# Advance Java By Madhura Anturkar

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.
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.jsp
Whenever the client sends the request to server for any resource such as a HTML document, it specifies

1. HTTP method
2. Request URI (eg : /bank/login.jsp)
3. Protocol version
4. Optional Header information.
5. After the client sends the request, server processes the request and sends the response.

HTTP Response contains

1. Response status information(404/200)
2. Response headers (eg : cookies/resp cont type
3. eg : text/html, application/json image/gif.... /cont length)
4. 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/*, */*
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=abc
The POST request method
The POST request method is typically used to access 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.

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

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

1. Some web servers limit the length of the URL.
2. 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.
3. For POST -- no such limitations.

Version Java EE 7 (J2EE 1.7)
What is J2EE ?

Consists of specifications only .
Which specs ? (Rules or contract )
Specifications of services required for any enterprise application.
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 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 etc...
Vendor of J2EE specs -- Oracle / Sun.
Implementation -- left to vendors (J2EE server vendors)
J2EE compliant web server --- Apache -- Tomcat (web container)
Services implemented --- servlet API,JSP API,Security,Connection pooling,JNDI
J2EE complaint application server --- web container + EJB container

- ALL J2EE services imple

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

1. Can support different types of clnts --- thin client(web clnt)
2. thick clnt --- application clnt
3. smart clnts -- mobile clnts
4. 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.
Layers involved in request-response flow
Web browser sends the request (URL)

http://www.abc.com:8080/first_web/index.html
Host --Web server--Web Container(server side JVM)--Web application---HTML/JSP/Servlet....
What is dyn web application --- server side appln --deployed on web server --- meant for servicing typically web clnts(thin) -- using application layer protocol HTTP /HTTPS
(ref : diag request-resp flow)
Read --HTTP basics , request & response structure.
Objective ?: Creating & deploying dyn web appln on Tomcat -- For HTML content
IDE auto creates J2EE compliant web application folder structure .
Its details -- Refer to diag (j2ee compliant web app folder structure)
What is Web container --- (WC) & its jobs

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

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

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

## What is a servlet ?

-- Java class -- represents dyn web comp - whose life cycle will be managed by WC
no main method
life cycle methods --- init,service,destroy
Job list

1. Request processing
2. B.L
3. response generation
4. Data access logic

5. Page navigation

## Servlet API details --refer to diag servlet-api

Creating & deploying Hello Servlet.

1. Using @WebServlet annotation
2. OR
3. Using XML tags
4. How to deploy a servlet w/o annotations --- via XML tags
5. web.xml
6. abc pages.TestServlet
7. abc /hello

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). If not found --- will check for @WebServlet annotation.

How to read request parameters in Servlet ?

javax.servlet.ServletRequest i/f methods

1. public String getParameter(String paramName)
2. public String[] getParameterValues(String paramName)

Servlet & JDBC Integration

Layers involved

Servlet --DAO(DBUtils) -- POJO -- DB

LoginServlet

D.M -- dao

init --- DAO inst

destroy --- Dao's clean up

doGet

1. set cont type
2. pw
3. read rq params
4. invoke Dao's --validate
5. null --- Invalid login --retry link
6. not null ---mesg , display customer dtls

URL

http://www.abc.com:8080/day1_web/validate?em=abc&pass=1234

Explain the parts.

URI --- /day1_web/validate?em=abc&pass=1234

1. What will happen --if u don't add "/" in url-pattern? --- web app doesn't get deployed ---- invalid url pattern (IllegalArgumentException)

2. What will happen --if u add "/url-pattern" in form action or href ? --- HTTP 404 (uses absolute url eg : http://host:port/hi)
3. Can 1 servlet have multiple url patterns ? ---YES
4. Can any 2 servlets have the same url pattern ? -- 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/jar
application/pdf
mages/jpeg
images/png
images/gif
audio/mp3
video/mp4A 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.What is a Session?
Session is a conversational 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) throughout current session

session = Represents duration or time interval
Consists of all requests/resps coming from/to same clnt from login to logout or till session expiration tmout.
There are several techniques for session tracking.

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) & downloaded (sent) to clnt browser--- within response header
Cookie represents data shared across multiple dyn pages from the SAME web appln.
Steps :

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

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

1. To retrieve the cookies :
2. HttpServletRequest :
3. 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

1. Web dev (servlet prog) has to manage cookies.
2. Cookies can handle only text data : storing Java obj or bin data difficult.
3. As no of cookies inc., it will result into increased net traffic.
4. 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 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.

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
2. API of HttpServletRequest ---
3. HttpSession getSession()
4. Desc --- Servlet requests WC to either create n return a NEW HS object(for new clnt) or ret the existing one.

Above method creates either a NEW HTTP session obj for new user or returns existing HTTP session object for old user.

HttpSession --- i/f from javax.servlet.http

NEW --- HttpSession --empty map

String,Object ---- (entry)= attribute

OR

HttpSession getSession(boolean create)

1. : How to save data in HttpSession?(scope=entire session)
2. API of HttpSession i/f
3. public void setAttribute(String attrName,Object attrVal)

equivalent to map.put(k,v)

eg : hs.setAttribute("cart",l1);

1. For retrieving session data(getting attributes)
2. public Object getAttribute(String attrName)
3. To get session ID (value of the cookie whose name is jsessionid -- unique per client)
4. String getId()

4.5 How to remove attribute from the session scope?

public void removeAttribute(String attrName)

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

7.How to find all attr names from the session ?

public Enumeration getAttributeNames()

--rets java.util.Enumeration of attr names.

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)

Have you tried - lab sequence ?

Revise

Why J2EE & what is it ?

Request response flow

URL --- http://www.abc.com:8080/day1_web/validate?em=abc&pass=1234

J2EE compliant web app folder structure

What is a Servlet & its jobs

API

Create n deploy servlets(using annotations)

eg : @WebServlet("/register)
public class RegServlet extends H.S {....}
What will happen if u don't add / in url pattern ? --error(web app doesn't get deployed --- invalid url pattern (IllegalArgumentException))

1. What will happen --if u add "/url-pattern" in form action or href ? --- HTTP 404 (uses absolute url eg : http://host:port/hi)

Can 1 servlet be deployed using multiple url-patterns? --
Can 2 servlets be deployed having same url-pattern? --
Can 2 web-apps have same ctx path ? --
What will happen if servlet class is not public ? --
What will happen if u supply a parameterize constr to the servlet ? ---
Why a servlet need not override service method ?
Deploy servlet with xml tags
How to read request params sent from the clnt ?
Servlet--JDBC integration

## Page navigation --client pull

@WebServlet annotation
load-on startup --def value =-1
eg : @WebServlet(value={"/hi","/hello"},loadOnStartup=1)
public class Servlet1 extends H.S {...}
@WebServlet("/test")
public class Servlet2 extends H.S {...}
, urlPatterns(value)
use case of loadOnStartup
--in case of time consuming init methods(eg : DispatcherServlet of Spring MVC)
Servlet life cycle with threads.(ref : diag thread pool executor)

## CGI vs Servlets

Servlet & JDBC Integration
Layers involved
Servlet --DAO(DBUtils) -- POJO -- DB
LoginServlet
D.M -- dao
init --- DAO inst
destroy --- Dao's clean up
doGet

1. set cont type
2. pw

3. read rq params
4. invoke Dao's --validate
5. null --- Invalid login --retry link
6. not null ---mesg , display customer dtls

Enter session tracking

IMPORTANT --- copy context.xml from "j2ee_help\day2_help\config files" in your /conf

## Then try out cookie based session tracking.

1. Can 1 servlet have multiple url patterns ? ---YES
2. Can any 2 servlets have the same url pattern ? -- NO

Page Navigation Techniques

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

2 Ways

1. Client Pull
2. Taking the client to the next page in the NEXT request
3. 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

public void sendRedirect(String redirectURL)

eg : For redirecting client from Servlet1 (/s1) to Servlet2 (/s2) , use response.sendRedirect("s2");

1. Server Pull.
2. Taking the client to the next page in the same request.
3. Also known as resource chaining or request dispatching technique.
4. 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)
2. API of ServletRequest
3. 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.

1. Include scenario
2. API of RequestDispatcher
3. 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.

## Regarding wrapper classes

1. What's need of wrapper classes?
2. ---1. to be able to add prim types to growable collection(growable data structure eg -- LinkedList)
3. --- 2. wrapper classes contain useful api(eg --- parseInt,parseFloat....,isDigit,isWhiteSpace...)
4. What are wrappers? --- Class equivalent for primitive types
5. -- Inheritance hierarchy
6. java.lang.Object --- Character (char)
7. java.lang.Object --- Boolean
8. Object -- Number -- Byte,Short,Integer,Long,Float,Double
9. Constrs & methods --- for boxing & unboxing
10. boxing= conversion from prim type to the wrapper type(class type)
11. un-boxing = conversion from wrapper type to the prim type
12. eg
13. Integer(int data) --- boxing
14. Integer i1=new Integer(100);
15. //un-boxing
16. int data=i1.intValue();

```
Integer i1=100;//no err from JDK 1.5
sop(i1);
int data=1234;
i1++;//Integer--->int(auto unboxing), inc ,auto box
Object o=123.45;//auto-boxing(double--->Double)--up casted to Object
```

Number n1=true;//auto-box----X(up casted) to Number
Object o2=false;//auto box -- up casting
Double d1=1234;//auto boxing (int --->Integer) ---X--Double

1. JDK 1.5 onwards --- boxing &unboxing performed automatically by java compiler,when required. --- auto-boxing , auto-unboxing,
2. examples

## Regarding synchronization

1. Only methods (or blocks) can be synchronized, not variables or classes.
2. Each object has just one lock.
3. Not all methods in a class need to be synchronized. A class can have both
4. synchronized and non-synchronized methods.
5. If two threads are about to execute a synchronized method in a class, and both threads are using the same instance of the class to invoke the method,only one thread at a time will be able to execute the method. The other thread will need to wait until the first one finishes its method call. In otherwords, once a thread acquires the lock on an object, no other thread can enter ANY of the synchronized methods in that class (for that object).
6. If a class has both synchronized and non-synchronized methods, multiple
7. threads can still access the class's non-synchronized methods. If you have methods that don't access the data you're trying to protect, then you don't
8. need to synchronize them. Synchronization can cause a hit in some cases (or
9. even deadlock if used incorrectly), so you should be careful not to overuse it.
10. If a thread goes to sleep(or invokes join,yield,notify) or encounters context switching , it holds any locks it has—it doesn't release them.
11. A thread can acquire more than one lock. For example, a thread can enter a
12. synchronized method, thus acquiring a lock, and then immediately invoke
13. a synchronized method on a different object, thus acquiring that lock as
14. well. As the stack unwinds, locks are released again. Also, if a thread acquires
15. a lock and then attempts to call a synchronized method on that same
16. object, no problem. The JVM knows that this thread already has the lock for
17. this object, so the thread is free to call other synchronized methods on the
18. same object, using the lock the thread already has.
19. eg :
20. class A {
21. private B b1;
22. synched void test()
23. {
24. ...

25. b1.testMe();
26. }
27. }

```
class B
{
synched void testMe()
{
//some B.L
}
}
```

1. You can synchronize a block of code rather than a method.
2. When to use synched blocks?
3. Because synchronization does hurt concurrency, you don't want to synchronize
4. any more code than is necessary to protect your data. So if the scope of a method is
5. more than needed, you can reduce the scope of the synchronized part to something

less than a full method—to just a block. OR when u are using Thread un-safe(un-sunchronized eg -- StringBuilder or HashMap or HashSet) classes in your appln.

Regarding static & non -static synchronized

1. Threads calling non-static synchronized methods in the same class will
2. only block each other if they're invoked using the same instance. That's
3. because they each lock on "this" instance, and if they're called using two different
4. instances, they get two locks, which do not interfere with each other.
5. Threads calling static synchronized methods in the same class will always
6. block each other—they all lock on the same Class instance.
7. A static synchronized method and a non-static synchronized method
8. will not block each other, ever. The static method locks on a Class
9. instance(java.lang.Class) while the non-static method locks on the "this" instance —these actions do not interfere with each other at all.

Refer to Thread state transitions figure

A -- transition from rdy to run -----> Running

Triggered by --- in time slice based scheduling --- time slot of earlier thrd over OR in pre-emptive multitasking -- higher prio thrd pre-empts lower prio thrd.

B --- transition from running ---> ready to run

Reverse of earlier transition OR

public static void yield()----

Requests underlying scheduler to swap out current thrd SO THAT some other lower prio thrd MAY get a chance to run. (to avoid thrd starvation -- i.e co-operative multi-

tasking.)
C --running state --- Only in this state --- run() method gets executed.
running --->dead --- Triggers -- run() method returns in healthy manner . OR run() aborts due to un-handled , un-checked excs.
D -- blocked ---> rdy to run --- when any of blocking condition is removed --- blocked thrd enters rdy pool & resumes competition among other thrds.
API Involved

1. Thread class constructor to be used in extends Thread scenario
2. 1.1 Thread() --- A new thrd is created BUT with JVM supplied name.
3. 1.2 Thread(String nm) -- creates named thread.
4. Thread class constructor to be used in implments scenario
5. 2.1 Thread (Runnable target/inst) --- Creates a new thrd --- by passing instance of the class which implements Runnable i/f.
6. Run time significance -- Whenever this thrd gets a chance to run --- underlying task scheduler -- will invoke(via JVM) this class's run() method.
7. 2.2 Thread (Runnable inst,String name)

public class Myclass extends Thread --- start()
Vs
public class Myclass implements Runnable --- This class simply represents a runnable task. Prog MUST create a thrd class inst BY attaching Runnable task to it.

## Thread class API

1.public String getName()

1. public void setName(String nm)
2. public static Thread currentThread() -- rets ref of the invoker thrd.
3. public int getPriority() -- rets current prio.
4. Prio scale -- 1---10(MIN_PRIORITY,MAX_PRIORITY)
5. NORM_PRIORITY ---- 5
6. 4.5 public void setPriority(int prio) --- must be invoked before start()

DO NOT rely on priority factor -- since it is ultimately specific to underlying OS prio range.-- may cause loss of portability.
t1 --- max-prio ---run() --- begin up-counter -- obs val after 10 secs
t1 --- min-prio ---run() --- begin up-counter -- obs val after 10 secs
t1 --- max-prio & t2 -- min prio

1. public String toString() --- to ret -- name,prio & thrd grp name
2. public static void sleep(long ms) throws InterruptedException

Objective -1. to test concurrency of thrd : in extends thrd scenario.
Create a Thrd class, add simple loop with dly in run method to display the exec. sequence.

Write main method : which will instantiate thrds(multi-thrded system) & test the concurrent exec. of main thrd along with other thrds.
To ensure that main thrd terminates last : no orphan thrds in the system.
Thread class Method :

1. public void join() throws InterruptedExc.
2. The invoker thrd gets blocked until the specified thrd joins it(i.e specified thrd becomes dead)
3. eg : 2 thrds t1 & t2 are running concurrently.
4. If u invoke : in run() method of t1 :
5. t2.join() ----> t1 gets blocked until t2 is over.
6. t3 -- t1.interrupt()
7. t1 : Blocked on join
8. Unblcking triggers -- t2 over,getting interrupt signal.
9. public void join(long ms) throws InterruptedExc
10. waits max for specified tmout .
11. Unblocking triggers -- t2 over,getting interrupt signal, tmout exceeded.

Objective -- Ensure no orphans , w/o touching child thread class.
Objective
Replace for loop from thrds, by indefinite loop (while true) & still ensure -- no orphans.
Start --- main + 3 child thrds. --- main(parent thrd waits patiently for 5 secs. & then somehow forces termination of child thrds & then terminates last.
API ---
public void interrupt() -- sends interrupt signal to the specified thread. If specified thrd has invoked any method (sleep,join,wait)-- having throws clause of InterruptedExc --- then only thrd gets UNBLOCKED due to InterruptedExc.
NOTE -- Thread which is blocked on I/O -- CANT be un-blocked by interrupt signal.
Threads blocked by invoking any method -- having InterruptedException (sleep,join,wait) can be unblocked by interrupt signal.

1. Objective - to test concurrency of thrds : in implements Runnable scenario.

2.5 implements scenarion

1. 3.

Objective : create 3 thrds with 3 random sleep durations(range is 500ms-5sec) & start them conc. & ensure that main terminates last.
For random nos--- java.util.Random() , nextInt,nextInt(int upperLim)
Objective : create 3 thrds & start them conc. & ensure that main terminates last.
2nd thrd should accept data from console, dont supply data & observe .
How to unblock a thread , which is blocked on I/O?
Objective : Apply multi threading to Swing application.

Create Swing application -- with start & stop buttons, in south region.
Create JPanel in center region , with some default color.
When start button is clicked, center panel should start changing color(random color) periodically.
When stop button is clicked, stop changing color.
Objective -- To store emp details , dept wise in SAME data file.(text buffered manner--- PrintWriter )
Design :

1. Emp class --- id,deptid,name,sal
2. Write Utils class
3. d.m --- pw
4. constr --pw inst ---
5. void writeData(Emp e) {...}

void cleanUp() {...}
constr ---create PW inst --
PW pw=new PW(new FW("emps.data"),true);
---add writeData(....) : instance method to write Emp dtls (first name, last name ,deptId of the emp)
to the file with small dly in between.(why dly ? --- for simulating practical scenario & also to add randomness to code)
do u need clean up method- -- yes -- close pw.

1. Write Dept Handler runnable task class --- implements
2. Override run() method which will invoke writeData method till 'exit' condition is encountered.
3. Add stop/exit method to enable 'exit' flag.
4. ....
5. {
6. Emp e;
7. Utils u;
8. constr(u,e)
9. {this.e=e;
10. //u=new Utils(); ------

public void run()
{
while(!exit)
u.writeData(e);
}

1. Write Tester class : accept some emp dtls . Create depthandler task per dept ,attach thrds & start them.

2. Wait for key stroke ---
3. System.in.read();
4. stop all child thrds
5. ensure no orphans
6. clean up -- pw

Wait for the key stroke : upon key stroke --- stop all child thrds & then finally terminate main.
Observed o/p ------garbled display or garbled data written in file.
Reason : multiple thrds trying to access the shared resource concurrently.
eg of shared resource : Console or file device,socket,DB table
Solution : Any time when asynch thrds need to access the SHARED resource : LOCK the shared resource --- so that after locking -- only single thread will be able to access the resource concurrently
When is synchronization(=applying thread safety=locking shared resource) required?
In multi-threaded java applns -- iff multiple thrds trying to access SAME copy of the shared resource(eg -- reservation tkt,db table,file or socket or any shared device) & some of the threads are reading n others updating the resource
How to lock the resource?
Using synchronized methods or synchronized blocks.
In either approach : the java code is executed from within the monitor & thus protects the concurrent access.
Note : sleeping thrd sleeps inside the monitor(i.e Thread invoking sleep(...) , DOESn't release the ownership of the monitor)
eg classes :
StringBuilder : thrd-unsafe.--- unsynchronized --- if multiple thrds try to access the same copy of the SB, SB may fail(wrong data)
StringBuffer --- thrd -safe ----synchronized internally--- if multiple thrds try to access the same copy of the SB, only 1 thrd can access the SB at any parti. instance.
which is reco class in single threaded appln? --- StringBuilder
multiple thrds -- having individual copies -- StingBuilder
multiple thrds -- sharing same copy -- StringBuilder --
identify code to be guarded -- sb 's api -- invoke thrd unsafe API -- from inside synched block.
ArrayList(inherently thrd un-safe) Vs Vector(inherently thrd safe)
HashMap(un-safe) Vs Hashtable(thrd safe)
synchronized block syntax --- to apply synchro. externally.
synchronized (Object to be locked--- shared resource)
{

//code to be synchronized --methods of shared res. -- thrd safe manner(from within monitor)
}

1. If any thrd is accessing any synched method of 1 obj, then same thrd or any other thrd CANT concurrently access same method of the same obj.(Tester1.java)
2. If any thrd is accessing any synched method of 1 obj, then same thrd or any other thrd CANT concurrently access same method or any other synchronized method of the same obj.(Tester2.java)
3. If thrds have their own independent copies of resources, synch IS NOT required. (Tester2.java)
4. 3.If u are using any thrd un-safe code(ie. ready code without source) --& want to apply thrd safety externally --- then just wrap the code within synched block to use locking.(Tester3.java)

Objective : Create Producer & Consumer thrds .
Producer produces data samples & consumer reads the same.
For simplicity : let the data be represented by : single Emp record
Producer produces emp rec sequentially & consumer reads the same.
Rules : 1 when producer is producing data , consumer thrd concurrently should not be allowed to read data & vice versa.
Any more rules?????????????
Yes --- correct sequencing is also necessary in such cases.
Rule 2 : Producer must 1st produce data sample ---consumer reads data sample & then producer can produce next data sample. Similarly consumer should not be able to read stale(same) data samples .
ITC --- API level
Object class API

1. public void wait() throws IE ---thrd MUST be owner of the monitor(i.e invoke wait/notify/notifyAll from within synched block or method) --- othewise MAY get IllegalMontitorStateExc

---causes blocking of the thrd outside montitor.
UnBlocking triggers --- interrupt(not reco --- since it may cause death of thrd) , notify/notifyAll --- reco.

