i/f
<T> org.hibernate.query.Query<T> createQuery(String hql/jpql,Class<T> resultType)

eg : Query<Book> q=hs.createQuery(hql/jpql,Book.class);

2.2. Execute query to get List of selected PERSISTENT POJOs
API of org.hibernate.query.Query i/f
(Taken from javax.persistence.TypedQuery<T>)

List<T> getResultList()

Execute a SELECT query and return the query results as a generic List<T>.
T -- type of POJO / Result

eg : hs,tx
String jpql="select b from Book b";
try {
    List<Book> l1=hs.createQuery(jpql,Book.class).getResultList();
}

 Usage ---
String hql="select b from BookPOJO b";
List<BookPOJO> l1=hibSession.createQuery(hql).getResultList();

3. Passing IN params to query. & execute it.
Objective : Display all books from specified author , with price < specified price.
API from org.hibernate.query.Query i/f

Query<R> setParameter(String name,Object value)

Bind a named query parameter using its inferred Type.
name -- query param name
value -- param value.I

String hql="select b from BookPOJO b where b.price < :sp_price and b.author
= :sp_auth";

How to set IN params ?
org.hibernate.query.Query<T> API
public Query<T> setPrameter(String pName,Object val)

List<Book> l1 =
hibSession.createQuery(hql,Book.class).setParameter("sp_price",user_price).setParam
eter("sp_auth",user_auth).getResultList();

Objective --Offer discount on all old books
i/p -- date , disc amt

4. Updating POJOs --- Can be done either with select followed by update or ONLY
with update queries(following is eg of 2nd option
commonly known as bulk update)
Objective : Reduce price of all books with author=specified author , published
before a sepecific date
String jpql = "update BookPOJO b set b.price = b.price - :disc where b.author = :au
and b.publishDate < :dt ";
set named In params
exec it (executeUpdate) ---
int updateCount= hs.createQuery(jpql).setParameter("disc", disc).setParameter("dt",
d1).executeUpdate();

---This approach is typically NOT recommended often, since it bypasses L1 cache .
Cascading is not supported. Doesn't support optismistic locking directly.
Use case -- for bulk updations to be performed on a standalone (un related) table)

5. Delete operations.
API of org.hibernate.Session

--void delete(Object o) throws HibernateException
i/p --persistent POJO ref

---POJO is marked for removal , corresponding row from DB will be deleted after comitting tx & will be removed form entity cache(l1 cache) closing of session.

OR
5.5
One can use directly "delete HQL" & perform deletions.
eg
int deletedRows = hibSession.createQuery ("delete Subscription s where s.subscriptionDate < :today").setParameter ("today", new Date ()).executeUpdate ();

Detailed API
0. SessionFactory API
getCurrentSession vs openSession

public Session openSession() throws HibernateExc
opens new session from SF,which has to be explicitely closed by prog.

public Session getCurrentSession() throws HibernateExc
Opens new session , if one doesn't exist , otherwise continues with the exisitng one.
Gets automatically closed upon Tx boundary or thread over(since current session is bound to current thread --mentioned in hibernate.cfg.xml property --- current_session_context_class ---thread)

Assumption :
Data Type of ID : Wrapper class(eg : Integer/Long) +
@GeneratedValue(strategy=IDENTITY)

1. Testing core api
persist ---
public void persist(Object transientRef)

if u give some non-null id (meaning non un saved value i.e non default) --(existing or non-existing) while calling persist(ref) --gives exc
org.hibernate.PersistentObjectException: detached entity passed to persist:
why its taken as detached ? ---non null id.

2.

public Serializable save(Object ref)
save --- if u give some non-null id(existing or non-existing in DB) while calling save(ref) --doesn't give any exc.
Ignores your passed id & creates its own id & inserts a row.

3. saveOrUpdate
public void saveOrUpdate(Object ref)
--either inserts/updates or throws exc.
null id -- fires insert (works as save)
non-null BUT existing id -- fires update (works as update)
non-null BUT non existing id -- throws StaleStateException --to indicate that we are trying to delete or update a row that does not exist.

3.5
merge

public Object merge(Object ref)
I/P -- either transient or detached POJO ref.
O/P --Rets PERSISTENT POJO ref.

null id -- fires insert (works as save)
non-null BUT existing id -- fires update (select , update)
non-null BUT non existing id -- no exc thrown --Ignores ur passed id & creates its own id & inserts a row.(select,insert)

4. get vs load
& LazyInitilalizationException.

5. update
Session API
public void update(Object detachedEntity)
Update the persistent instance with the identifier of the given detached instance.
I/P --detached POJO containing updated state.
Same POJO becomes persistent.

Exception associated :
1. org.hibernate.TransientObjectException: The given object has a null identifier:
i.e while calling update if u give null id. (transient ----X ---persistent via update)

2. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.
3.
org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session
eg : Employee e=session.get(100,Employee.class);//assume : id exists , persistent entity
session.update(e);
What will happen ? //thorws NonUniqueObjectException

6. public Object merge(Object ref)
Can Transition from transient -->persistent & detached --->persistent.
Regarding Hibernate merge
1. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().
2. API of Session
Object merge(Object object)
3.
Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
3.If there is no persistent instance currently associated with the session, it will be loaded.
4.Return the persistent instance. If the given instance is unsaved, save a copy of and return it as a newly persistent instance. The given instance does not become associated with the session.
5. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

7.public void evict(Object persistentPojoRef)

It detaches a particular persistent object
 from the session level cache(L1 cache)
(Remove this instance from the session cache. Changes to the instance will not be synchronized with the database.)

8.
void clear()

When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.
 But Databse Connection is not returned to connection pool.
(Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)

9. void close()

When close() is called on session object all
the persistent objects associated with the session object become detached(l1 cache is cleared) and also closes the Database Connection.

10. void flush()

When the object is in persistent state , whatever changes we made to the object state will be reflected in the databse only at the end of transaction.

BUT If we want to reflect the changes before the end of transaction
(i.e before commiting the transaction)
 call the flush method.
(Flushing is the process of synchronizing the underlying DB state with persistable state of session cache)

11. boolean contains(Object ref)

The method indicates whether the object is
associated with session or not.(i.e is it a part of l1 cache ?)

12.
void refresh(Object ref) -- ref --persistent or detached

This method is used to get the latest data from database and make corresponding modifications to the persistent object state.
(Re-reads the state of the given instance from the underlying database

API of org.hibernate.query.Query<T>

1. Iterator iterate() throws HibernateException

Return the query results as an Iterator. If the query contains multiple results per row, the results are returned Object[].

Entities returned --- in lazy manner

Pagination

2. Query setMaxResults(int maxResults)

Set the maximum number of rows to retrieve. If not set, there is no limit to the number of rows retrieved.

3. Query setFirstResult(int firstResult)

Set the first row to retrieve. If not set, rows will be retrieved beginnning from row 0. (NOTE row num starts from 0)

eg --- List<CustomerPOJO> l1=sess.createQuery("select c from CustomerPOJO c").setFirstResult(30).setMaxResults(10).list();

4.How to count rows & use it in pagination techniques?

```
int pageSize = 10;
String countQ = "Select count (f.id) from Foo f";
Query countQuery = session.createQuery(countQ);
Long countResults = (Long) countQuery.uniqueResult();
int lastPageNumber = (int) ((countResults / pageSize) + 1);

Query selectQuery = session.createQuery("From Foo");
selectQuery.setFirstResult((lastPageNumber - 1) * pageSize);
selectQuery.setMaxResults(pageSize);
List<Foo> lastPage = selectQuery.list();
```

5. org.hibernate.query.Query API

<T> T getSingleResult()

Executes a SELECT query that returns a single result.

Returns: Returns a single instance(persistent) that matches the query.

Throws:
NoResultException - if there is no result
NonUniqueResultException - if more than one result
IllegalStateException - if called for a JPQL UPDATE or DELETE statement

6. How to get Scrollable Result from Query?

ScrollableResults scroll(ScrollMode scrollMode) throws HibernateException

Return the query results as ScrollableResults. The scrollability of the returned results depends upon JDBC driver support for scrollable ResultSets.

Then can use methods of ScrollableResults ---first,next,last,scroll(n) .

7. How to create Named query from Session i/f?
What is a named query ?
Its a technique to group the HQL statements in single location(typically in POJOS)

and lately refer them by some name whenever need to use them. It helps largely in code cleanup because these HQL statements are no longer scattered in whole code.

Fail fast: Their syntax is checked when the session factory is created, making the application fail fast in case of an error.
Reusable: They can be accessed and used from several places which increase re-usability.

eg : In POJO class, at class level , one can declare Named Queries

```
@Entity
@NamedQueries
({@NamedQuery(name=DepartmentEntity.GET_DEPARTMENT_BY_ID, query="select d from
DepartmentEntity d where d.id = :id")}
public class Department{....}
```

Usgae

```
Department d1 = (Department)
session.getNamedQuery(DepartmentEntity.GET_DEPARTMENT_BY_ID).setInteger("id", 1);
```

8. How to invoke native sql from hibernate?

```
Query q=hs.createSQLQuery("select * from books").addEntity(BookPOJO.class);
 l1 = q.list();
```

9. Hibernate Criteria API
A powerful and elegent alternative to HQL
Well adapted for dynamic search functionalities where complex Hibernate queries have to be generated 'on-the-fly'.

Typical steps are -- Create a criteria for POJO, add restrictions , projections ,add order & then fire query(via list() or uniqueResult())

10. For composite primary key
Rules on prim key class
Annotation -- @Embeddable (& NOT @Entity)
Must be Serializable.
Must implement hashCode & equals as per general contract.

In Owning Entity class
Add usual annotation -- @Id.

1.1 Testing core api
persist ---

```
public void persist(Object transientRef)
```

if u give some non-null id (existing or non-existing) while calling persist(ref) -- gives exc
org.hibernate.PersistentObjectException: detached entity passed to persist:
why its taken as detached ? ---non null id.

2.

```
public Serializable save(Object ref)
```

save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exception
Ignores your passed id & creates its own id & inserts a row.

3. saveOrUpdate

public void saveOrUpdate(Object ref)
--either inserts/updates or throws exc.
null id -- fires insert (works as save)
non-null BUT existing id -- fires update (works as update)
non-null BUT non existing id -- throws StaleStateException --to indicate that we are trying to delete or update a row that does not exist.

3.5
merge
public Object merge(Object ref)
I/P -- either transient or detached POJO ref.
O/P --Rets PERSISTENT POJO ref.

null id -- fires insert (works as save)
non-null BUT existing id -- fires update (select , update)
non-null BUT non existing id -- no exc thrown --Ignores ur passed id & creates its own id & inserts a row.(select,insert)

4. get vs load
& LazyInitilalizationException.

5. update
Session API
public void update(Object object)
Update the persistent instance with the identifier of the given detached instance.
I/P --detached POJO containing updated state.
Same POJO becomes persistent.

Exception associated :
1. org.hibernate.TransientObjectException: The given object has a null identifier:
i.e while calling update if u give null id. (transient ----X ---persistent via update)

2. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.
3.
org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session

6. public Object merge(Object ref)
Can Transition from transient -->persistent & detached --->persistent.
Regarding Hibernate merge
1. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().
2. API of Session
Object merge(Object object)
3.
Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
3.If there is no persistent instance currently associated with the session, it will be loaded.
4.Return the persistent instance. If the given instance is unsaved, save a copy of and return it as a newly persistent instance. The given instance does not become associated with the session.
5. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

7.public void evict(Object persistentPojoRef)

It detaches a particular persistent object
detaches or disassociates from the session level cache(L1 cache)
(Remove this instance from the session cache. Changes to the instance will not be synchronized with the database.)

8.
void clear()

When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.
But Databse Connection is not returned to connection pool.
(Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)

9. void close()

When close() is called on session object all
the persistent objects associated with the session object become detached(l1 cache is cleared) and also closes the Database Connection.

10. void flush()

When the object is in persistent state , whatever changes we made to the object state will be reflected in the databse only at the end of transaction.

BUT If we want to reflect the changes before the end of transaction
(i.e before commiting the transaction)
call the flush method.
(Flushing is the process of synchronizing the underlying DB state with persistable state of session cache)

11. boolean contains(Object ref)

The method indicates whether the object is
associated with session or not.(i.e is it a part of l1 cache ?)

12.
void refresh(Object ref) -- ref --persistent or detached

This method is used to get the latest data from database and make
corresponding modifications to the persistent object state.
(Re-reads the state of the given instance from the underlying database

Advanced Hibernate

Relationship between Entity n Entity
(Inheritance , Association : HAS-A)

Types of associations

one-to-one
one-to-many
many-to-one
many-to-many

Objective --Using one-to-many & many-to-one association between entities

eg : Course 1 <---->* Student
Different type of relationships between entities
One To One
One To Many
Many To One
Many To Many

1. One To Many bi directional relationship between the entities
JPA Annotations : @OneToMany & @ManyToOne

eg : Course 1 <----->* Student
Table Relationship
courses Table columns : id,title, start_date , end_date , fees , capacity
students Table columns : id,name,email + Foreign Key(FK) : course_id

Since courses table has a OneToMany relationship with the students table , a single course row can be referenced by multiple student rows.

The course_id column in the students table , maps this relationship via a foreign key that references the primary key of the courses table.

Since you can't insert a student record , w/o course record
parent-side (@OneToMany) : course
child-side (@ManyToOne) : student

The @ManyToOne association is responsible for synchronizing the foreign key column with the Persistence Context (the First Level Cache).

As a thumb rule (for perfomance benefits) : DO NOT use uni directional @OneToMany associations

Owning side of the association
The side having the join column in its table is called the owning side or the owner of the relationship.

Non owning (inverse side)
The side that does not have the join column is called the non#owning or inverse side.

Entities involved :
Course Entity
Student Entity

Description
Course : one , parent , inverse

Student : many , child , owning side (FK)

Best Practices to code a bidirectional @OneToMany association
eg : Course 1 <------>* Student
Entity Relationships
Course POJO properties : id,title, startDate , endDate , fees +
@OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL,orphanRemoval=true)
private List<Student> students=new ArrayList<>();
Note : Always init collection to empty one , to avoid null pointer excpetion

Student POJO properties : id,name,email +
@ManyToOne
@JoinColumn(name="course_id")
private Course selectedCourse

Detailed explanation
1. Add Suitable mapping annotations : @OneToMany & @ManyToOne
otherwise JPA / Hibernate throws MappingException

2. Add mappedBy attribute in the inverse side of the association

What is mappedBy & when it's mandatory?
Mandatory only in case of bi-dir associations
It's attribute of the @OneToMany / @ManyToMany / @OneToOne annotation.

What will happen if you don't add this attribute , in case of one-to-many
 Additional table (un necessary for the relationship mapping) gets created

It MUST appear in the inverse side of the association.
What should be value of mappedBy ?
name of the association property as it appears in the owning side.

eg : In Course POJO : inverse side
@OneToMany(mappedBy="selectedCourse")
public List<Student> getStudents() {..}

3. Use @JoinColumn to Specify the Join Column Name (FK column)
Use it to override hibernate's default naming strategy for column names.

4. Cascade from Parent-Side to Child-Side
If you don't add cascade option : what will happen ?
eg : When try to save Course object, with multiple students, insert query gets
fired only on courses table.
Reason -- default cascade type = none
Solution --Add suitable cascade type & observe.
eg : @OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL)
public List<Student> getStudents(){...}

5. What will happen if simply add student reference into the list?
eg :
eg : Course newCourse=new Course(....);
newCourse.getStudents().add(newStudent1);
newCourse.getStudents().add(newStudent2);
newCourse.getStudents().add(newStudent3);
session.persist(newCourse);

Ans : 1 record will be inserted into courses table. Thanks to cascade option , 3 records will be inserted into students table. BUT value of FK will be null.
Why : No linking from child ----> parent &

Which is the best way to establish bi-dir linking (As per THE founder of Hibernate : Gavin King)
Add helper methods in the parent side of the POJO
eg : In Course POJO

```
public void addStudent(Student s)
{
  students.add(s);
  s.setSelectedCourse(this);
}
```

For removing bi dir link

```
public void removeStudent(Student s)
{
  students.remove(s);
  s.setSelectedCourse(null);
}
```

Above approach is recommended to keep both sides of the association in sync.

6. Set orphanRemoval on the Parent-Side
Setting orphanRemoval on the parent-side guarantees the removal of children without references.
It is good for cleaning up dependent objects that should not exist without a reference from an owner object.

eg : Cancel Student admission

Excellent reference : https://vladmihalcea.com/orphanremoval-jpa-hibernate/

Objectives :

1. Objective : Launch new course
DAO
ICourseDao

```
String launchCourse(Course c);
```

2. Admit student
I/p -- student name, email, course name
o/p -- student details inserted + linked with FK

DAO --IStudentDao

```
String admitNewStudent(String courseName,Student s);
```

3. Cancel Course
i/p : course id

4. Objective :
Launch a new course , having no of students
eg : Course : hibernate.....
s1,s2,s3,s4 : have already paid the fees for the course
Expected o/p : 1 rec should be inserted in course table & 4 recs in student tbl + linked with FK

If you don't add cascade option : problem observed
When try to save Course object, with multiple students, insert query gets fired only on courses table.
Reason -- def cascade type = none
Solution --Add suitable cascade type & observe.
eg : @OneToMany(mappedBy="selectedCourse",
cascade=CascadeType.ALL)
public List<Student> getStudents(){...}

5. Cancel Admission
String cancelAdmission(String courseName,int studentId);
Hint : use helper method.
Any problems ?????
Solution : orphanRemoval

6 Get Course Details
i/p : course name

Display course details

Display enrolled student details
Any problems observed ?

Problem associated with one to many
org.hibernate.LazyInitializationException
Trigger : GetCourseDetails : while accessing the Student details

WHY ?

Hibernate follows default fetching policies for different types of associations
one-to-one : EAGER
one-to-many : LAZY
many-to-one : EAGER
many-to-many : LAZY

one-to-many : LAZY
Meaning : If you try to fetch details of one side(eg : Course) , will it fetch auto details of many side ?
NO (i.e select query will be fired only on courses table)
Why ? : for performance

When will hibernate throw LazyInitializationException ?
Any time you are trying to access un-fetched data from DB , in a detached manner(outside the session scope)
cases : one-to-many
many-many
session's load

un fetched data : i.e student list in Course obj : represented by : proxy (substitution) : collection of proxies
proxy => un fetched data from DB

Solutions
1. Change the fetching policy of hibernate for one-to-many to : EAGER

eg :
```
@OneToMany(mappedBy = "selectedCourse",cascade =
CascadeType.ALL,fetch=FetchType.EAGER)
      private List<Student> students=new ArrayList<>();
```

Is it recommneded soln : NO (since even if you just want to access one side details , hib will fire query on many side) --will lead to worst performance.

2.
```
@OneToMany(mappedBy = "selectedCourse",cascade = CascadeType.ALL)
      private List<Student> students=new ArrayList<>();
```
Solution : Access the size of the collection within session scope : soln will be applied in DAO layer

Dis Adv : Hibernate fires multiple queries to get the complete details

3. How to fetch the complete details , in a single join query ?
Using "join fetch" keyword in JPQL
```
String jpql = "select c from Course c join fetch c.students where c.title=:ti";
```

Another trigger for lazy init exception
: Session's API
load.

DAY 9

Revise
POJO states : transient , persistent , detached
Session API : save , get , JPQL , createQuery , setParameter,
getResultList,getSingleResult
persistent vs detached

Continue with Session API
(refer to : "\day8-data\day8_help\readmes\hibernate session api.txt")

Objectives
1. save vs persist

2. Un subscribe user
i/p user id
o/p user details removed from DB

3. Lab work (Bulk Deletion)
Objective --delete user details for those users reg date > dt.
via Bulk delete
String jpql="delete from User u where u.regDate > :dt";

4. Save n restore images to / from DB
FileUtils from Apache commons-io

```
<!-- https://mvnrepository.com/artifact/commons-io/commons-io -->
            <dependency>
                  <groupId>commons-io</groupId>
                  <artifactId>commons-io</artifactId>
                  <version>2.11.0</version>
            </dependency>
```

Methods

1. public static byte[] readFileToByteArray(File file)
                                    throws IOException
Reads the contents of a file into a byte array. The file is always closed.

2. public static void writeByteArrayToFile(File file,
                        byte[] data)
                                    throws IOException
Writes a byte array to a file creating the file if it does not exist.

----------Basic Hibernate Over----------

Handling E-R in Hibernate
one-one
one-many
many-one
many-many

In a Typical eCommerce Scenario :
Admin Role
Add A Category
Add Products under the Category
Update Category
Update Product
Delete Category

Delete Product
Update order status (after the payment succeeds)

Customer Role
View Categories
View All Products
View Products By Category
Add to Cart
View Cart
Remove Product from Cart
Place Order --Supply Delivery Address

Objective : one-to-many bi dir assocation between Category 1<----->* Product
Tables involved :

1. categories :category id(PK) , categoryName ,description
2. products: product_id(PK), name,price,description,inStock + category_id(FK --> references category id from categories)

POJOs (Entities)
Category : categoryId,name,description

Product : productId, name ,price,description,inStock +
private Category productCategory;

Above will be the example of : uni dir relatioship

Can you replace above by a bi-dir relationship ? YES

Bi dir relationship(association) between entities
Category 1<----->* Product

1. Product : productId, name ,price,description,inStock +
private Category productCategory;

2. Category : categoryId, categoryName, desc +
List<Product> products=new AL<>(); //as per Gavin King's suggestion , always init the collection

POJO (Entities)
parent , one side , inverse side

Category : one , parent, inverse side(=non owning side i.e DOES NOT have FK field)

Product : many , child , owning side of the association(FK : category_id)

Category : categoryId,name,description +
private List<Product> products=new ArrayList<>();

//many side of the asso : OWNING side
Product : productId, name ,price,description,inStock +

private Category productCategory;

Technical terms :
parent : category
child : product
one : category
many : product
owning side of the asso. : product(since mapping info : FK , appears here)
non-owning side of the asso : category

JPA inheritance
1. @MappedSuperclass
Class level annotation , to be added on abstract or concrete super class
Hibernate will NOT generate any table for it.
One can add all common fields in this class
All other entities can extend n inherit from the common super class

2. Association between entities
@OneToMany
@ManyToOne
@JoinColumn
mappedBy

1. Add a new category
i/p : category name , description

2. Add Product to the category
i/p : category id , product details (name,price,desc) : Product instance
Steps
validate exp date --success
get category from it's id(get)
if valid ---establish bi-dir relationship --
session.persist(product);

2.1 (understand bi-dir relationship)
2.2 Helper methods

Entity Types :

1. If an object has its own database identity (primary key value) then it's type is Entity Type.
2. An entity has its own lifecycle. It may exist independently of any other entity.
3. An object reference to an entity instance is persisted as a reference in the database (a foreign key value).

eg : College is an Entity Type. It has it's own database identity (It has primary key).

Value Types :

1. If an object don't have its own database identity (no primary key value) then it's type is Value Type.
2. Value Type object belongs to an Entity Type Object.
3. It's embedded in the owning entity and it represents the table column in the database.
4. The lifespan of a value type instance is bounded by the lifespan of the owning entity instance.

Different types of Value Types

Basic, Composite, Collection Value Types :

1. Basic Value Types :

Basic value types are : they map a single database value (column) to a single, non-aggregated Java type.
Hibernate provides a number of built-in basic types.
String, Character, Boolean, Integer, Long, Byte, … etc.

2.

Composite Value Types :

In JPA composite types also called Embedded Types. Hibernate traditionally called them Components.
2.1 Composite Value type looks like exactly an Entity, but does not own lifecycle and identifier.

Annotations Used

1. @Embeddable :

Defines a class whose instances are stored as an intrinsic part of an owning entity and share the identity of the entity. Each of the persistent properties or fields of the embedded object is mapped to the database table for the entity. It doesn't have own identifier.
eg : Address is eg of Embeddable
Student HAS-A Address(eg of Composition --i.e Address can't exist w/o its owning Entity i.e Student)
College HAS-A Address (eg of Composition --i.e Address can't exist w/o its owning Entity i.e College)
BUT Student will have its own copy of Address & so will College(i.e Value Types don't support shared reference)

2. @Embedded :

Specifies a persistent field or property of an entity whose value is an instance of an embeddable class. The embeddable class must be annotated as Embeddable.
eg : Address is embedded in College and User Objects.

3. @AttributesOverride :

Used to override the mapping of a Basic (whether explicit or default) property or field or Id property or field.

In Database tables observe the column names. Student table having STREET_ADDRESS column and College table having STREET column. These two columns should map with same Address field streetAddress.
@AttributeOverride gives solution for this.
To override multiple column names for the same field use @AtributeOverrides annotation.
eg : In Student class :

```
@Embedded
 @AttributeOverride(name="streetAddress",
column=@Column(name="STREET_ADDRESS"))
private Address address;
```

where , name --POJO property name in Address class

3.
Collection Value Types :
Hibernate allows to persist collections.
But Collection value Types can be either collection of Basic value types, Composite types and custom types.
eg :
Collection mapping means mapping group of values to the single field or property. But we can't store list of values in single table column in database. It has to be done in a separate table.

eg : Collection of embeddables

```
@ElementCollection
      @CollectionTable(name="CONTACT_ADDRESS",
joinColumns=@JoinColumn(name="USER_ID"))
      @AttributeOverride(name="streetAddress",
column=@Column(name="STREET_ADDRESS"))
      private List<ContactAddress> address;
```

eg : collection of basic type

```
      @ElementCollection
      @CollectionTable(name="Contacts",
joinColumns=@JoinColumn(name="ID"))
      @Column(name="CONTACT_NO")
      private Collection<String> contacts;
```

Revise
Inheritance in hibernate
@MappedSuperclass : class level JPA annotation , to tell hib. -- following class is super class to all other entities n should not create asso. table !

one to many bi dir association with entities (weaker form of association : aggregation)
eg : Category 1<----->* Product

one/many , parent/child owning/inverse

Category : one , parent , inverse (non owning --it does NOT contain any physical mapping : FK)
Product : many , child , owning (FK)

Which is the additional property in Category POJO ? : ...

+ private List<Product> products=new ArrayList<>();

Which is the additional property in Product POJO ? ....
+ private Category productCategory;

What will happen if you don't add any annotations ? throws MappingExc

Which annotations ?In Category class
@OneToMany
private List<Product> products=new ArrayList<>();

Product POJO ....
@ManyToOne
 private Category productCategory;

What will happen if you don't add mappedBy element ? hib creates an extra table : link table (product_id n category_id : FKs)

Why ? Sice you have not yet supplied any info regarding owning side n inverse side.

When adding "mappedBy" mandatory ? in a bi-dir asso. only !

Where it appears ? inverse side (i.e Category)

What should be it's value ?
Name of the asso property as it appears in the owning side.
eg :
@OneToMany(mappedBy="productCategory")
private List<Product> products=new ArrayList<>();

Product POJO ....
@ManyToOne
private Category productCategory;
In above case who will decide the name of FK col ? hibernate
How to customize it ?
@JoinColumn(name="category_id") //FK col name refs ---> PK of categories table.

1. Add a new category
i/p : category name , description

2. Add Product to the category
i/p : category id , product details (name,price,desc)
steps in ProductDaoImpl
get category from it's id (get)
null chking --if not null -->
cat.addProduct(newProduct);
session.save(newProduct);

2.1 (understand how to establish bi-dir relationship)
2.2 Helper methods

3. Add a new category with products (w/o cascading)
Observe n conclude
w/o cascading :
establish asso.
save category
save products

Then add cascading
Add a property cascade in @OneToMany annotation to support cascading of the changes from parent --> child entity.

4. Remove a category
i/p : category name
Confirm cascading
Objective : String deleteCategory(String categoryName);
jpql : select c from Category c where c.categoryName=:nm
getSingleResult --->no exc --> session.delete(category)

5. Remove a product from a Category
i/p : category id , product id

Steps
get category n product from it's id : session.get
null chking --not null
cat.removeProduct(p);//helper method

Any problem noticed ? YES , hib simply de linked the entities (i.e FK : null) , BUT child rec wasn't deleted
Solution : add new property to @OneToMany : orphanRemoval=true

5. Display category details

i/p : category name

6. Display category n product details
i/p : category name
IMPORTANT
Any problem noticed : org.hibernate.LazyInitializationException
Cause : Any time you are trying to access un-fetched data , in a detached mode(i.e outside session scope) , hib throws the exc.

WHY ?

Hibernate follows default fetching policies for different types of associations
one-to-one : EAGER
one-to-many : LAZY
many-to-one : EAGER
many-to-many : LAZY

one-to-many : LAZY
Meaning : If you try to fetch details of one side(eg : Category) , will it fetch auto details of many side(i.e : Product) ?
NO (i.e select query will be fired only on categories table)
Why ? : for performance

When will hibernate throw LazyInitializationException ?
Any time you are trying to access un-fetched data from DB , in a detached manner(outside the session scope)
cases : one-to-many
many-many
session's load

un fetched data : i.e product list in Category obj : represented by : proxy (substitution) : collection of proxies
proxy => un fetched data from DB

Solutions
1. Change the fetching policy of hibernate for one-to-many to : EAGER
eg :

Is it recommneded soln ?

NO (since even if you just want to access one side details , hib will fire query on many side) --will lead to worst performance.
Use case : when the size of many is small !
(eg : BankCustomer ---> BankAccount)

2. Better Solution

Solution : Access the size of the collection within session scope : soln will be applied in DAO layer

Dis Adv : Hibernate fires multiple queries to get the complete details

3. How to fetch the complete details , in a single join query ?
Using "join fetch" keyword in JPQL

```
String jpql = "select c from Category c join fetch c.products where
c.category=:nm"
--inner join

String jpql = "select c from Category c left outer join fetch c.products
where c.category=:nm"
--left outer join
```

7. Register New User
i/p user details
o/p message indicating success or failure
Exists already (project : day9.1)

8. Use case :
User login
i/p email , password
In case of successful login , check the role --if cutomer role
--check if cart exists for the logged in customer . If no --create one .