1. 1. public void wait(long ms) throws IE
2. UnBlocking triggers --- interrupt(not reco --- since it may cause death of thrd) , notify/notifyAll --- reco.,tmout exceeded

2.2 public void notify() -- MUST be invoked from within monitor , ow may get IllegalMonitorStateExc
Un-blocks ANY waiting thread , blocked on SAME MONITOR

2.3 public void notifyAll() -- Un-blocks ALL waiting threads , blocked on SAME MONITOR
notify/notifyAll--- DOESN't BLOCK the thread & Doesn't release lock on monitor. --- send wake up signal -- to thrd/s waiting on same monitor.
wait --- Blocks the thread --- Releases lock on the monitor.
volatile --- java keyword, applicable at data member.
typically used in multi-threaded scenario only when multiple thrds are accessing the same data member.
Use --- to specify-- that data var. is being used by multiple thrds concurrently -- so dont apply any optimizations(OR the value of the variable can get modified outside the current thrd) . With volatile keyword -- its guaranteed to give most recent value.
The volatile modifier tells the JVM that a thread accessing the variable must always get its own private copy of the variable with the main copy in memory
A race condition is a special condition that may occur inside a critical section. A critical section is a section of code that is executed by multiple threads and where the sequence of execution for the threads makes a difference in the result of the concurrent execution of the critical section.
When the result of multiple threads executing a critical section may differ depending on the sequence in which the threads execute, the critical section is said to contain a race condition.
Race condition means that the threads are racing through the critical section, and that the result of that race impacts the result of executing the critical section.
Critical Sections
Running more than one thread inside the same application does not by itself cause problems. The problems arise when multiple threads access the same resources. For instance the same memory (variables, arrays, or objects), systems (databases, web services etc.) or files.
In fact, problems only arise if one or more of the threads write to these resources. It is safe to let multiple threads read the same resources, as long as the resources do not change.
Here is a critical section Java code example that may fail if executed by multiple threads simultaneously:
public class Counter {

```
protected long count = 0;


public void add(long value){
    this.count = this.count + value;
}
```

}
Imagine if two threads, A and B, are executing the add method on the same instance of the Counter class. There is no way to know when the operating system(scheduler) switches between the two threads. The code in the add() method is not executed as a single atomic instruction by the Java virtual machine. Rather it is executed as a set of smaller instructions, similar to this:
Read this.count from memory into PC register.
Add value to PC register.
Write register to memory.
Observe what happens with the following mixed execution of threads A and B:

```
    this.count = 0;
```

A: Reads this.count into a register (0)
B: Reads this.count into a register (0)
B: Adds value 2 to register
B: Writes register value (2) back to memory. this.count now equals 2
A: Adds value 3 to register
A: Writes register value (3) back to memory. this.count now equals 3
The two threads wanted to add the values 2 and 3 to the counter. Thus the value should have been 5 after the two threads complete execution. However, since the execution of the two threads is interleaved, the result ends up being different.
In the execution sequence example listed above, both threads read the value 0 from memory. Then they add their individual values, 2 and 3, to the value, and write the result back to memory. Instead of 5, the value left in this.count will be the value written by the last thread to write its value. In the above case it is thread A, but it could as well have been thread B.
Race Conditions in Critical Sections
The code in the add() method in the example earlier contains a critical section. When multiple threads execute this critical section, race conditions occur.
More formally, the situation where two threads compete for the same resource, where the sequence in which the resource is accessed is significant, is called race conditions. A code section that leads to race conditions is called a critical section.
Preventing Race Conditions
To prevent race conditions from occurring you must make sure that the critical section is executed as an atomic instruction. That means that once a single thread is executing it, no other threads can execute it until the first thread has left the critical section.Thread related API
Starting point
1. java.lang.Runnable --functional i/f
2. SAM -- public void run()

3. Prog MUST override run() -- to supply thread exec. logic.
4. java.lang.Thread --class -- imple . Runnable
5. It has imple. run() -- blank method.
6. Constrs of Thread class in "extends" scenario
7. 3.1 Thread() -- Creates a new un-named thrd.
8. 3.2 Thread(String name) -- Creates a new named thrd.
9. Constrs of Thread class in "implements" scenario
10. 4.1 Thread(Runnable instance) -- Creates a new un-named thrd.
11. 4.2 Thread(Runnable instance,String name) -- Creates a new named thrd.
12. Runnable instance ---instance of the class that implements Runnable i/f(or ano inner class or lambda expression)

Methods of Thread class

1. public void start() -- To cause transition from NEW -- RUNNABLE
2. throws IllegalThreadStateException -- if thrd is alrdy runnable or dead.
3. public static void yield() -- Requests the underlying native scheduler to release CPU for current thread , so that curnt thread enters rdy pool.(It's a request)
4. Use case -- co operative multi tasking(to allow lesser prio thrds to access processor) Doesn't release the locks.
5. public void setName(String nm)
6. public String getName()
7. Priority scale -- 1---10
8. Thread class consts --MIN_PRIO=1 , MAX_PRIO=10 , NORM_PRIO =5
9. public void setPriority(int prio)
10. public static Thread currentThread() -- rets invoker(current) thrd ref.
11. public String toString() -- Overrides Object class method , to ret
12. Thread name,priority,name of thrd grp.
13. 8.public static void sleep(long ms) throws InterruptedException
14. --Blocks the invoker thread for specified ms delay.(TIMED_WAITING)

8.5 public boolean isAlive()

1. public void join() throws InterruptedException
2. --Causes the invoker thread to block till specified thread gets over.

10 public void join(long ms) throws InterruptedException

--Causes the invoker thread to block till specified thread gets over OR tmout elapsed

1. public void interrupt() -- interrupts(un blocks ) the threads blocked on --- sleep/join/wait
    a. Test concurrency (concurrent exec of multiple thrds) using extends Thread.

1.1 Create a class ---extends Thread

1.2 constr --name

1.3 --override run() ---

delay loop
1.4 Tester ---main --main thrd --- to create child thrds --start & confirm asynch nature.

1. Test concurrency (concurrent exec of multiple thrds) using imple. Runnable -- separate class / ano inner class /lambda expression
2. 3.

Expression Language implicit variables(case sensitive)

1. pageContext : PageContext object (javax.servlet.jsp.PageContext) asso.
2. with current page.
3. pageScope - a Map that contains page-scoped attribute names and their
4. values.
5. requestScope - a Map that contains request-scoped attribute names and their
6. values.
7. sessionScope - a Map that contains session-scoped attribute names and their
8. values.
9. applicationScope - a Map that contains application-scoped attribute names
10. and their values.
11. param - a Map that contains rq. parameter names to a single String parameter
12. value (obtained by calling ServletRequest.getParameter(String name)).
13. paramValues - a Map that contains rq. param name to a String[] of all values
14. for that parameter (similar to calling ServletRequest.getParameterValues(name)
15. initParam - a Map that contains context initialization parameter names and their
16. String value (obtained by calling ServletContext.getInitParameter(String name)).
17. eg : ${initParam.db_drvr}
18. cookie : Map.Entry of cookies. (entrySet of cookies)
19. eg : ${cookie.cookiename.value}

key ---cookie name
value ---javax.servlet.http.Cookie
${cookie.JSESSIONID.value}
---cookie.get("JSESSIOIND").getValue()

1. To retrieve err details from Error handling page.
2. ERR causing URI : ${pageContext.errorData.requestURI }
3. ERR code : ${pageContext.errorData.statusCode}
4. ERR Mesg : ${pageContext.exception.message }
5. Throwable : ${pageContext.errorData.throwable}
6. Throwable Root cause: ${pageContext.errorData.throwable.cause}

eg :

${sessionScope.abc}

<%@ page language="java" contentType="text/html; charset=${encoding}" pageEncoding="${encoding}"%> Insert title here ${cursor} 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 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 public void sendRedirect(String redirectURL) eg : For redirecting client from Servlet1 (/s1) to Servlet2 (/s2) , use response.sendRedirect("s2"); 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. Regarding synchronization 1. Only methods (or blocks) can be synchronized, not variables or classes. 2. Each object has just one lock. 3. Not all methods in a class need to be synchronized. A class can have both synchronized and non-synchronized methods. 4. If two threads are about to execute a synchronized method in a class, and both threads are using the same instance of the class to invoke the method,only one thread at a time will be able to execute the method. The other thread will need to wait until the first one finishes its method call. In otherwords, once a thread acquires the lock on an object, no other thread can enter ANY of the synchronized methods in that class (for that object). 5. If a class has both synchronized and non-synchronized methods, multiple threads can still access the class's non-synchronized methods. If you have methods that don't access the data you're trying to protect, then you don't need to synchronize them.

Synchronization can cause a hit in some cases (or even deadlock if used incorrectly), so you should be careful not to overuse it. 6. If a thread goes to sleep(or invokes join,yield,notify) or encounters context switching , it holds any locks it has—it doesn't release them. 7. A thread can acquire more than one lock. For example, a thread can enter a synchronized method, thus acquiring a lock, and then immediately invoke a synchronized method on a different object, thus acquiring that lock as well. As the stack unwinds, locks are released again. Also, if a thread acquires a lock and then attempts to call a synchronized method on that same object, no problem. The JVM knows that this thread already has the lock for this object, so the thread is free to call other synchronized methods on the same object, using the lock the thread already has. eg : class A { private B b1; synched void test() { ... b1.testMe(); } } class B { synched void testMe() { //some B.L } } 8. You can synchronize a block of code rather than a method. When to use synched blocks? Because synchronization does hurt concurrency, you don't want to synchronize any more code than is necessary to protect your data. So if the scope of a method is more than needed, you can reduce the scope of the synchronized part to something less than a full method—to just a block. OR when u are using Thread un-safe(un-sunchronized eg -- StringBuilder or HashMap or HashSet) classes in your appln. ----------
Regarding static & non -static synchronized
1. Threads calling non-static synchronized methods in the same class will only block each other if they're invoked using the same instance. That's because they each lock on "this" instance, and if they're called using two different instances, they get two locks, which do not interfere with each other.
2. Threads calling static synchronized methods in the same class will always block each other—they all lock on the same Class instance.
3. A static synchronized method and a non-static synchronized method will not block each other, ever. The static method locks on a Class instance(java.lang.Class) while the non-static method locks on the "this" instance—these actions do not interfere with each other at all. What is JSP? (Java server pages) Dynamic Web page template(having typically HTML 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 JSP API jsp-api.jar --- /lib Contains JSP API implementation classses. 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 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 : 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) -- avlable to scriptlets,exprs 2.1 out - javax.servlet.jsp.JspWriter : represents the buffered writer stream connected to the clnt via HttpServletResponse(similar to your PW 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 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 . (till Javabeans & custom tags or JSTL, we will use use the scriptlets to add : Req. processing logic, B.L & P.L) syntax : <% java code..... %> location inside the translated page : within _jspService usage : till JBs 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") %>
<%= 1234456 %>
<%= session.getAttribute("user_dtls") %>
<%= session.setAttribute("nm",val) %>
<%= session.getId() %>
Better alternative to JSP Expressions : EL syntax (Expression Lang. : avlble from JSP 1.2 onwards)
syntax : ${expr to evaluate} (to be added directly in body tag)
EL syntax will evaluate the expr ---toString --sends it clnt browser.
JSP implicit object --- request,response,session....---accessible from scriptlets & JSP exprs. --- accessible to scriptlets/exprs
EL implicit objects --- can be accessible only via EL syntax
param = map of request parameters
pageScope=map of 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
---avlable ONLY to EL syntax ${...}
---to be added directly within ...
eg : ${param.user_nm} ---to string ---clnt browser
request.getParameter("user_nm") ---to string --sent to clnt browser.
${requestScope.abc} ---request.getAttribute("abc") ---to string --sent to clnt browser.
eg : suppose ctx scoped attr --- loan_scheme

${applicationScope.loan_scheme} ---
getServletContext().getAttirbute("loan_scheme") ---to string --sent to clnt
${user_dtls} --- null -- blank
${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}
${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} -----pageCotext.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 %>
Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)

1. Also can be used for overriding life cycle methods (jspInit,jspDestroy)

location inside the translated page : outside any of life-cycle meths & within the translated servlet class.
JSP Directives --- commands/messages for JSP Engine(=JSP container=WC) -- to be used @Translation time.
Syntax ---
<%@ Directive name attrList %>

1. page directive
2. --- all commands applicable to current page only.
3. Syntax

4. <%@ page import="comma separated list of pkgs" contentType="text/html" %>
5. eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>
6. Imp page directive attributes
7. import --- comma separated list of pkgs
8. session --- boolean attribute. def=true.
9. To disable session tracking, spectify session="false"
10. errorPage="URI of err handling page" ---
11. tells WC to forward user to err handler page.
12. isErrorPage="true|false" def = false
13. If u 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 }

ERR code : ${pageContext.errorData.statusCode}

ERR Mesg : ${pageContext.exception.message }

Throwable : ${pageContext.errorData.throwable}

Throwable Root cause: ${pageContext.errorData.throwable.cause}

1. isThreadSafe="true|false" default=true. "true" is recommended
2. true=>informing WC--- JSP is already written in thrd -safe manner ---- DONT 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 reco? --- isThreadSafe=true(def.) --- identify critical code--wrap 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 NOT recommended.

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

---

JSP actions ---- commands/mesgs meant for WC
to be interpreted @ translation time & applied @ req. processing time.(run time)
Syntax ---standard actions --implementation classes are present in jsp-api.jar.
Body of the tag/action
JSP Using Java beans
Why Java Beans
---1. allows prog to seperate B.L in JBs.(Req processing logic, Page navigation & resp generation will be still part of JSP)
JBs can store conversational state of clnt(JB 's properties will reflect clnt state) + supplies Business logic methods.

1. simple sharing of JBS across multiple web pages---gives rise to re-usability.
2. Auto. translation between req. params & JB props(string--->primitive data types auto. done by WC)

What is JB?

1. pkged public Java class
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)
4. In proper words --- Java bean props reflect the conversational state of the clnt.
5. per property -- if RW
6. naming conventions of JB
7. supply getter & setter.
8. Rules for setter
9. public void setPropertyName(Type val)
10. Type -- prop type.

11. eg -- private double regAmount;
12. public void setRegAmount(double val)
13. {...}
14. Rules for getter
15. public Type getPropertyName()
16. Type -- prop type.
17. eg -- public double getRegAmount(){...}
18. Business Logic --- methods

## public methods --- no other restrictions

Using Java Beans from JSP Via standard actions

1. W.C invokes JB life-cycle
    a. WC chks if specified Bean inst alrdy exists in specified scope
    b. java api --- request.getAttribute("user")
    c. ---null=>JB doesn't exist
    d. ---loc/load/inst JB class
    e. UserBean u1=new UserBean();
    f. --add JB inst to the specified scope
    g. java api -- request.setAttribute("user",u1);
    h. --- not-null -- WC continues....
    i. JSP using JB action
    j. 2.1
    k. Usage--
    l.
    m. WC invokes --- session.getAttribute("user").setEmail("a@b");
2. value="<%= request.getParameter("f1") %>"/>
3. OR via EL
4. value="${param.f1}"/>
5. WC invokes ---
6. session.getAttribute("user").setEmail(request.getParameter("f1"));
7. 2.2
8. Usage eg --
9. WC invokes ---
10. ((Userbean)request.getAttribute("user")).setEmail(request.getParameter("f1"));
11. 2.3
12. usage
13. eg -- If rq. param names are email & password(i.e matching with JB prop names) then ---matching setters(2) will get called
    a.

    b. Usage --
    c.
    d. WC ---
    e. session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.
14. Better equivalent -- EL syntax
15. ${sessionScope.user.email} ---
16. session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.
17. ${requestScope.user.validUser.email}
18. request.getAttribute("user").getValidUser().getEmail()
19. ${pageContext.exception.message}
20. 4.JSP std actions related to Request Dispatcher
21. RD's forward scenario
22.
23. eg : In one.jsp
24.
25. WC invokes ---RD rd=reuqest.getRD("two.jsp");
26. rd.forward(request,response);
27. RD's include scenario
28. Why JSTL ? JSP standard tag library
29. When JSP std actions are in-sufficient to solve B.L
30. w/o writing scriptlets --- use additional std actions --- supplied as JSTL actions
31. JSP standard Tag Library
32. --- has become std part of J2EE 1.5 onwards.
33. ---support exists in form JAR ---
    a. jstl-1.2.jar
    b. For using JSTL steps
    c. 1.Copy above JAR into ur run-time classpath(copy jars either in /lib OR /lib
    d. Use taglib directive to include JSTL tag library into ur JSP pages.
    e. tag=action
    f. tag library=collection of tags
    g. supplier = JSTL vendor(spec vendor=Sun, JAR vendor=Sun/any J2EE compliant web/app server)
    h. jstl.jar --- consist of Tag implementation classes
    i. Tag libr- TLD -- Tag library descriptor -- desc of tags -- how to use tags
    j. <%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>
34. eg --- To import JSTL core lib
35. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
    a. Invoke JSTL tag/action
    b. 3.1 eg

c.
d. ---WC
e. pageContext.setAttribute("abc",request.getParameter("f1"))

36. WC invokes --- session.setAttribute("abc",request.getparameter("f1"));
37. menaing of sets the specified attr to specified scope.
38.
39. WC
40. pageContext.setAttribute("details",session.getAttribute("abc"));
    a.
    b. WC ---request.removeAttribute("abc") ---removes the attr from req scope.
41. 3.2 JB --- ShopBean -- property --
42. private AL categories; --g & s
43. ${cat}
44. WC invokes ---
45. for(Category cat : session.getAttribute("shop").listCategories())
46. out.print(cat);
47. eg :
48. ${acct.acctID} ${acct.type} ${acct.balance}
49. http://localhost:8080/day6_web/close_acct.jsp?acId=101
50. formaction="transactions.jsp" />
51. formaction="transactions.jsp" />
52. <% request.getPrameter("btn").equals("Deposit") --- %> .... in deposit in withdraw http://localhost:8080/day6_web/transactions.jsp?acId=102&amount=500&btn=Deposit WC --- response.sendRedirect(session.getAttribute("my_bank").closeAccount()); ---------- logout.jsp Hello user name log out mesg clean up dao invalidate sess. auto redirect 3.3 JSTL action --- for URL rewriting eg : WC invokes --- pageContext.setAttribute("abc",resp.encodeURL("next.jsp")); Next var -- loop var items -- any JB 's prop --- array based,coll based (List or set) map based. Java syntax for(Category c : categories) out.write(c.getName() 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) Note : when u dont specify form action , its submitted to the same page. Refer to Thread state transitions figure A -- transition from rdy to run -----> Running Triggered by --- in time slice based scheduling --- time slot of earlier thrd over OR in pre-emptive multitasking -- higher prio thrd pre-empts lower prio thrd. B --- transition from running ---> ready to run Reverse of earlier transition OR public static void yield()---- Requests underlying scheduler to swap out current thrd SO THAT some other lower prio

thrd MAY get a chance to run. (to avoid thrd starvation -- i.e co-operative multi-tasking.) C --running state --- Only in this state --- run() method gets executed. running --->dead --- Triggers -- run() method returns in healthy manner . OR run() aborts due to un-handled , un-checked excs. D -- blocked ---> rdy to run --- when any of blocking condition is removed --- blocked thrd enters rdy pool & resumes competition among other thrds. API Involved 1. Thread class constructor to be used in extends Thread scenario 1.1 Thread() --- A new thrd is created BUT with JVM supplied name. 1.2 Thread(String nm) -- creates named thread. 2. Thread class constructor to be used in implments scenario 2.1 Thread (Runnable target/inst) --- Creates a new thrd --- by passing instance of the class which implements Runnable i/f. Run time significance -- Whenever this thrd gets a chance to run --- underlying task scheduler -- will invoke(via JVM) this class's run() method. 2.2 Thread (Runnable inst,String name) public class Myclass extends Thread --- start() Vs public class Myclass implements Runnable --- This class simply represents a runnable task. Prog MUST create a thrd class inst BY attaching Runnable task to it. Thread class API 1.public String getName() 2. public void setName(String nm) 3. public static Thread currentThread() -- rets ref of the invoker thrd. 4. public int getPriority() -- rets current prio. Prio scale -- 1--10(MIN_PRIORITY,MAX_PRIORITY) NORM_PRIORITY ---- 5 4.5 public void setPriority(int prio) --- must be invoked before start() DO NOT rely on priority factor -- since it is ultimately specific to underlying OS prio range.-- may cause loss of portability. t1 --- max-prio ---run() --- begin up-counter -- obs val after 10 secs t1 --- min-prio ---run() --- begin up-counter -- obs val after 10 secs t1 --- max-prio & t2 -- min prio 5. public String toString() --- to ret -- name,prio & thrd grp name 6. public static void sleep(long ms) throws InterruptedException Objective -1. to test concurrency of thrd : in extends thrd scenario. Create a Thrd class, add simple loop with dly in run method to display the exec. sequence. Write main method : which will instantiate thrds(multi-thrded system) & test the concurrent exec. of main thrd along with other thrds. To ensure that main thrd terminates last : no orphan thrds in the system. Thread class Method : 1. public void join() throws InterruptedExc. The invoker thrd gets blocked until the specified thrd joins it(i.e specified thrd becomes dead) eg : 2 thrds t1 & t2 are running concurrently. If u invoke : in run() method of t1 : t2.join() ----> t1 gets blocked until t2 is over. t3 -- t1.interrupt() t1 : Blocked on join Unblcking triggers -- t2 over,getting interrupt signal. 2. public void join(long ms) throws InterruptedExc waits max for specified tmout . Unblocking triggers -- t2 over,getting interrupt signal, tmout exceeded. Objective -- Ensure no orphans , w/o touching child thread class. Objective Replace for loop from thrds, by indefinite loop (while true) & still ensure -- no orphans. Start --- main + 3 child

thrds. --- main(parent thrd waits patiently for 5 secs. & then somehow forces termination of child thrds & then terminates last. API --- public void interrupt() -- sends interrupt signal to the specified thread. If specified thrd has invoked any method (sleep,join,wait)-- having throws clause of InterruptedExc --- then only thrd gets UNBLOCKED due to InterruptedExc. NOTE -- Thread which is blocked on I/O -- CANT be un-blocked by interrupt signal. Threads blocked by invoking any method -- having InterruptedException (sleep,join,wait) can be unblocked by interrupt signal. 2. Objective - to test concurrency of thrds : in implements Runnable scenario. 2.5 implements scenarion 3. Objective : create 3 thrds with 3 random sleep durations(range is 500ms-5sec) & start them conc. & ensure that main terminates last. For random nos--- java.util.Random() , nextInt,nextInt(int upperLim) Objective : create 3 thrds & start them conc. & ensure that main terminates last. 2nd thrd should accept data from console, dont supply data & observe . How to unblock a thread , which is blocked on I/O? Objective : Apply multi threading to Swing application. Create Swing application -- with start & stop buttons, in south region. Create JPanel in center region , with some default color. When start button is clicked, center panel should start changing color(random color) periodically. When stop button is clicked, stop changing color. ----------

53. Objective -- To store emp details , dept wise in SAME data file.(text buffered manner--- PrintWriter ) Design : 0. Emp class --- id,deptid,name,sal 1. Write Utils class d.m --- pw constr --pw inst --- void writeData(Emp e) {...} void cleanUp() {...} constr ---create PW inst -- PW pw=new PW(new FW("emps.data"),true); ---add writeData(....) : instance method to write Emp dtls (first name, last name ,deptId of the emp) to the file with small dly in between.(why dly ? --- for simulating practical scenario & also to add randomness to code) do u need clean up method- -- yes -- close pw. 2. Write Dept Handler runnable task class --- implements Override run() method which will invoke writeData method till 'exit' condition is encountered. Add stop/exit method to enable 'exit' flag. .... { Emp e; Utils u; constr(u,e) {this.e=e; //u=new Utils(); ------ public void run() { while(!exit) u.writeData(e); } 3. Write Tester class : accept some emp dtls . Create depthandler task per dept ,attach thrds & start them. Wait for key stroke --- System.in.read(); stop all child thrds ensure no orphans clean up -- pw Wait for the key stroke : upon key stroke --- stop all child thrds & then finally terminate main. Observed o/p ------garbled display or garbled data written in file. Reason : multiple thrds trying to access the shared resource concurrently. eg of shared resource : Console or file device,socket,DB table Solution : Any time when asynch thrds need to access the SHARED resource : LOCK the shared resource --- so that after locking -- only single thread will be able to access the resource

concurrently When is synchronization(=applying thread safety=locking shared resource) required? In multi-threaded java applns -- iff multiple thrds trying to access SAME copy of the shared resource(eg -- reservation tkt,db table,file or socket or any shared device) & some of the threads are reading n others updating the resource How to lock the resource? Using synchronized methods or synchronized blocks. In either approach : the java code is executed from within the monitor & thus protects the concurrent access. Note : sleeping thrd sleeps inside the monitor(i.e Thread invoking sleep(...) , DOESn't release the ownership of the monitor) eg classes : StringBuilder : thrd-unsafe.--- unsynchronized --- if multiple thrds try to access the same copy of the SB, SB may fail(wrong data) StringBuffer --- thrd -safe ----synchronized internally--- if multiple thrds try to access the same copy of the SB, only 1 thrd can access the SB at any parti. instance. which is reco class in single threaded appln? --- StringBuilder multiple thrds -- having individual copies -- StingBuilder multiple thrds -- sharing same copy -- StringBuilder -- identify code to be guarded -- sb 's api -- invoke thrd unsafe API -- from inside synched block. ArrayList(inherently thrd un-safe) Vs Vector(inherently thrd safe) HashMap(un-safe) Vs Hashtable(thrd safe) synchronized block syntax --- to apply synchro. externally. synchronized (Object to be locked--- shared resource) { //code to be synchronized --methods of shared res. -- thrd safe manner(from within monitor) } 1. If any thrd is accessing any synched method of 1 obj, then same thrd or any other thrd CANT concurrently access same method of the same obj.(Tester1.java) 2. If any thrd is accessing any synched method of 1 obj, then same thrd or any other thrd CANT concurrently access same method or any other synchronized method of the same obj. (Tester2.java) 2. If thrds have their own independent copies of resources, synch IS NOT required.(Tester2.java) 3.If u are using any thrd un-safe code(ie. ready code without source) --& want to apply thrd safety externally --- then just wrap the code within synched block to use locking.(Tester3.java) Objective : Create Producer & Consumer thrds . Producer produces data samples & consumer reads the same. For simplicity : let the data be represented by : single Emp record Producer produces emp rec sequentially & consumer reads the same. Rules : 1 when producer is producing data , consumer
thrd concurrently should not be allowed to read data & vice versa. Any more rules????????????? Yes --- correct sequencing is also necessary in such cases. Rule 2 : Producer must 1st produce data sample ---consumer reads data sample & then producer can produce next data sample. Similarly consumer should not be able to read stale(same) data samples . ITC --- API level Object class API 1. public void wait() throws IE ---thrd MUST be owner of the monitor(i.e invoke wait/notify/notifyAll from within synched block or method) --- othewise MAY get

IllegalMontitorStateExc ---causes blocking of the thrd outside montitor. UnBlocking triggers --- interrupt(not reco --- since it may cause death of thrd) , notify/notifyAll --- reco. 2. 1. public void wait(long ms) throws IE UnBlocking triggers --- interrupt(not reco --- since it may cause death of thrd) , notify/notifyAll --- reco.,tmout exceeded 2.2 public void notify() -- MUST be invoked from within monitor , ow may get IllegalMonitorStateExc Un-blocks ANY waiting thread , blocked on SAME MONITOR 2.3 public void notifyAll() -- Un-blocks ALL waiting threads , blocked on SAME MONITOR notify/notifyAll--- DOESN't BLOCK the thread & Doesn't release lock on monitor. --- send wake up signal -- to thrd/s waiting on same monitor. wait --- Blocks the thread --- Releases lock on the monitor. volatile --- java keyword, applicable at data member. typically used in multi-threaded scenario only when multiple thrds are accessing the same data member. Use --- to specify-- that data var. is being used by multiple thrds concurrently -- so dont apply any optimizations(OR the value of the variable can get modified outside the current thrd) . With volatile keyword -- its guaranteed to give most recent value. The volatile modifier tells the JVM that a thread accessing the variable must always get its own private copy of the variable with the main copy in memory A race condition is a special condition that may occur inside a critical section. A critical section is a section of code that is executed by multiple threads and where the sequence of execution for the threads makes a difference in the result of the concurrent execution of the critical section. When the result of multiple threads executing a critical section may differ depending on the sequence in which the threads execute, the critical section is said to contain a race condition. Race condition means that the threads are racing through the critical section, and that the result of that race impacts the result of executing the critical section. Critical Sections Running more than one thread inside the same application does not by itself cause problems. The problems arise when multiple threads access the same resources. For instance the same memory (variables, arrays, or objects), systems (databases, web services etc.) or files. In fact, problems only arise if one or more of the threads write to these resources. It is safe to let multiple threads read the same resources, as long as the resources do not change. Here is a critical section Java code example that may fail if executed by multiple threads simultaneously: public class Counter { protected long count = 0; public void add(long value){ this.count = this.count + value; } } Imagine if two threads, A and B, are executing the add method on the same instance of the Counter class. There is no way to know when the operating system(scheduler) switches between the two threads. The code in the add() method is not executed as a single atomic instruction by the Java virtual machine. Rather it is executed as a set of smaller instructions, similar to this: Read this.count from memory into PC register. Add value to PC register. Write register to memory. Observe what happens with the

following mixed execution of threads A and B: this.count = 0; A: Reads this.count into a register (0) B: Reads this.count into a register (0) B: Adds value 2 to register B: Writes register value (2) back to memory. this.count now equals 2 A: Adds value 3 to register A: Writes register value (3) back to memory. this.count now equals 3 The two threads wanted to add the values 2 and 3 to the counter. Thus the value should have been 5 after the two threads complete execution. However, since the execution of the two threads is interleaved, the result ends up being different. In the execution sequence example listed above, both threads read the value 0 from memory. Then they add their individual values, 2 and 3, to the value, and write the result back to memory. Instead of 5, the value left in this.count will be the value written by the last thread to write its value. In the above case it is thread A, but it could as well have been thread B. Race Conditions in Critical Sections The code in the add() method in the example earlier contains a critical section. When multiple threads execute this critical section, race conditions occur. More formally, the situation where two threads compete for the same resource, where the sequence in which the resource is accessed is significant, is called race conditions. A code section that leads to race conditions is called a critical section. Preventing Race Conditions To prevent race conditions from occurring you must make sure that the critical section is executed as an atomic instruction. That means that once a single thread is executing it, no other threads can execute it until the first thread has left the critical section.Regarding SERVLET CONFIG A servlet specific configuration object used 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, 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 init ex.TestInitParam name value init /test_init 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. Regarding javax.servlet.ServletContext 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 scoped attributes API public void setAttribute(String nm,Object val) NOTE : Access them always in thread safe manner 5.3 To access global(scope=entire web application) parameters How to add context scoped parameters ? In web.xml name value 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. 5.4 Creating request dispatcher H.W Session Tracking tchnique : HttpSession + URL rewriting 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 . 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 JseesionID 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 Revise server pull --refer to "page navigation readme" servlet life cycle complete -- refer to diag. Servlet config vs context -- refer to readmes init params ---------- URL rewriting -- refer to readme (refer to readme jsp) Enter JSP 1. What is it ? 2. Why JSP ? 3. JSP API 4. JSP Life cycle 5. JSP Tags Thread related API Starting point 1. java.lang.Runnable -- functional i/f SAM -- public void run() Prog MUST override run() -- to supply thread exec. logic. 2. java.lang.Thread --class -- imple . Runnable It has imple. run() -- blank method. 3. Constrs of Thread class in "extends" scenario 3.1 Thread() -- Creates a new un-named thrd. 3.2 Thread(String name) -- Creates a new named thrd. 4. Constrs of Thread class in "implements" scenario 4.1 Thread(Runnable instance) -- Creates a new un-named thrd. 4.2 Thread(Runnable instance,String name) -- Creates a new named thrd. Runnable instance ---instance of the class that implements Runnable i/f(or ano inner class or lambda expression) Methods of Thread class 1. public void start() -- To cause transition from NEW -- RUNNABLE throws IllegalThreadStateException -- if thrd is alrdy runnable or dead. 2. public static void yield() -- Requests the underlying native scheduler to release CPU for current thread , so that curnt thread enters rdy pool.(It's a request) Use case -- co operative multi tasking(to allow lesser prio thrds to access processor) Doesn't release the locks. 3. public void setName(String nm) 4. public String getName() 5. Priority scale -- 1---10 Thread

class consts --MIN_PRIO=1 , MAX_PRIO=10 , NORM_PRIO =5 public void setPriority(int prio) 6. public static Thread currentThread() -- rets invoker(current) thrd ref. 7. public String toString() -- Overrides Object class method , to ret Thread name,priority,name of thrd grp. 8.public static void sleep(long ms) throws InterruptedException --Blocks the invoker thread for specified ms delay. (TIMED_WAITING) 8.5 public boolean isAlive() 9. public void join() throws InterruptedException --Causes the invoker thread to block till specified thread gets over. 10 public void join(long ms) throws InterruptedException --Causes the invoker thread to block till specified thread gets over OR tmout elapsed 11. public void interrupt() -- interrupts(un blocks ) the threads blocked on ---sleep/join/wait 1. Test concurrency (concurrent exec of multiple thrds) using extends Thread. 1.1 Create a class ---extends Thread 1.2 constr --name 1.3 --override run() --- delay loop 1.4 Tester ---main --main thrd --- to create child thrds --start & confirm asynch nature. 2. Test concurrency (concurrent exec of multiple thrds) using imple. Runnable -- separate class / ano inner class /lambda expression 3. What is JSP? (Java server pages) Dynamic Web page template(having typically HTML 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 JSP API jsp-api.jar --- /lib Contains JSP API implementation classses. 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 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 : 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) -- avlable to scriptlets,exprs 2.1 out - javax.servlet.jsp.JspWriter : represents the buffered writer stream connected to the clnt via HttpServletResponse(similar to your PW 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 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 . (till Javabeans & custom tags or JSTL, we will use use the scriptlets to add : Req. processing logic, B.L & P.L) syntax : <% java code..... %> location inside the translated page : within _jspService usage : till JBs 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() %>

53. expr to evaluate : java method invocation which rets a value OR
54. const expr or attributes(getAttribute) or variables(instance vars or method local)
55. location inside the translated page : within _jspService
56. significance : the expr gets evaluated---> to string -> automatically sent to clnt browser.
57. eg <%= new Date() %>

58. eg <%= request.getAttribute("user_dtls") %>
59. <%= 1234456 %>
60. <%= session.getAttribute("user_dtls") %>
61. <%= session.setAttribute("nm",val) %>
62. <%= session.getId() %>
63. Better alternative to JSP Expressions : EL syntax (Expression Lang. : avlble from JSP 1.2 onwards)
64. syntax : ${expr to evaluate} (to be added directly in body tag)
65. EL syntax will evaluate the expr ---toString --sends it clnt browser.
66. JSP implicit object --- request,response,session....---accessible from scriptlets & JSP exprs. --- accessible to scriptlets/exprs
67. EL implicit objects --- can be accessible only via EL syntax
68. param = map of request parameters
69. pageScope=map of page scoped attrs
70. requestScope=map of request scoped attrs
71. sessionScope=map of session scoped attrs
72. applicationScope=map of application(=context) scoped attrs
73. pageContext --- instance of PageContext's sub class
74. ---avlable ONLY to EL syntax ${...}
75. ---to be added directly within ...
76. eg : ${param.user_nm} ---to string ---clnt browser
77. request.getParameter("user_nm") ---to string --sent to clnt browser.
78. ${requestScope.abc} ---request.getAttribute("abc") ---to string --sent to clnt browser.
79. eg : suppose ctx scoped attr --- loan_scheme
80. ${applicationScope.loan_scheme} --- getServletContext().getAttirbute("loan_scheme") ---to string --sent to clnt
81. ${user_dtls} --- null -- blank
82. ${abc} ---
83. pageContext.getAttribute("abc") ---not null -- to string -clnt
84. null
85. --request.getAttribute("abc") -- not null -- to string -clnt
86. null
87. session.getAttribute("abc") ---
88. null
89. getServletContext().getAttirbute("abc") --not null -- to string -clnt
90. null ---BLANK to clnt browser.
91. eg : ${sessionScope.nm}
92. ${pageContext.session.id}

93. --pageContext.getSession().getId() --- val of JessionId cookie w/o java code.
94. ${pageContext.request.contextPath} ---/day5_web
${pageContext.session.maxInactiveInterval}
1. ${param}
2. {user_nm=asdf, user_pass=123456}
3. eg : ${param.f1} ---> request.getParameter("f1").toString()---> sent to browser
4. param ----map of req parameters.
5. param : req. param map
6. ${requestScope.abc} ----- out.print(request.getAttribute("abc").toString())
7. ${abc} -----pageCotext.getAttribute("abc")----null ---request ---session--- application ---null ---EL prints blank.
8. 3.3 JSP declarations (private members of the translated servlet class)
9. syntax : <%! JSP declaration block %>
10. Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)
    a. Also can be used for overriding life cycle methods (jspInit,jspDestroy)

location inside the translated page : outside any of life-cycle meths & within the translated servlet class.

1. JSP Directives --- commands/messages for JSP Engine(=JSP container=WC) -- to be used @Translation time.
2. Syntax ---
3. <%@ Directive name attrList %>
    a. page directive
    b. --- all commands applicable to current page only.
    c. Syntax
    d. <%@ page import="comma separated list of pkgs" contentType="text/html" %>
    e. eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>
    f. Imp page directive attributes
    g. import --- comma separated list of pkgs
    h. session --- boolean attribute. def=true.
    i. To disable session tracking, spectify session="false"
    j. errorPage="URI of err handling page" ---
    k. tells WC to forward user to err handler page.
    l. isErrorPage="true|false" def = false
    m. If u enable this to true--- one can access 'exception' implicit object from this page.