9. What will be the relationship between User n Cart ??? : one to one asso.

User 1<----->1 ShoppingCart

ShoppingCart Entity extends BaseEntity
state : totalItems,totalCartPrice,createdOn,updatedOn,
Additional annotations : @CreationTimestamp @UpdateTimestamp

How to establish bi-dir one-to-one relationship between User 1<---->1 ShoppingCart

First identify
one, parent/child owning/inverse

User : one , parent , inverse
ShoppingCart : one , child , owning

Which additional properties ?
In User class :

```
@OneToOne(mappedBy="cartOwner",cascade=CascasdeTYpe.ALL,orphanRemoval=tru
e)
private ShoppingCart cart;
```

In ShoppingCart

```
...+
@OneToOne
@JoinColumn(name="customer_id",nullable=false)
private user cartOwner;
```

Which annotations ? Mentioned above !

Add <mapping> entry n run TestHibernate , check if the tables are created properly ?

10. What will be the relationship between Cart n Product?

Can 1 cart contain multiple products ?? YES
Can 1 product be added to multiple carts for different customers ?? YES

So it's many-to-many
What will happen if you use @ManyToMany annotation ?
(Try it out !)

Result : a join table will be created by hibernate , having composite PK
eg : cart_items
2 columns only : cart_id n product_id
(Having FK constraints)
cart_id : FK --> references PK of carts table
product_id : FK --> references PK of products table

Is it suitable if you want to maintain additional columns ? : quantity n total_price

Solution : Create a separate entity CartItem (representing DB tables)
What will the columns ? id (PK), quantity , total_price, cart_id(FK) , product_id(FK)
Which properties : quantity , totalPrice , cart , product

Establish following relationships , using the annotations

ShoppingCart 1<----->* CartItem

CartItem 1----->1 Product

----------PENDING----------

11. Enter value types
Customer HAS-A Adhar Card

11.1 Customer : entity : @Entity n @Id
Adhar Card : embeddable : @Embeddable
field : cardNo , date , loc

11.2 : Customer HAS-A hobbies
(collection of basic types)
Annotations : @ElementCollection n @CollectionTable

11.3 Customer HAS-A Card (can have multiple credit/debit cards)
(collection of embeddables)
Annotations : @ElementCollection n @CollectionTable

Good Articles/Books to refer to JPA Association Mapping

SPRING

WHY Spring ?

To simplify Java development.

What is spring?

Its a container & a framework both.

Spring is an open source framework since February 2003.
Created by Rod Johnson.(currently hosted on pivotal/vmware)

One line answer to what is spring--- spring is not a J2EE specification BUT its created to make developing complex J2EE applications easier.

Why learn one more frmwork --- when u already have EJB,Struts,Hibernate etc....

Spring helps you to

1.Build applications from plain old Java objects (POJOs) (known as spring beans)
2. Apply enterprise services non-invasively.
(w/o invasion means --- POJOs DONT implement or extend from spring APIs)
This capability applies to the Java SE programming model and to full and partial Java EE.

Examples of how you, as an application developer, can use the Spring platform advantage:

Make a Java method execute in a database transaction without having to deal with transaction APIs.

Make a local Java method a remote procedure without having to deal with remote APIs.

Make a local Java method an ORM operation without having to deal with overheads of ORM set up.

Make a local Java method a web service end point , without having to deal with JAX WS or JAX RS setups.

Simple answer to WHY Spring

Spring simplifies Java development.

Since above is a bold stmt -- to justify it --- there are main 4 reasons

Reasons ---it applies 4 key strategies

1. lightweight & min intrusive(POJOs not tied to spring) development with POJOs

2. Loose coupling thro' DI/IoC & with usage of i/f

3.Declarative prog(XML or anno or java config) + thro' Aspects & common conventions

4. Boilerplate code reduction thro aspects & templates.

7. Def. of Spring ----

Spring is a lightweight , dependency injection and aspect-oriented container and framework. works on Ioc(Inversion of control)

Meaning

7.1 Lightweight Lesser no of JARs
JavaBeans / POJOs in Spring-enabled application often have no dependencies on Spring-specific classes.
7.2 Dependency Injection Spring allows loose coupling through dependency injection (DI).

DI =instead of an object looking up dependencies from a container(in EJB from EJB container or in RMI from RMIRegistry) or creating its own dependency(in Fixed JDBC , conn = DM.getCn(....)) , the container gives the dependencies to the object at instantiation without waiting to be asked.

You can think of D.I as JNDI(Java naming & directory iterface -- Naming service) in reverse.

7.3 Aspect-oriented Spring supports aspect-oriented programming(AOP)

(AOP) allows separating application business logic from system services (such as auditing and transaction management,logging , security,transactions).
Application objects perform only business logic and nothing more.

They are not responsible for (or even aware of) other system concerns, such as eg : logging , transactions, or security

7.4
Why Spring is a Container ?

Spring is a container which manages
the lifecycle and configuration of application objects.(spring beans)
In Spring, using config XMLs or annotations or java config, you can declare - how to create each of your application objects(spring beans)
- how to configure them
- how they should be associated with each other.(collaboration/wiring/coupling=connecting dependencies with dependent objs)

7.5 Why Spring is a framework ?

Spring allows you to configure and compose complex applications from simpler components. In Spring, application objects are composed declaratively, typically in an XML file or using annotations

Spring also provides you with ready made implemetations of services like - transaction management, persistence framework,web mvc etc.

Spring is unique, for several reasons:

1. It helps you in important areas that many other popular frameworks don't. eg : readymade Hibernate templates.

2. Provides a way to manage your business objects - based on Dependeny injection(DI)

3.Spring is both comprehensive and modular.
Offers you lot many features, yet gives you choice to integrate layers one by one, test it & then add new features via new layers.

4 Spring is an ideal framework for test driven projects.

5. It is basically a one-stop shop, addressing most of the concerns of typical enterprise applications.

6.Using Spring one can centrally describe collaborating objects. From the earliest versions of Spring, there was an XML file that was used to describe the object graph.
Contents of XML ---- Consists of beans. Each bean element describes an object that will be created and given an id. Each property element describes a setter method on the object and the value that should be given to it. These setters are called for you by the Spring application container.

What is Spring ?

1. An open source framework since February 2003.
Created by Rod Johnson and described in his book Expert One-on-One: J2EE Design and Development.

Allows us to give capability of EJBs to plain JavaBeans without using an application server.

Any Java SE application can also use Spring to get simplicity, testability, and loose coupling.

2.Spring has been hosted on SourceForge.

3. Spring is a lightweight framework. Most of your Java classes will have no dependency on Spring. This means that you can easily transition your application from the Spring framework to any other frmwork. (Framework independence)

4. All Java applications that consist of multiple classes have inter-dependencies or coupling between classes. Spring helps us develop applications that minimize the negative effects of coupling and encourages the use of interfaces in application development.

5. Using interfaces in our applications to specify type helps make our applications easier to maintain and enhance later.

6. The Spring framework helps developers for separation of responsibilities.

eg scenario --
Think of a situation Your manager tells you to do your normal development work(eg - write stock trading appln) + write down everything you do and how long it takes you.

A better situation would be you do your normal work, but another person observes what you re doing and records it and measures how long it took.

Even better would be if you were totally unaware of that other person and that other person was able to also observe and record , not just yours but any other people s work and time.

That s separation of responsibilities. --- This is what spring offers u thro AOP

8 Spring framework Modules

9. Advantages of ApplicationContext over BeanFactory
9.1Application contexts resolve text messages, including support for internationalization (I18N).
9.2 Application contexts provide a generic way to load file resources, such as images.
9.3Application contexts can publish events to beans that are registered as listeners.

IOC -- rather a generic term
Inversion of Control (IoC) is an object-oriented programming practice where the object coupling(dependent obj bound with dependency) is bound at run time by an assembler object(eg spring container) and is typically not known at compile time using static analysis.

Unlike in traditional prog -- where dependent obj creates dependencies leading to tight coupling , container sets the dependencies (not US --not a prog or not a dependent obj) ---so its inversion of control

Dependency. injection=Ioc+dependency inversion
Why IoC or Dependency Injection (advantages of IoC)

* There is a decoupling of the execution of a certain task from implementation.
* Every module can focus on what it is designed for.
* Modules make no assumptions about what other systems do but rely on their contracts/specs (=i/f)
* Replacing modules has no side effect on other modules.

Inversion of Control is sometimes referred to as the "Hollywood Principle: Don't call us, we'll call you",

Dependency injection (DI) is the ability to inject dependencies. DI can help make your code architecturally pure. It aids in using a design by interface approach as well as test driven development by providing a consistent way to inject dependencies. For example a data access object (DAO) may need a database connection. Thus the DAO depends on the database connection. Instead of looking up the database connection with JNDI, you could inject it. or another eg is JMS -- conn factory or destination

One way to think about a DI container like Spring is to think of JNDI turned inside out. Instead of the objects looking up other objects that it needs to get its job done (dependencies), with DI the container injects those dependent objects. This is the so-called Hollywood

principle, you don't call us (lookup objects), we will call you (inject objects).

More on ApplicationContext

The instantiation of the ApplicationContext creates the container that consists of the objects defined in that XML file.
The purpose of this XML file is to create the beans and their relationship.
This XML file is then provided to the ApplicationContext instance, which creates a container with these beans and their object graphs along with relationships. The Spring container is simply a holder of the bean instances that were created from the XML file.
An API (getBean) is provided to query these beans and use them accordingly from our client application.

More on Ioc

IoC is also known as dependency injection (DI).

It is a process whereby objects define their dependencies,that is, the other objects they work with, only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method.

The container then injects those dependencies when it creates the bean.

This process is fundamentally the inverse,(hence the name Inversion of Control (IoC)), of the bean itself controlling the instantiation or location of its dependencies by using direct construction of classes, or a mechanism such as the Service Locator pattern.

The org.springframework.beans and org.springframework.context packages are the
basis for Spring Framework's IoC container. The BeanFactory interface provides an advanced
configuration mechanism capable of managing any type of object. ApplicationContext is a
sub-interface of BeanFactory. It adds easier integration with Spring's AOP features; message resource handling (for use in internationalization), event publication; and application-layer specific contexts such as the WebApplicationContext for use in web applications.

Spring Framework Modules

1. Core Container

It consists of the Core, Beans, Context, and Expression Language modules

1.1 The Core module provides the fundamental parts of the framework, including the IoC based upon Dependency Injection

1.2 The Bean module provides BeanFactory , which is a readymade implementation of the factory pattern.

1.3 The Context module is based upon the Core and Beans modules .It provides a way to access any spring beans(objects managed by Spring container) which are defined and configured. The ApplicationContext interface is the heart of Context module.

1.4 The SpEL module provides a powerful expression language for querying and manipulating an object graph at runtime.

2. Data Access/Integration module

The Data Access/Integration layer consists of the JDBC, ORM, OXM, JMS and Transaction modules

2.1 The JDBC module provides a JDBC-abstraction layer that removes the need for boilerplat JDBC related coding.

2.2 The ORM module provides integration layers for popular object-relational mapping APIs, including JPA, JDO, Hibernate, and iBatis.

2.3 The OXM module provides an abstraction layer that supports Object/XML mapping implementations for JAXB, XMLBeans, , XStream etc

2.4 JMS module contains features for producing and consuming messages sent across application components

2.5 The Transaction module supports programmatic and declarative transaction management for typically service layer or DAO layer spring beans.

3. Web

The Web layer consists of the Web, Web-MVC, Web-Socket, and Web-Portlet modules .

The Web module provides basic web integration features such as MVC support,multipart file-upload functionality ,init of IoC container using servlet listeners , web application context , I18N etc.

3.1 The Web-MVC module contains Spring's Model-View-Controller (MVC) implementation for web applications.

3.2 The Web-Socket module provides support for WebSocket-based, two-way communication between the client and the server in web applications.

3.3 The Web-Portlet module provides the MVC implementation to be used in a portlet environment

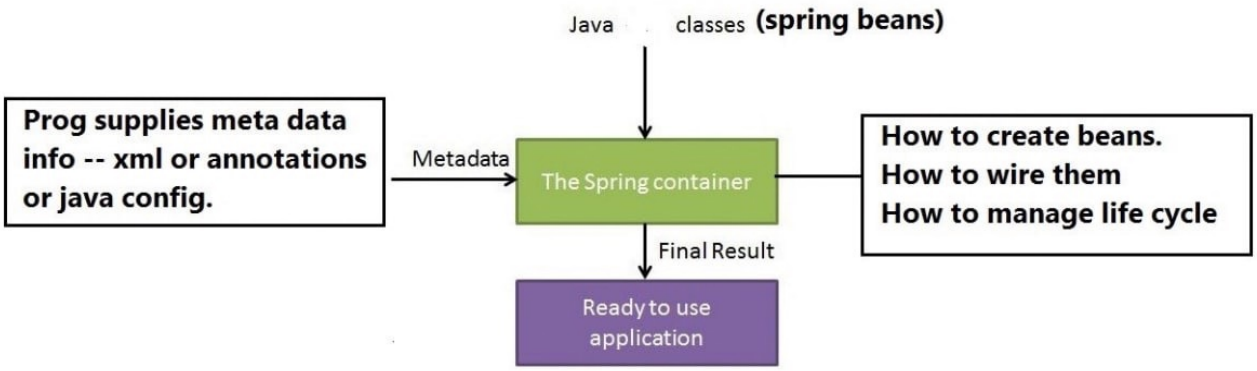
Other important Modules

1. The AOP module provides an aspect-oriented programming implementation allowing you to define method-interceptors and pointcuts to cleanly decouple code that implements functionality that should be separated.

2. The Instrumentation module

3. The Messaging module provides support for STOMP as the WebSocket sub-protocol to use in web socket applications.

4. The Test module supports the testing of Spring components with JUnit or TestNG frameworks.



Singleton Vs Prototype

1. Single instance of bean instance is shared across multiple demands made to Spring Container(SC)	1. SC will create a separate bean instance per demand(getBean)
2. Default loading policy = eager. Can be changed to lazy (using lazy-init)	2. Prototype beans are always loaded lazily (upon demand). lazy-init is not applicable
3. SC will automatically invoke destroy style method , upon ctx closing.	3. SC doesn't invoke destroy style method.
4. Created as stateless beans (Can't contain the conversational state of the client)	4. Stateful Beans. Can hold conversational state of the client.

Spring Container related API

org.springframework.beans.factory.BeanFactory --super i/f for starting SC(spring container)
BeanFactory =provider of spring beans (SC)
--i/f to SC
Not so suitable for web apps / I18N...

o.s.context.ApplicationContext --sub i/f of BeanFactory (will be used in Core Java + spring & web java +spring)

Implemented by o.s.context.support.ClasspathXmlApplicationContext
Meaning - SC started from meta data instructions --xml based , placed in run time classpath.
Constr : ClasspathXmlApplicationContext(String configFile) throws BeansException --unchecked exc.

How to get ready to use bean/s from SC ?
Inherited from BeanFactory i/f
public <T> T getBean(String beanId,Class<T> cls) throws BeansException

---

Wiring=dependency injection=Making dependencies available to dependent beans (POJO) @ runtime via 3rd Party (currently spring container)

Types Of Wiring (Depenency Injection) in Spring Framework

Explicit Wiring --Must supply setters/constrs/factory methods + XML configuration for beans

① Setter based D.I
1. Provide setters in dependent bean.
2. Provide <property name & value/ref > tag in xml config file

② Constructor based D.I
1. Provide parameterized constructor
2. Provide <constructor-arg name/type/index value/ref > tag in xml config file.

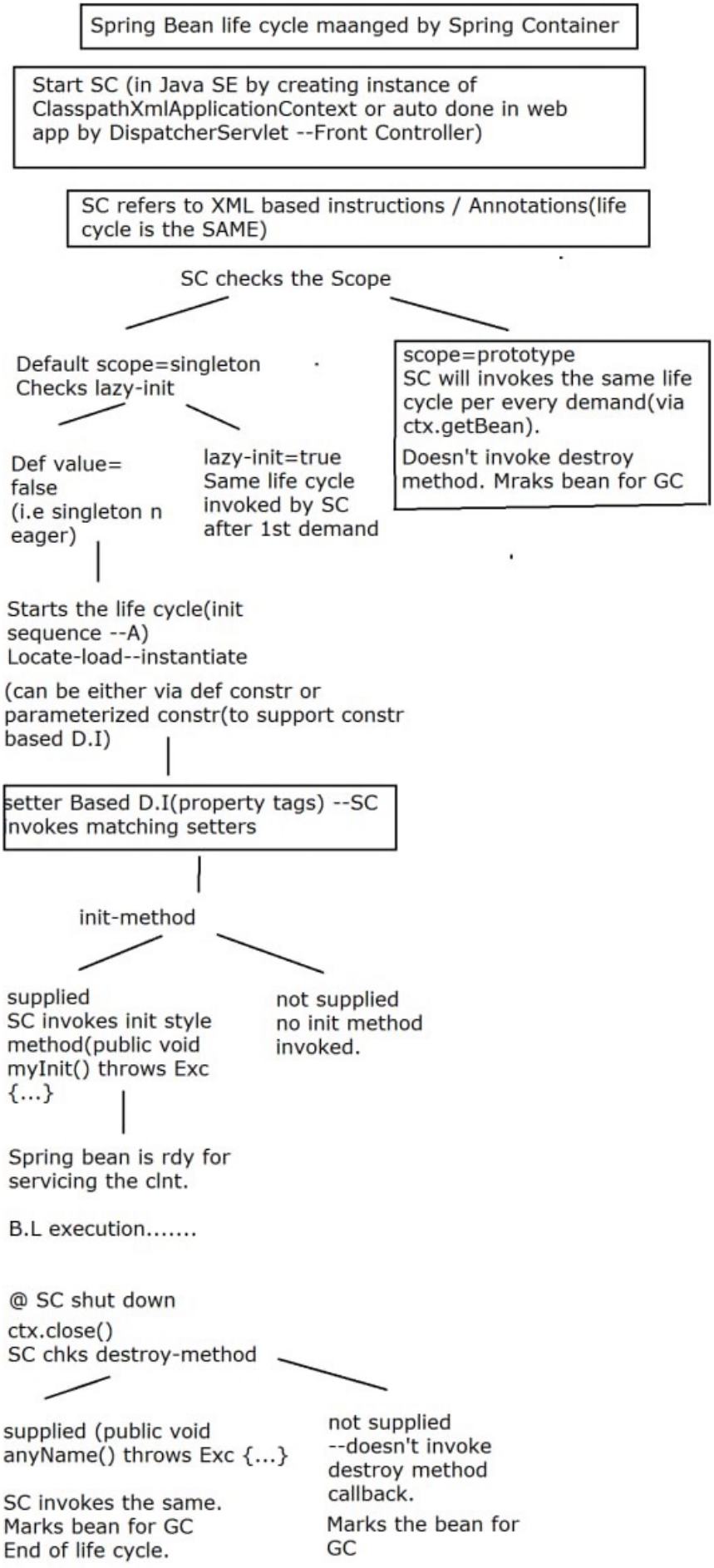
③ Factory Method Based D.I
1. Can provide private constructor & no setters in dependent bean.
2. Provide parameterized factory method (public static instance returning) in dependent bean.
3.Provide in <bean > tag factory-method name & provide <constructor-arg> tags one per method argument.

Implicit Wiring (auto wiring) --Must supply setters or constrs . No XML config. Just choose autowire mode

autowire=byName
1. Provide setters in dependent bean class.
2. No <property> tags required in xml config files.
3.Must match property setter names to dependency bean id.
4. No exception thrown if match not found.

autowire=byType
1. Provide setters in dependent bean class.
2. No <property> tags required in xml config file.
3. SC tries to match data type of the property to type of the dependency bean.
4. SC throws NoUniqueBeanDefException in case of ambiguity

autoWire=constructor
1.Provide parameterised constructor in dependent bean class.
2. No <constructor-arg> tag required in xml config file.
3. Similar to autoWire=byType. SC tries to match data type of the constructor argument to type of dependency bean.
4. SC throws NoUniqueBeanDefExc in case of ambiguity


Spring Bean life cycle maanged by Spring Container
Start SC (in Java SE by creating instance of ClasspathXmlApplicationContext or auto done in web app by DispatcherServlet --Front Controller)
SC refers to XML based instructions / Annotations(life cycle is the SAME)
SC checks the Scope
Default scope=singleton
Checks lazy-init
scope=prototype
SC will invokes the same life cycle per every demand(via ctx.getBean).
Doesn't invoke destroy method. Mraks bean for GC
Def value= false
(i.e singleton n eager)
lazy-init=true
Same life cycle invoked by SC after 1st demand
Starts the life cycle(init sequence --A)
Locate-load--instantiate
(can be either via def constr or parameterized constr(to support constr based D.I)
setter Based D.I(property tags) --SC nvokes matching setters
init-method
supplied
SC invokes init style method(public void myInit() throws Exc {...}
not supplied
no init method invoked.
Spring bean is rdy for servicing the clnt.
B.L execution.....
@ SC shut down
ctx.close()
SC chks destroy-method
supplied (public void anyName() throws Exc {...}
SC invokes the same.
Marks bean for GC
End of life cycle.
not supplied
--doesn't invoke destroy method callback.
Marks the bean for GC

Why Spring

It simplifies Java development. It's one-stop-shop.
Excellent for integration with existing frameworks.
Reduces boiler-plate code.
Allows to build applications using loose coupling achieved via IoC(Inversion of control) & AOP(aspect oriented programming)

What is it ?

It is a container + framework.

Why Container

It manages the life cycle of spring beans (eg : controller/rest controller , dao,service)
spring bean : java object whose life cycle is managed by spring container(SC)

Why Framework ?

It provides readymade implementation of patterns & helps in building enterprise applications.

It is the most popular application development framework for enterprise Java. It is used to create high performant, easily testable, and reusable code.

Spring framework is an open source Java platform.
Founder is Rod Johnson and was first released under the Apache license in June 2003.
Currently hosted on Pivotal/VMware

Why Spring

1. Spring is lightweight .The basic version of Spring framework is around 2MB.

2. It supports in developing any Java application, but there are extensions(web MVC) for building web applications on top of the Java EE platform.

3. It helps programmers to make J2EE development easier to use and promotes good programming practices by enabling a POJO-based programming model.
4. Excellent n easy testing support.(Thanks to D.I)

5. Supports smooth integration with ORM

6. Easy integration with web MVC applications including web sockets (for async communication between server & client)
Spring's web framework is a web MVC framework, which is a great alternative to web frameworks such as Struts

7. It is organized in a modular fashion. Even though it's extensive , you have to worry only about the those modules that you need and ignore the rest.

8. Spring does not re-invent the wheel, instead it makes use of already existing frameworks like Hibernate , making its integration easier.

9. It translates technology-specific exceptions (thrown by JDBC, Hibernate, or JDO, for example) into consistent, unchecked exceptions.

10.
It provides a consistent transaction management interface that can support a local transaction (using a single database) as well as global transactions (using JTA over multiple databases).

Main winning feature of Spring is : loose coupling between the modules.

How does it achieve loose coupling ?
1. IoC -- IoC is achieved using Dependency Injection(D.I)
2. Aspect Oriented Programming(AOP)

Steps for Spring based Java SE application

0. In eclipse , change perspective to Java
1. Create Java project
2. Create User lib --containing spring/hibernate/jdbc drvr/REST....
JARs.(from day11-data\day11-help\spring-help\spring-hibernate-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 --<resources> & 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)
7. Create a tester application , to start Spring container & run this as java application , to confirm spring in core java.

Enter hybrid approach

Hybrid approach(Reduced xml, reduced java code + majority annotations)

Steps in Spring programming using annotations

0. To enable annotation support -- add context namespace & add the following
<context:annotation-config/> --- To tell SC --to enable annotation support(eg --- AutoWired,PostConstruct,Predestroy,.....)

0.5 --- How to specify location(base pkg) of spring beans to SC?
<context:component-scan base-package="comma sep list of pkgs"/>---
SC starts searching(scanning) in specified pkgs (including sub-pkgs) --- for classes anno with stereotype annotations ---
@Component,@Service,@Repository,@Controller,@RestController,@ControllerAdvice...

Required throughout

Basic class level annotations meant for SC
Super type
@Component --- spring bean class
sub - type annotations
@Controller --- In Web MVC scenario -- for request handling.
@Service --- Service layer (B.L) + transaction management
@Repository --- DAO layer
@RestController -- RESTful service provider

These are all called stereotype annotation coz it comes from class. Stereotype

Annotation XML

1. @Component --- <bean id , class....> --- SC interprets it & starts bean life-cycle.
eg ---
```
package beans;
@Component("abc")
public class MyBean {...}
```
xml --- <bean id="abc" class="beans.MyBean"/>
OR
```
@Component
public class MyBean {...}
```
Derived bean id pattern
xml --- <bean id="myBean" class="beans.MyBean"/>

2. @Controller -- spring web mvc controller
3. @Repository --- DAO layer class
4. @Service --- for service layer beans --- transactions.
5. @Scope(value="singleton|prototype|request|session")--- class level annotaion --- in xml via scope attribute.
6. @Lazy(true|false) ----class level anno -- lazy-init attribute
7. @PostConstruct ---method level anno - init-method ---method level
8. @PreDestroy ---method level anno-- destroy-method --- method level
9. @Required(true|false) --- setter method or paramed constr ---tells SC if depenency is mandatory or optional-- def=true Explicit wired
10. @AutoWired ---setter method or paramed constr or field level
eg --- TestTransport implements Transport {...}
autowire="byType" (Canbe added over setter parameterized constructor /method /direct field level)
eg -- field level annotation ---in ATMImpl bean (dependent)
```
@AutoWired //autowire=byType , mandatory by default (required=true)
private Transport myTransport;
```
Meaning -- no parameterised constr, no setter , no xml containing bean definition is required.

SC --- chks for any bean of Transport by type & injects it in ATMImpl
What if : SC comes across multiple matches : SC throws
NoUniqueBeanDefinitionException
What if : SC doesn't find even a single match : SC throws
UnSatisfiedDependencyException

11. @AutoWired//(required=true)
@Qualifier("test")
private Transport myTransport; ---- =>autowire="byName"
---spring supplied anno.
SC searches for a bean with id="test"
Match found : NO excs , field level D.I succeds!
Match not found : SC throws exception (UnSatisfiedDependencyException :
NoBeanDefFoundExc)
Multiple match found: ambiguity → sc throws nubdexc

OR
@Resource(name="soap")
private Transport myTransport; ---- autowire="byName"

--J2EE supplied via javax.annotation

SpEL --- spring expression language
dynamic expression languge ---spring(3.x) supplied -- to evaluate
expressions dynamically.
#{SpEL expression} --- similar to JSP EL --- SpEL allows ---
getters,setters,constr invocation, static & non-static method
invocations.

Entity Types :
1. If an object has its own database identity (primary key value) then it’s type is Entity Type.
2. An entity has its own lifecycle. It may exist independently of any other entity.
3. An object reference to an entity instance is persisted as a reference in the database (a foreign key value).
eg : College is an Entity Type. It has it’s own database identity (It has primary key).

Value Types :
1. If an object don’t have its own database identity (no primary key value) then it’s type is Value Type.
2. Value Type object belongs to an Entity Type Object.
3. It’s embedded in the owning entity and it represents the table column in the database.
4. The lifespan of a value type instance is bounded by the lifespan of the owning entity instance.

Different types of Value Types

Basic, Composite, Collection Value Types :
1. Basic Value Types :
Basic value types are : they map a single database value (column) to a single, non-aggregated Java type.
Hibernate provides a number of built-in basic types.
String, Character, Boolean, Integer, Long, Byte, … etc.

2.
Composite Value Types :
In JPA composite types also called Embedded Types. Hibernate traditionally called them Components.
2.1 Composite Value type looks like exactly an Entity, but does not own lifecycle and identifier.

Annotations Used

1. @Embeddable :
Defines a class whose instances are stored as an intrinsic part of an owning entity and share the identity of the entity. Each of the persistent properties or fields of the embedded object is mapped to the database table for the entity. It doesn't have own identifier.
eg : Address is eg of Embeddable
Student HAS-A Address(eg of Composition --i.e Address can't exist w/o its owning Entity i.e Student)
College HAS-A Address (eg of Composition --i.e Address can't exist w/o its owning Entity i.e College)
BUT Student will have its own copy of Address & so will College(i.e Value Types don't support shared reference)

2. @Embedded :
Specifies a persistent field or property of an entity whose value is an instance of an embeddable class. The embeddable class must be annotated as Embeddable.
eg : Address is embedded in College and User Objects.

3. @AttributesOverride :

Used to override the mapping of a Basic (whether explicit or default) property or field or Id property or field.

In Database tables observe the column names. Student table having STREET_ADDRESS column and College table having STREET column. These two columns should map with same Address field streetAddress.
@AttributeOverride gives solution for this.
To override multiple column names for the same field use @AtributeOverrides annotation.
eg : In Student class :

```
@Embedded
 @AttributeOverride(name="streetAddress",
column=@Column(name="STREET_ADDRESS"))
private Address address;
```

where , name --POJO property name in Address class

3.
Collection Value Types :
Hibernate allows to persist collections.
But Collection value Types can be either collection of Basic value types, Composite types and custom types.
eg :
Collection mapping means mapping group of values to the single field or property. But we can't store list of values in single table column in database. It has to be done in a separate table.

eg : Collection of embeddables

```
@ElementCollection
      @CollectionTable(name="CONTACT_ADDRESS",
joinColumns=@JoinColumn(name="USER_ID"))
      @AttributeOverride(name="streetAddress",
column=@Column(name="STREET_ADDRESS"))
      private List<ContactAddress> address;
```

eg : collection of basic type

```
      @ElementCollection
      @CollectionTable(name="Contacts",
joinColumns=@JoinColumn(name="ID"))
      @Column(name="CONTACT_NO")
      private Collection<String> contacts;
```

Problems noticed in the lab n suggested solutions
1. JARs did not get downloaded.
Solution : delete <repository> under .m2 from user's home directory.
Re start IDE , it will build the repo again.

2. Create new database advjava2 was not working
Give root user credentials in your db settings n it will work.

Revise
Inheritance in hibernate
@MappedSuperclass : class level JPA annotation , to tell hib. --
following class is super class to all other entities n should not create asso. table !

one to many bi dir association with entities (weaker form of association : aggregation)
eg : Category 1<----->* Product

one/many , parent/child owning/inverse

Category : one , parent , inverse (non owning --it does NOT contain any physical mapping : FK)
Product : many , child , owning (FK)

Which is the additional property in Category POJO ? : ...

+ private List<Product> products=new ArrayList<>();

Which is the additional property in Product POJO ? ....
+ private Category productCategory;

What will happen if you don't add any annotations ? throws MappingExc

Which annotations ?In Category class
@OneToMany
private List<Product> products=new ArrayList<>();

Product POJO ....
@ManyToOne
 private Category productCategory;

What will happen if you don't add mappedBy element ? hib creates an extra table : link table (product_id n category_id : FKs)

Why ? Sice you have not yet supplied any info regarding owning side n inverse side.

When adding "mappedBy" mandatory ? in a bi-dir asso. only !

Where it appears ? inverse side (i.e Category)

What should be it's value ?
Name of the asso property as it appears in the owning side.
eg :
@OneToMany(mappedBy="productCategory")
private List<Product> products=new ArrayList<>();

Product POJO ....
@ManyToOne
private Category productCategory;
In above case who will decide the name of FK col ? hibernate
How to customize it ?
@JoinColumn(name="category_id") //FK col name refs ---> PK of categories table.

1. Add a new category
i/p : category name , description

2. Add Product to the category
i/p : category id , product details (name,price,desc)
steps in ProductDaoImpl
get category from it's id (get)
null chking --if not null -->
cat.addProduct(newProduct);
session.save(newProduct);

2.1 (understand how to establish bi-dir relationship)
2.2 Helper methods

3. Add a new category with products (w/o cascading)
Observe n conclude
w/o cascading :
establish asso.
save category
save products

Then add cascading
Add a property cascade in @OneToMany annotation to support cascading of the changes from parent --> child entity.

4. Remove a category
i/p : category name
Confirm cascading
Objective : String deleteCategory(String categoryName);
jpql : select c from Category c where c.categoryName=:nm
getSingleResult --->no exc --> session.delete(category)

5. Remove a product from a Category
i/p : category id , product id

Steps
get category n product from it's id : session.get
null chking --not null
cat.removeProduct(p);//helper method

Any problem noticed ? YES , hib simply de linked the entities (i.e FK : null) , BUT child rec wasn't deleted
Solution : add new property to @OneToMany : orphanRemoval=true

5. Display category details

i/p : category name

6. Display category n product details
i/p : category name
IMPORTANT
Any problem noticed : org.hibernate.LazyInitializationException
Cause : Any time you are trying to access un-fetched data , in a detached mode(i.e outside session scope) , hib throws the exc.

WHY ?

Hibernate follows default fetching policies for different types of associations
one-to-one : EAGER
one-to-many : LAZY
many-to-one : EAGER
many-to-many : LAZY

one-to-many : LAZY
Meaning : If you try to fetch details of one side(eg : Category) , will it fetch auto details of many side(i.e : Product) ?
NO (i.e select query will be fired only on categories table)
Why ? : for performance

When will hibernate throw LazyInitializationException ?
Any time you are trying to access un-fetched data from DB , in a detached manner(outside the session scope)
cases : one-to-many
many-many
session's load

un fetched data : i.e product list in Category obj : represented by : proxy (substitution) : collection of proxies
proxy => un fetched data from DB

Solutions
1. Change the fetching policy of hibernate for one-to-many to : EAGER
eg :

Is it recommneded soln ?

NO (since even if you just want to access one side details , hib will fire query on many side) --will lead to worst performance.
Use case : when the size of many is small !
(eg : BankCustomer ---> BankAccount)

2. Better Solution

Solution : Access the size of the collection within session scope : soln will be applied in DAO layer

Dis Adv : Hibernate fires multiple queries to get the complete details

3. How to fetch the complete details , in a single join query ?
Using "join fetch" keyword in JPQL

```
String jpql = "select c from Category c join fetch c.products where
c.category=:nm"
--inner join

String jpql = "select c from Category c left outer join fetch c.products
where c.category=:nm"
--left outer join
```

7. Register New User
i/p user details
o/p message indicating success or failure
Exists already (project : day9.1)

8. Use case :
User login
i/p email , password
In case of successful login , check the role --if cutomer role
--check if cart exists for the logged in customer . If no --create one .

9. What will be the relationship between User n Cart ??? : one to one asso.

User 1<----->1 ShoppingCart

ShoppingCart Entity extends BaseEntity
state : totalItems,totalCartPrice,createdOn,updatedOn,
Additional annotations : @CreationTimestamp @UpdateTimestamp

How to establish bi-dir one-to-one relationship between User 1<---->1 ShoppingCart

First identify
one, parent/child owning/inverse

User : one , parent , inverse
ShoppingCart : one , child , owning

Which additional properties ?
In User class :

```
@OneToOne(mappedBy="cartOwner",cascade=CascasdeTYpe.ALL,orphanRemoval=tru
e)
private ShoppingCart cart;
```

In ShoppingCart

```
...+
@OneToOne
@JoinColumn(name="customer_id",nullable=false)
private user cartOwner;
```

Which annotations ? Mentioned above !

Add <mapping> entry n run TestHibernate , check if the tables are created properly ?

10. What will be the relationship between Cart n Product?

Can 1 cart contain multiple products ?? YES
Can 1 product be added to multiple carts for different customers ?? YES

So it's many-to-many
What will happen if you use @ManyToMany annotation ?
(Try it out !)

Result : a join table will be created by hibernate , having composite PK
eg : cart_items
2 columns only : cart_id n product_id
(Having FK constraints)
cart_id : FK --> references PK of carts table
product_id : FK --> references PK of products table

Is it suitable if you want to maintain additional columns ? : quantity n total_price

Solution : Create a separate entity CartItem (representing DB tables)
What will the columns ? id (PK), quantity , total_price, cart_id(FK) , product_id(FK)
Which properties : quantity , totalPrice , cart , product

Establish following relationships , using the annotations

ShoppingCart 1<----->* CartItem

CartItem 1----->1 Product

----------PENDING----------

11. Enter value types
Customer HAS-A Adhar Card

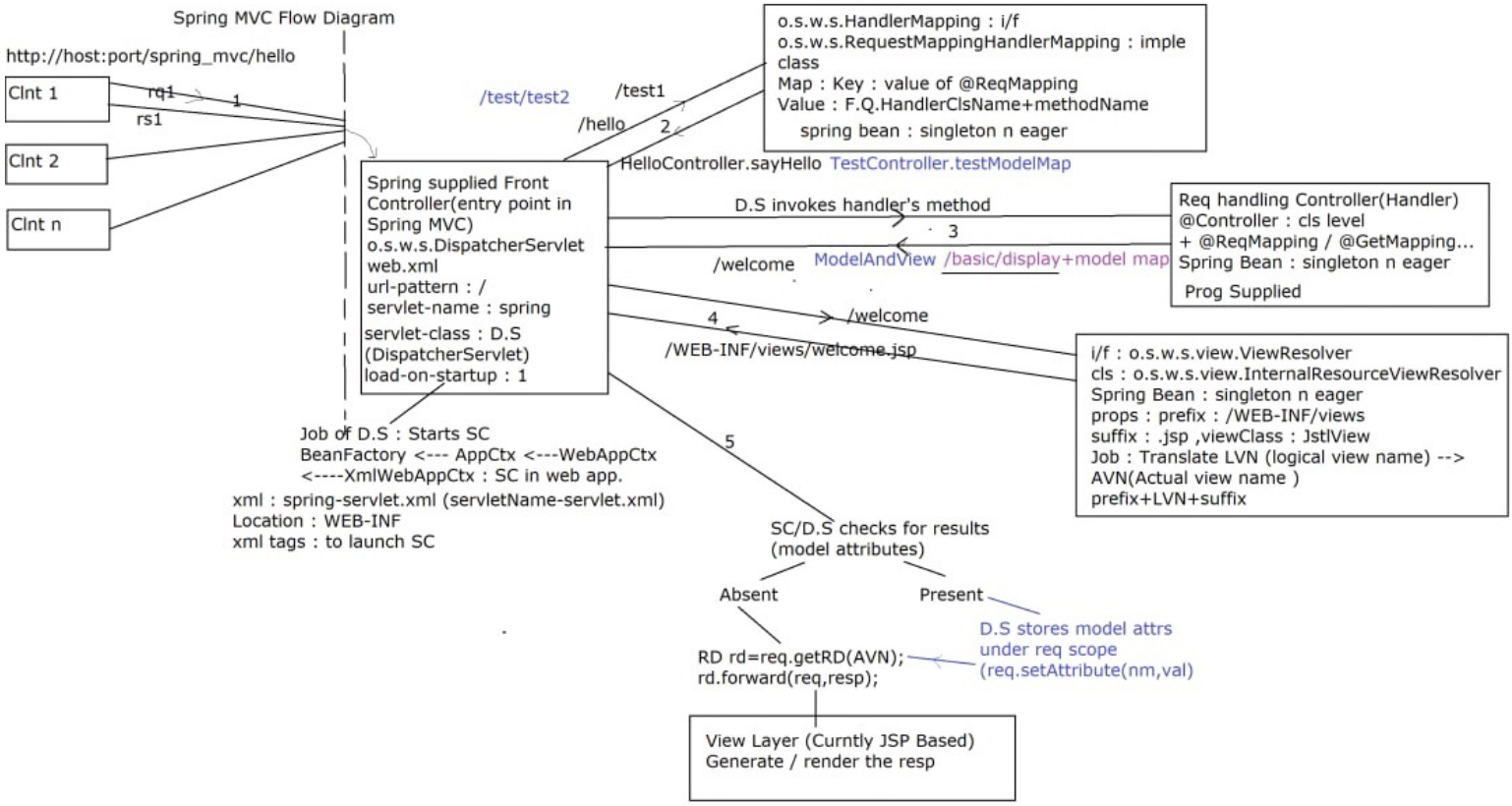
11.1 Customer : entity : @Entity n @Id
Adhar Card : embeddable : @Embeddable
field : cardNo , date , loc

11.2 : Customer HAS-A hobbies
(collection of basic types)
Annotations : @ElementCollection n @CollectionTable

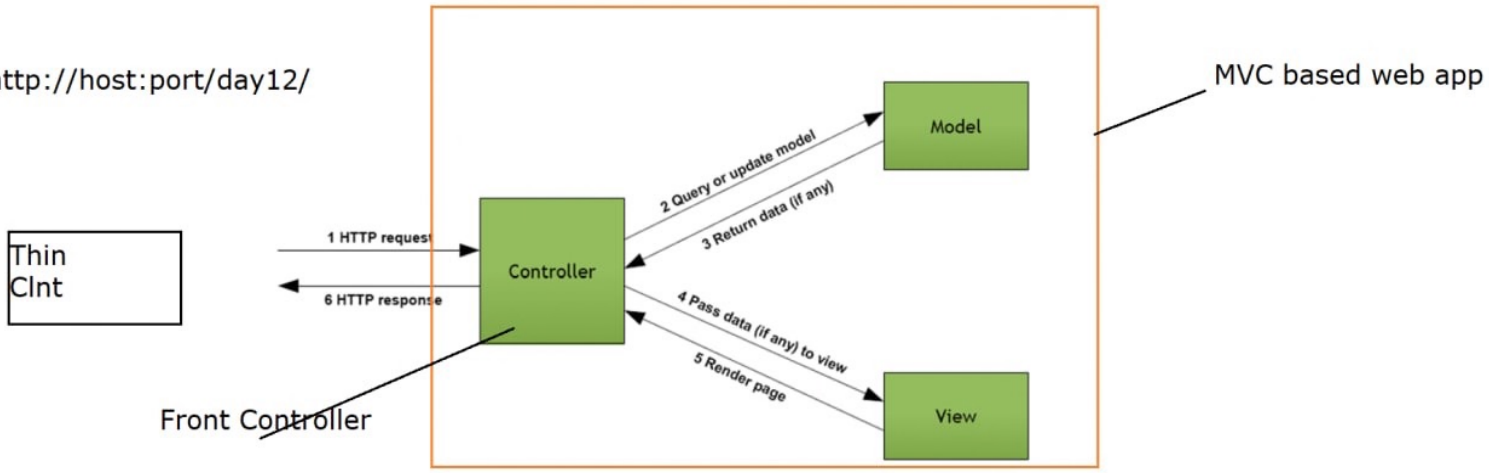
11.3 Customer HAS-A Card (can have multiple credit/debit cards)
(collection of embeddables)
Annotations : @ElementCollection n @CollectionTable

Good Articles/Books to refer to JPA Association Mapping

1. e Book(already shared in hibernate_help) : "hibernate-help\hibernate books\Pro JPA 2 in Java EE 8, 3rd Edition.pdf"
2. link : https://howtodoinjava.com/hibernate/how-to-define-association-mappings-between-hibernate-entities/
3. https://thorben-janssen.com/ultimate-guide-association-mappings-jpa-hibernate/

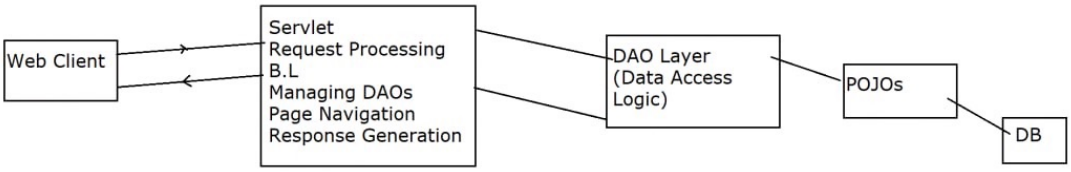


http://host:port/day12/

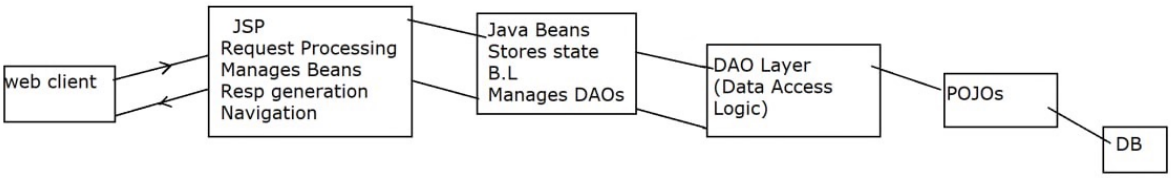


http://host:port/day12/validate

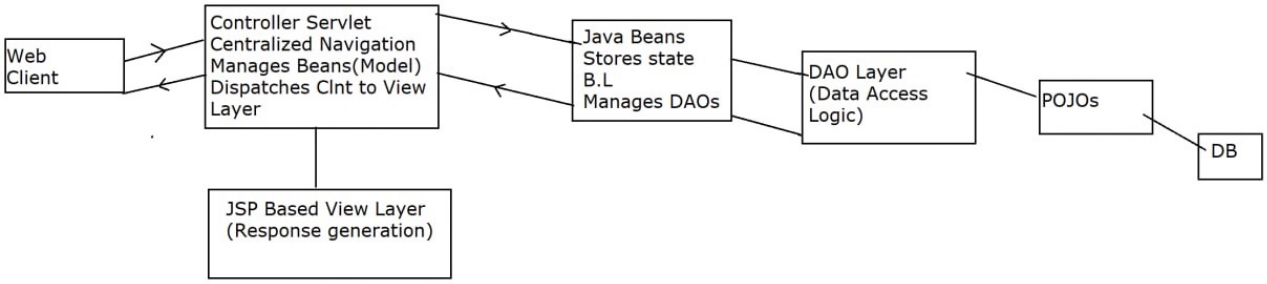
Model I Architecture with Servlets

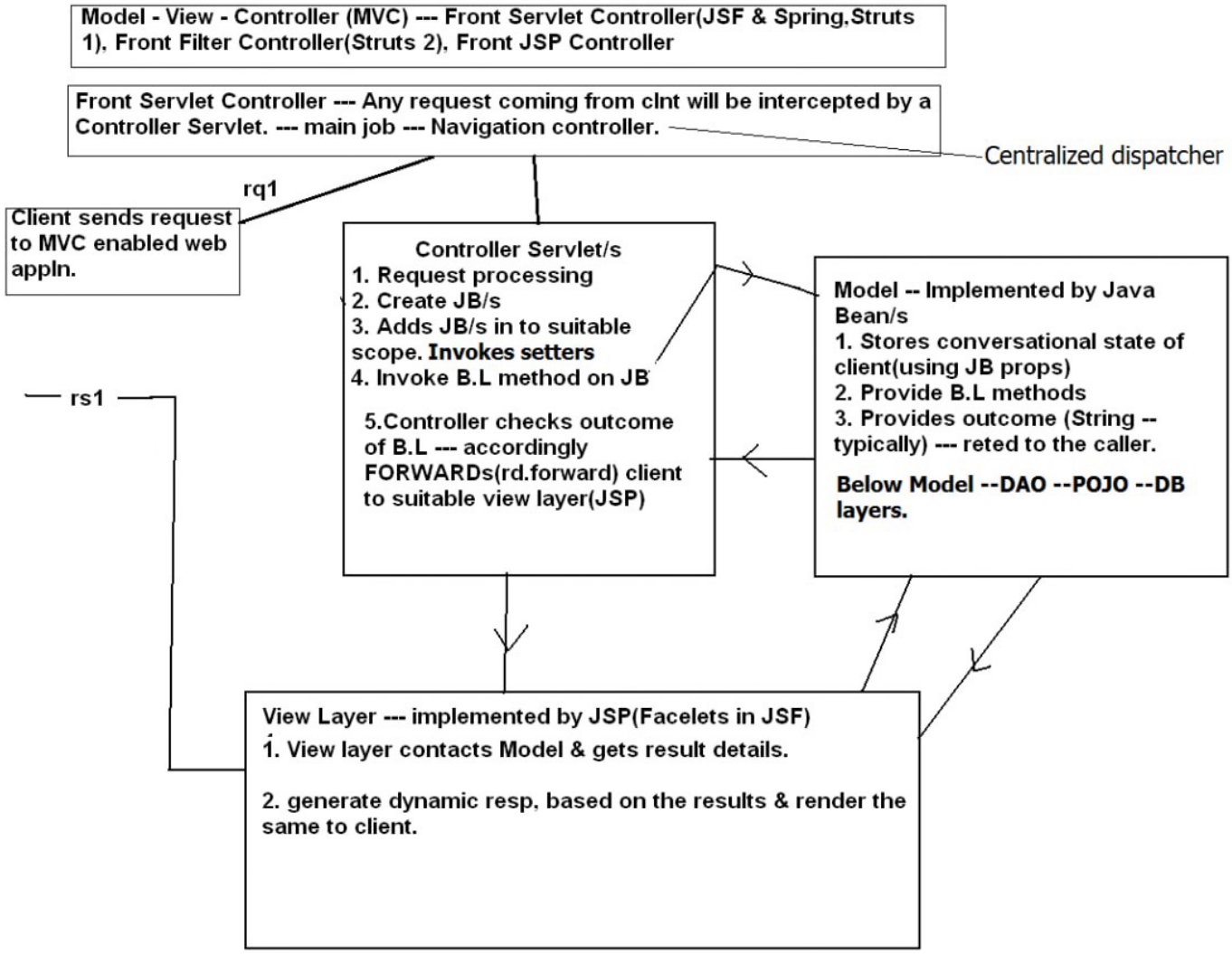


Model I Architecture with JSP



Model II Architecture (MVC)
Model View Controller




Model - View - Controller (MVC) --- Front Servlet Controller(JSF & Spring,Struts 1), Front Filter Controller(Struts 2), Front JSP Controller
Front Servlet Controller --- Any request coming from clnt will be intercepted by a Controller Servlet. --- main job --- Navigation controller.
Centralized dispatcher
rq1
Client sends request to MVC enabled web appln.
Controller Servlet/s
1. Request processing
2. Create JB/s
3. Adds JB/s in to suitable scope. Invokes setters
4. Invoke B.L method on JB
5.Controller checks outcome of B.L --- accordingly FORWARDs(rd.forward) client to suitable view layer(JSP)
Model -- Implemented by Java Bean/s
1. Stores conversational state of client(using JB props)
2. Provide B.L methods
3. Provides outcome (String -- typically) --- reted to the caller.
Below Model --DAO --POJO --DB layers.
rs1
View Layer --- implemented by JSP(Facelets in JSF)
1. View layer contacts Model & gets result details.
2. generate dynamic resp, based on the results & render the same to client.

Enter Spring

Why Spring ?
Simplifies overall java development

What is it ?
container --manages life cycle of spring beans
(spring bean --- java obj whose life cycle completely managed by SC(spring container)
eg : rest controller, controller, service,DAO.
framework --rdy made implementation of std patterns(eg :MVC,Proxy,singleton,factory, ORM ...)

Spring is modular n extensive framework.

Why Spring : loosely coupled application
Via : D.I / AOP

What is dependency injection ?

In JSP---JB---DAO(Utils) -- POJO --DB layers
Dependent Objs -- JavaBean , Hibernate based DAO, JDBC Based DAO
Dependencies --- DAO,HibUtils(SessionFactory) , DBUtils(DB connection)

All of above are examples of tight coupling.

Why --Any time the nature of the dependency changes , dependent obj is affected(i.e u will have to make changes in dependent obj)
eg : When the dependency of Java Bean changes from JDBC Based DAO to Hibernate based DAO , in case of user authentication , javabean class has to be modified to handle invalid login case(i.e handle NoResultException)

Tight coupling --strongly un desirable.
Why -- difficult to maintain or extend.

In above examples , Java bean creates the instance of DAO.
Hibernate based DAO , gets SF from HibUtils.
JDBC based DAO ,gets db connection from DBUtils.

i.e dependent objects are managing their dependencies. --- traditional/conventional programming model.

What is D.I ?(Dependency injection=wiring=collaboration between dependent & dependency)
Instead of dependent objs managing their dependencies , 3rd party containers(eg : Angular / Spring/ EJB/ WC) will auto create the dependecies & make it available to dependents, directly @ run time.

Since dependent are no longer managing dependencies --its called as IoC --Inversion of control

Hollywood principle --You don't call us , we will call you....

SC --- > Dependent objs (i.e SC will create the dependencies for the dependent objs)

eg : UserController
@Autowired
private IUserService service;

In DAO layer
@AutoWired
private SessionFactory sf;

Pre requisite : Already added STS plug-in / STS 3.9.x
Steps for spring nature to Java project

Important : Extract spring api-docs
Objective : Create Spring based Java SE project

1. Create Maven based Java SE project with spring dependencies
2. Create dependent n depency classes
4. Refer : <resources> & create spring bean config xml file.(Using STS support)
5. Add namespace <beans>

More details about <bean> tag
Attributes
1. id --mandatory --bean unique id
2. class --- mandatory -- Fully qualified bean class name
3. scope --- In Java SE --- singleton | prototype
In web app singleton | prototype | request | session | global session
Default scope = singleton
singleton --- SC will share single bean instance for multiple requests/demands(via ctx.getBean)
prototype -- SC creates NEW bean instance per request/demand.

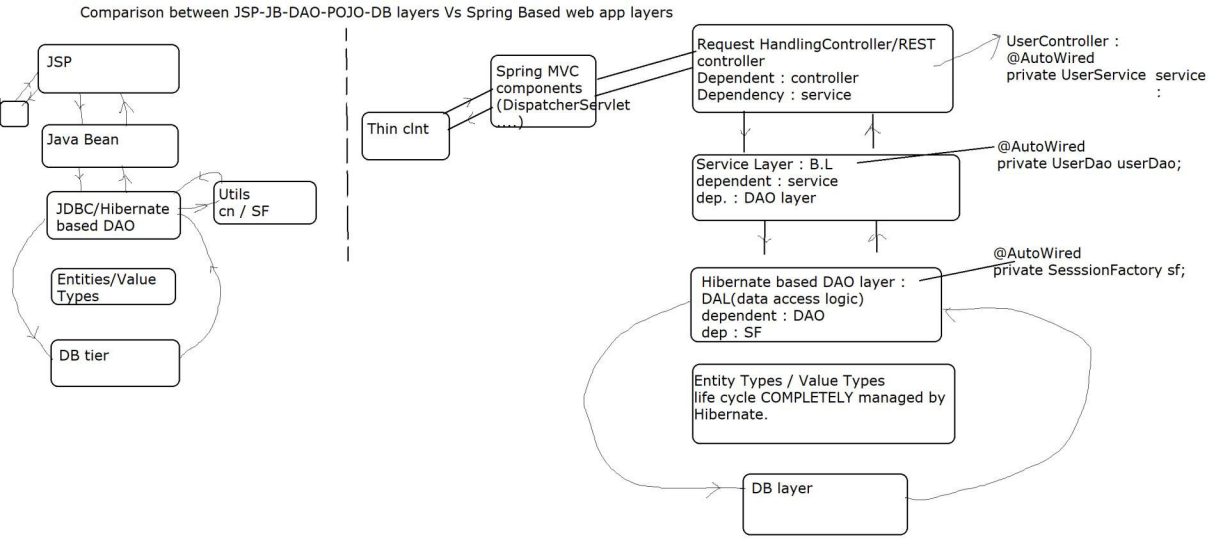
4. lazy-init --- boolean attribute. default value=false.
Applicable only to singleton beans.
SC will auto create singleton spring bean instance --- @ SC start up.

5. init-method --name of init style method(public void anyName() throws Exception{..})
called by SC after setter based D.I

6. destroy-method --name of destroy style method
(public void anyName() throws Exception{..})
called by SC before GC of spring bean (applicable only to singleton beans)

API
How to get ready to use spring beans from SC ?
API of BeanFactory
public <T> T getBean(String beanId,Class<T> beanClass) throws BeansException

Spring bean life cycle
Types of wiring


Comparison between JSP-JB-DAO-POJO-DB layers Vs Spring Based web app layers
JSP
Java Bean
JDBC/Hibernate based DAO
Utils cn / SF
Entities/Value Types
DB tier
Thin clnt
Spring MVC components (DispatcherServlet ....)
Request HandlingController/REST controller
Dependent : controller
Dependency : service
UserController :
@AutoWired
private UserService service :
Service Layer : B.L
dependent : service
dep. : DAO layer
@AutoWired
private UserDao userDao;
Hibernate based DAO layer :
DAL(data access logic)
dependent : DAO
dep : SF
@AutoWired
private SesssionFactory sf;
Entity Types / Value Types
life cycle COMPLETELY managed by Hibernate.
DB layer

Revise
Solve :
0. Course : id , title , start date , end date , fees capacity
courses table

1. one to one mapping
eg : Student HAS-A Address
Student <-----> Address
How will you configure a bi-dir one-to-one mapping between them ?
Student POJO : one , parent , inverse
id , name,.....
+
@OneToOne(mappedBy="owner",cascade=CascadeType.ALL,orphanRemoval=true,fetch=FetchType.LAZY)
private Address myAddress;

Address : one , child , owning
id ,line1 , line 2......
@OneToOne
@JOinColumn(name="student_id",nullable=false)
+private Student owner;

DAO
String jpql="select s from Student s left outer join fetch s.myAddress where s.email=:em";
exec query ---> Student s=......
//s.getAddress().getCity();//hint
tx.commit();
return s;//s : DETACHED

JSP-JB-DAO-POJO-DB

JB --> DAO's method
Student student=dao.getStudentDetails();
sop(student);//1
sop(student.getMyAddress());//2 --: LazyInitExc.

2. Many-to-Many (w/o additional columns)
Student *<------>* Project
We just want to maintain : student_id n project_id in the join table.

Student POJO : many , parent , owning
....
+
@ManyToMany //mandatory
@JoinTable(name="student_projects",joinColumns=@JoinColumn(name="student_id"),inverseJoinColumns=@JoinColumn(name="project_id")) //OPTIONAL BUT reco for customization
private Set<Project> projects=new HashSet<>();
//override hashCode n equals : as per the contract

Project POJO : many , parent , inverse
....+
@ManyToMany(mappedBy="projects")
private Set<Student> students=new HashSet<>();
//override hashCode n equals : as per the contract

3. Student Admissions
Course *<----->* Student

We want to maintain : application date n admission status(APPLIED
,ADMITTED,REJECTED,WAIT_LISTED,CANCELLED) , admission_date

You laready have : Student n Course : parent
Add another entity : Admission

state : id (PK) , applicationDate......
+
//Student 1<------* Admission
@ManyToOne
@JoinColumn(name="student_id",nullable=false)
private Student applicant;
//Course 1<-----* Admission
@ManyToOne
@JoinColumn(name="course_id",nullable=false)
private Course selectedCourse;

Use case :
student admission(applying for a course)
---

1. Enter value types
User HAS-A Adhar Card

1.1 User : entity : @Entity n @Id
Adhar Card : embeddable : @Embeddable
field : cardNo , date , loc

1.2 : User HAS-A hobbies
(collection of basic types)
Annotations : @ElementCollection n @CollectionTable

1.3 User HAS-A Card (can have multiple credit/debit cards)
(collection of embeddables)
Annotations : @ElementCollection n @CollectionTable
Lab work

Enter Spring

A question : Has every one tried creating 1st spring based stand alone application ?

Revise Spring introduction
Spring Container(SC) API
Spring Beans life cycle
Discuss different ways of D.I(modes of wiring)
Enter annotations based configuration
Java config based instructions (no xml)
Enter MVC : A design pattern
Spring MVC

Revision
Why Spring ?
.....+
It allows you to develop an enterprise app , in completely loosely coupled manner

What is it ?
It's container +modular n extensive , min intrusive framework

What is IoC?

Instead of dependent objs managing their dependencies , the control of managing the dependencies lies with SC
eg : UserController :
dep : @AutoWired
private UserService service; //field level D.I

What is D.I? : Making the dependencies available to dep. objs @ run time (dynamic) by SC

eg : dependent obj : ATMImpl
dep : Transport layer (i/f) <----- TestTransport,.....
No tight coupling !