4. This exception obj is stored under current page ---i.e under pageContext (type=javax.servlet.jsp.PageContext -- class which represents curnt JSP)
5. EL expresssion to display error mesg
6. ${pageContext.exception.message}
7. -- evals to pageContext.getException().getMessage()
8. Additional EL syntax
9. EL syntax to be used in error handling pages
10. ERR causing URI : ${pageContext.errorData.requestURI }
11.
12. ERR code : ${pageContext.errorData.statusCode}
13.
14. ERR Mesg : ${pageContext.exception.message}
15.
16. Throwable : ${pageContext.errorData.throwable}
17.
18. Throwable Root cause: ${pageContext.errorData.throwable.cause}
    a. isThreadSafe="true|false" default=true. "true" is recommended
    b. true=>informing WC--- JSP is already written in thrd -safe manner ---- DONT apply thrd safety.
19. false=>informing WC --- apply thrd safety.
20. (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.
21. What is reco? --- isThreadSafe=true(def.) --- identify critical code--wrap it in synchronized block.
22. eg ---Context scoped attrs are inherently thrd -un safe. So access them always from within synched block.
23. Equivalent step in Servlet
24. Servlet class can imple. tag i/f -- javax.servlet.SingleThreadModel(DEPRECATED) -- WC ensures only 1thread (representing clnt request) can invoke service method. --NOT NOT recommended.
    a. include directive
    b. <%@ include file="URI of the page to be included" %>
    c. Via include directive ---- contents are included @ Translation time.--- indicates page scope(continuation of the same page).
    d. Typically used -- for including static content (can be used to include dyn conts)
    e. eg ---one.jsp

f. ....<%@ include file="two.jsp" %>
g. two.jsp.....

## JSP actions ---- commands/mesgs meant for WC

1. to be interpreted @ translation time & applied @ req. processing time.(run time)
2. Syntax ---standard actions --implementation classes are present in jsp-api.jar.
3. Body of the tag/action
4. JSP Using Java beans
5. Why Java Beans
6. ---1. allows prog to seperate B.L in JBs.(Req processing logic, Page navigation & resp generation will be still part of JSP)
7. JBs can store conversational state of clnt(JB 's properties will reflect clnt state) + supplies Business logic methods.
   a. simple sharing of JBS across multiple web pages---gives rise to re-usability.
   b. Auto. translation between req. params & JB props(string--->primitive data types auto. done by WC)
8. What is JB?
   a. pkged public Java class
   b. Must have def constr.(MUST in JSP using JB scenario)
   c. 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)
   d. In proper words --- Java bean props reflect the conversational state of the clnt.
   e. per property -- if RW
   f. naming conventions of JB
   g. supply getter & setter.
   h. Rules for setter
   i. public void setPropertyName(Type val)
   j. Type -- prop type.
   k. eg -- private double regAmount;
   l. public void setRegAmount(double val)
   m. {...}
   n. Rules for getter
   o. public Type getPropertyName()
   p. Type -- prop type.
   q. eg -- public double getRegAmount(){...}
   r. Business Logic --- methods

## public methods --- no other restrictions

Using Java Beans from JSP Via standard actions

a. W.C invokes JB life-cycle
    i. WC chks if specified Bean inst alrdy exists in specified scope
    ii. java api --- request.getAttribute("user")
    iii. ---null=>JB doesn't exist
    iv. ---loc/load/inst JB class
    v. UserBean u1=new UserBean();
    vi. --add JB inst to the specified scope
    vii. java api -- request.setAttribute("user",u1);
    viii. --- not-null -- WC continues....
    ix. JSP using JB action
    x. 2.1
    xi. Usage--
    xii.
    xiii. WC invokes --- session.getAttribute("user").setEmail("a@b");
b. value="<%= request.getParameter("f1") %>"/>
c. OR via EL
d. value="${param.f1}"/>
e. WC invokes ---
f. session.getAttribute("user").setEmail(request.getParameter("f1"));
g. 2.2
h. Usage eg --
i. WC invokes ---
j. ((Userbean)request.getAttribute("user")).setEmail(request.getParameter("f1"));
k. 2.3
l. usage
m. eg -- If rq. param names are email & password(i.e matching with JB prop names) then ---matching setters(2) will get called
    i.
    ii. Usage --
    iii.
    iv. WC ---
    v. session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.
n. Better equivalent -- EL syntax
o. ${sessionScope.user.email} ---
p. session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.
q. ${requestScope.user.validUser.email}
r. request.getAttribute("user").getValidUser().getEmail()
s. ${pageContext.exception.message}

t. 4.JSP std actions related to Request Dispatcher
u. RD's forward scenario
v.
w. eg : In one.jsp
x.
y. WC invokes ---RD rd=reuqest.getRD("two.jsp");
z. rd.forward(request,response);

## RD's include scenario

a. Why JSTL ? JSP standard tag library
b. When JSP std actions are in-sufficient to solve B.L
c. w/o writing scriptlets --- use additional std actions --- supplied as JSTL actions
d. JSP standard Tag Library
e. --- has become std part of J2EE 1.5 onwards.
f. ---support exists in form JAR ---
    i. jstl-1.2.jar
    ii. For using JSTL steps
    iii. 1.Copy above JAR into ur run-time classpath(copy jars either in /lib OR /lib
    iv. Use taglib directive to include JSTL tag library into ur JSP pages.
    v. tag=action
    vi. tag library=collection of tags
    vii. supplier = JSTL vendor(spec vendor=Sun, JAR vendor=Sun/any J2EE compliant web/app server)
    viii. jstl.jar --- consist of Tag implementation classes
    ix. Tag libr- TLD -- Tag library descriptor -- desc of tags -- how to use tags
    x. <%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>
g. eg --- To import JSTL core lib
h. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
    i. Invoke JSTL tag/action
    ii. 3.1 eg
    iii.
    iv. ---WC
    v. pageContext.setAttribute("abc",request.getParameter("f1"))
i. WC invokes --- session.setAttribute("abc",request.getparameter("f1"));
j. menaing of sets the specified attr to specified scope.
k.
l. WC
m. pageContext.setAttribute("details",session.getAttribute("abc"));

i.
ii. WC ---request.removeAttribute("abc") ---removes the attr from req scope.

n. 3.2 JB --- ShopBean -- property --
o. private AL categories; --g & s
p. ${cat}
q. WC invokes ---
r. for(Category cat : session.getAttribute("shop").listCategories())
s. out.print(cat);
t. eg :
u. ${acct.acctID} ${acct.type} ${acct.balance}
v. http://localhost:8080/day6_web/close_acct.jsp?acId=101
w. formaction="transactions.jsp" />
x. formaction="transactions.jsp" />
y. <% request.getPrameter("btn").equals("Deposit") --- %> .... in deposit in withdraw http://localhost:8080/day6_web/transactions.jsp?acId=102&amount=500&btn=Deposit WC --- response.sendRedirect(session.getAttribute("my_bank").closeAccount()); ---------- logout.jsp Hello user name log out mesg clean up dao invalidate sess. auto redirect 3.3 JSTL action --- for URL rewriting eg : WC invokes --- pageContext.setAttribute("abc",resp.encodeURL("next.jsp")); Next var -- loop var items -- any JB 's prop --- array based,coll based (List or set) map based. Java syntax for(Category c : categories) out.write(c.getName() 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) Note : when u dont specify form action , its submitted to the same page. why JSP ? 1. separation between P.L -- JSP (view layer) B.L -- Java Bean / custom tag 2. Dev cycle easier WC -- auto translate , compile , deploy 3. RAD tools JSP API -- J2EE spec -- Oracle/Sun Servlet i/f ---JspPage (jspInit,jspDestroy -- can be overridden) -- HttpJspPage (_jspService(rq,rs) throws SE,IOExc --DON'T override --body content --auto translated by WC into _jspService) --Imple left to J2EE server vendors Apache -- tomcat --lib --jsp-api.jar ---jasper.jar JSP life cycle clnt sends 1st req to index.jsp WC --- 1. translation phase -- once index.jsp ---- index_jsp.java(only for Tomcat) --translated servlet page 2. compilation phase --once index_jsp.class (folder -- server specific folder --eg : work) 3. loc-load-instantiate(=creates an obj) using def constr 4. WC create ServletConfig -- populates init-params(if any) 5. WC invokes --- public void jspInit ----------

z. init seq over WC -- thrd pool @WC start up (@server start up) server.xml -- reads thrd pool config (init size,max pool size...) thrd pool = collection of thrds clnt sends req --- pools out ANY thrd --run --_jspService -->run ---> --thrd simply rets to pool --scalable ---req processing phase/run time phase ---------- @server shut down/un-dep/re-dep ---- jspDestroy ---GC of translated page class inst(page) --end ---------- ${param.em} WC -- out.write(request.getParameter("em")) ${sessionScope.user_dtls} out.write(session.getAttribute("user_dtls").toString()) ${pageContext.session.invalidate()} WC -- pageContext.getSession().invalidate() How to display session id using EL syntax ? ${pageContext.session.id} WC -- pageContext.getSession().getId() -- sent to clnt How to display session expiration time using EL syntax ? ${pageContext.session.maxInactiveInterval} WC -- pageContext.getSession().getMaxInactiveInterval() --sent clnt How to display context path (web app name ) using EL syntax ? ${pageContext.request.contextPath} WC --- pageContext.getRequest().getContextPath() --sent clnt ---------- JSP standard actions for managing Java Bean 1. def value of scope=page eg : What will WC call ? refer to diag. 2. eg : What will WC invoke ? Invokes ALL MATCHING setters of JB Request param names MUST match with JB property setters. eg : email --- setEmail .... 3. How to invoke B.L method of JB ? --via EL syntax. eg :${sessionScope.shop.authenticateCustomer()} WC --- session.getAttribute("shop").authenticateCustomer() --sent to clnt 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 address; eg : collection of basic type @ElementCollection @CollectionTable(name="Contacts", joinColumns=@JoinColumn(name="ID")) @Column(name="CONTACT_NO") private Collection contacts;The two rules, for bidirectional one-to-one associations 1 The @JoinColumn annotation goes on the mapping of the entity that is mapped to the table containing the join column, or the owner of the relationship. This might be on either side of the association. 2 The mappedBy element should be specified in the @OneToOne annotation in the entity that does not define a join column, or the inverse side

of the relationship. Illegal to have a bidirectional association that had mappedBy on both sides. Incorrect not have it on either side. Hibernate will assume each side was the owner and each will have a join column. ---------- When an entity is associated with a Collection of other entities, it is in form of a one-to-many mapping. When a relationship is bidirectional, there are actually two mappings, one for each direction. A bidirectional one-to-many relationship always implies a many-to-one mapping back to the source. eg : Course & Students In this , there is a one-to-many mapping from Course -----> Student and a many-to-one mapping from Student -----> Course. Which table must have a foreign key ? When a Course entity has an number of Student entities stored in its collection, there is no definite way to store those references in the database table. Instead Student table MUST have foreign keys back to the Course So the one-to-many association is almost always bidirectional and never the owning side. In Course entity u need to map the Students with @OneToMany annotation. This doesn't have foreign key , so its an inverse side of relationship. Since this is the inverse side of the relationship, MUST include the mappedBy attribute. eg : In Course Entity mappedBy -- must refer to the prop name in the associated table --to specify ownership of the asso. @OneToMany(cascade = CascadeType.ALL,mappedBy = "course") students getter. (getStudents) ---mappedBy tells hibernate that instead of having a separate join table --map students by course i.e course = name of the property appearing in Student class , anno by @ManyToOne In Student Entity @ManyToOne @JoinColumn(name = "course_id") getter for course Rules 1. The many-to-one side is the owning side, so the join column is defined on that side. 2. The one-to-many mapping is the inverse side, so the mappedBy element must be used. ---------- Collection of 3 types --- entities , embeddables & basic types. In Course 1<----->n Student the Course entity has a collection of Student instances. It is called a multivalued relationship. BUT collections of embeddable and basic types are not relationships; they are simply collections of elements . They are called element collections. Relationships define associations between independent entities , whereas element collections contain objects that are dependent upon the referencing entity, and can be retrieved only through the entity that contains them. Annotation Used -- @ElementCollection --mandatory BUT then hibernate creates its own table to store these embed dables If u want to name the table , optional anno is @CollectionTable @ElementCollection @CollectionTable(name = "table_name", joinColumns = @JoinColumn(name = "join_col_name")) Followed by getter of embeddables or basic types. Hibernate 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 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 Hibernate Caching Caching is a facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate achieves this by implementing first level cache. First level cache in hibernate is enabled by default and you do not need to do anything to get this functionality working. In fact, you can not disable it even forcefully.(typically) Its associated with Session object. Session object is created on demand from session factory and it is lost, once the session is closed. Similarly, first level cache associated with session object is available only till session object is live. It is available to session object only and is not accessible to any other session object in any other part of application. NOTE 1. First level cache is associated with session object and other session objects in application can not see it. 2. The scope of cache objects is of session. Once session is closed, cached objects are gone forever. 3. First level cache is enabled by default and you can not disable it. 4. When we query an entity first time, it is retrieved from database and stored in first level cache associated with hibernate session. 5. If we query same object again with same session object, it will be loaded from cache and no sql query will be executed. (completely true only for get or load) 6. The loaded entity can be removed from session using evict(Object ref) method. The next loading of this entity will again make a database call if it has been removed using evict() method. 7. The whole session cache can be removed using clear() method. It will remove all the entities

stored in cache. Hibernate Caching Caching is a facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate achieves this by implementing first level cache. First level cache in hibernate is enabled by default and you do not need to do anything to get this functionality working. Typically can not be disabled. Its associated with Session object. Session object is created on demand from session factory and it is lost, once the session is closed. Similarly, first level cache associated with session object is available only till session object is live. It is available to session object only and is not accessible to any other session object in any other part of application. NOTE 1. First level cache is associated with session object and other session objects in application can not see it. 2. The scope of cache objects is of session. Once session is closed, cached objects are gone forever. 3. First level cache is enabled by default and you can not disable it. 4. When we query an entity first time, it is retrieved from database and stored in first level cache associated with hibernate session. 5. If we query same object again with same session object, it will be loaded from cache and no sql query will be executed. 6. The loaded entity can be removed from session using evict(Object ref) method. The next loading of this entity will again make a database call if it has been removed using evict() method. 7. The whole session cache can be removed using clear() method. It will remove all the entities stored in cache. ---------- ---------- Caching is facility provided by ORM frameworks which help users to get fast running web application, while help framework itself to reduce number of queries made to database in a single transaction. Hibernate also provide this caching functionality, in two layers. Fist level cache: This is enabled by default and works in session scope. Read more about hibernate first level cache. Second level cache: This is apart from first level cache which is available to be used globally in session factory scope. Above statement means, second level cache is created in session factory scope and is available to be used in all sessions which are created using that particular session factory. It also means that once session factory is closed, all cache associated with it die and cache manager also closed down. How second level cache works? 1. Whenever hibernate session try to load an entity, the very first place it look for cached copy of entity in first level cache (associated with particular hibernate session). 2. If cached copy of entity is present in first level cache, it is returned as result of load method. 3.If there is no cached entity in first level cache, then second level cache is looked up for cached entity. 4.If second level cache has cached entity, it is returned as result of load method. But, before returning the entity, it is stored in first level cache also so that next invocation to load method for entity will return the entity from first level cache itself, and there will not be need to go to second level cache again. If

entity is not found in first level cache and second level cache also, then database query is executed and entity is stored in both cache levels, before returning as response of load() method. Implementation steps : forward approach or top ---down approach 1. create core java project. 2. Add external jars.(hibernate 5 jars + JDBC driver jar) (in a user library) 3. Create hibernate.cfg.xml(copy from help folder & make necessary changes) : location : run-time classpath(Create (source folder)--from IDE & store hibernate.cfg.xml) 4. Hibernate configuration steps (bootstrapping sequence) for creating singleton instance of the Hibernate SF -- in HibernateUtils class.(using static init block) Steps 1. StandardServiceRegistry reg = new StandardServiceRegistryBuilder().configure().build(); 2. SessionFactory sf=new MetadataSources(reg).buildMetadata().buildSessionFactory(); 5. Identify persistence requirements & wrap it in POJO class Features of Hib. based POJO/JPA based Entity 5.1 : public , pkged class, may implement Serializable(not mandatory from hibernate specifications) 5.2 Must provide def. constr.(mandatory) 5.3 can supply optionally paramed constr. 5.4 declare private D.M , non-static ,non-transient---properties of POJO generate getters & setters. 5.5 One property -- must be unique ID for POJO. --- Data type of property MUST be Serializable 6.Describe the mapping(ORM) to the Hibernate frmwork Either Generate the POJO.hbm.xml : OR use hibernate annotations to avoid writing HBM document eg : BookPOJO Annotation support from pkg : javax.persistence (JPA) Help for annotations : 1. Mark the entity or POJO class with @Entity annotation @Entity : class level ---mandatory 2. @Table(name="table name") : class level annotation ---optional 3. @Id : can be field level OR getter method level. ---mandatory 4. optional @GeneratedValue(strategy=GenerationType.AUTO) --> id generator supplied by persistence provider(app srvr's or ORM frmwork i.e hibernate) & not by DB Mandatory Rule for Identifier property type---must be Serializable 5. @Column(name="col name ") --> not mandatory if same names @Column(columnDefinition= "double(10,2)") private double price; 3,4,5 : applicable to Entity id property 6.@Column(name="upper-cased col name") : for rest of prop to col names mapping(optional) eg : for additional @Column annotations (method level annotation) @Entity public class Flight implements Serializable { @Column(updatable = false, name = "flight_name", nullable = false, length=50) public String getName() { ... } 6.IMPORTANT --- Add POJO (mapping) entry in hibernate.cfg.xml NOTE : Annotations like --@Id,@GeneratedValue,@Column,@ElementCollection,@Temporal --can either be applied @ field level(before data member) or property level(before getter) @Temporal --- can be applied to --java.util.Date,Calendar, GregorianCalendar 7. Create Hibernate based DAO Layer. DAO Layer --- No constructor or clean up required. Directly Create CRUD style methods Create Hib sesion.(using

openSession initially) -- typically just before CRUD.(from DAO layer) 7.1 beginTx -- transaction just represents a piece of work. 7.2 Perform CRUD operations using mainly Session API(eg : save/persist/delete/update/HQL) 7.3 if no errs : commit Tx & close the session.--in case of errs , rollback the tx. & close Hib session. Objective 1 - Simplest --- Test hib frmwork -- using create part of 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 (currently) hib frmwork. NOTE :If entity type has a generated identifier, the value is associated to the instance when the save or persist is called. If the identifier is not automatically generated, the application-assigned (usually natural) key value has to be set on the instance before save or persist is called. 2. Hibernate session API -- for data retrieval API (method) of org.hibernate.Session public get(Class 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); BookPOJO b1= (BookPOJO)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 --- HQL --- "from BookPOJO" eg JPQL -- "select b from BookPOJO b" 2.1 Create Query Object --- from Session i/f org.hibernate.query.Query--- i/f Query createQuery(String queryString, Class resultType) eg : Query 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) List getResultList() Execute a SELECT query and return the query results as a generic List. T -- type of POJO / Result eg : hs,tx String jpql="select b from Book b"; try { List l1=hs.createQuery(jpql,Book.class).getResultList(); } Usage --- String hql="select b from BookPOJO b"; List 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 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 API public Query setPrameter(String pName,Object val) List l1 = hibSession.createQuery(hql,Book.class).setParameter("sp_price",user_price).setParameter("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) Objective : dec. price of all books with author=specified author. String hql = "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(hql).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. 5. Delete operations. API of org.hibernate.Session --void delete(Object object) throws HibernateException ---POJO is marked for removal , corresponding row from DB will be deleted after comitting tx & 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 (); API of org.hibernate.query.Query 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 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 lastPage = selectQuery.list(); 5. org.hibernate.query.Qyery API getSingleResult() Executes a SELECT query that returns a single typed 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 Java Persistence query language 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. ---------- Annotations related to relationships(associations) between entities 0. one -----> one : unidirectional relationship. In Customer Entity class --- @OneToOne(cascade=CascadeType.ALL) @JoinColumn(name="addr_id") private Address adr; ---owning side In Address Entity --- no need of additional annotations. 1. one 1 <----> one : bidirectional relationship. @OneToOne(cascade=CascadeType.ALL,mappedBy="cust") private Address adr; ---owning side @OneToOne @JoinColumn(name="cust_id") private Customer cust; 2. one 1 <---->* many : bi-directional At one side : field level annotaion @OneToMany(cascade=CascadeType.ALL,mappedBy="propertyName in many side") NOTE -- cascade is optional attribute. can be skipped . At many side : @ManyToOne @JoinColumn(name="prim key column name of one side") Meaning - Acts as Foreign key column referred from one side eg -- Course 1----* Students Table structure for understanding --- Course table --- course_id(PK),name,start_date,end_date,fees Students table --- id(PK),name,addr,course_id(FK) @Id @GeneratedValue(strategy=GenerationType.AUTO) private int courseId; ..... @OneToMany(cascade=CascadeType.ALL,mappedBy="myCourse") private List students; In Student POJO @ManyToOne @JoinColumn(name="courseId") private Course myCourse; eg One User having multiple Vehicles. In User class @OneToMany(mappedBy="user") private Collection vehicles=new ArrayList<>(); ---