XML based config

What are 3 different ways of supplying metadata instrs to SC
1. pure xml (legacy!)
2. Hybrid approach (lesser xml + majority annotations)
3. Java config class + annotations

1.Pure XML based approach(legacy)
<bean> tag attributes
id : unique bean id
class : F.Q bean class name
scope :
Java SE : singleton | prototype
def scope : singleton (=> SC creates a single instance of this bean : to be shared across multiple demands i.e getBean)
prototype : => SC creates a separate instance of the bean : as per demand i.e getBean)
lazy-init : def value =false (applicable to ONLY singleton beans)

init-method : name of custom init method
pattern : public void anyName() throws Exc {....}

Will be invoked for singleton as well as prototype beans

destroy-method : name of custom destroy method : SC invokes it just before GC
pattern : public void anyName() throws Exc {....}
Will be invoked only for singleton beans

How to launch SC in a standalone application ?

Understand SC API (refer to a diagram)
API
How to get ready to use spring beans from SC ?
API of BeanFactory
public <T> T getBean(String beanId,Class<T> beanClass) throws BeansException
T : type of the spring bean

Understand Spring bean life cycle (refer to a diagram)

Different Modes of Wiring (D.I.) : (refer to a diagram)

2. Hybrid Approach (reduced XML n majority annotations)
(refer to readme : "spring sequence for annotations")

eg : UserController : dependency : UserService : i/f , (UserServiceImpl) : impl class

@AutoWired //=> autowire=byType
private UserService myUserService;
What if : SC comes across multiple matches : SC throws : NoUniqueBeanDefinitionException
What if : SC doesn't find even a single match : SC throws NOBeandefFoundExc

eg : UserController : dependency : UserService : i/f , (UserServiceImpl) : impl class
@AutoWired (required=false) //=> autowire=byType
private UserService myUserService;
What if : SC comes across multiple matches : SC throws : NOUnqBeandefexc
What if : SC doesn't find even a single match : SC DOES NOT throw NOBeandefFoundExc
Business logic failure : NPExc

11.
eg : UserController : dependency : UserService : i/f , (UserServiceImpl) : impl class
@AutoWired
@Qualifier("abc") //=> autowire=byName
private UserService myUserService;
What if : SC comes across exactly 1 match : D.I succeeds !
What if : SC comes across multiple matches : SC throws : NOUnqBeandefexc
What if : SC doesn't find even a single match : SC DOES NOT throw NOBeandefFoundExc

OR

```
@Resource(name="abc") //=> autowire=byName (Java EE annotation)
private UserService myUserService;


SpEL --- spring expression language
dynamic expression languge ---spring(3.x) supplied -- to evaluate
expressions dynamically.
#{SpEL expression} --- similar to  JSP EL --- SpEL allows ---
getters,setters,constr invocation, static & non-static method
invocations.
OR
${SpEL expr}
-----------------------------------
What is MVC ?
Model-View-Controller --Standard design pattern , meant for separation of
concerns(=tasks=responsibilities)

Model -- Java Bean (conversational state holder + B.L supplier) & POJOs

View layer --JSP , Thymeleaf/velocity/Angular/React/Vue
Represents UI / presentation logic  (processing requests & generating
response)

Controller -- Typically a servlet(used in Spring MVC ) or a filter(used
in Struts 2 framework)
Manages navigation & beans.

Front Controller -- A design pattern -- which ensures ANY request coming
from ANY client , for this web app , will be intercepted by a common gate
keeper(or a centralized dispatcher)
It will dispatch clnt request to further components , based upon nature
of the req.

MVC flow (without spring)

refer : MVC diagrams , MVC Details
mvc-overview.png


Implementation using servlet / JSP & JavaBean  --Shared as a readymade
demo.
Refer to : eclipse projects\mvc_hibernate
Flow  diagram : mvc-flow.png

Before entering spring-mvc

For  comparison of layers between JSP--JB--DAO --POJO --DB Vs Spring MVC
layers.
refer : comparison of layers JSP-JB vs Spring MVC.png


Enter Spring MVC --concept & implementation steps
Refer to : "lab sequence for spring mvc.txt"
----------------------
1. Test Spring MVC flow.
(refer to lab sequence)

2. What is a model attribute?
It's the attribute(server side entry=k n value pair)
```

Purpose : to store the results of B.L
Who creates --- Req handling Controller(Handler)
Who sends it to whom : Handler ---> D.S
After D.S gets actual view name from V.R :
D.S chks : are there any model attrs :
Yes : D.S saves model attrs under Request scope & then forwards the clnt to view layer .
NO : D.S forwards the clnt to view layer .
How to access these model attrs from JSP ?
${requestScope.attrName}

What are different ways for Controller to add model attrs ?
1. Via is o.s.w.s.ModelAndView?

o.s.w.s.ModelAndView : class => holder for model attrs + logical view name

ModelAndView(String viewName,String modelAttrName,Object modelAttrVal)
eg : what can be valid ret type of req handling method
 ModelAndView

2. Simpler way ?

Use o.s.ui.Model : i/f ---holder (Map) of model attributes
How do u add model attributes ?
public Model addAttribute(String modelAttrName,Object modelAttrVal)
eg : How to add 2 model attrs?

Who will supply empty Model map ? : SC
IoC : DI
How to tell SC that u need a model map ? : add it as the arg of req handling method
When controller rets string : logical view name (controller impl. rets all model map)

3. How to hide index.jsp under WEB-INF (i.e how to ensure that index page is served to client via D.S)
Add HomeController to serve index.jsp

4. Handling request parameters in Controller ?
eg : @RequestParam("price") double productPrice
SC : double productPrice
=Double.parseDouble(request.getParameter("price"))
Reco : Match req param names with method arg names.
URL :
http://localhost:8080/day13.1/test/product?nm=pen&qty=10&price=40.5&manuDate=2020-1-1
SC : def date time format : mon/day/year
Problem : HTTP 400 : Bad request => some request data coming from client is invalid
Def dat format : mon/day/yr
How to tell SC : about exact Date time format : annotation.

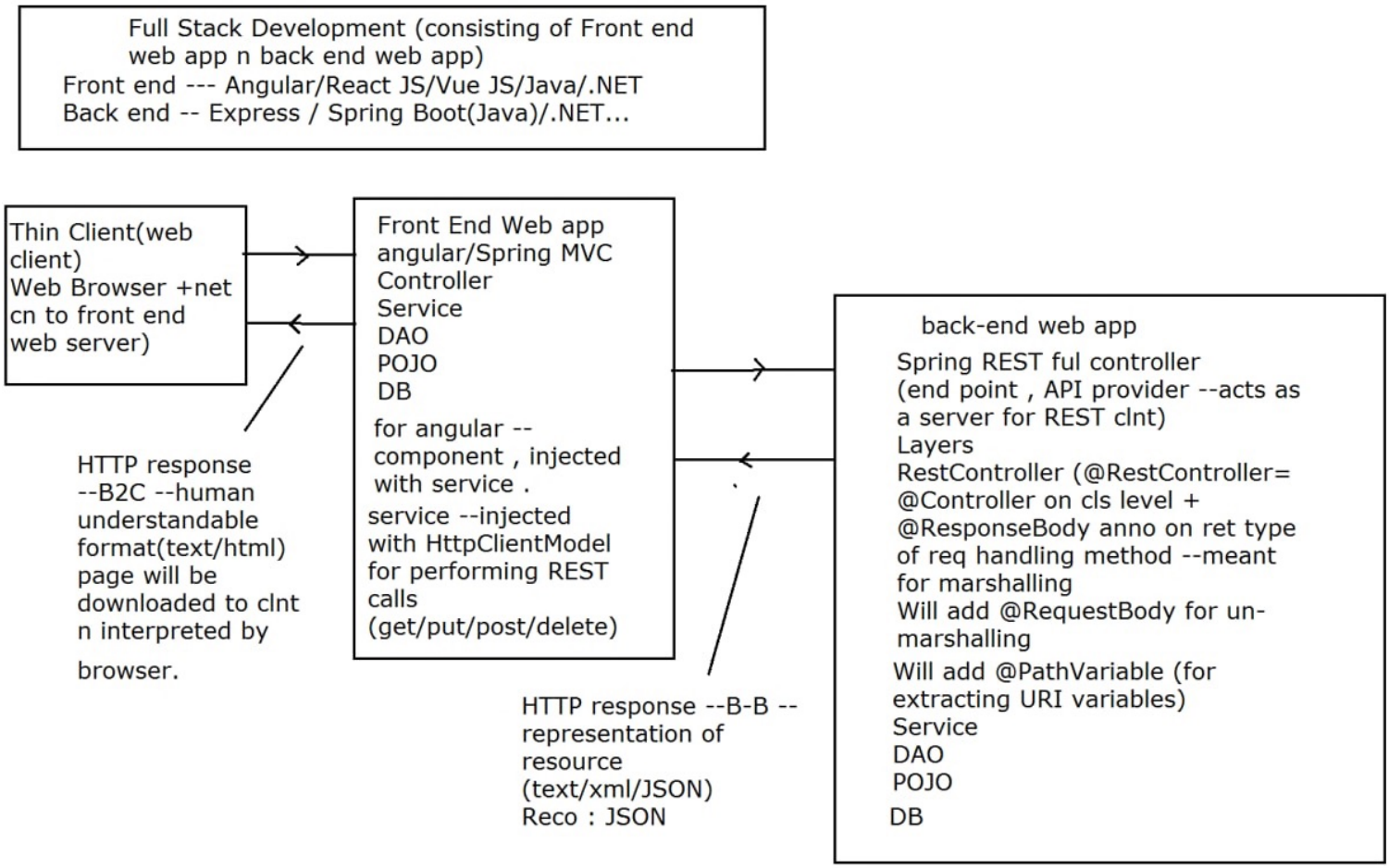
5. What's the meaning of HTTP status 400 : Bad Client Error

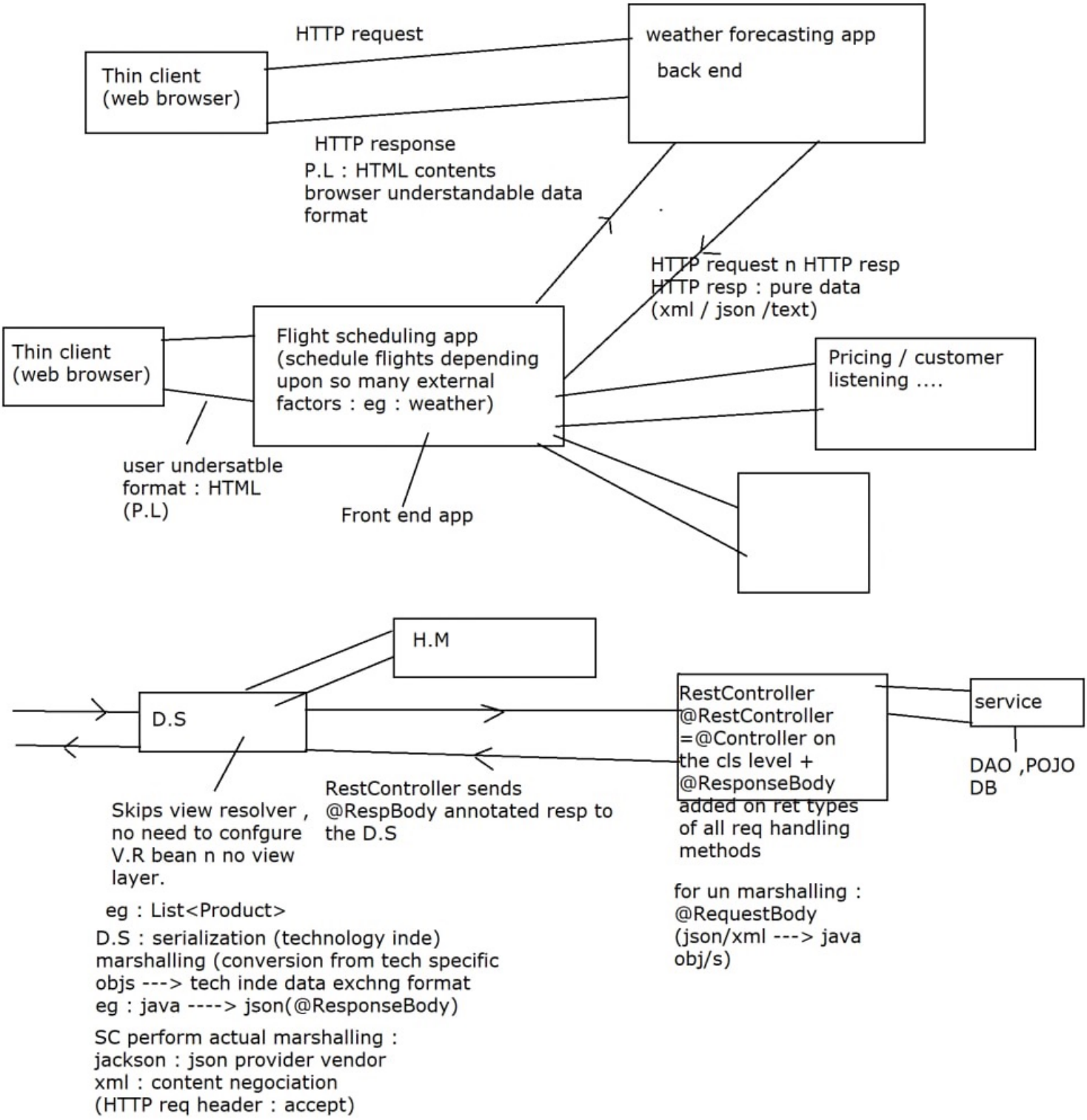
Typically it represents some request parameter conversion error.

Default date format used by SC : MM/dd/yyyy
In order to change it use : @DateTimeFormat annotation.

eg : @RequestParam("exp_date") @DateTimeFormat(pattern="yyyy-MM-dd")
Date expDate
SC invokes : SimpleDateFormat sdf=new SDF("yyyy-MM-dd");
Date expDate=sdf.parse(request.getParameter("exp_date"));

6. Integrate it with Hibernate




HTTP request
weather forecasting app
back end
Thin client
(web browser)
HTTP response
P.L : HTML contents
browser understandable data
format
HTTP request n HTTP resp
HTTP resp : pure data
(xml / json /text)
Thin client
(web browser)
Flight scheduling app
(schedule flights depending
upon so many external
factors : eg : weather)
Pricing / customer
listening ....
user undersatble
format : HTML
(P.L)
Front end app
H.M
D.S
RestController
@RestController
=@Controller on
the cls level +
@ResponseBody
added on ret types
of all req handling
methods
service
DAO ,POJO
DB
Skips view resolver ,
no need to confgure
V.R bean n no view
layer.
RestController sends
@RespBody annotated resp to
the D.S
eg : List<Product>
D.S : serialization (technology inde)
marshalling (conversion from tech specific
objs ---> tech inde data exchng format
eg : java ----> json(@ResponseBody)
SC perform actual marshalling :
jackson : json provider vendor
xml : content negociation
(HTTP req header : accept)
for un marshalling :
@RequestBody
(json/xml ---> java
obj/s)

Today's topics
Revision
Enter MVC : A design pattern

(w/o Spring Boot)
Spring MVC
Spring MVC +Hibernate

Revision
1. Spring Bean Life cycle
2. Different techniques of D.I
3. Hybrid Approach (reduced XML n majority annotations)

eg : UserController
@AutoWired
@Qulifier("my_usr_service")
private UserService userService;

SpEL --- spring expression language
dynamic expression languge ---spring(3.x) supplied -- to evaluate expressions dynamically.
#{SpEL expression} --- similar to JSP EL --- SpEL allows --- getters,setters,constr invocation, static & non-static method invocations.
OR
${SpEL expr}

What is MVC ?
Model-View-Controller --Standard design pattern , meant for separation of concerns(=tasks=responsibilities)

Model -- Java Bean (conversational state holder + B.L supplier) & POJOs

View layer --JSP , Thymeleaf/velocity/Angular/React/Vue
Represents UI / presentation logic (processing requests & generating response)

Controller -- Typically a servlet(used in Spring MVC) or a filter(used in Struts 2 framework)
Manages navigation & beans.

Front Controller -- A design pattern -- which ensures ANY request coming from ANY client , for this web app , will be intercepted by a common gate keeper(or a centralized dispatcher)
It will dispatch clnt request to further components , based upon nature of the req.

MVC flow (without spring)

refer : MVC diagrams , MVC Details
mvc-overview.png

Implementation using servlet / JSP & JavaBean --Shared as a readymade demo.

Refer to : eclipse projects\mvc_hibernate
Flow diagram : mvc-flow.png

Before entering spring-mvc

For comparison of layers between JSP--JB--DAO --POJO --DB Vs Spring MVC layers.

refer : comparison of layers JSP-JB vs Spring MVC.png

Enter Spring MVC --concept & implementation steps
Refer to : "lab sequence for spring mvc.txt"
---
1. Test Spring MVC flow.
(refer to lab sequence)

2. What is a model attribute?
It's the attribute(server side entry=k n value pair)
Purpose : to store the results of B.L
Who creates --- Req handling Controller(Handler)
Who sends it to whom : Handler ---> D.S
After D.S gets actual view name from V.R :
D.S chks : are there any model attrs :
Yes : D.S saves model attrs under Request scope & then forwards the clnt to view layer .
NO : D.S forwards the clnt to view layer .
How to access these model attrs from JSP ?
${requestScope.attrName}

What are different ways for Controller to add model attrs ?
1. Via is o.s.w.s.ModelAndView?

o.s.w.s.ModelAndView : class => holder for model attrs + logical view name

ModelAndView(String viewName,String modelAttrName,Object modelAttrVal)
eg : what can be valid ret type of req handling method
 ModelAndView

2. Any Simpler way of sending model attributes to D.S ?

Use o.s.ui.Model : i/f ---holder (Map) of model attributes
How do u add model attributes ?
public Model addAttribute(String modelAttrName,Object modelAttrVal)
eg : How to add 2 model attrs?
Use method chaining .

Who will supply empty Model map ? : SC
IoC : DI
How to tell SC that u need a model map ? : add it as the arg of req handling method
When controller rets string : logical view name (controller impl. rets all model map)

3. How to hide index.jsp under WEB-INF (i.e how to ensure that index page is served to client via D.S)
Add HomeController to serve index.jsp

Enter the case study
(online shopping)

----------Pending ----------

4. Handling request parameters in Controller ?
eg : @RequestParam("price") double productPrice
SC : double productPrice
=Double.parseDouble(request.getParameter("price"))
Reco : Match req param names with method arg names.
URL :
http://localhost:8080/day13.1/test/product?nm=pen&qty=10&price=40.5&manuDate=2020-1-1
SC : def date time format : mon/day/year
Problem : HTTP 400 : Bad request => some request data coming from client is invalid
Def date format : mon/day/yr
How to tell SC : about exact Date time format : annotation.

5. What's the meaning of HTTP status 400 : Bad Client Error
Typically it represents some request parameter conversion error.

Default date format used by SC : MM/dd/yyyy
In order to change it use : @DateTimeFormat annotation.

eg : @RequestParam("exp_date") @DateTimeFormat(pattern="yyyy-MM-dd")
Date expDate
SC invokes : SimpleDateFormat sdf=new SDF("yyyy-MM-dd");
Date expDate=sdf.parse(request.getParameter("exp_date"));

6. Integrate it with Hibernate

Model 1 and Model 2 (MVC) Architecture

Before developing any web application, we need to have idea about design models.
There are two types of programming models (design models)

Model 1 Architecture
Model 2 (MVC) Architecture

Model 1 Architecture

Servlet and JSP are the main technologies to develop the web applications.

Servlet is considered faster alternative to CGI. Servlet technology doesn't create process, rather it uses threads from a thread pool to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area , leading to better performance. Thus many subsequent requests can be easily handled by servlet.

Problem in Servlet technology Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

JSP overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated. You don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.

Advantage of Model 1 Architecture

Easy and Quick to develop web application

Disadvantage of Model 1 Architecture

Navigation control is decentralized since every page contains the logic to determine the next page.

If JSPs names are modified, we need to change it in all the pages leadining to the maintenance problem.

Time consuming.
Hard to extend It is better for small applications but not for large applications.

Enter Model 2 (MVC) Architecture

Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

Model The model represents the state (data) and business logic of the application. (JavaBean , POJOs)

View The view module is responsible to display data i.e. it represents the presentation.(JSP)

Controller : The Front controller module(Servlet/Filter) acts as an interface between view and model. It intercepts all the requests i.e. receives input and sends commands to Model / View to change accordingly.

Advantage of Model 2 (MVC) Architecture

Navigation control is centralized .
Now only controller contains the logic to determine the next page.
Easy to maintain
Easy to extend
Easy to test
Better separation of concerns

Disadvantage of Model 2 (MVC) Architecture
Developing centralized dispatcher (a navigation controller) is tedius, especially when web app size grows. That's the reason , Spring MVC is so popular , which implements Servlet based MVC pattern n supplies readymade components : Front Controller , Handler mapping , View Resolver ...

Today's Topics
Revision
Spring MVC n Hibernate revision
Enter Spring Boot n port earlier application
JPA
Spring Data JPA

Revise
1.Spring MVC with hibernate implementation steps n concepts(flow)
1.1 Refer : "day13_data\day13_help\diagrams\comparison of layers JSP-JB vs Spring MVC.png"

1.2 Refer to steps n Spring MVC flow diagram
(refer to the detailed explanation below)

Which are the important blocks of Spring MVC ?

1. o.s.w.s.DispatcherServlet
Entry point of the spring MVC. Front Controller pattern
Centralized dispatcher
Will be intercepting all reqs coming from all clnts.
Managed by WC
web.xml . Life cycle started at the deployment time .
init ---> job of the D.S --> hybrid --xml
Def Name of master config file to start SC : spring-servlet.xml
Def loc : WEB-INF
read by D.S (DispatcherServlet)
API of SC : o.s.b.f.BeanFactory <---- o.s.c.ApplicationContext : for Java SE : in hybrid approach <---o.s.c.s.ClasspathXmlApplcationContext : instance , to start SC

In web app : API of SC : o.s.b.f.BeanFactory <----
o.s.c.ApplicationContext <---- WebApplicationContext<-----
XmlWebApplicationContext : it's instance created by D.S @ web app dep time.

2. Request Handling Controller (Handler)
: prog supplied
Mandatory annotations :
@Controller : cls level annotation
@RequestMapping : method level annotation
which all HTTP requests , it can intercept ? : GET , POST , PUT , DELETE .....
@GetMapping : get
@PostMapping : post
@PutMapping : put
@DeleteMapping : delete

3. HandlerMapping Bean
auto populated by SC : by looking at req handling methods (@RequestMapping) present in req handling controller(@Controller)
Map
key : value of @ReqMapping(or it's sub type) annotation eg : /hello
value : F.Q Handler class Name+ Method name

4. View Resolver :
Job : Translation from Logical view (forward view name) ---> Actual view name

propeties : prefix , suffix + viewClass : JstlView (to enable JSTL tags/actions)
AVN(actual view name) = prefix + LVN + suffix
eg : /WEB-INF/views+LVN+.jsp

5. View Layer (JSP) : prog supplied

6. What is a model attribute?
It's the attribute(server side entry=k n value pair : String, Object)
Purpose : to store the results of B.L
Who creates --- Req handling Controller(handler) --prog supplied
Who sends it to whom : Handler ---> D.S
After D.S gets actual view name from V.R :
D.S chks : are there any model attrs :
Yes : D.S saves model attrs under Request scope & then forwards the clnt to view layer .
NO : D.S forwards the clnt to view layer .
How to access these model attrs from JSP ?
${requestScope.attrName}

What are different ways for handler to add model attrs ?
6.1 Via   o.s.w.s.ModelAndView?

o.s.w.s.ModelAndView : class => holder for model attrs + logical view name
Ctor :
ModelAndView(String viewName,String modelAttrName,Object modelAttrVal)
eg : what can be valid ret type of req handling method
String OR ModelAndView : ModelAndView

6.2Any Simpler way to send model attributes from Handler --> D.S ?

Use o.s.ui.Model : i/f ---holder (Map) of model attributes
How do u add model attributes ?
public Model addAttribute(String modelAttrName,Object modelAttrVal)
eg : How to add 2 model attrs? : method chaining

Who will supply empty Model map (as the dependency)? : SC
IoC : DI
How to tell SC that handler method needs a model map ? : add it as the arg of req handling method
When req handling controller rets string : logical view name (Handler implicitly rets all model attr map to D.S)

7. Handling request parameters in Controller ?
Request Handling method argument levele annotation
@RequestParm
To bind request paramters to the method arguments

eg : @RequestParam("price") double productPrice
SC : double productPrice
=Double.parseDouble(request.getParameter("price"))

Reco : Match req param names with method arg names.

eg : URL :
http://localhost:8080/day13.1/test/product?nm=pen&qty=10&price=40.5&manuDate=2020-1-1

Problem : HTTP 400 :
Bad request => some request data coming from client is invalid
Default date format : MM/dd/yyyy
To tell SC : use @DateTimeFormat annotation , along with @RequestParam

Test Data

Add this data in DB

```
id            | bigint      | NO   | PRI | NULL    | auto_increment |
| card_no       | varchar(20) | YES  | UNI | NULL    |                |
| creation_date | date        | YES  |     | NULL    |                |
| location      | varchar(30) | YES  |     | NULL    |                |
| email         | varchar(25) | YES  | UNI | NULL    |                |
| first_name    | varchar(20) | YES  |     | NULL    |                |
| last_name     | varchar(20) | YES  |     | NULL    |                |
| password      | varchar(20) | NO   |     | NULL    |                |
| user_role
```

```
insert into users_tbl values(default,'abc1234567','2019-1-
1','Pune','rama@gmail.com','Rama','Kadam','12345','ADMIN');
insert into users_tbl
values(default,'riya@gmail.com','Riya','Patel','1345','CUSTOMER','abc1234
568','2018-1-1','Nagpur');
insert into users_tbl
values(default,'mihir@gmail.com','Mihir','Rao','2345','CUSTOMER','abc1534
567','2020-1-1','Delhi');

insert into categories values(default,'Food Items','Tasty Food
Items!!!!');
insert into categories values(default,'Clothes','Smart Clothes!!!!');

insert into products values(default,'Different Types of
Flour',1,75,'Wheat Flour',1);
insert into products values(default,'Different Types of
Flour',1,70,'Jowar Flour',1);
insert into products values(default,'Different Types of Flour',65,1,'Rice
Flour',1);
insert into products values(default,'Different Types of
Flour',80,1,'Besan',1);

insert into products values(default,'Stylish T Shirts',1,500,'T
Shirt',2);
insert into products values(default,'Comfortable Jeans',1,1500,'Comfort
Jeans',2);
insert into products values(default,'Smart Kurta',1,900,'Kurta',2);
insert into products values(default,'Fitting
Trousers',1,1800,'Trouser',2);
```

7. Enter spring boot.

(refer to : https://docs.spring.io/spring-boot/docs/current/api/ , for spring boot latest API docs)

7.1 Port existing web app to spring boot

8. 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:/customer/categories";
Internals
D.S skips the V.R & sends temp redirect response to the clnt browser.
How ?
D.S invokes ---
response.sendRedirect(response.encodeRedirectURL("/customer/categories"))
;
So clnt browser will send a next request ---with method=GET
URL --
http://host:port/day13_boot/customer/categories

URL after choosing a category :
http://localhost:8080/day13_boot/customer/products;jsessionid=8210C7A572E0BF63BBB8176C40602A86?catId=1

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

8. How to remember the details(attributes) till only 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.

9. How to ensure all links(href)/form actions relative to current web app + add URL rewriting support ?

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

2. Use the tag.
<a href="<spring:url value='/user/logout'/>">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
<spring:url var="url" value='/admin/list'/>
eg : <form action="${var}">
form inputs
</form>

10. 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=any url you want to redirect to (eg : home page url (root of web app))
10 : delay in seconds

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

Problems with traditional/legacy spring framework

We use different modules from spring such as core module, to do dependency injection.The MVC module to develop the web layer for our application or even the restful web services layer.And then the DAO layer where we use the spring JDBC/ORM which makes our life easy to develop a data access layer for our application. When we are using ORM tools like Hibernate, we can use spring data JPA and we use these modules and more that are valuable from Spring.

Initially we used XML based configuration or annotations based configuration,This configuration can get difficult and hard to maintain over time.And also we need to make sure that each of these modules is available for our application by defining all the dependencies in the Maven pom xml.And at runtime we have to be sure that these versions of various Modules that we use are compatible with each other.But it's our responsibility to do all that, and once we have all that in place we will build our application and will have to deploy to an external web container to test it .

Spring boot will automate all this for us.

What are Spring Boot Features ?

1. The first of those super cool features is auto configuration - Spring Boot automatically configures everything that is required for our application. We don't have to use XML or Java configuration anymore.

For example if you are using Spring MVC or the web to develop a web application or a restful web service application spring boot will automatically configure the dispatcher servlet and does all the request mapping for us. We don't have to use any xml or annotation based configuration to configure this servlet.

Similarly if you are using spring data or object relational mapping while working with tools like Hibernate to perform database crud we no longer have to configure the data source or even the transaction manager. Spring boot will automatically configure these for our application.

2. The next super cool feature is spring boot starters .With the spring boot starters spring boot takes the problem off module availability we need. Before Spring Boot we had to make sure that a particular library required for our project is available and also the versions of different libraries are compatible.But we don't have to do that anymore thanks to spring boot starters every spring boot project will have a parent project.This project has all the version information of various libraries that we will be using in our project so we need not worry about version compatibility. The spring developers have already done it for us and put all that information into this spring boot starter parent .

Secondly we have starters for different types of projects if we are developing a web project then we simply need to include the starter web . We don't have to include any other libraries or dependencies. Spring boot will automatically pull all the other libraries that are required because this project here or this or dependency transitively depends on other libraries that are required. Maven will automatically pull those libraries. The spring boot developers have given us starter dependencies which when we use in our projects will not have the Modular availability problem and the version compatibility problem.

Another example is the spring boot starter data jpa .When you want to work with Hibernate you simply include the single dependency in your maven pom xml and all the other libraries will be pulled accordingly. And also the correct versions of those libraries will be included because all that version information is available in this spring boot starter parent which is a parent for every spring boot project.