mappedBy attribute tells hibernate that instead of having a separate join table -- map vehicles by user i.e user = name of the property appearing in Vehicle class , anno by @ManyToOne In Vehicle class -- to supply our own name for the join col (o.w it will take def name) @ManyToOne @JoinColumn(name="USER_ID") private User user; Annotation for Date @Temporal(TemporalType.DATE) private Date startDate; Creates date type of column in underlying DB(java.util.Date or java.util.Calendar or GC) Annotation for Time @Temporal(TemporalType.TIME) private Date openingTime; Creates time type of column in underlying DB Annotation for TimeStamp @Temporal(TemporalType.TIMESTAMP) private Date closingDateTime; Creates datetime type of column in underlying DB More details on one ---many The association may be bidirectional. In a bidirectional relationship, only one of the sides has to be the owner: --- the owner is responsible for the association column(s) update. To declare a side as not responsible for the relationship, the attribute mappedBy is used. mappedBy refers to the property name of the association on the owner side. You MUST NOT declare the join column since it has already been declared on the owners side. Some more annotations @Entity @Table(name = "stock", catalog = "scott", uniqueConstraints = { @UniqueConstraint(columnNames = "STOCK_NAME"), @UniqueConstraint(columnNames = "STOCK_CODE") }) For Identity generator @GeneratedValue(strategy=GenerationType.IDENTITY) @Column(name="b_id") private int bookId; ---------- Steps for oracle 1. create a sequence using PL-SQL ---- CREATE SEQUENCE my_seq MINVALUE 1 START WITH 1 INCREMENT BY 1 CACHE 20; 2.In BookPOJO @SequenceGenerator(name="seq_gen",sequenceName="my_seq",allocationSize=1) @Id @GeneratedValue(strategy = GenerationType.SEQUENCEgenerator="seq_gen") @Column(name="book_id") private int bookId; 3. If book_id say has reached 30, & then u drop table & generate next row , it shows 31. If u want to reset it to 1 ---drop sequence my_seq & then it starts with id vals 1..... ---------- LOB handling annotations @Lob private byte[] data; To restart hibernate sequence 1. DROP SEQUENCE hibernate_sequence; 2. CREATE SEQUENCE hibernate_sequence START WITH 1; ---------- For HAS-A relationship eg : Student HAS - A address. What is Session? ---------- 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? ---------- It is a factory 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. Persistent Object 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 Note ------ When the object is in persistent state it will be in synchronization with the matching record i.e if we make any changes to the state of persistent object it will be reflected in the database.(after commiting tx) -- i.e automatic dirty checking will be performed. 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);

What is Hibernate ? 0. Complete solution to the problem of managing persistence in Java. 1. ORM tool.(Object Relational Mapping) used mainly in data access layer or DAO layer. 2. Provides automatic & transperent persistence. 3. JPA(Java Persistence API) implementor JPA vs Hibernat JPA ---part of J2EE specification --vendor --J2EE (sun) Implementation classes -- JAR ---hibenrate core JARs Provides automatic & transparent persistence framework to store & retrieve data from database. Open Source Java based framework founded by Gavin King in 2001, hosted on hibernate.org Currently hosted on sourceforge.net Java Persistence API (JPA) compliant Current version Hibernate 5.x Other popular ORM Frameworks EclipseLink,iBATIS etc. WHY Hibernate? It mediates the applications interaction with a relational database, leaving the developer free to concentrate on the business problem at hand. J2EE developer does not have to use JDBC API & manage data persistence at RDBMS level. No need to go to Table/Query/Column level. One has to bootstrap Hibernate framework , create transient(=not yet persistent) POJOs & then rely entirely on Hibernate frmwork to manage persistence ref : why hibernate readme ---------- Details There is huge mismatch between Object & Relational world. Formally referred as -- Object-Relational Impedance Mismatch' (sometimes called the 'paradigm mismatch) Important Mismatch Points 1. Granularity 2. Sub Types or inheritance n polymorphism 3. Identity 4. Associations 5. Data Navigation Cost of Mismatch 1.SQL queries in Java code 2.Iterating through ResultSet & mapping it to POJOs or entities. 3.SQL Exception handling. 4. Transaction management 5. Caching 6. Connection pooling 7. Boiler plate code Hibernate Frmwork --- popular ORM Tool ---JPA (Java perssitence API) provider Hibernate 4.x --- JPA compliant --- Java persistence API --- Its part of J2EE specifications. ---Is fully JPA compliant BUT it also has additional services / annotations --- specific to Hibernate. Dev MUST add hibernate JARs ---while deploying appln on web server. Need not add JPA provider JARs , while working on appln server. Transparent persistence provider.(As POJOs or Entities are not bound to any Persistence API --- its written completely independent of Persistence Provider.) --Fully supports OOP features --- association,inheritance & polymorphism --can persist object graphs , consisting of asso. objects --caches data which is fetched repeatedly (via L1 & L2 cache) -- thus reduces DB traffic(L1 cache - at session level -- built in. L2 cache - pluggable) (More on caching at end of document) --supports lazy loading -- thus increases DB performance. (Meaning --- Lazy fetchingThe associated object or collection is fetched lazily, when its first accessed. This results in a new request to the database (unless the associated object is cached). Eager fetchingThe associated object or collection is fetched together with the owning object, using an SQL outer join, and no further database request is required. --supports Objectified version of SQL -- HQL --works on objects &

properties --Hibernate usually obtains exactly the right lock level automatically . so developer need not worry about applying Read/Write lock. Some basics 1. Hibernate uses runtime reflection to determine the persistent properties of a class. 2. The objects to be persisted(called as POJO or Entity) are defined in a mapping document or marked with annotations. Either these HBM XML docs or annotations serves to describe the persistent fields and associations, as well as any subclasses or proxies of the persistent object. 3. The mapping documents or annotations are compiled at application startup time and provide the framework with necessary information for a persistent class. 4. What is Hibernate config.? An instance of Hib Configuration allows the application to specify properties and mapping documents to be used at the frmwork start-up. The Configuration : initialization-time object. 5. SessionFactory is created from the compiled collection of mapping documents . The SessionFactory provides the mechanism for managing persistent classes, the Session interface. 6. A web application or Java SE apllication will create a single Configuration, build a single instance of SessionFactory and then instantiate multiple Sessions in threads servicing client requests. SessionFactory : immutable and does not reflect any changes done later to the Configuration. 7. The Session class provides the interface between the persistent data store and the application. The Session interface wraps a JDBC connection, which can be user-managed or controlled by Hibernate. Hibernate Session A Hibernate Session is a set of managed entity instances that exist in a particular data store. Managing an Entity Instances Life Cycle You manage entity instances(or POJOs) by invoking operations on the entity/POJO using EntityManager/Session instance. Entity instances are in one of four states (2 imp aspects of it : its asso. with the hibernate session & sync of its state with the underlying DB) States : new or transient , managed or persistent, detached, removed. New entity instances have no persistent identity and are not yet associated with a hib. session (transient) Managed entity instances have a persistent identity and are associated with a hib. session.(persistent : via save() or saveOrUpdate()) Changes to DB will be done when tx is commited. Detached entity instances have a persistent identity and are not currently associated with a persistence context/Hib session. Removed entity instances have a persistent identity, are associated with a persistent context and are scheduled for removal from the data store.(removed via session.delete(obj)) Introduction to Hibernate Caching While working with Hibernate web applications we will face so many problems in its performance due to database traffic. That too when the database traffic is very heavy . Actually hibernate is well used just because of its high performance only. So some techniques are necessary to maintain its performance. Caching is the best technique to solve this problem. The performance of Hibernate web applications is improved using caching by optimizing the database applications. The cache actually

stores the data already loaded from the database, so that the traffic between our application and the database will be reduced when the application want to access that data again. At maximum the application will work with the data in the cache only. Whenever some another data is needed, the database will be accessed. Because the time needed to access the database is more when compared with the time needed to access the cache. So obviously the access time and traffic will be reduced between the application and the database. Here the cache stores only the data related to current running application. In order to do that, the cache must be cleared time to time whenever the applications are changing. Difference in get & load 1. Both use common API (i.e load or get(Class c,Serializable id)) Ret type = T In get --- if id doesn't exist --- rets null In load --- if id doesn't exist & u are accessing it from within hib session --- throws ObjectNotFoundExc 2. In get --- Hibernate uses eager fetching policy ---- meaning will generate select query always & load the state from DB in persistent POJO ref. --- so even if u access the same from within the session(persistent pojo) or outside (detached) the hib session --- NO EXCEPTION(proxy + state) 3. In load --- Hib uses lazy fetching policy ---- meaning it will , by default NOT generate any select query --- so what u have is ONLY PROXY(wrapper ---with no state loaded from DB) --- on such a proxy --- if u access anything outside the hib session(detached) ---- U WILL GET ---LazyInitializationExc Fix --- 1. Change fetch type --- to eager (NOT AT ALL reco.=> no caching , disabling L1 cache) 2. If u want to access any POJO in detached manner(i.e outside hib session scope) - fire non-id get method from within session & then hib has to load entire state from DB ---NO LazyInitializationExc Session API update Vs merge Both methods transition detached object to persistent state. Update():- if you are sure that the session does not contain an already persistent instance with the same identifier then use update to save the data in hibernate. If session has such object with same id , then it throws --- org.hibernate.NonUniqueObjectException: a different object with the same identifier value was
already associated with the session: Merge():-if you want to save your modificatiions at any time with out knowing about the state of an session then use merge() in hibernate. Lazy fetching (becomes important in relationships or in Load Vs get) When a client requests an entity(eg - Course POJO) and its associated graph of objects(eg -Student POJO) from the database, it isnt usually necessary to retrieve the whole graph of every (indirectly) associated object. You wouldnt want to load the whole database into memory at once; eg: loading a single Category shouldnt trigger the loading of all Items in that category(one-->many) ---------- What is Session? Represents a wrapper around pooled out jdbc connection. ---------- 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 Example save() - Inserting the record get() / load() - Retrieveing the record update() - Updating the record delete() - Deleting the record What is SessionFactory? ---------- It is a factory(provider) of session objects. we use sessionfactory object to create session object It is a heavy weight object, therefore it has to be created only once for an application(typically @ appln start up time) -- typically one per DB per web application. Its immutable --- Once SF is created , changes made to hibernate.cfg.xml will not be auto reflected in SF. What is Configuration Object ? ---------- Configuration object is used to create the SessionFactory object. Object Oriented Representation of Hibernate configuration file and mapping files(or annotations) is nothing but Configuration object. When we call configure() method on configuration object ,hibernate configuration file(hibernate.cfg.xml from run time classpath) and mapping files (or resources) are loaded in the memory. ---------- Why connection pooling? Java applications should use connection pools because : Acquiring a new connection is too expensive Maintaining many idle connections is expensive Creating prepared statements is expensive Hibernate provides basic or primitive connection pool -- useful only for classroom testing. Replace it by 3rd party vendor supplied connection pools(eg Apache or C3P0) for production grade applications. Natural Key Vs Surrogate Key If u have User reg system -- then u have a business rule that --- user email must be distinct. So if u want to make this as a prim key --then user will have to supply this during regsitration. This is called as natural key. Since its value will be user supplied , u cant tell hibernate to generate it for u---i.e cant use @GeneratedValue at all. Where as -- if u say I will reserve user id only for mapping purposes(similar to serial no ), it need not come from user at all & can definitely use hib. to auto generate it for u---this is ur surrogate key & can then use @GeneratedValue. What is JSP? (Java server pages) Dynamic Web page template(having typically HTML 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 JSP API jsp-api.jar --- /lib Contains JSP API implementation classses. 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 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 : 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) -- avlable to scriptlets,exprs 2.1 out - javax.servlet.jsp.JspWriter : represents the buffered writer stream connected to the clnt via HttpServletResponse(similar to your PW 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 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 . (till Javabeans & custom tags or JSTL, we will use use the scriptlets to add : Req. processing logic,

B.L & P.L) syntax : <% java code..... %> location inside the translated page : within _jspService usage : till JBs 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

a. const expr or attributes(getAttribute) or variables(instance vars or method local)
b. location inside the translated page : within _jspService
c. significance : the expr gets evaluated---> to string -> automatically sent to clnt browser.
d. eg <%= new Date() %>
e. eg <%= request.getAttribute("user_dtls") %>
f. <%= 1234456 %>
g. <%= session.getAttribute("user_dtls") %>
h. <%= session.setAttribute("nm",val) %>
i. <%= session.getId() %>
j. Better alternative to JSP Expressions : EL syntax (Expression Lang. : avlble from JSP 1.2 onwards)
k. syntax : ${expr to evaluate} (to be added directly in body tag)
l. EL syntax will evaluate the expr ---toString --sends it clnt browser.
m. JSP implicit object --- request,response,session....---accessible from scriptlets & JSP exprs. --- accessible to scriptlets/exprs
n. EL implicit objects --- can be accessible only via EL syntax
o. param = map of request parameters
p. pageScope=map of page scoped attrs
q. requestScope=map of request scoped attrs
r. sessionScope=map of session scoped attrs
s. applicationScope=map of application(=context) scoped attrs
t. pageContext --- instance of PageContext's sub class
u. ---avlable ONLY to EL syntax ${...}
v. ---to be added directly within ...
w. eg : ${param.user_nm} ---to string ---clnt browser
x. request.getParameter("user_nm") ---to string --sent to clnt browser.
y. ${requestScope.abc} ---request.getAttribute("abc") ---to string --sent to clnt browser.
z. eg : suppose ctx scoped attr --- loan_scheme

${applicationScope.loan_scheme} --- getServletContext().getAttirbute("loan_scheme") ---to string --sent to clnt

a. ${user_dtls} --- null -- blank
b. ${abc} ---
c. pageContext.getAttribute("abc") ---not null -- to string -clnt
d. null
e. --request.getAttribute("abc") -- not null -- to string -clnt
f. null
g. session.getAttribute("abc") ---
h. null
i. getServletContext().getAttirbute("abc") --not null -- to string -clnt
j. null ---BLANK to clnt browser.
k. eg : ${sessionScope.nm}
l. ${pageContext.session.id}
m. --pageContext.getSession().getId() --- val of JessionId cookie w/o java code.
n. ${pageContext.request.contextPath} ---/day5_web
${pageContext.session.maxInactiveInterval}

a. ${param}
b. {user_nm=asdf, user_pass=123456}
c. eg : ${param.f1} ---> request.getParameter("f1").toString()---> sent to browser
d. param ----map of req parameters.
e. param : req. param map
f. ${requestScope.abc} ----- out.print(request.getAttribute("abc").toString())
g. ${abc} -----pageCotext.getAttribute("abc")----null ---request ---session---application ---null ---EL prints blank.
h. 3.3 JSP declarations (private members of the translated servlet class)
i. syntax : <%! JSP declaration block %>
j. Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)
    i. Also can be used for overriding life cycle methods (jspInit,jspDestroy)

location inside the translated page : outside any of life-cycle meths & within the translated servlet class.

a. JSP Directives --- commands/messages for JSP Engine(=JSP container=WC) -- to be used @Translation time.
b. Syntax ---
c. <%@ Directive name attrList %>
    i. page directive
    ii. --- all commands applicable to current page only.
    iii. Syntax
    iv. <%@ page import="comma separated list of pkgs" contentType="text/html" %>

v. eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>
vi. Imp page directive attributes
vii. import --- comma separated list of pkgs
viii. session --- boolean attribute. def=true.
ix. To disable session tracking, spectify session="false"
x. errorPage="URI of err handling page" ---
xi. tells WC to forward user to err handler page.
xii. isErrorPage="true|false" def = false
xiii. If u enable this to true--- one can access 'exception' implicit object from this page.

d. This exception obj is stored under current page ---i.e under pageContext (type=javax.servlet.jsp.PageContext -- class which represents curnt JSP)
e. EL expresssion to display error mesg
f. ${pageContext.exception.message}
g. -- evals to pageContext.getException().getMessage()
h. Additional EL syntax
i. EL syntax to be used in error handling pages
j. ERR causing URI : ${pageContext.errorData.requestURI }
k.
l. ERR code : ${pageContext.errorData.statusCode}
m.
n. ERR Mesg : ${pageContext.exception.message}
o.
p. Throwable : ${pageContext.errorData.throwable}
q.
r. Throwable Root cause: ${pageContext.errorData.throwable.cause}
i. isThreadSafe="true|false" default=true. "true" is recommended
ii. true=>informing WC--- JSP is already written in thrd -safe manner ---- DONT apply thrd safety.

s. false=>informing WC --- apply thrd safety.
t. (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.
u. What is reco? --- isThreadSafe=true(def.) --- identify critical code--wrap it in synchronized block.
v. eg ---Context scoped attrs are inherently thrd -un safe. So access them always from within synched block.
w. Equivalent step in Servlet

x. Servlet class can imple. tag i/f -- javax.servlet.SingleThreadModel(DEPRECATED) -- WC ensures only 1thread (representing clnt request) can invoke service method. --NOT NOT recommended.
   i. include directive
   ii. <%@ include file="URI of the page to be included" %>
   iii. Via include directive ---- contents are included @ Translation time.--- indicates page scope(continuation of the same page).
   iv. Typically used -- for including static content (can be used to include dyn conts)
   v. eg ---one.jsp
   vi. ....<%@ include file="two.jsp" %>
   vii. two.jsp.....

## JSP actions ---- commands/mesgs meant for WC

a. to be interpreted @ translation time & applied @ req. processing time.(run time)
b. Syntax ---standard actions --implementation classes are present in jsp-api.jar.
c. Body of the tag/action
d. JSP Using Java beans
e. Why Java Beans
f. ---1. allows prog to seperate B.L in JBs.(Req processing logic, Page navigation & resp generation will be still part of JSP)
g. JBs can store conversational state of clnt(JB 's properties will reflect clnt state) + supplies Business logic methods.
   i. simple sharing of JBS across multiple web pages---gives rise to re-usability.
   ii. Auto. translation between req. params & JB props(string--->primitive data types auto. done by WC)
h. What is JB?
   i. pkged public Java class
   ii. Must have def constr.(MUST in JSP using JB scenario)
   iii. 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)
   iv. In proper words --- Java bean props reflect the conversational state of the clnt.
   v. per property -- if RW
   vi. naming conventions of JB

vii. supply getter & setter.
viii. Rules for setter
ix. public void setPropertyName(Type val)
x. Type -- prop type.
xi. eg -- private double regAmount;
xii. public void setRegAmount(double val)
xiii. {...}
xiv. Rules for getter
xv. public Type getPropertyName()
xvi. Type -- prop type.
xvii. eg -- public double getRegAmount(){...}
xviii. Business Logic --- methods

## public methods --- no other restrictions

eh. Using Java Beans from JSP Via standard actions
i. W.C invokes JB life-cycle
1. WC chks if specified Bean inst alrdy exists in specified scope
2. java api --- request.getAttribute("user")
3. ---null=>JB doesn't exist
4. ---loc/load/inst JB class
5. UserBean u1=new UserBean();
6. --add JB inst to the specified scope
7. java api -- request.setAttribute("user",u1);
8. --- not-null -- WC continues....
9. JSP using JB action
10. 2.1
11. Usage--
12.
13. WC invokes --- session.getAttribute("user").setEmail("a@b");

value="<%= request.getParameter("f1") %>"/>

i. OR via EL
ii. value="${param.f1}"/>
iii. WC invokes ---
iv. session.getAttribute("user").setEmail(request.getParameter("f1"));
v. 2.2
vi. Usage eg --
vii. WC invokes ---
viii.
((Userbean)request.getAttribute("user")).setEmail(request.getParameter("f1

"));

ix. 2.3
x. usage
xi. eg -- If rq. param names are email & password(i.e matching with JB prop names) then ---matching setters(2) will get called
    1.
    2. Usage --
    3.
    4. WC ---
    5. session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.
xii. Better equivalent -- EL syntax

${sessionScope.user.email} ---

i. session.getAttribute("user").getEmail()--- toString --- sent to clnt browser.
ii. ${requestScope.user.validUser.email}
iii. request.getAttribute("user").getValidUser().getEmail()
iv. ${pageContext.exception.message}
v. 4.JSP std actions related to Request Dispatcher
vi. RD's forward scenario
vii.
viii. eg : In one.jsp
ix.
x. WC invokes ---RD rd=reuqest.getRD("two.jsp");
xi. rd.forward(request,response);
xii. RD's include scenario
xiii. Why JSTL ? JSP standard tag library
xiv. When JSP std actions are in-sufficient to solve B.L
xv. w/o writing scriptlets --- use additional std actions --- supplied as JSTL actions
xvi. JSP standard Tag Library
xvii. --- has become std part of J2EE 1.5 onwards.