3. The third super cool feature we don't have to worry about deploying our applications to external container spring boot comes with an embedded servlet container and these containers.

By default it is Tomcat but you can also use Jetty and undertow or any other external server. So no longer external deployment you can simply right click on your project and run it and your application will be launched on its embedded Tomcat server by default.

4. Last and very important spring boot gives us a lot of health checks of our application for free through the spring boot actuators.We can use different types of health checks that come for free and we can use these even on production when our application is running. These can be activated easily and will display all the auto configuration reports and everything that is automatically configured for our application.

What is Spring Boot
=Spring Framework + Embedded web server (Tomcat) -(complex config setting / complex dependency management in pom.xml)

Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

	public static void main(String[] args) {
		SpringApplication.run(HelloSpringBootApplication.class, args);
	}

}
```

About : org.springframework.boot.SpringApplication
It's Class used to bootstrap and launch a Spring application from a Java main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC)
2. Manages life cycle of spring beans
3. Launches the embedded Tomcat Container

@SpringBootApplication - This is where all the spring boot magic happens. It consists of following 3 annotations.

1. @SpringBootConfiguration

It tells spring boot that this class here can have several bean definitions(@Bean annotation configured methods , equivalent to <bean> tag in xml based configuration)
We can define various spring beans here and those beans will be available at run time .

2. @EnableAutoConfiguration
It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source,using the properties in application.properties file.

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

3. @ComponentScan (equivalent to xml tag : context:component-scan)

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans .
Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :
@ComponentScan(basePackages = "com")
For scanning entities : (equivalent to packagesToScan)
@EntityScan(basePackages = "com.app.pojos")

Below is the starting point of a Spring Boot Application

@SpringBootApplication

public class HellospringbootApplication { p.s.v.m(...) {...}}

@SpringBootApplication - This is where all the spring boot magic happens.

The SpringBootApplication is a key annotation which is a top level annotation which contains several other annotations on it.

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan

The first one @SpringBootConfiguration tells spring boot or the container that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time so you define a method here .

The second annotation @EnableAutoConfiguration is a very important annotation at enable Auto configuration.This annotation tells spring boot

to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml as automatically Spring Boot will create a data source. We will also provide other information like username password etc. but spring boot will scan through all these dependencies and it will automatically configure the data source required for us.

Another example is spring web, If we have spring web in your dependencies.

Then spring boot will automatically create the dispatcher servlet on all that configuration file you for free.

All the xml, all the java based configuration is now gone. We as developers need not do all that configuration it all comes for free thanks to spring boots to enable auto configuration annotation.

@ComponentScan
So this tells us that spring boot or spring should scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans . Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :
@ComponentScan(basePackages = "com")
For scanning entities :
@EntityScan(basePackages = "com.app.pojos")

1. @SpringBootApplication : is added on the main class , in spring boot application , to run it as a standalone JAR / WAR

@SpringBootApplication =

1.1 @Configuration – Designates this main class as a configuration class for Java configuration. In addition to beans configured via component scanning, an application may want to configure some additional beans via the @Bean annotation as demonstrated here. Thus, the return value of methods having the @Bean annotation in this class are registered as beans.

1.2 @EnableAutoConfiguration – This enables Spring Boot's autoconfiguration mechanism. Auto-configuration refers to creating beans automatically by scanning the classpath.

1.3 @ComponentScan – Typically, in a Spring application, annotations like @Component, @Configuration, @Service, @Repository are specified on classes to mark them as Spring beans. The @ComponentScan annotation basically tells Spring Boot to scan the current package and its sub-packages in order to identify annotated classes and configure them as Spring beans. Thus, it treats the current package as the root package for component scanning.

Annotation added automatically :
@EnableWebMvc: Marks the application as a web application and activates setting up a DispatcherServlet , HandlerMapping , ViewResolvers . Spring Boot adds it automatically when it sees spring-webmvc on the classpath.

2. How to use the @SpringBootApplication annotation
In order to run a Spring Boot application, it needs to have a class with the @SpringBootApplication annotation.

eg :

```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class Main {
    public static void main(String[] args) {
          SpringApplication.run(Main.class, args);
      }
}
```

The Main class has the @SpringBootApplication annotation

It simply invokes the SpringApplication.run method.
This starts the Spring application as a standalone application, runs the embedded servers and loads the beans.

Normally, such a main class is placed in a root package above other packages. This enables component scanning to scan all the sub-packages for beans.

Steps

1. File --New --Spring starter project -- add project name , group id ,artifact id ,pkg names , keep packaging as JAR for Spring MVC web application.

2. Add dependencies
web -- web
sql -- Spring Data JPA, MYSQL
dev -- DevTools
Lombok
validation

3. Copy the entries from supplied application.properties & edit DB configuration.

4.For Spring MVC (with JSP view layer demo) using spring boot project

Add following dependencies ONLY for Spring MVC with JSP as View Layer in pom.xml

```
	<!-- Additional dependencies for Spring MVC -->
		<dependency>
			<groupId>org.apache.tomcat.embed</groupId>
			<artifactId>tomcat-embed-jasper</artifactId>
		</dependency>

		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
		</dependency>
```

5. Create under src/main : /webapp/WEB-INF/views   folder

6. R Click on the spring boot project --Run As --Spring Boot App

7. Port earlier spring MVC app , observe the problems.
& fix it.
Any problems : Yes
No session context !

Solution : Replace hibernate's native API (org.hibernate) by JPA
(refer : day13-data\day13_help\diagrams\jpa-entitymgr-session-layers.png)

In DAO layer : replace native hibernate API by JPA
i.e instead of auto wiring SF in DAO layer : inject JPA' EntityManager directly in DAO.

How ?
```
@PersistenceContext
//OR can continue to use @AutoWired : spring supplied annotation
private EntityManager mgr;
//uses def persistence unit , created auto by spring boot using db setting added //in application.properties file , 1 per app / DB
```

Use directly EntityManager API (refer :)

8. Test Entire application.

1. Revise n complete Customer Flow till logout
1.1 Use lombok annotations to reduce boilerplate code
1.2 Revise n complete flow.
(at least till displaying the product details n adding to cart mesg)

2. Add Option for User Registration (Data Binding technique) : Tomorrow or in the lab
+ Creating an empty cart n associating it with user.

3. Enter Spring Data JPA. Modify the DAO layer.
(refer : regarding Spring Data JPA)
eg : Emp e=empRepo.findById(100).orElseThrow(() -> new ResNotFoundExc("invalid emp id"));
3.1 Inherited APIs (eg : findAll , findById,save, deleteAll,count ....)
3.2 Derived Query methods
eg : Optional<Employee> findByFirstNameOrLastName(String fName,String lastName)
....
3.3 Custom Query Methods
Can inject JPQL using @Query annotations
```
@Query("select p from Product p where p.productCategory.name=?1")
List<Product> fetchProductsByCategoryName(String name);
```
OR
```
@Query("select p from Product p where p.productCategory.name=:nm")
List<Product> fetchProductsByCategoryName(@Param(name="nm") String name);
```

If you want to use (as a last resort in the project phase! : not reco)
sql :
```
@Query(value="select * from products where name=?1",nativeQuery=true)
List<Product> fetchProductsByCategoryName(@Param(name="nm") String desc);
```

4. REST concepts

refer to :
Diag : web app vs web service
Diag : Full Stack Development
Readme : RestController vs MVC Controller n Annotations.txt

Steps
1. Create spring boot app : using spring boot starter project (choose packaging as JAR)
2. Use same spring boot starters as earlier.
Spring web , Mysql driver , Spring data JPA , Lombok , Spring Dev Tools,validation
3. NO additional dependencies for view layer(i.e no jstl n no tomcat-embeded jasper dependencies , in pom.xml

4. Copy application.properties from earlier spring boot project
(Do not add view resolver related properties)

5. Build the layers in bottoms up manner, for the following objectives.

Objective : Complete backend for User Management

1. Get All Users

Later Use ResponseEntity , to wrap response body + response headers.
2. Add User Details :

3. Delete User Details
4. Get User details by id
i/p : user id
o/p : exsiting user details
5. Update User details
i/p : user id , updated user details

Test it with postman & then with React front end(in future!)

Unit Testing / Integration Testing

What is a web service ?
A solution to distributed computing.

Integral part of SOA (service oriented architecture)
service = Business functionality to be exported to remote clnts via standard protocols(HTTP/s)
server -- service provider
clnt --service accessor

Why -- To export the Business logic (functional logic --banking, customer service, payment gateway , stock exchange server BSE , NSE ...) to remote clients over standard set of protocols.

Its equivalent to Java RMI (remote method invocation)
In Java RMI -- java clnt object can directly invoke the remote method (hosted on the remote host) & get n process results. (i.e it gives you location transperency)
BUT Java RMI ---is 100% java solution.
There is no inter operability in that(i.e its a technologoy specific soln)
How to arrive at technology inde soln ?

CORBA --- Common obj request borker architecture
tough to set up. (IDL ---i/f def language)
Better alternative --- web services
In 2002, the World Wide Web consortium(W3C) had released the definition of WSDL(web service definition language) and SOAP web services. This formed the standard of how web services are implemented.
Earlier (J2EE 1.4) -- JAX-RPC
Java API for XML based remote procedure calls
Today replaced by JAX-WS

In 2004, the web consortium also released the definition of an additional standard called RESTful. Over the past couple of years, this standard has become quite popular. And is being used by many of the popular websites around the world which include Facebook and Twitter.

Corresponding J2EE specification JAX RS

JAX WS -- Java API for XML based web services -- Based upon
Protocol --SOAP -- simple object access protocol (runs over HTTP)
Has additional header & message format .
+ Have to set up Naming service (UDDI --Universal Description, Discovery, and Integration)
+ Have to set up WSDL (web service def. language)-- xml based web service def lang.
--supports only XML as data exchange format.

Too much to set up & eats up larger bandwidth !!
So a simple soln is JAX RS -- Java API for RESTful web service
JAX RS --- is a part of J2EE specifications
Known Vendors --Apache , JBoss
& products --RESTeasy,Apache CXF
BUT it's still difficult to set up.
So spring , being integration master , comes to the rescue.....

The six architectural constraints of REST APIs

1. Client-server architecture
An API's job is to connect two pieces of software without limiting their own functionalities. This objective is one of the core restrictions of REST: the client (that makes requests) and the server (that gives responses) stay separate and independent.

When done properly, the client and server can update and evolve in different directions without having an impact on the quality of their data exchange. This is especially important in various cases where there are plenty of different clients a server has to cater to. Think about weather APIs — they have to send data to tons of different clients (all types of mobile devices are good examples) from a single database.

2. Statelessness
For an API to be stateless, it has to handle calls independently of each other. Each API call has to contain the data and commands necessary to complete the desired action.

An example of a non-stateless API would be if, during a session, only the first call has to contain the API key, which is then stored server-side. The following API calls depend on that first one since it provides the client's credentials.

In the same case, a stateless API will ensure that each call contains the API key and the server expects to see proof of access each time.

Stateless APIs have the advantage that one bad or failed call doesn't derail the ones that follow.

3. Uniform Interface
While the client and the server change in different ways, it's important that the API can still facilitate communication. To that end, REST APIs impose a uniform interface that can easily accommodate all connected software.

In most cases, that interface is based on the HTTP protocols. Besides the fact that it sets rules as to how the clients and the server may interact, it also has the advantage of being widely known and used on the Internet. Data is stored and exchanged through JSON files because of their versatility.

4. Layered system
To keep the API easy to understand and scale, RESTful architecture dictates that the design is structured into layers that operate together.

With a clear hierarchy for these layers, executing a command means that each layer does its function and then sends the data to the next one. Connected layers communicate with each other, but not with every component of the program. This way, the overall security of the API is also improved.

If the scope of the API changes, layers can be added, modified, or taken out without compromising other components of the interface.

5. Cacheability
It's not uncommon for a stateless API's requests to have large overhead. In some cases, that's unavoidable, but for repeated requests that need the same data, caching said information can make a huge difference.

The concept is simple: the client has the option to locally store certain pieces of data for a predetermined period of time. When they make a request for that data, instead of the server sending it again, they use the stored version.

The result is simple: instead of the client sending several difficult or costly requests in a short span of time, they only have to do it once.

6. Code on Demand

Unlike the other constraints we talked about up to this point, the last one is optional. The reason for making “code on demand” optional is simple: it can be a large security risk.

The concept is to allow code or applets(now obsolete!) to be sent through the API and used for the application. As you can imagine, unknown code from a shady source could do some damage, so this constraint is best left for internal APIs where you have less to fear from hackers and people with bad intentions. Another drawback is that the code has to be in the appropriate programming language for the application, which isn’t always the case.

The upside is that “code on demand” can help the client implement their own features on the go, with less work being necessary on the API or server. In essence, it permits the whole system to be much more scalable and agile.

What is REST ?

REST stands for REpresentational State Transfer.

REST is web standards based architecture and uses HTTP Protocol for data communication.

It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents the resources.

Here each resource is identified by URIs

REST uses various representations to represent a resource like text, JSON and XML. Most popular light weight data exchange format used in web services = JSON

HTTP Methods
Following well known HTTP methods are commonly used in REST based architecture.

GET - Provides a read only access to a resource.

POST - Used to create a new resource.

DELETE - Used to remove a resource.

PUT - Used to update a existing resource or create a new resource.

PATCH -- Used to perform partial updates to the resource.

REST API Meaning

REST stands for REpresentational State Transfer.
API stands for Application Program Interface.

REST is a software architectural style that defines the set of rules to be used for creating web services(that are resource oriented)

Web services which follow the REST architectural style are known as RESTful web services.

It allows requesting systems)(REST Client) to access and manipulate web resources by using a uniform and predefined set of rules , using standard Hypertext Transfer Protocol (HTTP).

What does a Restful web system consists of

1. client who requests for the resources.
2. server who has the resources.

It is MANDATORY to create REST API as per industry standards which results in ease of development and increase client support.

Architectural Constraints of RESTful API

Uniform Interface
Stateless
Cacheable
Client-Server
Layered System
Code on Demand
The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

Details :

What is Restful Web Service?

REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. REST stands for REpresentational State Transfer

The key elements of a RESTful implementation are as follows:

1. Resources The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is http://www.server.com. Now in order to access an employee record resource via REST, one can issue the command http://www.server.com/employee/1 - This command tells the web server to please provide the details of the employee whose employee number is 1.

2. Request Verbs - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example http://www.server.com/employee/1 , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.

3. Request Headers These are additional instructions sent with the request. These might define the type of response required or the authorization details.

4. Request Body - Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.

5. Response Body This is the main body of the response. So in our example, if we were to query the web server via the request http://www.server.com/employee/1 , the web server might return an XML document with all the details of the employee in the Response Body.

6. Response Status codes These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

Links
1. https://gist.github.com/alexserver/2fcc26f7e1ebcfc9f6d8

A key difference between a traditional MVC controller and the RESTful web service controller is the way that the HTTP response body is created.Instead of relying on a view technology(JSP / Thymeleaf) to perform server-side rendering of the data to HTML, typically a RESTful web service controller simply populates and returns a java object. The object data will be written directly to the HTTP response as JSON/XML/Text

To do this, the @ResponseBody annotation on the ret type of the request handling method tells Spring MVC that it does not need to render the java object through a server-side view layer.

Instead it tells that the java object returned is the response body, and should be written out directly.

The java object must be converted to JSON. Thanks to Spring's HTTP message converter support, you don't need to do this conversion manually. Because Jackson Jar is on the classpath, SC can automatically convert the java object to JSON & vice versa (using 2 annotations @ReponseBody & @RequestBody)

API --Starting point
o.s.http.converter.HttpMessageConverter<T>
--Interface that specifies a converter that can convert from and to HTTP requests and responses.
T --type of request/response body.

Implementation classes
1. o.s.http.converter.xml.Jaxb2RootElementHttpMessageConverter
-- Implementation of HttpMessageConverter that can read and write XML using JAXB2.(Java architecture for XML binding)

2. o.s.http.converter.json.
MappingJackson2HttpMessageConverter
--Implementation of HttpMessageConverter that can read and write JSON using Jackson 2.x's ObjectMapper class API

Important Annotations
1. @ResponseBody

Applied at : return value of the request handling method , annotated with @RequestMapping or @GetMapping / @PostMapping / @PutMapping / @DeleteMapping

It is used to marshall(serialize) the return value into the HTTP response body. Spring comes with converters that convert the Java object into a format understandable for a client(text/xml/json)

eg :
```
@Controller
@RequestMapping("/employees")
public class EmpController
{
   @GetMapping(....)
   public @ResponseBody Emp fetchEmpDetails(int empId)
   {
      //get emp dtls from DB through layers
      return e;
   }
```

```
}
```

2. @RestController
Class level annotation

Good news is @RestController = @Controller(at the class level) + @ResponseBody added on ret types of ALL request handling methods

eg :

```
@RestController
@RequestMapping("/employee")
public class EmpController
{
   @GetMapping(....)
   public Emp fetchEmpDetails(int empId)
   {
      //get emp dtls from DB through layers
      return e;
   }.....
}
```

3. @PathVariable --- handles URI templates.(URI variables or path variables)
Where to apply : on the method argument
Purpose : to access a path variable

eg : URL -- http://host:port/products/1234

Method of ProductController

```
@RestController
@RequestMapping("/products") {
@GetMapping("/{pid}")
public Product getDetails(@PathVariable(name="pid") int pid1234)
{...}
}
```

Can change pie to anything

In the above URL , the path variable {pid} is mapped to an int . Therefore all of the URIs such as /products/1 or /products/10 will map to the same method in the controller.

4. The @RequestBody annotation, unmarshalls the HTTP request body into a Java object injected in the method.
Applied on the method argument of the reuqest handling methods , containing request body
eg : Typically in POST , PUT , PATCH

@RequestBody must be still added on a method argument of request handling method , for un marshaling(de serialization)

5. @CrossOrigin @Crossorigin (Origins = http://local host:3000)
Class/Method level annotation
To be placed over class typically for connecting it to any browser side app

What is CORS ?
Cross-Origin Resource Sharing (CORS)

CORS is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading of resources.

A cross-origin HTTP request is a request to a specific resource, which is located at a different origin, namely a domain, protocol and port, than the one of the client performing the request.

For security reasons, browsers can request several cross-origin resources, including images, CSS, JavaScript files etc. By default, a browser' security model will deny any cross-origin HTTP request performed by client-side scripts.

While this behavior is desired, to prevent different types of Ajax-based attacks, sometimes we need to instruct the browser to allow cross-origin HTTP requests from JavaScript clients with CORS.

eg : React client running on http://localhost:3000, and a Spring Boot RESTful web service API listening at http://host:port/products

In such a case, the client should be able to consume the REST API, which by default would be forbidden.

To make this work , enable CORS by simply annotating the class / methods of the RESTful web service API responsible for handling client requests with the @CrossOrigin annotation

```
eg : @CrossOrigin(origins = "http://localhost:3000")
@RestController
public class ProductController{....}
```

Spring boot internals explained :

Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

	public static void main(String[] args) {
		SpringApplication.run(HelloSpringBootApplication.class, args);
	}

}
```

About : org.springframework.boot.SpringApplication — class
It's Class used to bootstrap and launch a Spring application from a Java main method. It will launch the SpringApplication from the main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC) Based upon default configuration
2. Manages life cycle of spring beans
3. Launches embedded Tomcat container

@SpringBootApplication - This is where all the spring boot magic happens. It consists of following 3 annotations.

1. @SpringBootConfiguration

this is to tell spring Boot Cont. this main class can contain several spring bean config.

It tells spring boot that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time . e.g – Model Mapper (used for mapping Entity ↔ DTO), password encoder

2. @EnableAutoConfiguration

It added default configuration

It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source,using the properties in application.properties file. Connection Pool

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

3. @ComponentScan (equivalent to xml tag : context:component-scan)

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans .
Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :

```
@ComponentScan(basePackages = "com")
```

For scanning entities : (equivalent to packagesToScan)

```
@EntityScan(basePackages = "com.app.pojos")
```

Enter Spring boot

1. What is Spring Boot?
Spring Boot is a Framework from "The Spring Team" to ease the bootstrapping and development of new Spring Applications.

It provides defaults for code and annotation configuration to quick-start new Spring projects within no time.

It follows "Opinionated Defaults Configuration" an approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

2. What is NOT Spring Boot?
Spring Boot Framework is not implemented from the scratch by The Spring Team
It's implemented on top of existing Spring Framework (Spring IO Platform).
It is not used for solving any new problems. It is used to solve same problems like Spring Framework.
(i.e to help in writing enterprise applications)

3. Advantages of Spring Boot:

It is very easy to develop Spring Based applications with Java
It reduces lots of development time and increases productivity.
It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.

It follows "Opinionated Defaults Configuration" Approach to reduce Developer effort

It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.

It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy)
Applications from command prompt very easily and quickly.

It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle

It provides lots of plugins to work with embedded and in-memory Databases very easily.

In short
Spring Boot = Spring Framework + Embedded HTTP Server(eg Tomcat) - XML Based configuration - efforts in identifying dependencies in pom.xml

4. What is that "Opinionated Defaults Configuration" ?

When we use Hibernate/JPA, we would need to configure a datasource, a session factory, a transaction manager among lot of other things.
Refer to our hibernate-persistence.xml , to recall what we did earlier .

Spring Boot says can we look at it differently ?

Can we auto-configure a Data Source(connection pool) / session factory / Tx manager if Hibernate jar is on the classpath?

It says :
When a spring mvc jar is added into an application, can we auto configure some beans automatically?
(eg HandlerMapping , ViewResolver n configure DispatcherServlet)

By the way :
There would be of course provisions to override the default auto configuration.

5. How does it work ?
Spring Boot looks at
1. Frameworks available on the CLASSPATH
2. Existing configuration for the application.
Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called Auto Configuration.

6. What is Spring Boot Starter ?
Starters are a set of convenient dependency descriptors that you can include in your application's pom.xml
.
eg : Suppose you want to develop a web application.

W/o Spring boot , we would need to identify the frameworks we want to use, which versions of frameworks to use and how to connect them together.

BUT all web application have similar needs.
 These include Spring MVC, Jackson Databind (for data binding), Hibernate-Validator (for server side validation using Java Validation API) and Log4j (for logging). Earlier while creating any web app, we had to choose the compatible versions of all these frameworks.

With Spring boot : You just add Spring Boot Starter Web.

Dependency for Spring Boot Starter Web
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Just by adding above starter , it will add lot of JARs under maven dependencies

Another eg : If you want to use Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.

7. Another cool feature of Spring boot is : we don't have to worry about deploying our applications to external container. It comes with an embedded servlet container.

8.Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

	public static void main(String[] args) {
		SpringApplication.run(HelloSpringBootApplication.class, args);
	}

}
```

About : org.springframework.boot.SpringApplication
It's Class used to bootstrap and launch a Spring application from a Java main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC)
2. Manages life cycle of spring beans

@SpringBootApplication - This is where all the spring boot magic happens. It consists of following 3 annotations.

1. @SpringBootConfiguration
It tells spring boot that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time .

2. @EnableAutoConfiguration
It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source,using the properties in application.properties file.

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

3. @ComponentScan (equivalent to xml tag : context:component-scan)

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans .
Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :
```
@ComponentScan(basePackages = "com")
```
For scanning entities : (equivalent to packagesToScan)
```
@EntityScan(basePackages = "com.app.pojos")
```

Steps

1. File --New --Spring starter project -- add project name , group id ,artifact id ,pkg names , keep packaging as JAR for Spring MVC web application.

2. Add dependencies
web -- web
sql -- Spring Data JPA, MYSQL
Core -- DevTools
Lombok
validation

3. Copy the entries from supplied application.properties & edit DB configuration.

4.For Spring MVC (with JSP view layer demo) using spring boot project

Add following dependencies ONLY for Spring MVC with JSP as View Layer in pom.xml

```
	<!-- Additional dependencies for Spring MVC -->
		<dependency>
			<groupId>org.apache.tomcat.embed</groupId>
			<artifactId>tomcat-embed-jasper</artifactId>
		</dependency>

		<dependency>
			<groupId>javax.servlet</groupId>
			<artifactId>jstl</artifactId>
		</dependency>
```

5. Create under src/main/webapp : WEB-INF folder

6. Create `r n test it.

7. Port earlier spring MVC app , observe the problems.
& fix it.

Problem observed : ??????

Reason : Could not find org.hibernate.SessionFactory (since Spring boot DOES NOT support any native hibenrate implementationj directly)

Solution : Replace hibernate's native API (org.hibernate) by JPA
(refer : day17-data\day17_help\diagrams\jpa-entitymgr-session-layers.png)

In DAO layer : replace native hibernate API by JPA
i.e instead of auto wiring SF in DAO layer : inject JPA' EntityManager directly in DAO.

How ?

```
@PersistenceContext
//OR can continue to use @AutoWired : spring supplied annotation
private EntityManager mgr;
//uses def persistence unit , created auto by spring boot using db setting added //in application.properties file , 1 per app / DB
```

Use directly EntityManager API (refer :)
OR
Unwrap hibernate Session from EntityManager

```
Session session = mgr.unwrap(Session.class);
```

Which one is preferred ? 1st soln.

8. Test Entire application.

The relationship between Spring Data JPA, JPA, and Hibernate/EclipseLink/Kodo n /JDBC

JPA is a part of Java EE/Jakarta EE specification that defines an API for ORM and for managing persistent objects. Hibernate and EclipseLink are popular implementations of this specification.

Spring Data JPA adds a layer on top of JPA. That means it uses all features defined by the JPA specification, especially the entity and association mappings, the entity lifecycle management, and JPA's query capabilities. On top of that, Spring Data JPA adds its own features like a no-code implementation of the repository pattern and the creation of database queries from method names.

If the JPA specification and its implementations provide most of the features that you use with Spring Data JPA, do you really need the additional layer? Can't you just use the Hibernate directly ?

You can, of course, do that. That's what a lot of Java applications do. Spring ORM provides a good integration for JPA (eg : Spring native Hibernate Integration or Spring JPA)

But the Spring Data team took the extra step to make your job a little bit easier. The additional layer on top of JPA enables them to integrate JPA into the Spring stack easily.
They also provide a lot of functionality that you otherwise would need to implement yourself.

Why Spring Data JPA

1. No-code Repositories
The repository pattern is one of the most popular persistence-related patterns. It hides the DB specific implementation details and enables you to implement your business logic with higher abstraction level.

eg : For Author Entity
How ?
to persist, update and remove one or multiple Author entities,
to find one or more Authors by their primary keys,
to count, get and remove all Authors and
to check if an Author with a given primary key exists.

All you need to do is :

```
public interface AuthorRepository extends JpaRepository<Author, Integer>
{}
```

2. Reduced boilerplate code
To make it even easier, Spring Data JPA provides a default implementation for each method defined by one of its repository interfaces. You don't need to implement these operations.

3. Generated queries
Another cool feature of Spring Data JPA is the generation of database queries based on method names.(finder method pattern)

eg : Write a method that gets a Book entity with a given title.
Internally, Spring generates a JPQL query based on the method name, sets

the provided method parameters as bind parameter values, executes the query and returns the result.

```
//JpaRepository<T,ID>
//T : entity type
//ID : Data type of id property(PK)
public interface BookRepository extends JpaRepository<Book, Integer> {

    Optional<Book> findByTitle(String title123);
}
```

Assumption : title : property of Book POJO

Using Spring Data JPA with Spring Boot

You only need to add the spring-boot-starter-data-jpa and your JDBC driver to your maven build. The Spring Boot Starter includes all required dependencies and activates the default configuration.
Add DB config properties in application.properties file

By default, Spring Boot expects that all repositories are located in a sub-packages of the class annotated with @SpringBootApplication.
If your application doesn't follow this default, you need to configure the packages of your repositories using an @EnableJpaRepositories annotation.

Repositories(API) in Spring Data JPA
package : o.s.data.repository
CrudRepository
PagingAndSortingRepository
JpaRepository
The CrudRepository interface defines a repository that offers standard create, read, update and delete operations.
The PagingAndSortingRepository extends the CrudRepository and adds findAll methods that enable you to sort the result and to retrieve it in a paginated way.
The JpaRepository adds JPA-specific methods, like flush() to trigger a flush on the persistence context or findAll(Example<S> example) to find entities

Defining an entity-specific repository
eg :
Book entity is a normal JPA entity with a generated primary key of type Long, a title and a many-to-many association to the Author entity.

```
@Entity
@Table(name="books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Version
    private int version; //for optimistic locking

    private String title;

    @ManyToMany
    @JoinTable(name = "book_author",
                joinColumns = { @JoinColumn(name = "fk_book") },
```

```
                inverseJoinColumns = { @JoinColumn(name = "fk_author") })
    private Set<Author> authors = new HashSet<>();

    ...
}
```

If you want to define a JPA repository for this entity, you need to extend Spring Data JpaRepository interface and type it to Book and Long.

```
public interface BookRepository extends JpaRepository<Book, Long> {

    Book findByTitle(String title);
}
```

Working with Repositories

After you defined your repository interface, you can use the @Autowired annotation to inject it into your service implementation. Spring Data will then provide you with a proxy implementation of your repository interface. This proxy provides default implementations for all methods defined in the interface.

For More API Details
refer " regarding Spring Data JPA"

Problems observed in the lab
1. No def ctor found , or empty JSON array of users ([{},{}])
Root cause : lombok is not working
If Lombok library is not working , then comment the annotations n use earlier approach
How to test ?
eg : User POJO , if u have added lombok annotations , then try from some other class , if u can call the getter / setter
This will be the confirmation test.

Revise : Full Stack Development
Diagram : Full Stack overview
Readme : RestController vs MVC Controller n Annotations.txt
Readme : REST simplified

Objective : Complete backend for User Management

1. Get All Users

Later Use ResponseEntity , to wrap response body + response headers.
2. Add User Details :
3. Delete User Details
4. Get User details by id
i/p : user id
o/p : exsiting user details
5. Update User details
i/p : user id , updated user details

Test it with postman & then with React front end(in future!)

Project Tips :

1. If you see json response , you will see cart : null , empty hobbies : []
How to exclude null or absent values ?
You can add at the POJO class level ,
@JsonInclude(Include.NON_EMPTY)
Check other options also.

2. How to control the property/field access during ser/de-serial
@JsonProperty(access = Access.WRITE_ONLY) : property/field will be ignored during serialization
@JsonProperty(access = Access.READ_ONLY) : property/field will be ignored during de serialization

3. Add Option for User Registration.
Creating an empty cart n associate it with user.
Add hobbies from the front end.
Observer n conclude.
Problem : In case of bi-dir association , jackson will cause the recursion, during serialization(java--> json)

Solution : How to tell SC using Jackson , to skip the properties
Add field level annotation
@JsonIgnoreProperties(name of the prop as it appears in the opposite side)
More Details :
eg : User 1<----->1 Cart

Problem : Infinite recursion (stack overflow !!!!!!!!!!!!!!)
Cause : Jackson (converter) tries to serialize Cart object , during the ser. of User obj.
AND
Jackson (converter) tries to serialize User object , during the ser. of Cart obj.
Soln :
In User entity, add
@JsonIgnorePropeties(value="cartOwner")
private ShoppingCart cart;

In ShoppingCart entity
@JsonIgnorePropeties(value="cart")
private User cartOwner;

Same problem n solution for any bi-dir relationship.

OR use DTO solution
Project Tip
As a standard design practice , do not expose entities directly to the REST client
(i.e DO not return them from the RestController)
Instead use DTO (data transfer object) , to separate entities from resources to be shared with the clnts.
eg : Send User DTO as resp , instead of User entity.

2. Introduction to ResponseEntity
o.s.http.ResponseEntity<T> : generic class
T : type of the response body
Replace actual Resource(eg : User) by ResponseEntity

Standard design practice for back end : DO NOT directly send response body , instead wrap it in ResponseEntity

o.s.http.ResponseEntity<T>
Ctor : ResponseEntity(T body, HttpStatus status);

OR Methods
ResponseEntity.status(HttpStatus status).body(T body)

Objective : Add ResponseEntity to UserController

5. REST Server side Validations
(refer : templates under ready code & "regarding spring boot exc handling n validations.txt")
5.1 Add Validation rules on Entity / DTO
eg : first name : can't be blank .(min : 4 chars , max =20 chars)
last name : can't be blank
valid email
valid strong password (alpha numeric, special character , min 5 max 20)
eg : ((?=.*\\d)(?=.*[a-z])(?=.*[#@$*]).{5,20})
reg amount in the range : 500 ---5000
reg date must be in future

Test it with postman client
(Same annotations are used in Spring MVC standalone App , in P.L: presentation logic validations)
eg : @NotBlank , @NotNull,@Email, @Pattern,@Future ,@Range....

5.2 Add @Valid , along with @RequestBody , in REST controller methods.

5.3 Additionally , add @Validated on class(RestController) , for validating path var / request param.

6. Any problems observed on the client side ?

YES : Since spring boot supplies a default exception handler , entire stack trace , along with exception details are sent to the front end.

NOTE : Validation failures CAN NOT be caught by controller method level exc handling(try-catch)
B.L failures CAN BE caught by controller method level exc handling(try-catch) --> but resulting into repeatative exc handling

Instead :
How to avoid both of these problems ?
Solution : Add centralized (global) exception handler
Steps
1. Create a separate class : extending from ResponseEntityExcHandler (so that err resp will sent back as a resp entity) n can override base class methods
2. Add cls level anno : @ControllerAdvice
To tell SC , following class is a common advice to : controllers n rest controller --regarding exc handling (cross cutting concern=common task)

3. For validation failures : @Valid
override the method : handleMethodArgNotValidException
Extract map of Field Errors --send it to the caller(front end) by wrapping it in the RespEntity.
Exc class : MethodArgNotValidException
It's super class : BindException has : Method --List<FieldError> getFieldErrors() .
FieldError : getField, getDefaultMessage

4. For ResourceNotFound or similar exceptions :
Instead of sending err mesg as a plain string , wrap it in Error response object n send it to the front end for simpler processing
Enter DTO pattern : Data Transfer object
ApiResponse : message , timestamp

Refer to readme : regarding spring boot exc handling n validations.txt

Project Tip

How to automatically map Entitiy --- DTO
```
<!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->
		<dependency>
			<groupId>org.modelmapper</groupId>
			<artifactId>modelmapper</artifactId>
			<version>3.0.0</version>
```

```
            </dependency>

@Bean
      public ModelMapper mapper() {
            ModelMapper modelMapper = new ModelMapper();

      modelMapper.getConfiguration().setMatchingStrategy(MatchingStrategi
es.STRICT);
            return modelMapper;
      }
```

Enter Spring security

4. Spring Data JPA more practice
Add 3 data members : regDate , regAmount , desc (about user)
DML :
update users_tbl set description='common desc' , reg_amount=500
,reg_date='2020-1-1' where id < 5;
update users_tbl set description='some other desc' , reg_amount=700
,reg_date='2020-11-21' where id >= 5'
(Finder methods , Custom Query Methods)
Solve n Test it with DAO layer Tests (Later test it with service layer tests)
4.1 Find the users by last name (add multiple users with same last name n test)
eg : UserRepository
4.2 Find the user by specific email n password (authentication)
4.3 Find users with reg amounts between start value n end value
4.4 Find users registered after a particular date , sorted as per their reg amount
4.5 Find users whose description contains a keyword.

Custom query
Get first name n last names of all users under a specific role , paying reg amount > specified amount

Spring Security

Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework.
It is supplied as a "ready made aspect" , from spring security framework , that can be easily plugged in spring MVC application.

It is "THE" standard for securing Spring-based applications. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.

Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.

Features

1. Comprehensive and extensible support for both Authentication and Authorization
2. Protection against attacks like session fixation, clickjacking, cross site request forgery, etc..
3.Servlet API integration (Uses Servlet Filter chain)
4.Integration with Spring Web MVC.

Common Security Terms

Credentials : Way of confirming the identity of the user (email /username , password typically)
Principal: Currently logged in user.
Authentication: Confirming truth of credentials(i.e confirming who you are)
Authorisation: Defines access policy for the Principal.(i.e confirming your permissions, i.e what you can do)
GrantedAuthority: Permission granted to the principal.
AuthenticationManager (i/f): Controller in the authentication process. Authenticates user details via authenticate() method.

Steps

1. Create a new spring boot project (RESTful web service) , adding usual dependenices . DO NOT add spring security yet. Copy earlier working application.properties.
NOTE : spring boot version downgrade : 2.6.7 : NO Longer needed !!!!!!!!!!!!!!!
Can continue with Spring boot 2.7 (spring sec 5.7.x)

2. Add a ProductController , with 3 end points

/home : (should be accessible to all)
/purchase : (should be for authenticated User in customer role)
/add_product : (should be for authenticated User in admin role)
/browse_categores : Any Authenticated user(any role) can browse through the categories
/delete_product : Only admin should be able to delete a product

Currently : respond to GET method , with simple string response.

3. Test it .(using browser/postman)
Did it work : YES

4. Add spring security dependency

. Test the end points again.
Are they accessible still ? NO

Observation : Suddenly n automatically all end points are now protected.
So on browser it will prompt you to login form (spring security supplied)
On postman it will give you HTTP 401 (Un authorized error)

We have not yet supplied any credentials .
Def credentials are : user n password(UUID : universally unique ID : 128 bit) from server console.

So w/o configuring anything , the moment spring security JARs are added , all your end points are secured automatically .

Thus Spring Boot(running on the top of the Spring Framework) + Spring security , provides a ready made aspect(solution to cross cutting concern like authetication n authorization) in form of spring security

After supplying correct credentials(i.e after authentication) , spring security will redirect you to the resource :
http://localhost:8080/products/view ,and you will be able to access it.
Supplies you automatically with a logout page (test it on the browser)

Observe on postman(w/o setting authorization header)
Response : (HTTP 401)

From authorization , choose Basic Authentication (referred as Basic Auth) ,
Add user name n password.

It will be encoded using base64 encoding.

Basic authentication, or “basic auth” is formally defined in the HTTP standard. When a client (your browser) connects to a web server, it sends a “WWW-Authenticate: Basic” message in the HTTP header. After that, it sends your login credentials to the server using a mild concealment technique called base64 encoding.

Not desirable , to use such credentials , so continue to next step.

5. Can you configure username n password n roles , in a property file ?
YES .

Add these 2 properties in application.properties file
spring.security.user.name=
spring.security.user.password=
spring.security.user.roles=

So now instead of spring security generated user name n pwd , these will be used for authentication.

6. Ultimate goal is using DB to store the authentication details .
BUT immediate next goal , to understand spring security is : Basic In memory authentication

The credentials will be stored in memory.
Comment earlier properties from app property file.

6.1 Add security config class.
Earlier we used to extend it from
o.s.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
It's a convenient base class , to customize security configuration.
Read above only as Exam objective)

BUT with latest spring boot version 2.7.x onwards (Spring security 5.7.x onwards) , it has become deprecated.
So will not be used as the super class.

Ref : https://docs.spring.io/spring-security/reference/servlet/configuration/java.html

6.2 Class level annotations
1. @EnableWebSecurity : for customizing spring security
2. @Configuration (annotation based approach equivalent to bean config xml file containing <bean id xml..../>) Meaning this class can declare one or more @Bean methods(for providing spring beans) and which will be auto processed by the SC . We can then inject these beans as dependencies whenever required.

6.3 Add a method , to configure a bean to provide in memory authentication , returning InMemoryUserDetailsManager

What is it ?
A class : Non-persistent implementation of UserDetailsManager which is backed by an in-memory map.
It implements UserDetailsService to provide support for username/password based authentication that is stored in memory.

InMemoryUserDetailsManager provides management of UserDetails by implementing the UserDetailsManager interface.

Mainly used for testing purpose , to be replaced by DAO based User details manager soon.

Constructor
InMemoryUserDetailsManager(UserDetails... users)

o.s.s.c.userdetails.UserDetails : i/f
Implementation class : o.s.s.c.userdetails.User
Builder Methods for configuration :
withUsername
password
roles (NOTE : NO ROLE_ prefix required here, it will be added implicitely by spring security framework)
Then build User instance.
Using : build() method

OR
Use the User ctor
User(String username,String password,Collection<? extends GrantedAuthority? authorities)

NOTE : Here in setting the authorities : ROLE_ is required. otherwise Authentication will fail.

6.4 Refer to diag : spring security architecture

Refer to readme : "spring sec auth flow"
Diagram : detailed flow.png

6.5 For supplying authorization details :

Objective :
/products/view : accessible to all
/products/add n delete : only to admin user
/products/purchase : accessible to customer role
/products/categories : accessible to any authenticated user.

Add a method , to configure a bean to provide , authorization rules.
Bean : o.s.s.web.SecurityFilterChain
Defines a filter chain which can be matched against incoming
 HttpServletRequest , in order to decide whether any filter in the chain applies to that request.

Dependency (Method Argument) : HttpSecurity

Steps
1.Authorize all requests (authorizeRequests())
2. Supply ant matcher patterns : for role specific authorization with methods like : hasRole ,hasAnyRole : no ROLE prefix or use the method : permitAll
3. Can supply the rule that remaining end points can be accessed only by authenticated users(irrespective of any role)
and :
4. Enables HTTP Basic and Form based authentication
5. Spring Security will automatically render a login page and logout success pages for you
6. Use buid() on HttpSecurity

A typical example would look like this :

```
http.authorizeRequests().
            antMatchers("/products/view").permitAll()
            .antMatchers("/products/purchase").hasRole("CUSTOMER")
            .antMatchers("/products/add").hasRole("ADMIN").
            anyRequest().authenticated().and().
            httpBasic().and().formLogin();
```

6.6 Run the application n try accessing any of the protected resource

Problem : java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"

Reason -- Prior to Spring Security 5.0 the default PasswordEncoder was NoOpPasswordEncoder which required plain text passwords.
From Spring Security 5, the default is DelegatingPasswordEncoder, which requires Password Storage Format.

Solution : provide Password encoder bean , in the main application class itself.

```
@Bean
      public PasswordEncoder encoder() {
            return new BCryptPasswordEncoder();
```

}

Test the application.

6.7 Add method level finer control over authorization
1. To enable method level authorization supprt , add annotation over sec configuration class
@EnableGlobalMethodSecurity(prePostEnabled = true)
prePostEnabled : to enable pre n post sec annotations (def value is false)

2. Add pre/post annotations over controller methods
eg :
@PreAuthorize("hasRole('ROLE_ADMIN')")
@GetMapping("/delete")
public String deleteProduct() {
 return "Only admin should be able to delete the products";
}
NOTE : Method level authorization rule will override that from sec config class

----------In Memory authentication over----------

7. Replace in memory authentication by DB based authentication.
Using Spring Data JPA.

7.1 Edit application.properties file with DB settings.
Can optionally add these for debugging.
debug=true
logging.level.org.springframework.security=DEBUG

7.2 In Security config class
How to replace in memory authentication , by UserDetailsService based auth mgr builder
1. Simply remove in-memory configuration bean In memory user detail manager

In the absence of any specific options , default authentication provider used is :
DaoAuthenticationProvider based upon UserDetailsService

(Meaning : NO extra configuration required !)

7.3
The org.springframework.security.core.userdetails.UserDetailsService interface is used to retrieve user-related data. It has one method named loadUserByUsername() which can be overridden to customize the process of finding the user.

It is used by the DaoAuthenticationProvider to load details about the user during authentication.
It is used throughout the framework as a user DAO and is the strategy used by the DaoAuthenticationProvider.

class DaoAuthenticationProvider :
Represents an AuthenticationProvider implementation that retrieves user details from a UserDetailsService.

Method

```
UserDetails loadUserByUsername(java.lang.String username)
                        throws UsernameNotFoundException
```

7.4 How to load user by user name ?

1. Create POJOs User with Role

```
UserEntity  extending from BaseEntity
Properties : firstName,lastName ,email,password,Role
Role : enum (ROLE_VISITOR,ROLE_CUSTOMER,ROLE_ADMIN) : adding ROLE_ prefix
is mandatory
```

```
2. DAO layer :
UserRepository :  findByEmail
```

```
3. Create custom implementation of o.s.s.c.userdetails.UserDetailsService
n implement
UserDetails loadUserByUsername(String username)
                        throws UsernameNotFoundException
In case , user entity not found , raise UsernameNotFoundException , with
suitable error message.
```

```
4. In case of success , create custom implementation of ,
org.springframework.security.core.userdetails.UserDetails i/f
, by passing to it's constructor , User entity details , lifted from DB

o.s.s.c.userdetails.UserDetails : represents core user information. It
stores
        user information which is later encapsulated into Authentication
object. This
        allows non-security related additional user information (eg :
email
        acct expiry, user enabled ... ) in addition to user name n
password to be stored in a convenient location.
```

```
One important method in above i/f to implement is
      public Collection<? extends GrantedAuthority> getAuthorities() ,
which should return , granted authorities (role based) for the loaded
user.
eg : user => UserEntity

List.of(new SimpleGrantedAuthority(user.getRole().name());
```

```
5. Implement all other methods , suitably .

6. Create UserService for user registration
UserRegDto registerUser(UserRegDto request);
```

```
How to run ?
1. Run service layer test cases : to add 3 roles : ROLE_ADMIN ,
ROLE_VISITOR , ROLE_CUSTOMER

2. Confirm from DB

3. Test it from postman
```

For hashing the password , you can refer :
https://bcrypt-generator.com/

For more details : https://dzone.com/articles/hashing-passwords-in-java-with-
bcrypt#:~:text=One%20way%20hashing%20%2D%20BCrypt%20is,hashes%20across%20each%20user's%20password.

Project Tip :
Later to test it with React/Angular front end :
use below for authorization.

```
      http.csrf().disable().
            cors().and().
            authorizeRequests().

            antMatchers(HttpMethod.OPTIONS, "/**").permitAll().
            antMatchers("/", "/home", "/api/signup").permitAll().
            antMatchers("/admin").hasRole("ADMIN").
            antMatchers("/user").hasAnyRole("USER", "ADMIN").
            and().httpBasic();
```

BEST n ULTIMATE resource for spring security
https://docs.spring.io/spring-security/reference/servlet/architecture.html

In that mainly
https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/basic.html
https://docs.spring.io/spring-security/reference/servlet/authentication/passwords/dao-authentication-provider.html

https://springhow.com/in-memory-userdetailsservice-in-spring-security/

How does spring security works internally?

Spring security is enabled automatically , by just adding the spring security starter jar. But, what happens internally and how does it make our application secure?

Common Terms

Principal: Currently logged in user.
Authentication: Confirming truth of credentials.
Authorisation: Defines access policy for the Principal.
GrantedAuthority: Permission granted to the principal.
AuthenticationManager (i/f): Controller in the authentication process. Authenticates user details via authenticate() method.

AuthenticationManager i/f implemented by : ProviderManager class .
Diagram : detailed flow .png
It Iterates an Authentication request through a list of AuthenticationProviders.
AuthenticationProviders are usually tried in order until one provides a non-null response. A non-null response indicates the provider had authority to decide on the authentication request and no further providers are tried.

AuthenticationProvider: Interface that maps to a data store that stores your data.
Authentication Object: Object that is created upon authentication. It holds the login credentials. It is an internal spring security interface.
UserDetails: Data object that contains the user credentials but also the role of that user.
UserDetailsService: Collects the user credentials, authorities (roles) and build an UserDetails object.

The Spring Security Architecture

When we add the spring security starter jar, it internally adds Filter to the application. A Filter is an object that is invoked at pre-processing and post-processing of a request. It can manipulate a request or even can stop it from reaching a servlet. There are multiple filters in spring security out of which one is the Authentication Filter, which initiates the process of authentication.

Once the request passes through the authentication filter, the credentials of the user are stored in the Authentication object. Now, what actually is responsible for authentication is AuthenticationProvider (Interface that has method authenticate()). A spring app can have multiple authentication providers, one may be using Dao based , JWT based , OAuth, LDAP ... To manage all of them, there is an AuthenticationManager.

The authentication manager finds the appropriate authentication provider by calling the supports() method of each authentication provider. The supports() method returns a boolean value. If true is returned, then the authentication manager calls its authenticate() method.

After the credentials are passed to the authentication provider, it looks for the existing user in the system by UserDetailsService. It returns a UserDetails instance which the authentication provider verifies and

authenticates. If success, the Authentication object is returned with the Principal and Authorities otherwise AuthenticationException is thrown.

Problems observed in the lab
1. No def ctor found , or empty JSON array of users ([{},{}])
Root cause : lombok is not working
If Lombok library is not working , then comment the annotations n use earlier approach
How to test ?
eg : User POJO , if u have added lombok annotations , then try from some other class , if u can call the getter / setter
This will be the confirmation test.

Revise : Full Stack Development
Diagram : Full Stack overview
Readme : RestController vs MVC Controller n Annotations.txt
Readme : REST simplified

Objective : Complete backend for User Management

1. Get All Users

Later Use ResponseEntity , to wrap response body + response headers.
2. Add User Details :
3. Delete User Details
4. Get User details by id
i/p : user id
o/p : exsiting user details
5. Update User details
i/p : user id , updated user details

Test it with postman & then with React front end(in future!)

Project Tips :

1. If you see json response , you will see cart : null , empty hobbies : []
How to exclude null or absent values ?
You can add at the POJO class level ,
@JsonInclude(Include.NON_EMPTY)
Check other options also.

2. How to control the property/field access during ser/de-serial
@JsonProperty(access = Access.WRITE_ONLY) : property/field will be ignored during serialization
@JsonProperty(access = Access.READ_ONLY) : property/field will be ignored during de serialization

3. Add Option for User Registration.
Creating an empty cart n associate it with user.
Add hobbies from the front end.
Observer n conclude.
Problem : In case of bi-dir association , jackson will cause the recursion, during serialization(java--> json)

Solution : How to tell SC using Jackson , to skip the properties
Add field level annotation
@JsonIgnoreProperties(name of the prop as it appears in the opposite side)
More Details :
eg : User 1<----->1 Cart

Problem : Infinite recursion (stack overflow !!!!!!!!!!!!!!)
Cause : Jackson (converter) tries to serialize Cart object , during the ser. of User obj.
AND
Jackson (converter) tries to serialize User object , during the ser. of Cart obj.
Soln :
In User entity, add
@JsonIgnorePropeties(value="cartOwner")
private ShoppingCart cart;

In ShoppingCart entity
@JsonIgnorePropeties(value="cart")
private User cartOwner;

Same problem n solution for any bi-dir relationship.

OR use DTO solution
Project Tip
As a standard design practice , do not expose entities directly to the REST client
(i.e DO not return them from the RestController)
Instead use DTO (data transfer object) , to separate entities from resources to be shared with the clnts.
eg : Send User DTO as resp , instead of User entity.

2. Introduction to ResponseEntity
o.s.http.ResponseEntity<T> : generic class
T : type of the response body
Replace actual Resource(eg : User) by ResponseEntity

Standard design practice for back end : DO NOT directly send response body , instead wrap it in ResponseEntity

o.s.http.ResponseEntity<T>
Ctor : ResponseEntity(T body, HttpStatus status);

OR Methods
ResponseEntity.status(HttpStatus status).body(T body)

Objective : Add ResponseEntity to UserController

5. REST Server side Validations
(refer : templates under ready code & "regarding spring boot exc handling n validations.txt")
5.1 Add Validation rules on Entity / DTO
eg : first name : can't be blank .(min : 4 chars , max =20 chars)
last name : can't be blank
valid email
valid strong password (alpha numeric, special character , min 5 max 20)
eg : ((?=.*\\d)(?=.*[a-z])(?=.*[#@$*]).{5,20})
reg amount in the range : 500 ---5000
reg date must be in future

Test it with postman client
(Same annotations are used in Spring MVC standalone App , in P.L: presentation logic validations)
eg : @NotBlank , @NotNull,@Email, @Pattern,@Future ,@Range....

5.2 Add @Valid , along with @RequestBody , in REST controller methods.

5.3 Additionally , add @Validated on class(RestController) , for validating path var / request param.

6. Any problems observed on the client side ?

YES : Since spring boot supplies a default exception handler , entire stack trace , along with exception details are sent to the front end.

NOTE : Validation failures CAN NOT be caught by controller method level exc handling(try-catch)
B.L failures CAN BE caught by controller method level exc handling(try-catch) --> but resulting into repeatative exc handling

Instead :
How to avoid both of these problems ?
Solution : Add centralized (global) exception handler
Steps
1. Create a separate class : extending from ResponseEntityExcHandler (so that err resp will sent back as a resp entity) n can override base class methods
2. Add cls level anno : @ControllerAdvice
To tell SC , following class is a common advice to : controllers n rest controller --regarding exc handling (cross cutting concern=common task)

3. For validation failures : @Valid
override the method : handleMethodArgNotValidException
Extract map of Field Errors --send it to the caller(front end) by wrapping it in the RespEntity.
Exc class : MethodArgNotValidException
It's super class : BindException has : Method --List<FieldError> getFieldErrors() .
FieldError : getField, getDefaultMessage

4. For ResourceNotFound or similar exceptions :
Instead of sending err mesg as a plain string , wrap it in Error response object n send it to the front end for simpler processing
Enter DTO pattern : Data Transfer object
ApiResponse : message , timestamp

Refer to readme : regarding spring boot exc handling n validations.txt

Project Tip

How to automatically map Entitiy --- DTO
```
<!-- https://mvnrepository.com/artifact/org.modelmapper/modelmapper -->
		<dependency>
			<groupId>org.modelmapper</groupId>
			<artifactId>modelmapper</artifactId>
			<version>3.0.0</version>
```

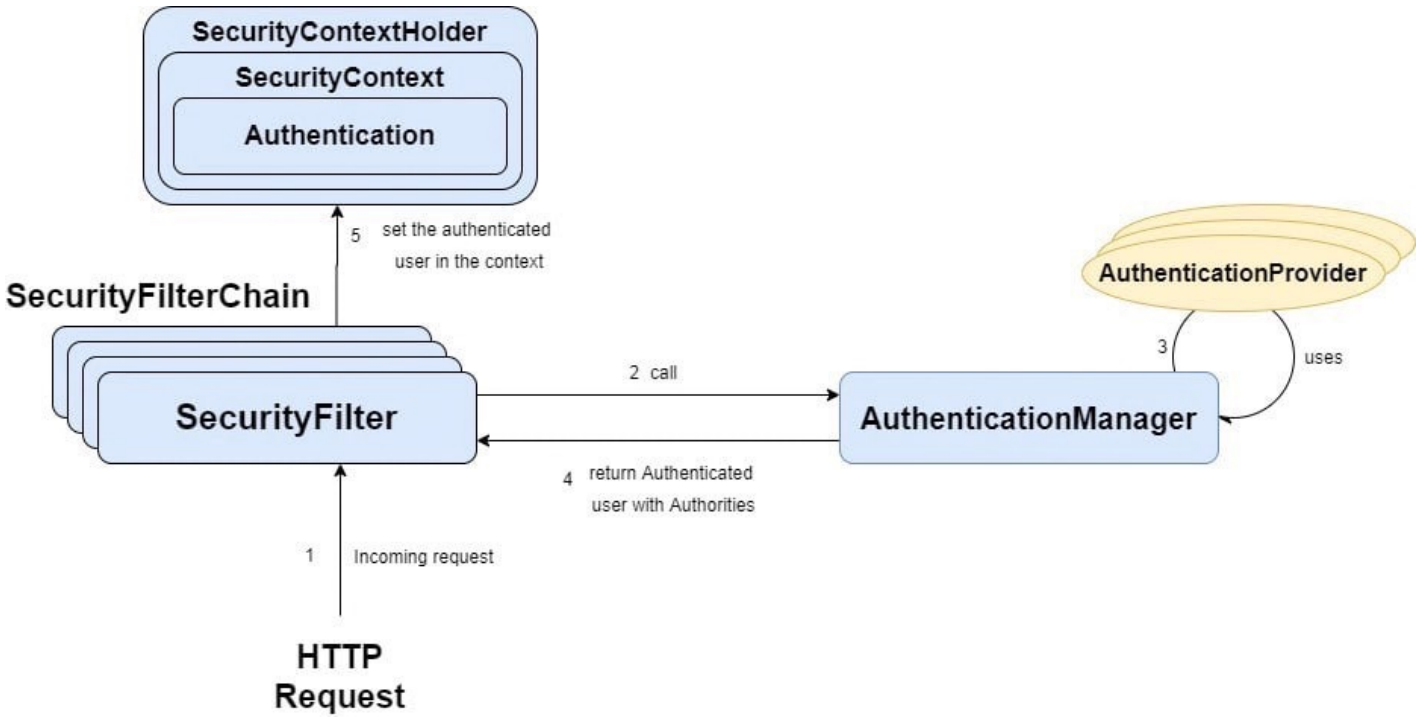
```
            </dependency>