---support exists in form JAR ---

1. jstl-1.2.jar
2. For using JSTL steps
3. 1.Copy above JAR into ur run-time classpath(copy jars either in /lib OR /lib
4. Use taglib directive to include JSTL tag library into ur JSP pages.
5. tag=action
6. tag library=collection of tags

7. supplier = JSTL vendor(spec vendor=Sun, JAR vendor=Sun/any J2EE compliant web/app server)
8. jstl.jar --- consist of Tag implementation classes
9. Tag libr- TLD -- Tag library descriptor -- desc of tags -- how to use tags
10. <%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>

i. eg --- To import JSTL core lib
ii. <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
    1. Invoke JSTL tag/action
    2. 3.1 eg
    3.
    4. ---WC
    5. pageContext.setAttribute("abc",request.getParameter("f1"))
iii. WC invokes --- session.setAttribute("abc",request.getparameter("f1"));
iv. menaing of sets the specified attr to specified scope.
v.
vi. WC
vii. pageContext.setAttribute("details",session.getAttribute("abc"));
    1.
    2. WC ---request.removeAttribute("abc") ---removes the attr from req scope.
viii. 3.2 JB --- ShopBean -- property --
ix. private AL categories; --g & s
x. ${cat}
xi. WC invokes ---
xii. for(Category cat : session.getAttribute("shop").listCategories())
xiii. out.print(cat);
xiv. eg :
xv. ${acct.acctID} ${acct.type} ${acct.balance}
xvi. http://localhost:8080/day6_web/close_acct.jsp?acId=101
xvii. formaction="transactions.jsp" />
xviii. formaction="transactions.jsp" />
xix. <% request.getPrameter("btn").equals("Deposit") --- %> .... in deposit in withdraw http://localhost:8080/day6_web/transactions.jsp?acId=102&amount=500&btn=Deposit WC --- response.sendRedirect(session.getAttribute("my_bank").closeAccount()); ---------- logout.jsp Hello user name log out mesg clean up dao invalidate sess. auto redirect 3.3 JSTL action --- for URL rewriting eg : WC invokes --- pageContext.setAttribute("abc",resp.encodeURL("next.jsp")); Next var -- loop var items -- any JB 's prop --- array based,coll based (List

or set) map based. Java syntax for(Category c : categories) out.write(c.getName() 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) Note : when u dont specify form action , its submitted to the same page. Layers involved in JSP ---DB JSP --P.L / request processing /naviagation JSP Actions --commands meant for WC --to be processed @ run time 3 categories --- 1.standard actions(tag) --actions whose imple classes are jsp-api.jar (jasper.jar--tomcat) 2.1 WC --- getServletContext().getAttribute("test").setEmail("abc@gmail.com"); 2.2 WC -- getServletContext().getAttribute("test").setEmail(request.getParameter("em")); 2.3 WC -- getServletContext().getAttribute("test").setEmail(request.getParamter("abc")) 2.4 eg : req param names --f1 , f2 , name JB property seters --- setEmail,setName,setAge WC ---- getServletContext().getAttribute("test").setName(request.getParameter("name")) 3. ${applicationScope.test.testMe()} WC ---- getServletContext().getAttribute("test").testMe() --- to string --sent clnt 4. ${applicationScope.test.profile} WC ---- getServletContext().getAttribute("test").getProfile() --- to string --sent clnt 3. eg : WC -- getServletContext().getAttribute("test").getProfile() -- tostring --sent clnt OR Via EL syntax ${applicationScope.test.profile} JB ---server side attribute JB (id/name) --- attr name (string) JB instance -- attr value scope = page | request | session| application jsp:useBean --- JB's def constr get inovoked / per scope jsp:setProperty property="*" --- JB's all MATCHING setters(req param names MUST match with JB property setters) jsp:getProerty -- JB's matching getter EL syntax -- JB's B.L method/getters. ---------- Enter JSTL (JSP standard tag library) --part of J2EE specs. vendor of spec --Oracle/sun imple left to server vendors --jstl1-2.jar(maven repository) Why ? In the absence of JSTL tag(action) -- JSP prog may need to write a scriptlet & mix up B.L & P.l Steps 1. Copy jstl1-2.jar under /lib 2. Import the JSTL supplied tag library tag library -- collection of tags <%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

1. Use JSTL tags.
2. 3.1
3. eg : In one.jsp
4.

5. WC --- response.sendRedirect(response.encodeRedirectURL("two.jsp"));

3.2

i. Body of for-each
ii. eg : How to generate dyn options to show catgeories ?
iii. ${cat}
iv. WC ---for(String cat : session.getAttribute("shop").getCategories()) out.write("<option value="+cat+.....);

i. Common doubts/queries
ii. page vs pageContext vs pageScope
iii. when to use session / sessionScope
iv. why did i invalidate the session even after keeping shop bean under page scope .....
v. There are many advantages of Hibernate Framework over JDBC
vi. 1) Opensource , Lightweight & DB independent : Hibernate framework is opensource & lightweight.
vii. 2) Fast performance: The performance of hibernate framework is fast because cache is internally used in hibernate framework. There are two types of cache in hibernate framework first level cache and second level cache. First level cache is enabled bydefault.
viii. Third type of cache is --query level cache.(not implicitely enabled)
ix. 3) Database Independent query: HQL (Hibernate Query Language) is the object-oriented version of SQL. It generates the database independent queries. So you don't need to write database specific queries. Before Hibernate, If database is changed for the project, we need to change the SQL query as well that leads to the maintenance problem.
x. 4) Automatic table creation: Hibernate framework provides the facility to create the tables of the database automatically. So there is no need to create tables in the database manually.
xi. 5) Simplifies complex join: To fetch data form multiple tables is easy in hibernate framework.
xii. eg : To display the course names ordered by desc no of participants (many-to-many)
xiii. select c.name from dac_courses c inner join course_studs cs on c.id = cs.
xiv. c_id inner join dac_students s on cs.s_id = s.stud_id group by c.id order by count*() desc;*
xv. *JPQL -- select c from Course c join fetch c.students group by c.id order by count()* desc

xvi. 6) Provides query statistics and database status: Hibernate supports Query cache and provide statistics about query and database status.
    1. Hibernate translates checked SQLException to un checked org.hibernate.HibernateException(super cls of all hibernate related errs) ---so that prog doesn't have to handle excs.

i. Advantages of hibernates:
    1. Hibernate supports Inheritance, Associations, Collections
    2. In hibernate if we save the derived class object, then its base class object will also be stored into the database, it means hibernate supporting inheritance
    3. Hibernate supports relationships like One-To-Many,One-To-One, Many-To-Many-to-Many, Many-To-One
    4. This will also supports collections like List,Set,Map (Only new collections)
    5. In jdbc all exceptions are checked exceptions, so we must write code in try, catch and throws, but in hibernate we only have Un-checked exceptions, so no need to write try, catch, or no need to write throws. Actually in hibernate we have the translator which converts checked to Un-checked ;)
    6. Hibernate has capability to generate primary keys automatically while we are storing the records into database
    7. Hibernate has its own query language, i.e hibernate query language which is database independent
    8. So if we change the database, then also our application will works as HQL is database independent
    9. HQL contains database independent commands
    10. While we are inserting any record, if we don't have any particular table in the database, JDBC will rises an error like "View not exist", and throws exception, but in case of hibernate, if it not found any table in the database this will create the table for us ;)
    11. Hibernate supports caching mechanism by this, the number of round trips between an application and the database will be reduced, by using this caching technique an application performance will be increased automatically.
    12. Hibernate supports annotations, apart from XML
    13. Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.
    14. Getting pagination in hibernate is quite simple.

ii. Vendor assignment status ?

iii. List Vendors
iv. Delete vendor

Add new vendor

i. Enter Hibernate
ii. What is it ?
iii. Why ?
iv. Hibernate architecture
v. Important blocks
vi. Implementation
vii. Adding hibernate support in Java SE (later in to web app)
viii. Steps
    1. Change the perspective to Java (core java)
    2. Create a java project
    3. Create user library containing hibernate jars
    4. window-preferences-user lib --new --add external jars
    5. Add user lib in build path
    6. Create as a new src folder
    7. R click on src --new src folder --
    8. Copy hibernate.cfg.xml under
    9. 1st check list --Create a tester to test bootstrapping of hibernate (creation of SF)
    10. Identify persistence requirements & create a POJO
    11. User --- id, name ,eamil,password,role,regAmount,regDate,userType,image
    12. Create DAO i/f .
    13. Create DAO implementation class.
    14. Create a Tester & test the application.
    15. Save user details
    16. Get user details by id
    17. Display all user details
    18. --replace openSesison by getCurrentSession
    19. Display list of users registered in curnt financial year.
    20. i/p begin date n end date
    21. o/p list of matching user
ix. Query i/f
x. public Query setParameter(String name , Object value) throws HibernateException
xi. Hibernate API
    1. SessionFactory API

2. getCurrentSession vs openSession

xii. public Session openSession() throws HibernateExc

xiii. opens new session from SF,which has to be explicitely closed by prog.

xiv. public Session getCurrentSession() throws HibernateExc

xv. Opens new session , if one doesn't exist , otherwise continues with the exisitng one.

xvi. 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)
2. API (method) of org.hibernate.Session
3. public Serializable save(Object o) throws HibernateException

xvii. I/P ---transient POJO ref.

xviii. save() method auto persists transient POJO on the DB(upon committing tx) & returns unique serializable ID generated by (currently) hib frmwork.

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

xix. Usage of Hibernate Session API's get()

xx. int id=101;

xxi. BookPOJO b1=hibSession.get(BookPOJO.class,id);

xxii. BookPOJO b1= (BookPOJO)hibSession.get(Class.forName("pojos.BookPOJO"),id);

Display all books info :

i. 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.

ii. (JPA--- Java Persistence API compliant syntax --- JPQL )

iii. eg --- HQL --- "from BookPOJO"

iv. eg JPQL -- "select b from BookPOJO b"

v. 2.1 Create Query Object --- from Session i/f

vi. org.hibernate.query.Query createQuery(String queryString,Class resultType)

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

## 2.2. Execute query to get List of selected PERSISTENT POJOs

i. API of org.hibernate.query.Query i/f
ii. (Taken from javax.persistence.TypedQuery)
iii. List getResultList()
iv. Execute a SELECT query and return the query results as a generic List.
v. T -- type of POJO / Result
vi. eg : hs,tx
vii. String jpql="select b from Book b";
viii. try {
ix. List l1=hs.createQuery(jpql,Book.class).getResultList();
x. }
xi. Usage ---
xii. String hql="select b from BookPOJO b";
xiii. List l1=hibSession.createQuery(hql).getResultList();
    1. Passing IN params to query. & execute it.
    2. Objective : Display all books from specified author , with price < specified price.
    3. API from org.hibernate.query.Query i/f
xiv. Query setParameter(String name,Object value)
xv. Bind a named query parameter using its inferred Type.
xvi. name -- query param name
xvii. value -- param value.l
xviii. String hql="select b from BookPOJO b where b.price < :sp_price and b.author = :sp_auth";
xix. How to set IN params ?
xx. org.hibernate.query.Query API
xxi. public Query setPrameter(String pName,Object val)
xxii. List l1 = hibSession.createQuery(hql,Book.class).setParameter("sp_price",user_price) .setParameter("sp_auth",user_auth).getResultList();
xxiii. Objective --Offer discount on all old books
xxiv. i/p -- date , disc amt
xxv. 4.Updating POJOs --- Can be done either with select followed by update or ONLY with update queries(following is eg of 2nd option--Bulk update scenario)
xxvi. Objective : dec. price of all books with author=specified author.

xxvii. String jpql = "update BookPOJO b set b.price = b.price - :disc where b.author = :au and b.publishDate < :dt ";
xxviii. set named In params
xxix. exec it (executeUpdate) ---
xxx. int updateCount= hs.createQuery(hql).setParameter("disc", disc).setParameter("dt", d1).executeUpdate();
xxxi. ---This approach is typically NOT recommended often, since it bypasses L1 cache . Cascading is not supported. Doesn't support optismistic locking directly.
    1. Delete operations.
    2. API of org.hibernate.Session
    3. --void delete(Object object) throws HibernateException
    4. ---POJO is marked for removal , corresponding row from DB will be deleted after comitting tx & closing of session.
xxxii. OR

## 5.5

i. One can use directly "delete HQL" & perform deletions.(Bulk delete)
ii. eg
iii. int deletedRows = hibSession.createQuery ("delete Subscription s WHERE s.subscriptionDate < :today").setParameter ("today", new Date ()).executeUpdate ();
iv. API of org.hibernate.query.Query
    1. Iterator iterate() throws HibernateException
v. Return the query results as an Iterator. If the query contains multiple results per row, the results are returned Object[].
vi. Entities returned --- in lazy manner
vii. Pagination
    1. Query setMaxResults(int maxResults)
    2. 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)
    4. Set the first row to retrieve. If not set, rows will be retrieved beginnning from row 0. (NOTE row num starts from 0)
viii. eg --- List l1=sess.createQuery("select c from CustomerPOJO c").setFirstResult(30).setMaxResults(10).list();
ix. 4.How to count rows & use it in pagination techniques?
x. int pageSize = 10;
xi. String countQ = "Select count (f.id) from Foo f";

xii. Query countQuery = session.createQuery(countQ);
xiii. Long countResults = (Long) countQuery.uniqueResult();
xiv. 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();
```

## org.hibernate.query.Query API

i. T getSingleResult()
ii. Executes a SELECT query that returns a single typed result.
iii. Returns: Returns a single instance(persistent) that matches the query.
iv. Throws:
v. NoResultException - if there is no result
vi. NonUniqueResultException - if more than one result
vii. IllegalStateException - if called for a Java Persistence query language UPDATE or DELETE statement
    1. How to get Scrollable Result from Query?
viii. ScrollableResults scroll(ScrollMode scrollMode) throws HibernateException
ix. Return the query results as ScrollableResults. The scrollability of the returned results depends upon JDBC driver support for scrollable ResultSets.
x. Then can use methods of ScrollableResults ---first,next,last,scroll(n) .
    1. How to create Named query from Session i/f?
    2. What is a named query ?
    3. 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.
xi. Fail fast: Their syntax is checked when the session factory is created, making the application fail fast in case of an error.
xii. Reusable: They can be accessed and used from several places which increase re-usability.
xiii. eg : In POJO class, at class level , one can declare Named Queries
xiv. @Entity
xv. @NamedQueries

xvi. ({@NamedQuery(name=DepartmentEntity.GET_DEPARTMENT_BY_ID, query="select d from DepartmentEntity d where d.id = :id")}
xvii. public class Department{....}
xviii. Usgae
xix. Department d1 = (Department) session.getNamedQuery(DepartmentEntity.GET_DEPARTMENT_BY_ID).setInteger("id", 1);
    1. How to invoke native sql from hibernate?
    2. Query q=hs.createSQLQuery("select * from books").addEntity(BookPOJO.class);
    3. l1 = q.list();
    4. Hibernate Criteria API
    5. A powerful and elegent alternative to HQL
    6. Well adapted for dynamic search functionalities where complex Hibernate queries have to be generated 'on-the-fly'.
xx. Typical steps are -- Create a criteria for POJO, add restrictions , projections ,add order & then fire query(via list() or uniqueResult())
    1. For composite primary key
    2. Rules on prim key class
    3. Annotation -- @Embeddable (& NOT @Entity)
    4. Must be Serializable.
    5. Must implement hashCode & equals as per general contract.
xxi. In Owning Entity class
xxii. Add usual annotation -- @Id.
xxiii. 1.1 Testing core api
xxiv. persist ---
xxv. public void persist(Object transientRef)
xxvi. ---persists trasient POJO .
xxvii. if u give some non-null id (existing or non-existing) while calling persist(ref) --gives exc
xxviii. org.hibernate.PersistentObjectException: detached entity passed to persist:
xxix. why its taken as detached ? ---non null id.
    1. public Serializable save(Object ref)
    2. save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exc.
    3. Ignores ur passed id & creates its own id & inserts a row.
    4. saveOrUpdate
    5. public void saveOrUpdate(Object ref)

6. --either inserts/updates or throws exc.
7. null id -- fires insert (works as save)
8. non-null BUT existing id -- fires update (works as update)
9. 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

i. merge
ii. public Object merge(Object ref)
iii. I/P -- either transient or detached POJO ref.
iv. O/P --Rets PERSISTENT POJO ref.
v. null id -- fires insert (works as save)
vi. non-null BUT existing id -- fires update (select , update)
vii. non-null BUT non existing id -- no exc thrown --Ignores ur passed id & creates its own id & inserts a row.(select,insert)
    1. get vs load
    2. & LazyInitilalizationException.
    3. update
    4. Session API
    5. public void update(Object object)
    6. Update the persistent instance with the identifier of the given detached instance.
    7. I/P --detached POJO containing updated state.
    8. Same POJO becomes persistent.
viii. Exception associated :
    1. org.hibernate.TransientObjectException: The given object has a null identifier:
    2. i.e while calling update if u give null id. (transient ----X ---persistent via update)
    3. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.
    4. 3.
    5. org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session
    6. public Object merge(Object ref)
    7. Can Transition from transient -->persistent & detached --->persistent.
    8. Regarding Hibernate merge
    9. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().

10. API of Session
11. Object merge(Object object)
12. 3.
13. Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
14. 3.If there is no persistent instance currently associated with the session, it will be loaded.
15. 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.
16. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

ix. 7.public void evict(Object persistentPojoRef)
x. It detaches a particular persistent object
xi. detaches or disassociates from the session level cache(L1 cache)
xii. (Remove this instance from the session cache. Changes to the instance will not be synchronized with the database. )
    1.
    2. void clear()
xiii. When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.
xiv. But Databse Connection is not returned to connection pool.
xv. (Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)
    1. void close()
xvi. When close() is called on session object all
xvii. the persistent objects associated with the session object become detached(l1 cache is cleared) and also closes the Database Connection.
    1. void flush()
xviii. When the object is in persistent state , whatever changes we made to the object
xix. state will be reflected in the databse only at the end of transaction.
xx. BUT If we want to reflect the changes before the end of transaction
xxi. (i.e before commiting the transaction )
xxii. call the flush method.
xxiii. (Flushing is the process of synchronizing the underlying DB state with persistable state of session cache )
    1. boolean contains(Object ref)
xxiv. The method indicates whether the object is

xxv. associated with session or not.(i.e is it a part of l1 cache ?)
xxvi. 12.
xxvii. void refresh(Object ref) -- ref --persistent or detached
xxviii. This method is used to get the latest data from database and make
xxix. corresponding modifications to the persistent object state.
xxx. (Re-reads the state of the given instance from the underlying database

## WHY Spring ?

i. To simplify Java development.
ii. What is spring?
iii. Its a container & a framework both.
iv. Spring is an open source framework since February 2003.
v. Created by Rod Johnson.(currently hosted on pivotal)
vi. One line answer --- spring is not a J2EE specification BUT its created to make developing complex JEE applications easier.
vii. Why learn one more frmwork --- when u already have EJB,Struts,Hibernate etc....
viii. Spring helps you to
ix. 1.Build applications from plain old Java objects (POJOs) (known as spring beans )
    1. Apply enterprise services non-invasively.
    2. (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.
x. Examples of how you, as an application developer, can use the Spring platform advantage:

```
Make a Java method execute in a database transaction without hav
ing to deal with transaction APIs.