@Bean
      public ModelMapper mapper() {
            ModelMapper modelMapper = new ModelMapper();

      modelMapper.getConfiguration().setMatchingStrategy(MatchingStrategi
es.STRICT);
            return modelMapper;
      }
```

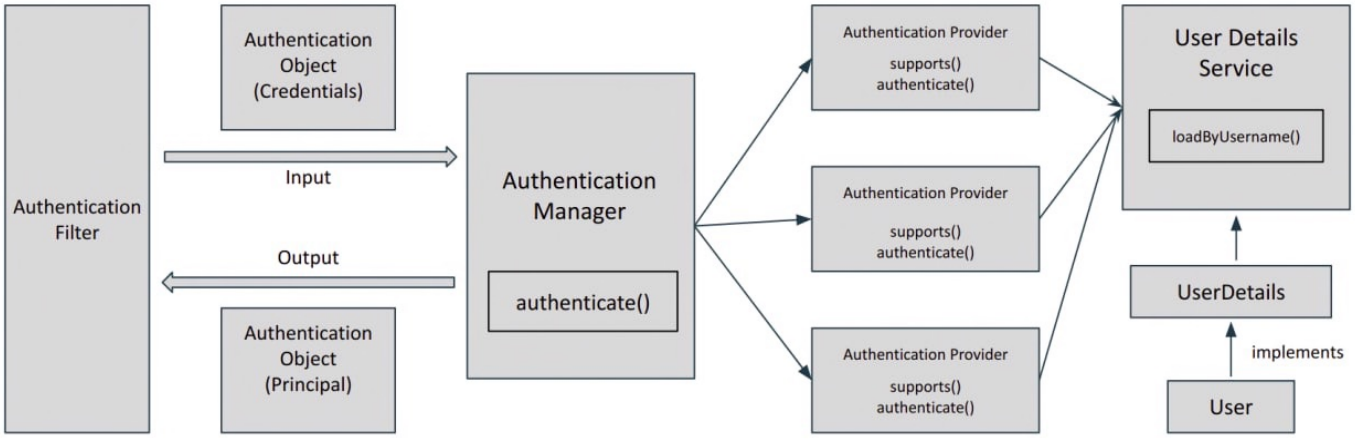
Enter Spring security

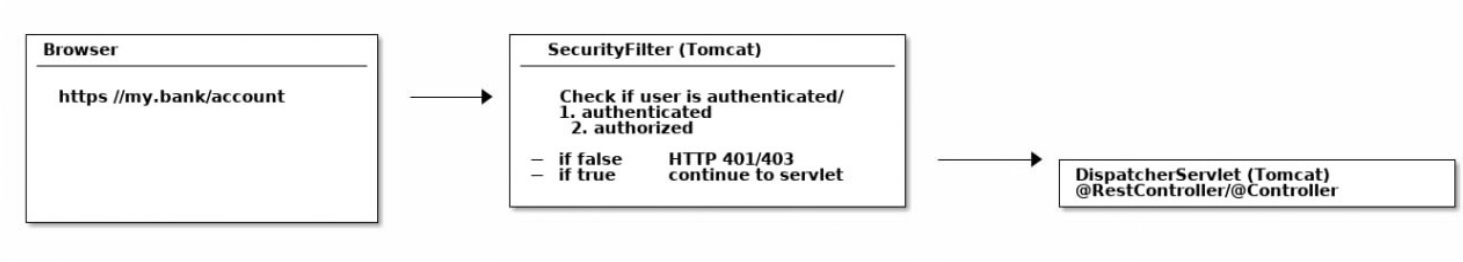
4. Spring Data JPA more practice
Add 3 data members : regDate , regAmount , desc (about user)
DML :
update users_tbl set description='common desc' , reg_amount=500
,reg_date='2020-1-1' where id < 5;
update users_tbl set description='some other desc' , reg_amount=700
,reg_date='2020-11-21' where id >= 5'
(Finder methods , Custom Query Methods)
Solve n Test it with DAO layer Tests (Later test it with service layer tests)
4.1 Find the users by last name (add multiple users with same last name n test)
eg : UserRepository
4.2 Find the user by specific email n password (authentication)
4.3 Find users with reg amounts between start value n end value
4.4 Find users registered after a particular date , sorted as per their reg amount
4.5 Find users whose description contains a keyword.

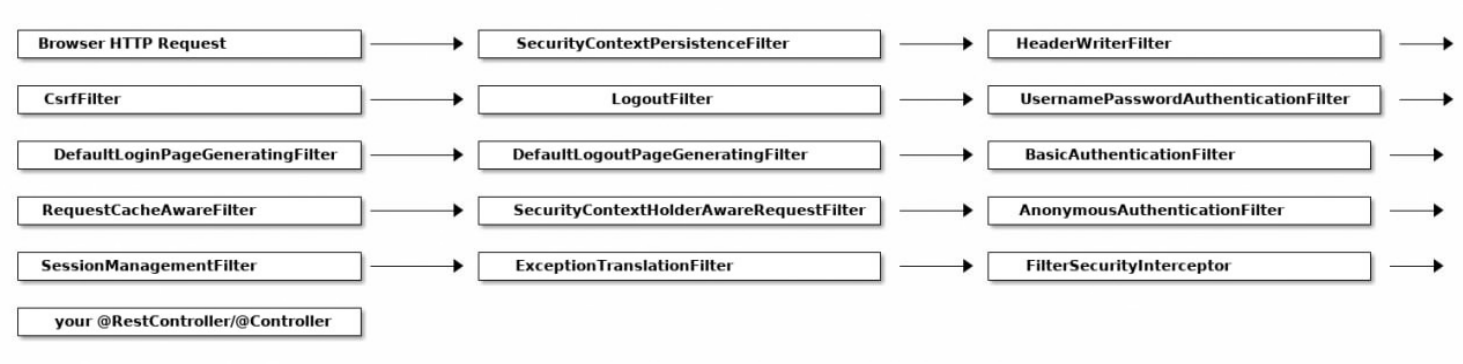
Custom query
Get first name n last names of all users under a specific role , paying reg amount > specified amount

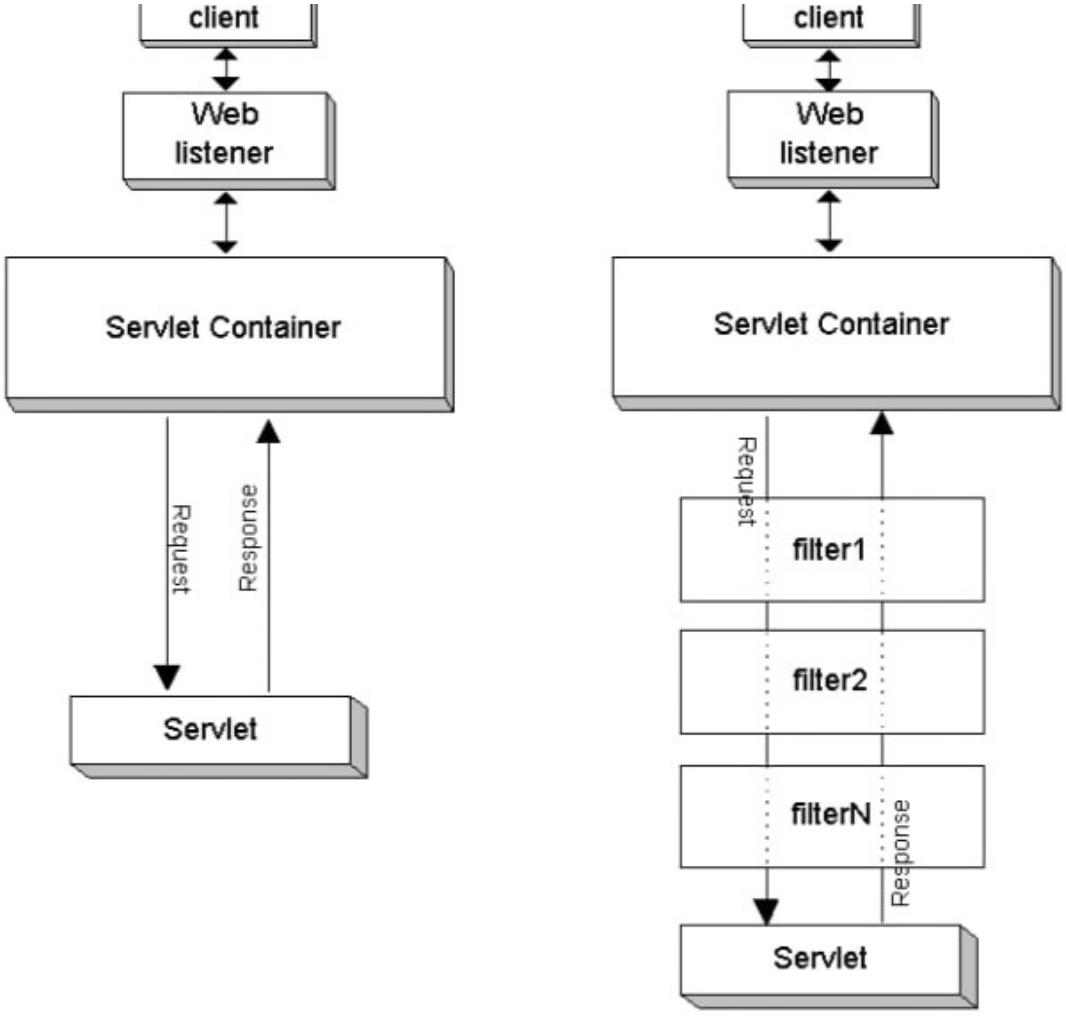
```
@NotEmpty
@Length(min=5,max=10)
@Email
private String email;
@NotBlank
@Pattern(regexp="((?=.*\\d)(?=.*[a-z])(?=.*[#@$*]).{5,20})")
private String password;
@NotNull
@Range(min=200,max=2000)
private double regAmt;
@NotNull
@DateTimeFormat(pattern="dd-MMM-yyyy")
private Date regDate;
```






Browser
https //my.bank/account
SecurityFilter (Tomcat)
Check if user is authenticated/
1. authenticated
2. authorized
– if false HTTP 401/403
– if true continue to servlet
DispatcherServlet (Tomcat)
@RestController/@Controller


Browser HTTP Request
SecurityContextPersistenceFilter
HeaderWriterFilter
CsrfFilter
LogoutFilter
UsernamePasswordAuthenticationFilter
DefaultLoginPageGeneratingFilter
DefaultLogoutPageGeneratingFilter
BasicAuthenticationFilter
RequestCacheAwareFilter
SecurityContextHolderAwareRequestFilter
AnonymousAuthenticationFilter
SessionManagementFilter
ExceptionTranslationFilter
FilterSecurityInterceptor
your @RestController/@Controller


client
Web listener
Servlet Container
Request
Response
Servlet
client
Web listener
Servlet Container
Request
filter1
filter2
filterN
Response
Servlet

Why Filters ?

1. Provides re-usability.
Meaning --- They provide the ability to encapsulate recurring tasks(=cross cutting concerns) in reusable units.
They provide clear cut separation between B.L & cross cutting concerns.

2. Can dynamically intercept req/resp to dyn or static content

What is Filter?

Dynamic web component just like servlet or JSP. Resides within web-appln.(WC)
Filter life-cycle managed by WC

It performs filtering tasks on either the request to a resource (a servlet,JSP or static content), or on the response from a resource, or both.

It can dynamically intercepts requests and responses .

Usage of Filters
1. Authentication Filters
2. Logging Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters
6. Session Check filter

How to create Filter Component?
1. Create Java class imple. javax.servlet.Filter i/f
2. Implements 3 life-cycle methods
2.1 public void init(FilterConfig filterConfig)
throws ServletException

Above called by WC --- only once during filter creation & initialization.(@appln start up time)
2.2
void doFilter(ServletRequest request,ServletResponse response,FilterChain chain) throws IOException, ServletException

Invoked by WC -- per every rq & resp processing time.

Here u can do pre-processing of req, then invoke chain.doFilter -- to invoke next component of filter chain --- finally it invokes service method of JSP/Servlet --- on its ret path visits filter chain in opposite manner & finally renders response to clnt browser.

2.3
public void destroy() ---invoked by WC at the end of filter life cycle.
Triggers --- server shut down/re-deploy/un deploy

How to deploy a Filter component ?
1. Annotation -- @WebFilter (class level annotation)
OR
2. XML tags (in web.xml)
<filter>
<filter-name>abc</..>
<filter-class>filters.AuthenticatioFilter</...>

```
 <init-param>
  <param-name>nm1</..>
  <param-value>val1</..>
 </...>
</filter>
<filter-mapping>
 <filter-name>abc</..>
 <url-pattern>/*</...>
</filter-mapping>
```

Detailed Description ---
Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.

Filters are important for a number of reasons. First, they provide the ability to encapsulate recurring tasks in reusable units. Organized developers are constantly on the lookout for ways to modularize their code. Modular code is more manageable and documentable, is easier to debug, and if done well, can be reused in another setting.

Second, filters can be used to transform the response from a servlet or a JSP page. A common task for the web application is to format data sent back to the client. Increasingly the clients require formats (for example, WML) other than just HTML. To accommodate these clients, there is usually a strong component of transformation or filtering in a fully featured web application. Many servlet and JSP containers have introduced proprietary filter mechanisms, resulting in a gain for the developer that deploys on that container, but reducing the reusability of such code. With the introduction of filters as part of the Java Servlet specification, developers now have the opportunity to write reusable transformation components that are portable across containers.

Spring boot internals explained :

Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

	public static void main(String[] args) {
		SpringApplication.run(HelloSpringBootApplication.class, args);
	}

}
```

About : org.springframework.boot.SpringApplication
It's Class used to bootstrap and launch a Spring application from a Java main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC) , based upon default configuration.
2. Manages life cycle of spring beans
3. Launches embedded Tomcat container

@SpringBootApplication - This is where all the spring boot magic happens. It consists of following 3 annotations.

1. @SpringBootConfiguration
It tells spring boot that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time .
eg : ModelMapper (used for mapping Entity <----> DTO)) , PasswordEncoder

2. @EnableAutoConfiguration
It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source(Connection Pool) ,using the properties in application.properties file.

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

3. @ComponentScan (equivalent to xml tag : context:component-scan)
Default base-pkg is the pkg in which main class is defined.

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans .
Can be overridden by
eg :
@ComponentScan(basePackages = "com")

```
For scanning entities :  (equivalent to packagesToScan)
@EntityScan(basePackages = "com.app.pojos")
```

Still Lombok issues ?
Confirm Eclipse STS location

0. For actual full stack app (for connecting it with any JS front end app eg : React / Angular)
Additional annotation : @CrossOrigin

1.Spring boot internals
(refer to readme)

2. Understand spring security architecture
(refer to spring security help)
2.1 Servlet Filters overview (filter help)
2.2 servletfilter-1 , servletfilter-2
2.3 spring security architecture diagram
2.4 detailed-flow diagram
(readme : Spring security flow)

Draw a detailed diagram

2.4 Continue to in mem based authentication
2.5 Replace it by DB based authentication

Copy User entity from ready code

(refer : Regarding Spring Security)
2.6 Write Service Layer tests for adding user records
@SpringBootTest

DAO layer tests
Class level annotations over Test class
@DataJpaTest
@AutoConfigureTestDatabase(replace = Replace.NONE)
=> DO NOT replace the actual DB by any test db.
If you need to disable the rollback @ end of the test case , add
@RollBack(false) : over the test method

eg : find out users by specific role
find out users by reg date between start date n end date

3.
Spring Boot AOP (mainly exam objective)

Before AOP :

Regarding logging framework in Java

Spring Boot being extremely helpful framework , it allows us to forget about the majority of the configuration settings, many of which it opinionatedly auto-tunes.

In the case of logging, we don't have to explicitly import it's starter , since a starter, like our spring-boot-starter-web, depends on spring-boot-starter-logging, which already pulls in spring-jcl for us.

(Jakarta Commons Logging API (JCL) is the only mandatory external dependency for Spring w/o boot)

When using starters, Logback is used for logging by default.

eg : different logging levels
Add one field in the Controller class :
Logger logger = LoggerFactory.getLogger(LoggingController.class);
OR use Lombok annotation
@SLF4J : at the class level -- It will auto inject a Logger in the field : log

eg :

```
@RestController
public class LoggingController {

    Logger logger = LoggerFactory.getLogger(LoggingController.class);

    @RequestMapping("/")
    public String index() {
//it's in asc manner : logging levels
        logger.trace("A TRACE Message");
        logger.debug("A DEBUG Message");
        logger.info("An INFO Message");
        logger.warn("A WARN Message");
        logger.error("An ERROR Message");

        return "Testing logging here....";
    }
}
```

Default setting in application.properties file :
logging.level.root=INFO
eg : logging.level.org.springframework.orm.hibernate5=DEBUG
logging.level.com.app.service=DEBUG

Enter AOP
NOTE : for adding AOP : spring boot aop starter is not required.
AOP Basics
Before n After demos
@Around advice
Use @Around from AOP , to measure the time taken for servicing of client request.
ref project : spring-boot-aop

4. Spring boot microservices demo.
Invoking a REST API from another REST API

5. Additional demos for
5.1 Image handling
NetbankingRestClient

5.2
Spring Boot MVC based monolithic web app
2 way data binding in User Registration.
spring_mvc_ecommerce
Will be adding soon :
Additional shopping cart logic , order placement n Payment gateway integration

5.3 get vs load

5.4 JPA inheritance

5.5 JWT

Spring's method-level dependency injection support, via the @Lookup annotation.

2. Why @Lookup?

A method annotated with @Lookup tells Spring to return an instance of the method's return type when we invoke it.

Essentially, Spring will override our annotated method and use our method's return type and parameters as arguments to BeanFactory#getBean.

@Lookup is useful for:

Injecting a prototype-scoped bean into a singleton bean (similar to Provider)
Injecting dependencies procedurally
Note also that @Lookup is the Java equivalent of the XML element lookup-method.

3. Using @Lookup

3.1. Injecting prototype-scoped Bean Into a Singleton Bean

If we happen to decide to have a prototype Spring bean, then we are almost immediately faced with the problem of how will our singleton Spring beans access these prototype Spring beans?

Now, Provider is certainly one way, though @Lookup is more versatile in some respects.

First, let's create a prototype bean that we will later inject into a singleton bean:

```
@Component
@Scope("prototype")
public class SchoolNotification {
    // ... prototype-scoped state
}
```

And if we create a singleton bean that uses @Lookup:

```
@Component
public class StudentServices {

    // ... member variables, etc.

    @Lookup
    public SchoolNotification getNotification() {
        return null;
    }

    // ... getters and setters
}
```

Using @Lookup, we can get an instance of SchoolNotification through our singleton bean:

```
@Test
public void whenLookupMethodCalled_thenNewInstanceReturned() {
    // ... initialize context
    StudentServices first = this.context.getBean(StudentServices.class);
    StudentServices second = this.context.getBean(StudentServices.class);
```

```
    assertEquals(first, second);
    assertNotEquals(first.getNotification(), second.getNotification());
}
```
Note that in StudentServices, we left the getNotification method as a stub.

This is because Spring overrides the method with a call to beanFactory.getBean(StudentNotification.class), so we can leave it empty.

3.2. Injecting Dependencies Procedurally

Still more powerful, though, is that @Lookup allows us to inject a dependency procedurally, something that we cannot do with Provider.

Let's enhance StudentNotification with some state:

```
@Component
@Scope("prototype")
public class SchoolNotification {
    @Autowired Grader grader;

    private String name;
    private Collection<Integer> marks;

    public SchoolNotification(String name) {
        // ... set fields
    }

    // ... getters and setters

    public String addMark(Integer mark) {
        this.marks.add(mark);
        return this.grader.grade(this.marks);
    }
}
```
Now, it is dependent on some Spring context and also additional context that we will provide procedurally.

AD

We can then add a method to StudentServices that takes student data and persists it:

```
public abstract class StudentServices {

    private Map<String, SchoolNotification> notes = new HashMap<>();

    @Lookup
    protected abstract SchoolNotification getNotification(String name);

    public String appendMark(String name, Integer mark) {
        SchoolNotification notification
          = notes.computeIfAbsent(name, exists ->
getNotification(name)));
        return notification.addMark(mark);
    }
}
```
At runtime, Spring will implement the method in the same way, with a couple of additional tricks.

First, note that it can call a complex constructor as well as inject other Spring beans, allowing us to treat SchoolNotification a bit more like a Spring-aware method.

It does this by implementing getSchoolNotification with a call to beanFactory.getBean(SchoolNotification.class, name).

Second, we can sometimes make the @Lookup-annotated method abstract, like the above example.

Using abstract is a bit nicer-looking than a stub, but we can only use it when we don't component-scan or @Bean-manage the surrounding bean:

```
@Test
public void whenAbstractGetterMethodInjects_thenNewInstanceReturned() {
    // ... initialize context

    StudentServices services = context.getBean(StudentServices.class);
    assertEquals("PASS", services.appendMark("Alex", 89));
    assertEquals("FAIL", services.appendMark("Bethany", 78));
    assertEquals("PASS", services.appendMark("Claire", 96));
}
```

With this setup, we can add Spring dependencies as well as method dependencies to SchoolNotification.

4. Limitations

Despite @Lookup's versatility, there are a few notable limitations:

@Lookup-annotated methods, like getNotification, must be concrete when the surrounding class, like Student, is component-scanned. This is because component scanning skips abstract beans.
@Lookup-annotated methods won't work at all when the surrounding class is @Bean-managed.
In those circumstances, if we need to inject a prototype bean into a singleton, we can look to Provider as an alternative.

5. Conclusion

In this quick article, we learned how and when to use Spring's @Lookup annotation, including how to use it to inject prototype-scoped beans into singleton beans and how to use it to inject dependencies procedurally.

SOAP vs REST web services

SOAP stands for simple object access protocol
REST stands for REpresentational State Transfer

Protocol vs Architectural style
SOAP is a standard protocol to create web services
Rest is architectural style to create web services.

Contract
Client and Server are bound with WSDL contract in SOAP
There is no contract between client and Server in REST

Format Support
SOAP supports only XML format
REST web services supports XML, json and plain text etc.

Maintainability
SOAP web services are hard to maintain as if we do any changes in WSDL , we need to create client stub again
REST web services are generally easy to maintain.

Service interfaces vs URI
SOAP uses Service interfaces to expose business logic
Rest uses URI to expose business logic

Security
SOAP has its own security : WS-security
Rest inherits its security from underlying transport layer.

Bandwidth
SOAP requires more bandwidth and resources as it uses XML messages to exchange information
REST requires less bandwith and resources. It can use JSON also.

Learning curve
SOAP web services are hard to learn as you need to understand WSDL , client stub.
REST web services are easy to understand as you need to annotate plain java class with JAX-RS annotations to use various HTTP methods .

1. If there multiple request params(use case -- register/update) --- bind POJO directly to a form.(2 way form binding technique)
How ?
1.1 For loading the form (in showForm method of the controller) , bind empty POJO (using def constr) in model map
How ?
Explicit --add Model as dependency & u add
map.addAttribute(nm,val)
OR better way
implicit -- add POJO as a dependency
eg : User registration
@GetMapping("/reg")
public String showForm(User u) {...}

What will SC do ?
SC --- User u=new User();
chks --- Are there any req params coming from client ? --- typically --no --- only getters will be called --
adds pojo as model attr (in Model map)
map.addAttribute("user",new User());

1.2 In form (view ---jsp) -use spring form tags along with modelAttribute
Steps
1. import spring supplied form tag lib
2. Specify the name of modelAttribute under which form data will be exposed.(name of model attr mentioned in the controller)
<s:form method="post" modelAttribute="user">
  <s:input path="email"/>.....
</s:form>

1.3 Upon form submission (clnt pull I)
clnt sends a new req --- containing req params
@PostMapping("/reg")
public String processForm(User u,RedirectAttributes flashMap,HttpSession hs) {
//SC calls
User u=new User();
SC invokes MATCHING (req param names --POJO prop setters)
setters. -- conversational state is transferred to Controller.
adds pojo as model attr (in Model map)
map.addAttribute("user",u)
Thus you get a populated POJO directly in controller w/o calling <jsp:setProperty> & w/o using any java bean.

Spring form tags
1. <form:input path="name" />

2. <form:input type="email" path="email" />

3. <form:password path="password" />

4. <form:textarea path="notes" rows="3" cols="20"/>

5. <form:checkboxes items="${languages}" path="favouriteLanguage" />

6. Male: <form:radiobutton path="sex" value="M"/>
Female: <form:radiobutton path="sex" value="F"/>

7. `<form:radiobuttons items="${jobItem}" path="job" />`

8.
```
<form:select path="book">
    <form:option value="-" label="--Please Select--"/>
    <form:options items="${books}" />
</form:select>
```

9. `<form:hidden path="id" value="12345" />`

10. `<form:errors path="name" cssClass="error" />`

1.Each incoming request passes through security filter chain for authentication and authorization process.
eg : UsernamePasswordAuthenticationFilter.
OR If the incoming request contains the authorization header "Basic base-64 encoded credentials" , this request goes through the chains of filters until it reaches BasicAuthenticationFilter.

What is the overall job of this filter?

Processes a HTTP request's Basic authorization header, putting the result into the SecurityContextHolder

2. AuthenticationToken is created based on User Credentials
For the user login, once the authentication request reaches the authentication filter, it will extract the username and password from the request payload. Spring security will create an Authentication object based on the username and password.

```
UsernamePasswordAuthenticationToken authentication
 = new UsernamePasswordAuthenticationToken(username, password);
```

3. Authentication filter delegates the request to Authentication Manager.

Authentication Manager is the core for the Spring security authentication process.

```
public interface AuthenticationManager {
    Authentication authenticate(Authentication authentication) throws
AuthenticationException;
}
```

4. This interface is implemented by ProviderManager class.

ProviderManager has no idea about which auth provider will support the current authentication.
So it iterates through the list of AuthenticationProviders n calls supports() method

5. AuthenticationProviders
The AuthenticationProvider are responsible to process the request and perform a specific authentication. It provides a mechanism for getting the user details with which we can perform authentication.

```
public interface AuthenticationProvider {

    Authentication authenticate(Authentication authentication) throws
AuthenticationException;
    boolean supports(Class<?> authentication);
}
```

eg of implementation classes :
DaoAuthenticationProvider.
RememberMeAuthenticationProvider
LdapAuthenticationProvider
...

You can also supply the custom authentication provider.

6. Typically used AuthenticationProvider is : DaoAuthenticationProvider
It depends upon
1. UserDetailsService
2. PasswordEncoder

6.1 Spring Security UserDetailsService

DaoAuthenticationProvider needs UserDetailsService to get the user details stored in the database by username

```
package org.springframework.security.core.userdetails;

public interface UserDetailsService {
    UserDetails loadUserByUsername(String userName/email) throws
UsernameNotFoundException;
}
```

6.2 In case of in memory based user details , UserDetailsService is implemented by InMemoryUserDetailsManager
So you can configure this as a spring bean (@Bean) in your spring sec configuration file

BUT in practical scenario , we use DB based credentials.

In that case , we will implement UserDetailsService to provide User Details

7. Authentication and Authentication Exception
During the authentication process, if the user authentication is successful, AuthenticationProvider will send a fully initialized Authentication object back.

For failed authentication, AuthenticationException will be thrown, filter will send the response to the client HTTP 401
For failed authorization , filter will send the response to the client HTTP 403 (forbidden)

Oherwise a fully populated authentication object carries the following details:

User credentials.
List of granted authorities (for authorization).
Authentication flag : set to true

8. Setting Authentication SecurityContext
The last step in the successful authentication is setting up the authentication object in the SecurityContext. It wraps the SecurityContext in the SecurityContextHolder.

9. The request is then delegated further to DispatcherServlet n continues with the usual flow.

JWT Details

Session-based Authentication & Token-based Authentication

For using any website, mobile app or desktop app , you need to create an account, then use it to login for accessing features of the app : Authentication.

So, how to authenticate a user?
Simple method used : Session-based Authentication.
(refer : session-based-authentication.png)

As per the diag , when a user logs into a website, the Server will create a new Session for that user and store it (in Memory or Database). Server also returns a SessionId for the Client to save it in Browser Cookie.

The Session on Server has an expiration time. After that time, this Session has expired and the user must re-login to create another Session.

If the user has logged in and the Session has not expired yet, the Cookie (including SessionId) always goes with all HTTP Request to Server. Server will compare this SessionId with stored Session to authenticate and return corresponding Response.

If it's been working fine , then why do we need Token-based Authentication?
The answer is we don't have the only consumer as a browser (end user : client a person), we may have different consumers of our services here.

So if you have a website which works well with thin client(browser client) & want to implement system for Mobile (Native Apps) and use the same Web app. You can't authenticate users who use Native App using Session-based Authentication because these native apps don't support cookies. Or if you are implementing a REST server , every REST request HAS TO be stateless , meaning you can't think of maintaining a Http Session in the back end. On the other hand , you can't expect your front end app , to send you the credentials (username / password) along with every request.

That's why Token-based Authentication was born.

With this method, the user login state is encoded into a JSON Web Token (JWT) by the Server and sent to the Client after successful authentication.

Instead of creating a Session, the Server generates a JWT from user login data and sends it to the Client. The Client saves the JWT and then onwards sends this JWT along with every request , typically in a header to the server. The Server will validate the JWT and return the Response. (Giving access to the secured resources)

For storing JWT on Client side, it depends on the platform you use:

Browser: Local Storage
IOS: Keychain
Android: SharedPreferences

3 important parts of a JWT:

Header
Payload

Signature

1. Header
The Header answers the question: How will we calculate JWT?
It’s a JSON object
eg :

```
{
  "typ": "JWT",
  "alg": "HS512"
}
```

- typ is ‘type’, indicates that Token type here is JWT.
- alg stands for ‘algorithm’ which is a hash algorithm for generating Token signature. Here HS256 is HMAC-SHA256/512 – the algorithm which uses Secret Key.

2. Payload
The Payload helps us to answer: What do we want to store in JWT?
This is a payload sample:

```
{
  "userId": "abcd123456",
  "username": "rama",
  "email": "rama@gmail.com",
  // standard fields
  "iss": "Issuer at developers.com",
  "iat": 1570238918,
  "exp": 1570238992
}
```

In the JSON object above, we store 3 user fields: userId, username, email. You can save any field you want.

We also have some Standart Fields. They are optional.

iss (Issuer): who issues the JWT
iat (Issued at): time the JWT was issued at
exp (Expiration Time): JWT expiration time

3. Signature
This part is where we use the Hash Algorithm : HMAC-SHA256
Look at the code for getting the Signature below:
In Java code :
java.util.Base64 class offers encoding n decoding.
Base64 is a binary-to-text encoding scheme. It represents binary data in a printable ASCII string format by translating it into a radix-64 representation.

The basic encoder keeps things simple and encodes the input as is, without any line separation.
The output is mapped to a set of characters in A-Za-z0-9+/ character set, and the decoder rejects any character outside of this set.
eg : In Java

```
String originalInput = "test input";
String encodedString =
Base64.getEncoder().encodeToString(originalInput.getBytes());
```

In Javascript

```
const data = Base64UrlEncode(header) + '.' + Base64UrlEncode(payload);
const hashedData = Hash(data, secret);
const signature = Base64UrlEncode(hashedData);
```

- First, it encodes Header and Payload, join them with a dot .
- Next, makes a hash of the data using Hash algorithm (defined at Header) with a secret string.
- Finally, encodes the hashing result to get Signature.
-

After having Header, Payload, Signature, combines them into JWT standard structure: header.payload.signature.

Does JWT secures our data ?
JWT does NOT secure your data

JWT does not hide, obscure, secure data at all. You can see that the process of generating JWT (Header, Payload, Signature) only encode & hash data, not encrypt data.

The purpose of JWT is to prove that the data is generated by an authentic source.

So, what if there is a Man-in-the-middle attack that can get JWT, then decode user information?
Yes, that is possible, so always make sure that your application has the HTTPS encryption.

How Server validates JWT from Client ?
For creating a JWT , we use a Secret string to create Signature. This Secret string is unique for every Application and must be stored securely in the server side.

When receiving JWT from Client, the Server get the Signature, verify that the Signature is correctly hashed by the same algorithm and Secret string as above. If it matches the Server’s signature, the JWT is valid.

Detailed steps

Refer : JWT Details n diagrams

For JWT details : https://jwt.io/introduction
Refer to debugger n introduction sections

0. Copy earlier project created for Basic Authentication

1. Add jjwt dependency in pom.xml
<dependency>
<groupId>io.jsonwebtoken</groupId>
<artifactId>jjwt</artifactId>
<version>0.9.1</version>
</dependency>

2. Add these properties in application.properties
#JWT properties
#JWT Secret key for signing n Verification , later can be encrypted using Jasypt
SECRET_KEY=mySecretKey1234
#JWT expiration timeout in msec : 24*3600*1000
EXP_TIMEOUT=86400000

3. Create JWTUtils class .Add @Component for enabling it's D.I in the controller later.
Methods in JWTUtils
3.1 generate JWT token

3.2 get user name/email from the token

3.3 validate token

4.To Intercept all incoming reuqests (except those mentioned with permit All in sec config class),
create a custom filter class : extending from
o.s.web.filter.OncePerRequestFilter
Spring security will call it's doFilterInternal method , once per request
Steps
4.1 Extract JWT from the request header
Get Authorization header from request n check if it is not null n starting with "Bearer "
4.2 If it's present , extract n validate it (using JWT utils validation method : to check if it's not been tampered !)
4.3 In case of valid token ,
extract user name or email from the token
4.4 If user name is not null n if not already authenticated (i.e authentication info is not yet stored in sec context)
then extract UserDetails from UserDetailsService n create Authentication object(class : UserNamePasswordAuthenticationToken)
4.5 Save this authentication token in the sec ctx.

5. Configure JWT in security configuration class.
5.1 NO changes since Basic Auth , i.e no method needed for authentication

5.2 Modify authorization rules
Summary :

1. enable cors n disable CSRF
http.cors().and().csrf().disable()
2. In addition to other patterns : allow any HTTP OPTIONS request (which is typically to allow a pre flight request coming from react like front end . NOTE : it's not required for testing it with postman
antMatchers(HttpMethod.OPTIONS, "/**").permitAll()

3. Configure session management policy
To tell Spring Security NEVER create an HttpSession & DO NOT use HttpSession to obtain the SecurityContext

sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

4. Add custom jwt filter before a filter for processing an authentication based upon form submission.
http.addFilterBefore(jwtFilter,
UsernamePasswordAuthenticationFilter.class);

5.3 In our authentication REST controller , we are going to need to access the authentication manager. By default, it's not publicly accessible, and we need to configure Auth Manager bean
@Bean
public AuthenticationManager manager() throws Exception
{
 return super.authenticationManagerBean();
}

6. Create AuthController : for signin n signup
For signup (user registration)
6.1 Request payload : User DTO (containing : user details +Set<UserRole : enum>)
6.2 Using User Service layer , store user details (with encoded password n enum of roles mapped to Role entities) in DB . Store a child record in user_roles table.

For signin
request payload : Auth req DTO : email n password
resp payload : In case of success : Auth Resp DTO : mesg + JWT token + SC 200
IN case of failure : SC 401

Dependencies : Authentication Manager n JWTUtils

Steps in authentication
7.1
Create UserNamePasswordAuth token to wrap ONLY user supplied credentials(em n pass) , no authorities n authenticated=false
API : UsernamePasswordAuthenticationToken(String userName,String password)

7.2 Invoke Auth mgr's authenticate method , by passing above auth token
It rets : fully populated Authentication object (including granted authorities)if successful.
In case of failure :
Throws :
 DisabledException : if an account is disabled
 LockedException : if an account is locked

BadCredentialsException : if incorrect credentials are presented by client

7.3 In case of success : return
ResponseEntity : SC OK
Resp DTO : Auth successful mesg & generate JWT using utils n add it .

Demo
1. Run Test case : TestRoleRepository , to add roles.

----------More Details----------
For JWT details : https://jwt.io/introduction
Refer to debugger n introduction sections

For spring security javadocs : https://docs.spring.io/spring-security/site/docs/current/api/

HttpSecurity is a builder class and provides numerous convenience methods that can be chained. Under the hood, each method adds a filter the HTTP request needs to pass through

3.1 API : org.springframework.security.core.Authentication i/f
Methods : getAuthorities , getPrincipal , getCredentials ....

3.2 Authentication i/f method
java.lang.Object getPrincipal()
The identity of the principal being authenticated. In the case of an authentication request with username and password, this would be the username.
Since we have added these user details in the customized implementation of o.s.s.c.userdetails.UserDetails , you can type cast it to CustomUserDetailsImpl.

3.3 o.s.s.c.userdetails.UserDetails : i/f
Provides core user information.
You have to implement it to store user information which is later encapsulated into Authentication object.

3.4 org.springframework.security.authentication.AuthenticationManager
Implementing Classes: ProviderManager
Method :
Authentication authenticate(Authentication authentication) throws AuthenticationException
Attempts to authenticate the passed Authentication object, returning a fully populated Authentication object (including granted authorities) if successful.
An AuthenticationManager must honour the following contract concerning exceptions:

A DisabledException must be thrown if an account is disabled and the AuthenticationManager can test for this state.
A LockedException must be thrown if an account is locked and the AuthenticationManager can test for account locking.
A BadCredentialsException must be thrown if incorrect credentials are presented. While the above exceptions are optional, an AuthenticationManager must always test credentials.

Exceptions should be tested for and if applicable thrown in the order expressed above (i.e. if an account is disabled or locked, the authentication request is immediately rejected and the credentials testing process is not performed). This prevents credentials being tested against disabled or locked accounts.
Parameters:
authentication - the authentication request object(credentials)
Returns:
a fully authenticated object including credentials (principal)
Throws:
AuthenticationException - if authentication fails

3.5
org.springframework.security.authentication.UsernamePasswordAuthenticationToken
An imple class of Authentication i/f which is designed for simple presentation of a username and password.
Simply said it's the holder of username n password (i.e user credentials)

Objective : Invoking one REST API from another REST API
URL : http://host:port/api/employees/{empId}/accounts/{acctNo}

Resource : employees
Sub Resource : accounts

1.1 NetBanking REST Server
Ready made Spring boot project : NetBankingRESTServer

DML
```
insert into bank_customers values('hdfc-00001','Rama','12345');
insert into bank_accounts values(default,0,'SAVING',234567,'hdfc-00001');
insert into bank_accounts values(default,0,'CURRENT',24567,'hdfc-00001');
```

REST API : /bank/accounts/{acctNo}

1.2 Employee App REST Server for React App & client to NetBanking
1.3 Front end Postman (later add it in React app)

Details
2. Objective : Testing E-R with REST API + REST Client(RestTemplate)
Test setup : Postman -- Emp Management API invoking REST Banking API

Get Account summary for a bank customer.
Resource : /accounts
I/P : acct no

O/P : In case of success : Account DTO
or in case of invalid credentials : Send Error resp code : HTTP 404 (not found)

Get acct details
Method =GET (/bank/accounts/{acctNo})

Layers :
REST Controller --Service --Repository--POJO --DB
Customer 1<------* BankAccount
Customer : customer id(eg of assigned id here) ,name, password

BankAccount : acct id (auto generation) AcctType(enum) ,balance + Customer owner

For Data Transfer : DTOs
LoginRequest : customerId , password
LoginResponse : customer name & List<BankAccount>

How to make a REST call from one web app to another ?
Use : org.springframework.web.client.RestTemplate

The RestTemplate class in Spring Framework is a synchronous HTTP client for making HTTP requests to consume RESTful web services.

It exposes a simple and easy-to-use template method API for sending an HTTP request and also handling the HTTP response.

The RestTemplate class also provides aliases for all supported HTTP request methods, such as GET, POST, PUT, PATCH , DELETE, and OPTIONS.

In a service layer : inject

```
public class ClntService {
private RestTemplate template;

      @Autowired //autowire=constructor
      public ClntService(RestTemplateBuilder builder) {
            template = builder.build();
      }


}
      // SpEL : spring expression language
      @Value("${REST_GET_URL}")
      private String authUrl;
```

Use Method of o.s.w.c.RestTemplate public <T> ResponseEntity<T>
1. public <T> ResponseEntity<T> getForEntity(String url,Class<T> responseType,Object... uriVariables) throws RestClientException

2. public <T> ResponseEntity<T> postForEntity(String url,@Nullable Object request, Class<T> responseType, Object... uriVariables) throws RestClientException

2. Spring Boot Image handling : project objective

Objective : Image upload n download --from server side folder

Use multipart request

An HTTP multipart request is an HTTP request that HTTP clients construct to send files and data over to an HTTP Server. It is commonly used by browsers and HTTP clients to upload files to the server.

org.springframework.web.multipart.MultipartFile => A representation of an uploaded file received in a multipart request.

A representation of an uploaded file received in a multipart request. The file contents are either stored in memory or temporarily on disk. In either case, the user is responsible for copying file contents to a session-level or persistent store as and if desired. The temporary storage will be cleared at the end of request processing.

Stpes
0. Add a property in Entity n DTO , to store the image path
eg : private String imagePath;

1. To upload a file to a server side folder add the property (any property name)
file.upload.location=images
(This will be a folder relative to current project)

#limiting max file size for upload
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=15MB

2. File upload
In Application class , create "images" folder , if not present
2.1 Application class implements CommandLineRunner i/f n implement "run" method
It will run exactly once @ spring boot app launching time
OR
Create ImageHandlingService : as singleton n eager service bean
n create "images" folder , if not present

2.2 Create a rest controller
Other emp details are already stored in DB. image path has to be stored in the table n image should be uploaded in the "images" folder.

i/p : emp id : path var
MultipartFile : request param
resp : Emp DTO (with image file name set) or Simply ApiResponse
eg : Method : POST
http://localhost:8080/api/employees/{empId}/images

2.3 Image Handling Service
1. Create total path , using folder location ,file separator , image file's original file name

2. Copy multipart file into server side folder structure.
API of java.nio.file.Files : helper class
public static long copy(InputStream in, Path target, CopyOption... options) throws IOException

Copies all bytes from an input stream to a file returning no of bytes copied.

3. Save image path in Emp entity (to trigger update query) --setter

4. Return Emp DTO to the caller.(service --> controller --> JSON representation to clnt)

5. Test it with postman

3. Serving images (download) from server side folder.
i/p : emp id

Steps
3.1 In Controller
Method = GET
Add "produces" in the annotation(@GetMapping): to include image MediaType : JPEG_VALUE , GIF_VALUE, PNG_VALUE

3.2 In service layer
1. get emp details from emp id
2. Get complete path of the image
emp's getter
3. API of java.nio.file.Files : helper class
Method :
public static byte[] readAllBytes(Path path) throws IOException
Reads all the bytes from a file. The method ensures that the file is closed when all bytes have been read or an I/O error, or other runtime exception, is thrown.

For java.nio.file.Path : Paths.get(String first,String ... more)
4. return contents of image file(byte[]) to the controller

From controller , simply add it in ResponseEntity n send it to the clnt.
4. For react frontend , use : in <img> : src as
Method : GET
http://localhost:8080/api/employees/{empId}/image

Inheritance is one of the fundamental design principles in OOP. BUT relational databases don't support inheritance

JPA suggests different strategies to support inheritance hierarchies.

Hibernate Inheritance strategies
1 Single Table Strategy

In this inheritance, a single table is used to store all the instances of the entire inheritance hierarchy. The Table will have a column for every attribute of every class in the hierarchy. Discriminator columns identifies which class a particular row belongs.

Annoations used in super class

```
@Entity
@Inheritance(strategy = InheritanceType.SINGLE_TABLE)
@DiscriminatorColumn(name = "emp_type")
@Table(name = "employees")
public class Employee {....}
```

In sub class :

```
@Entity
@DiscriminatorValue("worker")
public class Worker extends Employee {..}

@Entity
@DiscriminatorValue("mgr")
public class Manager extends Employee {..}
```

With this mapping strategy, only a single table will be created for both concrete classes (Manager and Worker). Hibernate will create a discriminator column named emp_type to differentiate each concrete type. The value of this column will be either worker or mgr

2. Joined Table Strategy

This is the most logical solution, as it mirrors the object structure in the database. In this approach, a separate database table is defined for each of the class in the hierarchy and each table stores only its local attributes.

```
@Entity
@Inheritance(strategy = InheritanceType.JOINED)
public class Product {
    @Id @GeneratedValue
    private Long id;
    private String name;

}

@Entity
public class Book extends Product {
    private String isbn;
}
```

The above mapping will create two tables, one for super class Product and another for entity Book. Both the tables will have Product id column. The primary key of table Book will have foreign key relationship with primary key of Product table.

3. Table Per class
Not supported by all JPA vendors .
A separate table is defined for each concrete class in the inheritance hierarchy to store all the attribute of that class and all its super classes.

4. @MappedSuperClass

We want to extract common behavior in a super class in JPA entities but without having a table for that super class. How would you achieve that?

If we create a normal class as the super class, then as per JPA specifications, the fields for that class are not persisted in the database tables. We need to create a super class extracting the common fields and then annotate that class with @MappedSuperClass in order to persist the fields of that super class in subclass tables. A mapped super class has no separate table defined for it.

MappedSuperClass example

```
@MappedSuperclass
public class Person {

    @Id
    private Long id;
    private String name;

}

@Entity
public class Employee extends Person {
    private String company;

}
```

The above configuration will result in a single table for employee with 3 columns (id, name and company), 1 from Employee class itself and 2 are inherited from Person class. Person class will never have its own table in this case.

Aspect Oriented Programming(AOP)

WHY

Separating cross-cutting concerns(=repeatative tasks) from the business logic/ request handling /persistence

IMPORTANT

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other(eg : Controller separate from Service or DAO layers) and AOP helps you decouple cross-cutting concerns from the objects that they affect.(eg : transactional code or security related code can be kept separate from B.L or exception handling code from request handling method)
eg of ready made aspects provided by SC : tx management,security , exc handling

eg scenario --

Think of a situation Your manager tells you to do your normal development work(eg - write stock trading appln) + write down everything you do and how long it takes you.

A better situation would be you do your normal work, but another person observes what youre doing and records it and measures how long it took.

Even better would be if you were totally unaware of that other person and that other person was able to also observe and record , not just yours but any other peoples work and time.

That's separation of responsibilities.   --- This is what spring offers you through AOP.

It is NOT an alternative to OOP BUT it complements OOP.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.
Aspects enable the modularization of concerns.(concern=task/responsibility) such as transaction management,logging,security --- that cut across multiple types and objects. (Such concerns are often termed crosscutting concerns in AOP jargon)

Enables the modularization of cross cutting concerns(=task)
Eg : Logging,Security,Transaction management, Exception Handling

Similar in functionality to ---In EJB framework -- EJBObject
Struts 2 -- interceptors
Servlet -- filters.
RMI -- stubs
Hibernate --- proxy (hib frmwork -- lazy --- load or any--many associations --rets typically un-inited proxy/proxies)

AOP with Spring Framework

One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework.

Like DI, AOP supports loose coupling of application objects.

The functionalities that span multiple points of an application are called cross-cutting concerns.

With AOP, applicationwide concerns(common concerns-responsibilities or cross-cutting concerns like eg - declarative transactions , security,logging,monitoring,auditing,exception handling....) are decoupled from the objects to which they are applied.

Its better for application objects(service layer/controller/rest controller/DAO) to focus on the business domain problems that they are designed for and leave certain ASPECTS to be handled by someone else.

Job of AOP framework is --- Separating these cross-cutting concerns(repeatative tasks) from the core business logic

AOP is like triggers in programming languages such as Perl, .NET.

Spring AOP module provides interceptors to intercept an application, for example, when a method is executed, you can add extra functionality before or after the method execution.

Key Terms of AOP

Advice : Action(=cross cutting concern) to take either before/after or around the method (req handling logic ,OR B.L OR data access logic) execution. eg: transactional logic(begin tx,commit,rollback,session closing)

Advice describes WHAT is to be done & WHEN it's to be done.

eg : logging. It is sure that each object will be using the logging framework to log the event happenings , by calling the log methods. So, each object will have its own code for logging. i.e the logging functionality requirement is spread across multiple objects (call this as Cross cutting Concern, since it cuts across multiple objects). Wont it be nice to have some mechanism to automatically execute the logging code, before executing the methods of several objects?

2. Join Point : Place in application WHERE advice should be applied.(i.e which B.L methods should be advised)
(Spring AOP, supports only method execution join point)

3. Pointcut : Collection of join points.
It is the expression used to define when a call to a method should be intercepted.
eg :
@Pointcut("execution (Vendor com.app.bank.*.*(double))")

advice logic{....}

4. Advisor Group of Advice and Pointcut into a single unit.

5. Aspect : class representing advisor(advice logic + point cut definition)-- @Asepct -- class level annotation.

6. Target : Application Object containing Core domain logic.(To which advice gets applied at specified join points) --supplied by Prog

7. Proxy : Object created after applying advice to the target object(created by SC dynamically by implementing typically service layer i/f) ---consists of cross cutting concern(repeatative jobs , eg : tx management,security, exc handling)

8.Weaving -- meshing(integration) cross cutting concern around B.L (3 ways --- compile time, class loading time or spring supported -- dynamic --method exec time or run time)

Examples of readymade aspects :
Transaction management & security.

Types of Advice --appear in Aspect class

@Before : This advice (cross cutting concern) logic gets Executed only before B.L method execution.
@AfterReturning Executes only after method returns in successful manner
@AfterThrowing - Executes only after method throws exception
@After -- Executes always after method execution(in case of success or failure)
@Around -- Most powerful, executes before & after.

Regarding pointcuts
Sometimes we have to use same Pointcut expression at multiple places, we can create an empty method with @Pointcut annotation and then use it as expression in advices.

eg of PointCut annotation syntax

```
@Before("execution (* com.app.bank.*.*(..))")

@Pointcut("execution (* com.app.bank.*.*(double))")

// point cut expression
@Pointcut("execution (* com.app.service.*.add*(..))")
      // point cut signature -- empty method .
      public void test() {
      }
```

eg of Applying point cut

```
1. @Before(value = "test()")
public void logBefore(JoinPoint p) {.........}

2.
@Pointcut("within(com.app.service.*)")
    public void allMethodsPointcut(){}

@Before("allMethodsPointcut()")
    public void allServiceMethodsAdvice(){...}
```

3.
```
@Before("execution(public void com.app.model..set*(*))")
 public void loggingAdvice(JoinPoint joinPoint){pre processing logic
....}
```

4. //Advice arguments, will be applied to bean methods with single String argument

```
    @Before("args(name)")
    public void logStringArguments(String name){....}
```

5. //Pointcut to execute on all the methods of classes in a package

```
    @Pointcut("within(com.app.service.*)")
    public void allMethodsPointcut(){}
```

```
6.@Pointcut("execution(* com.core.app.service.*.*(..))") // expression
private void meth1() {}  // signature

7.@Pointcut("execution(* com.app.core.Student.getName(..))")
private void test() {}
```

Steps in AOP Implementation

1. Create core java project.
2. Add AOP jars to runtime classpath.
3. Add aop namespace to spring config xml.
4. To Enable the use of the @AspectJ style of Spring AOP & automatic proxy generation, add <aop:aspectj-autoproxy/>
5. Create Business object class. (using stereotype anotations)
6. Create Aspect class, annotated with @Aspect & @Component
7. Define one or more point cuts as per requirement

Eg of Point cut definition.

```
@PointCut("execution (* com.aop.service.Account.add*(..))")
public void test() {}
OR
@Before("execution (* com.aop.service.Account.add*(..))")
public void logIt()
{
   //logging advice code
}
```

Use such point cut to define suitable type of advice.

Test the application.

execution --- exec of B.L method

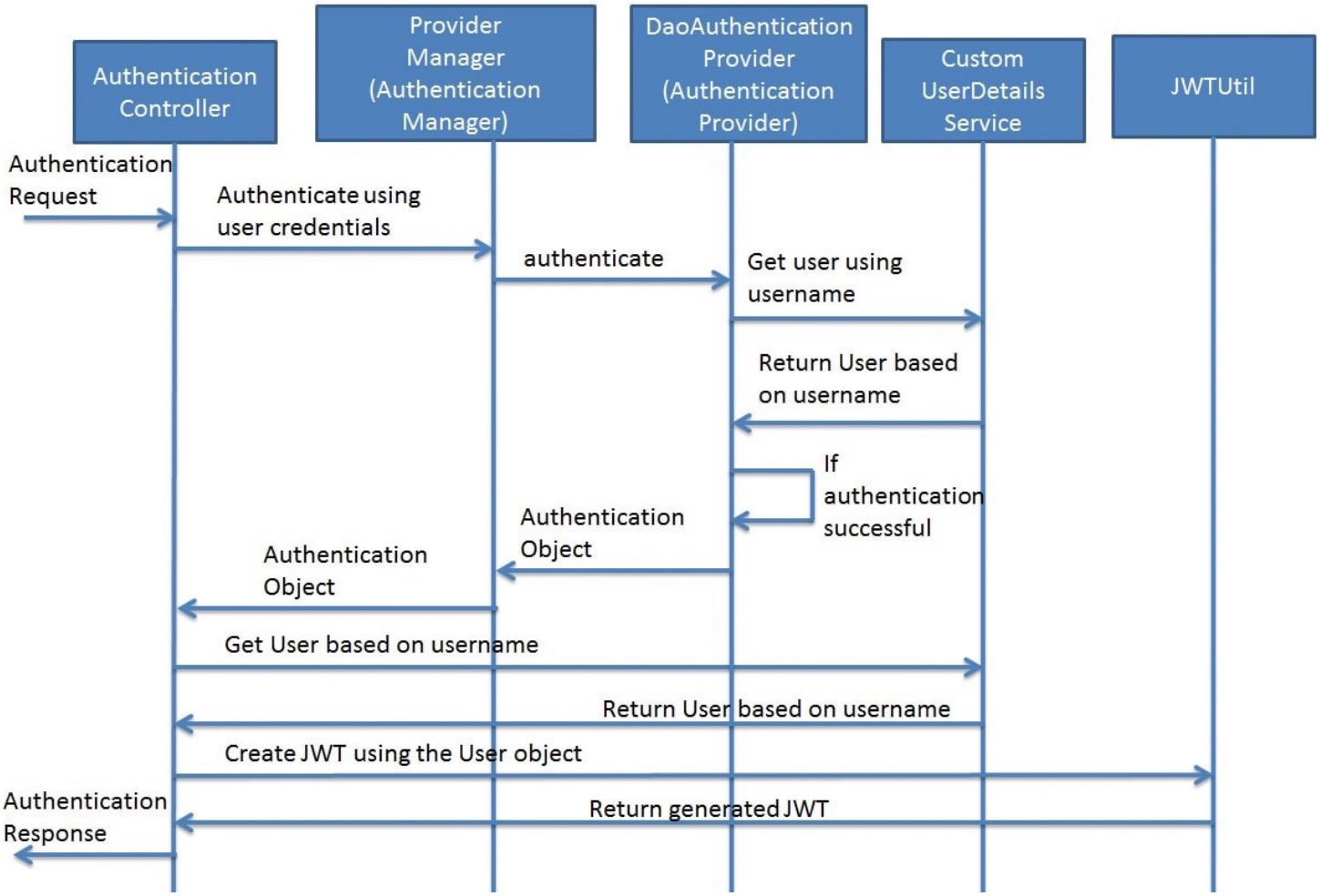
```
eg : @Before("execution (* com.app.bank.*.*(..))")
public void logIt()  {...}
```

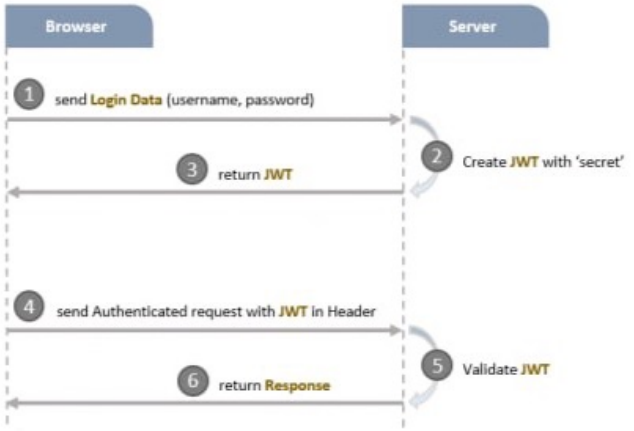
Above tell SC ---- to intercept ---ANY B.L method ---
having ANY ret type, from ANY class from pkg -- com.app.bank
having ANY args
Before its execution.

Access to the current JoinPoint

Any advice method may declare as its first parameter, a parameter of type org.aspectj.lang.JoinPoint (In around advice this is replaced by ProceedingJoinPoint, which is a subclass of JoinPoint.)

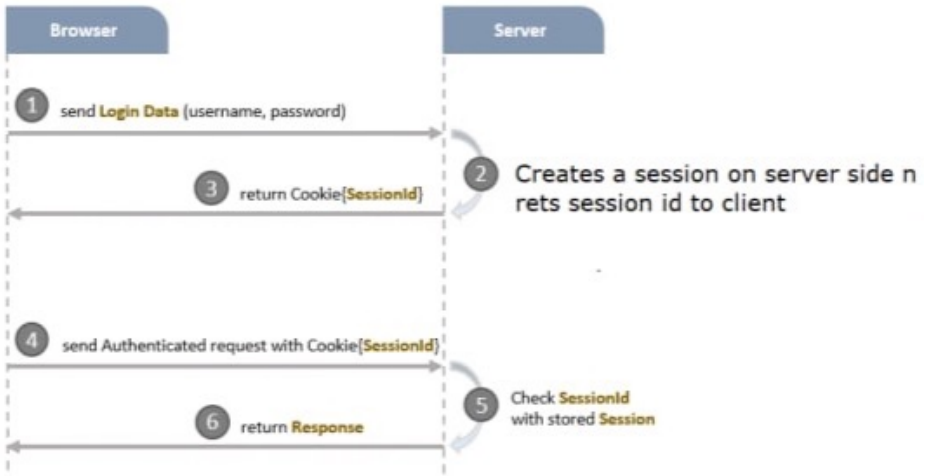
The org.aspectj.lang.JoinPointJoinPoint interface methods
1. Object[] getArgs() -- returns the method arguments.
2. Object getThis() --returns the proxy object
3 Object getTarget() --returns the target object
4. Signature getSignature() -- returns a description of the method that is being advised
5. String toString() -- description of the method being advised

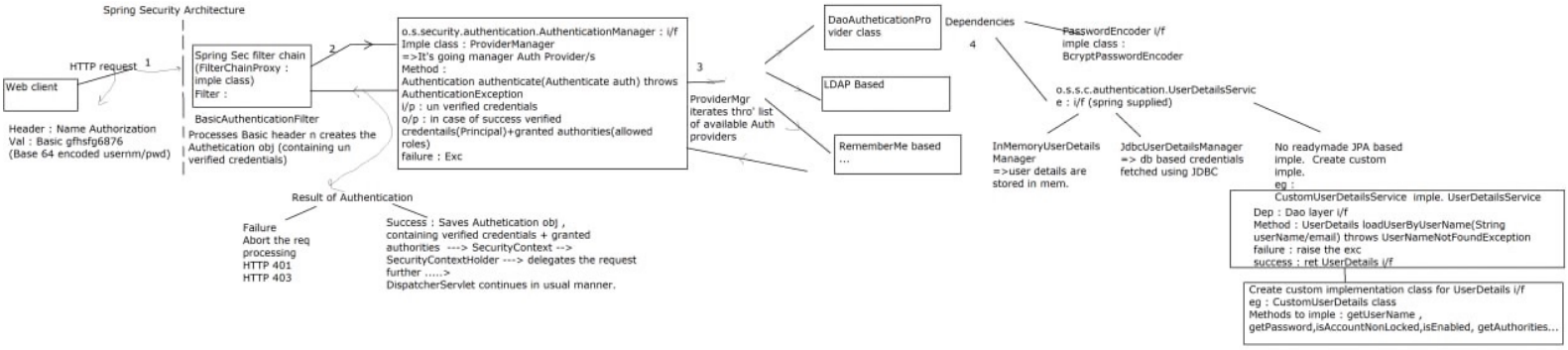




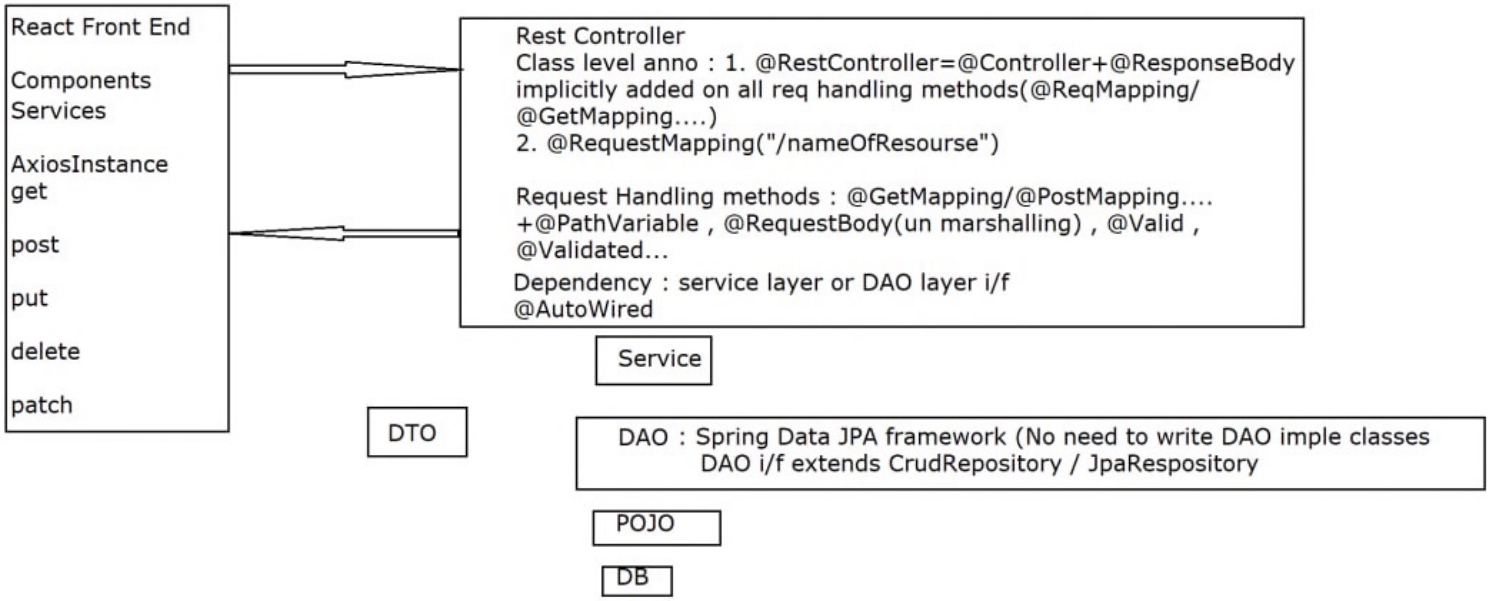
Session ID tokens : reference tokens (refer to the state on the server : HttpSession object)

JWT : value tokens (containing the encoded client details itself)





- REST API back end(REST server) for front end React App



Microservices Style Architecture (Not a full fledged one!)
Demo of one web service calling another via RestTemplate