Make a local Java method a remote procedure without having to de
al 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.

i. Simple answer to WHY Spring
ii. Spring simplifies Java development.
iii. Since above is a bold stmt -- to justify it --- there are main 4 reasons
iv. 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
v. 3.Declarative prog(XML or anno or java config) + thro' Aspects & common conventions
    1. Boilerplate code reduction thro aspects & templates.
    2. Def. of Spring ----
    3. Spring is a lightweight , dependency injection and aspect-oriented container and framework. works on Ioc(Inversion of control)
vi. Meaning
vii. 7.1 Lightweight Lesser no of JARs
viii. JavaBeans / POJOs in Spring-enabled application often have no dependencies on Spring-specific classes.
ix. 7.2 Dependency Injection Spring allows loose coupling through dependency injection (DI).
x. 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.
xi. You can think of D.I as JNDI(Java naming & directory iterface -- Naming service) in reverse.
xii. 7.3 Aspect-oriented Spring supports aspect-oriented programming(AOP)
xiii. (AOP) allows separating application business logic from system services (such as auditing and transaction management,logging , security,transactions).
xiv. Application objects perform only business logic and nothing more.
xv. They are not responsible for (or even aware of) other system concerns, such as eg : logging , transactions, or security
xvi. 7.4

## Why Spring is a Container ?

i. Spring is a container which manages
ii. the lifecycle and configuration of application objects.(spring beans)
iii. In Spring, using config XMLs or annotations or java config, you can
iv. declare - how to create each of your application objects(spring beans)

v. - how to configure them
vi. - how they should be associated with each other. (collaboration/wiring/coupling=connecting dependencies with dependent objs)

## 7.5 Why Spring is a framework ?

i. 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
ii. Spring also provides you with ready made implemetations of services like - transaction management, persistence framework,web mvc etc.
iii. 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)
iv. 3.Spring is both comprehensive and modular.
v. Offers you lot many features, yet gives you choice to integrate layers one by one, test it & then add new features via new layers.
vi. 4 Spring is an ideal framework for test driven projects.
    1. It is basically a one-stop shop, addressing most of the concerns of typical enterprise applications.
vii. 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.
viii. 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.
2. Created by Rod Johnson and described in his book Expert One-on-One: J2EE Design and Development.

i. Allows us to give capability of EJBs to plain JavaBeans without using an application server.
ii. Any Java SE application can also use Spring to get simplicity, testability, and loose coupling.
iii. 2.Spring has been hosted on SourceForge.

1. 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)
2. 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.
3. Using interfaces in our applications to specify type helps make our applications easier to maintain and enhance later.
4. The Spring framework helps developers for separation of responsibilities.

iv. eg scenario --
v. 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.
vi. 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.
vii. 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.
viii. That s separation of responsibilities. --- This is what spring offers u thro AOP

# 8 Spring framework Modules

1. Advantages of ApplicationContext over BeanFactory
2. 9.1Application contexts resolve text messages, including support for internationalization (I18N).
3. 9.2 Application contexts provide a generic way to load file resources, such as images.
4. 9.3Application contexts can publish events to beans that are registered as listeners.

i. IOC -- rather a generic term
ii. 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.
iii. 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

iv. Dependency. injection=loc+dependency inversion
v. Why loC or Dependency Injection (advantages of loC)

```
* 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",

i. 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
ii. 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).
iii. More on ApplicationContext
iv. The instantiation of the ApplicationContext creates the container that consists of the objects defined in that XML file.
v. The purpose of this XML file is to create the beans and their relationship.
vi. 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.
vii. An API (getBean) is provided to query these beans and use them accordingly from our client application.
viii. More on Ioc
ix. loC is also known as dependency injection (DI).
x. 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.

xi. The container then injects those dependencies when it creates the bean.
xii. 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.
xiii. The org.springframework.beans and org.springframework.context packages are the
xiv. basis for Spring Framework's IoC container. The BeanFactory interface provides an advanced
xv. configuration mechanism capable of managing any type of object. ApplicationContext is a
xvi. 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.
xvii. Revise
xviii. Rules to be followed on hibernate POJO?
xix. public n pkged class
xx. def constr
xxi. per non -static n non transient porperty ---getter & setter
xxii. POJO class need not be Serializable. BUT id property MUST be serilizable (imple Serializable i/f)
xxiii. Can it be primtive type --YES
xxiv. BUT reco to be wrapper type for efficient working.
xxv. eg : Integer/Long

Typical suggestion(thumb rule ) --keep 1 separate property (Integer/Long) ONLY for HBM purpose.

## What is Hibernate ?

i. Why ?
ii. Architecture
iii. Development blocks
iv. hibernate.cfg.xml details
v. HibernateUtils class details
vi. API
vii. openSession vs getCurrentSession
viii. save

ix. get
x. createQuery
xi. L1 cache
xii. POJO state transitions
xiii. Update
    1. Objective : user wants to update password & active status.
    2. i/p --user id , new pwd & sts.
    3. Bulk Update
    4. objective -- reduce reg amount of all active users by specified amount.
xiv. i/p --- discount amt
xv. String jpql="update User u set u.regAmount=u.regAmount-:disc where u.isActive=true";
xvi. int updateCount=hs.createQuery(jpql).setParameter("disc",discount).executeUpdate();
    1. Un subscribe customer by email & password
    2. i/p em,pass
    3. o/p string
    4. Bulk delete users
    5. Objective : delete vendors by city.
    6. i/p --city
    7. o/p mesg
    8. String jpql="delete Vendor v where v.city=:ci";
    9. int delCount=hs.createQuery(jpql).setParameter("ci",city).executeUpdate();
    10. Bulk delete
    11. update(..) API
    12. ----------CRUD complete.....
xvii. display vendor name who has paid max reg amount
    1. Save image to DB
    2. i/p customer id & file name to read
    3. Retrieve image from DB
    4. i/p customer id & file name to store
xviii. Sequence for reference
    1. Diags ---spring_ioc_container & spring-modules-overview
    2. readme_spring --- IoC --- generic term & 4 key points of why spring
    3. eclipse project --- why_spring
    4. for understanding --- scope,lazy-init,setter based D.I , init , destroy method call backs.

5. ref project -- why_spring
6. Ref ---eclipse project spring_basics
7. Sequence
8. com.app.core.atm -- setter based D.I --spring-config.xml
9. com.app.core.atm1 -- constructor based D.I ,meta data -- spring-config1.xml
10. com.app.core.atm2 -- factory method based D.I
11. com.app.core.atm3 -- combo of setter based +constructor based D.I
12. com.app.simple --- auto-wiring byName
13. com.app.simple1 --- auto-wiring byType
14. com.app.simple2 --- auto-wiring constructor
15. com.app.simple6 -- handling collection of values.
16. com.app.simple6_ref -- handling collection of refs + autowire
17. com.app.spel -- demo for Spring Expression language --- similar in capability to OGNL -- meaning can invoke ---getters,setters,static , non static methods.

# Quick Revision

i. eg xml :
ii. --- skipped
iii. --one ctx.getBean(...)
iv. def scope=singleton
v. lazy-init -- def value=false --Meaning --- whenever SC comes across bean def() it will loc/load/inst ....invoke bean life cycle
vi. lazy-init=true --- implies loc/load/inst beans only after clnt demands for it (ctx.getBean(id))
vii. above applicable for scope=singleton
viii. For prototype scope --- bean will get loc/load & instantiated only on demand.
ix. init & destroy methods in spring beans
    1. In Bean class , declare methods
    2. public void anyName() throws Exception(....)
    3. In xml , 2 attributes of tag
    4. init-method --name of init method
    5. destroy-method --name of destroy method
    6. SC will auto invoke init style n destroy style methods
    7. Life cycle
    8. loc--load --inst --setter based D.I --init
    9. B.L

10. destroy style method(only called for singleton beans)
11. GC

x. automatic wiring
xi. wiring = injecting dependecies into dependent objects
xii. auto wiring means ---- bean definition in xml need not contain property tag or construcutor arg tag. This can resolved auto by container.
xiii. Types of auto wiring
xiv. autowire="byName"----SC tries to match bean property by name. In dependent bean class , u must have matching setter & in xml config file -- MUST have --bean id with the matched name.
xv. auto-wired="byType" --- SC tries match bean property by type of the property --- if it comes across multiple beans of the same type --- NoUniqueBean exception is raised.
xvi. eg --- In dependent --- ATMImpl
xvii. private Transport myTransport;---- setter
xviii. Transport imple --- test,soap,http
xix. auto-wired=constructor ---
xx. paramed constr --- byName --- constr arg name MUST match with dependency bean id ----if not found --- byType---
xxi. Spring programming using annotations
    1. To enable annotation support -- add context namespace & add the following
    2. --- To tell SC --to enable annotation support(eg --- AutoWired,PostConstruct,Predestroy,.....)
xxii. 0.5 --- How to specify location(base pkg) of spring beans to SC?
xxiii. ---
xxiv. SC starts searching(scanning) in specified pkgs (including sub-pkgs) ---for classes anno with stereotype anno --- @Component,@Service,@Repository,@Controller
xxv. Basic class level annotations meant for SC
xxvi. Super type
xxvii. @Component --- spring bean class
xxviii. sub - type annotations
xxix. @Controller --- In Web MVC scenario -- for request handling.
xxx. @Service --- Service layer (B.L) + transaction management
xxxi. @Repository --- DAO layer
xxxii. @RestController -- RESTful service provider
    1. @Component --- --- SC interprets it & starts bean life-cycle.
    2. eg ---

3. @Component("abc")
4. public class MyBean {...}
5. xml ---
6. OR
7. @Component
8. public class MyBean
9. xml ---
10. @Controller -- spring web mvc controller
11. @Repository --- DAO layer class
12. @Service --- for service layer beans --- transactions.
13. @Scope(value="singleton|prototype|request|session")--- class level annotaion --- in xml via scope atribute.
14. @Lazy(true|false) ----class level anno -- lazy-init attribute
15. @PostConstruct ---method level anno - init-method ---method level
16. @PreDestroy ---method level anno-- destroy-method --- method level
17. @Required(true|false) --- setter method or paramed constr or field level ---tells SC if depenency is mandatory or optional-- def=true
18. @AutoWired ---setter method or paramed constr or field level
19. eg --- TestTransport imple Transport
20. auto-wired="byType"
21. eg -- field level annotation ---in ATMImpl bean (dependent)
22. @AutoWired
23. private Transport myTransport;
24. Meaning -- no parameterised constr, no setter , no xml containing bean definition is required.
25. SC --- chks for any bean of Transport by type & injects it in ATMImpl
26. @AutoWired
27. @Qualifier("test")
28. private Transport myTransport; ---- auto-wired="byName"
29. ---spring supplied anno.

## OR

i. @Resource(name="advSpellChecker")
ii. private SpellChecker checker; ---- auto-wired="byName"
iii. --J2EE supplied via javax.annotation
iv. SpEL --- spring expression language
v. dynamic expression languge ---spring(3.x) supplied -- to evaluate expressions dynamically.

vi. #{SpEL expression} --- more powerful than JSP EL ---JSP EL allows only getters , where as SpEL allows --- getters,setters,constr invocation, static & non-static method invocations.

1. Change perspective to Java EE
2. Import existing day6.1 in your workspace
3. Add User lib(hibernate lib) in 2 places
4. 2.1 Add it in build path
5. 2.2 Add it in dep assembly.(WEB-INF/lib --- user lib --hib lib)
6. Create new src folder & copy ur own working hibernate.cfg.xml in it & confirm db settings
7. Edit Customer & Book pojos to be managed by hibernate
8. 4.1 Replace int id by Integer & modify s/g
9. 4.2 Replace java.sql.Date by java.util.Date
10. 4.3 add JPA annotations
11. Add POJO mapping entries in hibernate.cfg.xml
12. Remove DBUtils & copy HibernateUtils
13. Modify DAO i/f -- remove throws Exception
14. Edit DAO implementation class --to replace JDBC by hibernate.
15. 8.1 No data members,constr & cleanup
16. 8.2 remove java.sql.* --org.hibernate.*
17. 8.2 Modify CRUD methods.
18. Make necessary changes in javabean & test the web app.

How to ensure creation of SF (i.e loading of entire hibernate frmwork) @ web applicsation start up n how to ensure its closing @ appln stopping time ?

1. Create Context Listener class.
2. How ?
3. Create a class --implements javax.servlet.ServletContextListener i/f & override the methods.
4. Deploy via annotation
5. @WebListener or xml tags in web.xml
6. 
7. F.Q listener cls name

i. Advanced Hibernate
ii. Associtation between entities
iii. one-to-one ---@OneToOne
iv. one-to-many ---@OneToMany
v. many-to-one --@ManyToOne
vi. many-to-many --@ManyToMany

1. one-to-many
2. many-to-one
3. eg : Course management system
4. Course 1<----->* Student

vii. Course POJO --- courseId(Integer/Long) , name (unique) , beginDate,endDate , fees, capacity
- List students

Student --- studentId(Integer/Long) , email(unique),name

i. +Course selectedCourse
ii. Tables (RDBMS)
iii. courses --parent
iv. students --child --FK column should be added.
v. owning side --- where actual FK column appears. (eg : many side -- students)
vi. inverse side(=non owning side) --- courses.
vii. Why hibernate is un-necessarily creating additional link table (in addition to FK column)?
viii. Hibernate doesn't have ANY info regarding which is owning & which is non-owning side of the bi-dir asso.
ix. How to tell hibernate about inverse side of the assoication ?
x. In inverse side(Course)
xi. @OneToMany(mappedBy="name of the property as it appears in the owning entity")
xii. public List getStudents() {...}
xiii. Objective --Launch new course with some students
xiv. Tester--DAO -- POJO --DB
xv. DAO i/f
xvi. String launchNewCourseWithStudents(Course c);

# 1.Create java se project

1. Add external Jars.--spring & hibenrate jars --in user lib
2. Create spring-config.xml --- in --add namespaces --beans & context
3. Add tags -- context:annotation-config,context:component-scan.
4. Create separate hibernate configuration file(under ) & import it in main config file -- add bean,context,p,tx namespaces to hibernate cfg xml file
5. Configuration steps for persistence.

i. 8.1 create database properties file & store it under run-time class path(either directly under src or under source folder) --consists of ---driver class name,db url, user name & password & dialect.

ii. 8.2 Supply location of db properties file using
iii. 8.3 Create data source bean(conn pool) --- using apache supplied cn pool based datasource.
iv. class="org.apache.commons.dbcp2.BasicDataSource"
v. p:driverClassName="${jdbc.driver}" p:url="${jdbc.url}"
vi. p:username="${jdbc.username}"
vii. p:password="${jdbc.password}"
viii. p:initialSize="1" p:maxTotal="2"
ix. destroy-method="close">
x. 8.4 Create Hibernate session factory bean
xi. class="org.springframework.orm.hibernate5.LocalSessionFactoryBean"
xii. p:packagesToScan="com.app.pojos" p:dataSource-ref="dataSource">
xiii. hibernate.dialect=${jdbc.dialect}
xiv. hibernate.format_sql=true
xv. hibernate.show_sql=true
xvi. hibernate.hbm2ddl.auto=update
xvii. 8.5 Declare transaction mgr bean by injecting session factory bean ref into it.
xviii. class="org.springframework.orm.hibernate5.HibernateTransactionManager"
xix. p:sessionFactory-ref="sessionFactory">
xx. 8.6
xxi. To enable annotated transaction supprt(spring container managed txs)
xxii.

## This finishes config steps.

1. Identify persistence requirements -- create annotated pojos or use rev eng to generate it.(class level annotation --- @Entity & @Id -- mandatory)
2. Create DAO layer i/f & add implementation class -- annotate DAO layer with @Repository annotation.

i. Inject (using @AutoWired or @Resource) , hib's session factory into DAO impl bean.
ii. Simply use ready made session factory instance to --- get current session --- use Session api for CRUD(eg --- createQuery,save,update,delete )

1. Create Service layer (optional but recommended --- since u can keep tx management dtls separate from DAO layer)

2. Create service i/f & its implementation class. -- annotated with @Service class level annotation.
3. This layer simply invokes DAO layer methods , by autowiring DAO instance.
4. Additional method level annotation or class level annotation
5. org.springframework.transaction.annotation.Transactional
6. @ can supply --- readonly sts,tx propagation,isolation,timeout,rollback rules etc.
7. Create a Tester ---
8. 12.1 Create ApplicationContext -- ClasspathXmlApplicationContext --- spring-config file
9. 12.2 Get service bean & call B.L method

## 12.3 Close Ctx to release resources.

## ^[a-z0-9_-]{3,15}$ ---- user name validations

i. ((?=.*d)(?=.[a-z])(?=.*[A-Z])(?=.*[@#$%]).{6,20}) ---- password validations
ii. Usage :
iii. @Pattern(regexp="((?=.*d)(?=.[a-z])(?=.*[#@$]).{5,20})")
iv. ^[_A-Za-z0-9-]+(\.[_A-Za-z0-9-]+)*@[A-Za-z0-9]+
v. (\.[A-Za-z0-9]+)*(\.[A-Za-z]{2,})$ --- email
vi. (0?[1-9]|[12][0-9]|3[01])/(0?[1-9]|1[012])/((19|20)\d\d) --- date(dd/mm/yyyy)
vii. Revise
viii. Hibernate one to many & many to one relationship
ix. reate core java project.
    1. Create POJOs for one-many bi-dir association between Author & Book.
    2. Author -- id,name(unique),+List books
    3. Book --id ,title,price,category,rating(1 to 5) , + Author bookAuthor
x. Author --- parent / non owning(inverse) --one
xi. -- id,name(unique),
xii. private List books=new AL<>();
xiii. //getter
xiv. @OneToMany(mappedBy="bkAuthor",cascade=CascadeType.ALL)
xv. public List getBooks() {...}
xvi. //as per Gavin King's suggestion -- add conveneience methods for bi-dir asso directly in POJO (parent)
xvii. public void addBook(Book b)
xviii. {

xix. books.add(b);//parent ----> child
xx. b.setBkAuthor(this); //child --> parent
xxi. }
xxii. public void removeBook(Book b)
xxiii. {
xxiv. books.remove(b);//parent ----> child
xxv. b.setBkAuthor(null); //child --> parent
xxvi. }
xxvii. Book --child / owning ---many
xxviii. id ,title,price,category,rating(1 to 5)
xxix. private Author bkAuthor;
xxx. //getter
xxxi. @ManyToOne
xxxii. @JoinColumn(name="auth_id")
xxxiii. public Author getBkAuthor()
xxxiv. {..}
xxxv. Objective --Display course details

## i/p course name o/p dtls / err mesg

# IMPORTANT

i. Default fetching policy of JPA/Hibernate
ii. one-to-one --eager
iii. one-to-many --lazy
iv. many-to-one --eager
v. many-to-many --lazy
vi. When will hibenrate throw org.hibernate.LazyInitializationException ?
vii. --Any time u are accessing un-fetched data from DB (represented by a proxy) , outside the session scope(=in DETACHED manner) .
viii. Triggers 1. ANY -To -Many association (will contain collection of proxies)
    1. Session's load method (will contain single proxy)
ix. Solution :
    1. Change the def fetch type of one-to- many , from lazy --->Eager
    2. --in POJO layer.
    3. eg :
    4. @OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL,fetch=FetchType.EAGER)
    5. public List getStudents() {

6. return students;
7. }
8. Use case -- size of many is small
9. eg : BankCustomer HAS Accounts
10. Question HAS Options
11. Apply soln in DAO layer
12. Access the size of collection of proxies , in session scope(=PERSISTENT state)
13. Disadvantage of 2nd approach -- multiple select queries(select n+1 problem)

## Soln ---join fetch

Student --HAS A --Address (one-to-one )--bi-dir between entities

i. Student --..... + --inverse
ii. private Address homeAdr;
iii. Address --entity ---owning
iv. id, city,state,country
v. +Student stud;
vi. FK column -- address table
vii. Objective
    1. Student admission with address details
    2. i/p --student dtls + adr details (confirm cascading)
    3. Student already exists , just link address
    4. i/p --student id + adr details

## 3. Cancel admission i/p student id

## Value Types.

i. Student HAS-A Adhar Card(component) --embeddable
ii. Student HAS-A EducationalQualifications (name,yr of passing , % secured) --collection of embeddable comps

Student HAS-A hobbies --collection of basic type of eg : private List hobbies;

i. Enter Spring
ii. Why Spring ?
iii. What is it ?
iv. What is dependency injection ?
v. eg : In Controller class --AdmissionController
vi. add a data member
vii. @AutoWired

viii. private IAdmissionService service;
ix. Why & understand it with examples.
x. Spring bean life cycle.
    1. Thin clnt sends request along with rq. params.(*.htm)
    2. First it reaches Spring's DispatcherServlet which acts like
    3. front controller in MVC model II . It is a common web-application pattern where a single servlet delegates responsibility for a request to other components of an application to perform the actual processing.
    4. The DispatcherServlet's job is to send the request on to a Spring MVC controller(prog supplied)
    5. As typical appln may have several controllers, DispatcherServlet consults Handler mapping to select controller.
    6. The Handler mapping will choose a controller based on rq. URL.
    7. Req thus reaches the controller. Controller may use one or more service layer objs for exec of B.L
xi. The result of B.L needs to be carried back to the user and displayed in the browser. This info. is the model. It has to be sent to JSP(or any other view template) for converting it to HTML like format.
    1. So the controller will package up the model data and the
    2. name of a view into a ModelAndView object. It rets the request+ ModelAndView back to the DispatcherServlet. (M&V
    3. doesn't carry a reference to the actual JSP but only carries a logical result name that will be used to look up the actual view that will produce the resulting HTML.
    4. DispatcherServlet now consults a view resolver to find the actual JSP. & delivers the model data to view JSP
    5. The view will use the model data to render a page that will be sent as dyn resp back to the clnt browser.
    6. object.
xii. The ViewResolver provides a mapping between view names and actual views.
xiii. UrlBasedViewResolver Simple implementation of the ViewResolver interface that
xiv. effects the direct resolution of logical view names to URLs, without
xv. an explicit mapping definition. This is appropriate if your logical
xvi. names match the names of your view resources in a straightforward
xvii. manner, without the need for arbitrary mappings.

1. Create dynamic web project
    1. Create User library --spring-hibernate-rest jars

2. DO NOT add any other library.
3. Add DispatcherServlet entry in web.xml -- to ensure all request pass through central dispatcher servlet.
4. Create spring-servlet.xml under -- To allow D.S to create Web application context using master config xml file.

i. 4.1 Copy earlier entries.(ctx,mvc & view resolver)
1. Create & copy database.properties & hibernate-persistence.xml from
2. What it contains ---
3. 5.1 DataSource bean --- Apache (Connection pool)
4. 5.2 SF bean -- Spring
5. 5.3 Tx Mgr --- Spring
6. 5.4 enabled anno support for Txs(@Transactional)

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

iii. Configuration steps over....
1. Identify persistence requirements & create POJO/Model/DTO.
2. POJO properites --- represent 1. DB cols 2.Request params --i.e clnt's conversational state.
- P.L validation rules --anno.
- class level --@Entity,@Table
- Anno -- field level --- @NotEmpty,@NotNull,@Email....
- Annotation -- prop level(getter) --@Id,@Column....
3. Create DAO layer

I/F -- Dao i/f --- validateCustomer

i. Implementation class ---

ii. dependency --- SessionFactory -- @AutoWired

iii. No need to manage Txs --directly get session from SF & perform CRUD operation.
1. Create Service Layer --i/f & then implementation class
2. @Service & @Transactional --- annotations.
3. Inject dependency of Dao Layer.

iv. 10 Create or copy existing controllers & test the flow.
1. MVC overview
2. Demo ---
3. Create New Dyn web project, create user library ,configure DispatcherServlet in web.xml --- to ensure all or any req coming from clnt will be intercepted by this servlet
4. spring org.springframework.web.servlet.DispatcherServlet 1
5. spring /

6. create master spring configuration file ---- Name ---- servletName-servlet.xml under ---- add beans,context , mvc & p namespaces.
7. 2.1 Add tags -- context:annotation-config,context:component-scan,mvc:annotation-driven(to enable annotated MVC controller support)
8. 2.2 declare view resolver bean --- Can use InternalResourceViewResolver or its super-class --- UrlBasedViewResolver --- Props are same for both beans --- viewClass(choose JSTL view for JSP view templates using JSTL actions), prefix -- typically under web-inf & suffix -- .jsp

v. 2.3 Write Hello Controller
vi. Annotations used --- @Controller(class level) & @RequestMapping(add at method level ---& pass only URL value)
vii. eg -- @RequestMapping("/hello") . Ret type -string --represents logical view name --- will get resolved by viewResolver bean.
viii. 2.4 Add to same controller , one more req handling method -- @RequestMapping("/welcome1") & ret type as ModelAndView.
ix. Using M&V constructor --- set model obj & view name --- test it.
x. 2.5 Add 1 more method --- ret Type String BUT annotated with @ResponseBody

-- tells SC -- not to pass it to view (JSP) layer, but directly add ret val as resp data & commit response.

1. Objective ---In Contact management Utility --- Contact authentication using Login form.
2. Layers used ---
3. 3.1 -- POJO (Contact) --email,password,name,regDate,regAmt
4. 3.2 -- Sevice layer bean -- @Service annotation -- i/f & implmenation class --
5. constr -- create & populate HM with contacts. B.L method ---- validateContact -- i/p -- contact with email & password , o/p --- validated pojo or null
6. 3.3 -- Controller bean -- @Controller annotated methods
7. For displaying form --- create empty model instance
8. API of org.springframework.ui.Model --represents request scoped attribute map
9. How to add attribute--
10. public Model addAttribute(Object attributeValue) -- attr name is derived from attribute value class type eg --- UserPOJO ---userPOJO
11. OR

12. addAttribute(String attributeName,Object attributeValue)

i. After adding model attribute --- return logical name of jsp page --- to navigate user to view layer.
ii. 3.4 Create login page using spring form tag lib & for displaying validation errs add style tag -- under
iii.
iv. eg of spring form tags
v. --- name of the model attr=request scoped attr name(similar to model driven approach in sturts2) -- def name is command
vi. Enter User Email --label as with ui comp
vii. --- path --name of the rq param name
viii. 3.5 Since action is typically not mentioned -- its submitted to same controller
ix. supply different req mapping method --- use method=RequestMethod.POST -- for processing the form.
x. min method args --POJO ref, BindingResult(org.springframework.validation.BindingResult), HttpSession hs -- if validated pojo needs to be stored in session scope.
xi. -- in method --always check first for P.L errors --
xii. API of BindingResult
xiii. boolean hasErrors() -- rets true in case of P.L errors. -- in this navigate user to input page.
xiv. In absence of P.L erros -- invoke service layer method -- in case of success --store dtls under session scope & navigate user to success page. --
xv. 3.6 Give logout link from success page -- in logout controller method --
xvi. min args required ---org.springframework.web.bind.support.SessionStatus
xvii. API of SessionStatus --
xviii. void setComplete() --- to discard session.

1. How to add P.L validations?
2. Annotations based approach ---
3. 4.1 -- Add validation annotations in POJO class.
4. (ref : from templates.txt)
5. 4.2 -- in the request handling method of Controller --- in the form processing --- (3.5)
6. simply add @Valid annotation for POJO class reference -- which will force SC to apply validation rules mentioned in annotated POJO class & BindingResult i/f is populated with validation results .

xix. 4.3 Problem in above scenarios is -- in case of validation failures --- page will display spring's default err mesgs.

# How to add custom err messages & perform validation using annotations?

1. Add entry in xml config file (spring-servlet.xml) --define name of message resource bundle.

i. NOTE -- bean id must be messageSource & can be declared either in spring-servlet.xml (i.e root xml for dispatcher servlet) or imported xml file.
   1. Add property file & see error msgs derived from property file.
   2. How to ?
   3. syntax for property name = constraint name.model attr name.model prop name
   4. eg
   5. NotEmpty.user.name=Name is required
   6. or in case of invalid date format ---
   7. typeMismatch.java.util.Date=Invalid date format
   8. How to add automatic exc handling support in spring MVC ?
   9. Add following bean definition in spring-servlet.xml

ii. exc-handler

iii. Above describes centralized exc handler page .(exc-handler.jsp)

iv. Spring From Tags examples -- import spring form tag library
   1. 1.
   2. 
   3. 3.
   4. 
   5. where countryList -- is model attribute of type List & Country has
   6. countryId & countryName -- as properties.
   7. 6
   8. 7
   9. How to add I18N support?

v. Internationalization is the process of designing a software application so that it can potentially be adapted to various languages and regions without engineering changes.

vi. Localization is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text .

vii. Spring frameworkgives you LocaleResolver bean to support the internationalization and thus localization as well.

viii. 6.1. Add locale resolver bean definition in spring-servlet.xml file.

ix. SessionLocaleResolver

x. SessionLocaleResolver resolves locales by inspecting a predefined attribute in a user's session. If the session attribute doesn't exist, this locale resolver determines the default locale from the accept-language HTTP header.
xi. 6.1.5 Add mvc interceptors for detecting change in locale
xii. LocaleChangeInterceptor interceptor detects if a special parameter is present in the current HTTP request. The parameter name can be customized with the paramName property of this interceptor. If such a parameter is present in the current request, this interceptor changes the user's locale according to the parameter value.
xiii. p:paramName="locale123" />
xiv. Above 2 can be either declared in spring-servlet.xml or also in imported xml file.
xv. 6.2 Create copies of message based property files
xvi. 6.3 Create JSP with links to add support for various locales -- using same param name
xvii. en English

## 6.4 Use

1. How to serve static resources?
2. Eg : CSS,JavaScript or images.

i. Configure a handler for serving static resources in spring-servlet.xml.
ii. For applying CSS
iii. 8 How to add content(file) uploading feature?
iv. Spring supports multipart i.e file uploading. Spring has built-in multipart support for file uploads in web applications.
v. A multipart content is the content with enctype="multipart/form-data".
vi. (def HTML form encoding type : application/x-www-form-urlencoded)
vii. API
viii. Interface : org.springframework.web.multipart.MultipartResolver
ix. One implementation of the MultipartResolver i/f is Commons FileUpload
x. Spring CommonsMultipartResolver is a MultipartResolver implementation for use with Apache Commons FileUpload.
xi. Additional jar required -- apache commons-fileupload.jar

## Steps to enable the Spring multipart handling:

1. Add a multipart resolver bean to the web application's context(either directly in spring-servlet.xml or import from separate file)

2. Spring provides org.springframework.web.multipart.MultipartFile which is a representation of an uploaded file received in a multipart request. It provides handy methods like getName(), getContentType(), getBytes(), getInputStream() etc.. which make it easier while retrieving information about file being uploaded.
3. Ref : javadocs.

i. Create a wrapper POJO having MutipartFile as a state(data member)
ii. (not mandatory)
    1. Write a controller .
    2. 3.1 Add req handling method to show upload form
    3. Add POJO instance to the Model map (model attribute)
    4. Create JSP view with a form (form:form) having same model attribute & encType="multipart/form-data"
iii. Add
iv. OR can be done also by a simple form.
    1. In the same controller add req handling method to process upload form
    2. Use MultipartFile API to transfer file from temp location or memeory to permanent file on server.
v. API of MultipartFile
vi. String getContentType()
vii. String getOriginalFileName()
viii. byte[] getBytes()
ix. long getSize()
x. public static void copy(byte[] in,File out) throws IOException
xi. Meaning
xii. Each request is inspected to see if it contains a multipart.
xiii. If no multipart is found, the request continues as expected.
xiv. If a multipart is found in the request, the MultipartResolver that has been declared in your context is used. The multipart attribute in your request is treated like any other attribute.

# Enter Spring

i. Why Spring ?
ii. What is it ?
iii. What is dependency injection ?
iv. eg : In Controller class --AdmissionController
v. add a data member
vi. @AutoWired

vii. private IAdmissionService service;
viii. Why & understand it with examples.
ix. eg :
x. Dependent -- JB , Hibernate Based DAO layer, JDBC based DAO ,
xi. Dependency --DAO , SF , JDBC cn
xii. Traditional/conventional approach --- Dependent objs HAVE TO manage(create/locate the dependency) the dependencies.
xiii. Leads to : tight coupling between dependents n dependencies.
xiv. i.e Any time nature of dependency changes --- dependent obj gets affected.
xv. Solution that leads to loose coupling = Dependency Injection resulting into IoC(Inversion of control)
xvi. D.I = Instead of dependent objs managing their dependencies , 3rd party constainers(Spring container/EJB container) will auto provide(locate-load-inst....) the dependencies --@ run time.

## Why its called IoC ?

-- dependent objs are no longer in charge of dependency creation --SC is .

i. Hollywood principle -- You don't call us , we will u!!!!!
ii. Steps for creating spring based Java SE application
    1. Change perspective to Java
    2. Create user lib containing spring JARs(from day10_help\spring4-hibernate5-rest-jars) --spring_all
    3. Create Java project.
    4. Add user lib in the build path & confirm.
    5. Create dependent & dependency beans(copy from day8_help --- dependent n dependency)
    6. Create -- src --R click --new src folder
    7. & create spring bean config file in it.
iii. Spring bean life cycle.

```
@NotEmpty
@Length(min=5,max=10)
@Email
private String email;
@NotEmpty
@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;
```

dlxxxvii. Spring 4 DB Transactions

dlxxxviii. Required JARS ---org.springframework.transaction & aop jars

dlxxxix. Basics

dxc. Benefits of Spring Transaction Management

```
* Very easy to use, does not require any underlying transaction API knowledge
* Your transaction management code will be independent of the transaction technology
* Both annotation- and XML-based configuration
* It does not require to run on a server - no server needed
```

## What is a Transaction?

i. A Transaction is a unit of work performed on the database and treated in a reliable way independent of other transaction. In database transaction processing ACID property refers to the Atomicity, Consistency, Isolation, Durability respectively.

ii. Atomicity- This property says that all the changes to the data is performed as if they form single operation. For example suppose in a bank application if a fund transfer from one account to another account the atomicity property ensures that is a debit is made successfully in one account the corresponding credit would be made in other account.

iii. Consistency- The consistency property of transaction says that the data remains in the consistence state when the transaction starts and ends. for example suppose in the same bank account, the fund transfer from one account to another account, the consistency property ensures that the total value(sum of both account ) value remains the same after the transaction ends.

iv. Isolation- This property says that, the intermediate state of transaction are hidden/ invisible to another transaction process.

v. Durability- The Durability says that when the transaction is completed successfully, the changes to the data persist and are not un-done, even in the event of system failure.A transaction is not considered durable until it commits. A system failure entails a database recovery, which includes a rollback procedure for all uncommitted transactions, ultimately leaving the database in a consistent state.
vi. Transaction Handling
vii. Now, in Java you can handle transactions with plain SQL, with plain JDBC (a bit higher level), using Hibernate (or any other ORM library), or on an even higher level - with EJB or, finally, Spring!
viii. EJBs require an application server, but spring based jdbc application doesn't.
ix. Ways of Transaction Handling

## Programmatic vs. Declarative

i. Spring offers two ways of handling transactions: programmatic and declarative. If you are familiar with EJB transaction handling, this corresponds to bean-managed and container-managed transaction management.
ii. Programmatic means you have transaction management code surrounding your business code. That gives you extreme flexibility, but is difficult to maintain and too much of boilerplate code.
iii. Declarative means you separate transaction management from the business code. You only use annotations or XML based configuration.
iv. As a summary

```
* programmatic management is more flexible during development time but less flexible during application life
* declarative management is less flexible during development time but more flexible during application life
```

## Global transactions Vs Local Transactions

i. Global transactions enable you to work with multiple transactional resources, typically multiple relational databases . The application server manages global transactions through the JTA. (complex to use through UserTransaction object)
ii. Local Transactions
iii. Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have

significant disadvantages as they cannot work across multiple transactional resources.

iv. Spring's solution

v. Spring resolves the disadvantages of global and local transactions. It enables application developers to use a consistent programming model in any environment. You write your code once, and it can benefit from different transaction management strategies in different environments. It supports both declarative and programmatic transaction management. Most users prefer declarative transaction management.

vi. API details

1. The key to the Spring transaction abstraction is the notion of a transaction strategy. Its the central interface in Spring's transaction infrastructure. A transaction strategy is defined by the org.springframework.transaction.PlatformTransactionManager
2. interface: which has TransactionStatus getTransaction(TransactionDefinition td) throws TransactionExc
3. TransactionException --- As in Spring's philosophy, the TransactionException that is thrown by any of the PlatformTransactionManager interface's methods , is unchecked Transaction infrastructure failures are generally fatal , managed by spring Tx frmwork & developer is NOT forced to handle this.

vii. The TransactionDefinition interface specifies:

viii. Isolation: The degree to which this transaction is isolated from the work of other transactions.

ix. Concurrent transactions cause problems that might be difficult to investigate.

```
* Lost update - two transactions both update a row, the second transaction aborts, both changes are lost
* Dirty read - reading changes that are not yet committed
* Unrepeatable read - a transactions reads twice the same row, getting different data each time
* Phantom read - similar to the previous one, except that the number of rows changed
```

Now, the perfect solution to these problems is maximum isolation, but in reality this would cost too much resources and could lead to deadlocks. So, instead, you will set one of five isolation levels (where the fifth one is actually the maximum isolation level):

i. Supported levels

ii. ISOLATION_DEFAULT -- Use the default isolation level of the underlying datastore.
iii. ISOLATION_READ_UNCOMMITTED --- Indicates that dirty reads, non-repeatable reads and phantom reads can occur.
iv. ISOLATION_READ_COMMITTED --- Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
v. ISOLATION_REPEATABLE_READ -- Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
vi. ISOLATION_SERIALIZABLE -- Indicates that dirty reads, non-repeatable reads and phantom reads are prevented.
vii. Transaction Propagation
viii. Whenever a one transactional method calls other transactional mehod , a decision is made - what to do with the transaction. Create a new one? Use an existing one if it exists, otherwise create a new one? Use an existing one only if it exists, otherwise fail?
ix. Supported behaviors ---
x. MANDATORY --Supports a current transaction; throws an exception if no current transaction exists.

REQUIRED -- default behavior.

i. Supports a current transaction; creates a new one if none exists.
ii. NESTED ---Executes within a nested transaction if a current transaction exists, otherwise same as REQUIRED
iii. SUPPORTS
iv. Supports a current transaction; executes non-transactionally if none exists.
v. REQUIRES_NEW
vi. Creates a new transaction, suspending the current transaction if one exists.
vii. NEVER
viii. Does not support a current transaction; throws an exception if a current transaction exists.
ix. NOT_SUPPORTED
x. Does not support a current transaction; always executes non-transactionally.
xi. Timeout: in seconds .How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.
xii. Default value = -1 , indefinite w/o time out
xiii. Otherwise specify value
xiv. eg

xv. @Transactional(timeout=100)
xvi. Read-only status: A read-only transaction can be used when your code reads but does not modify data.
xvii. Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.
xviii. eg -- @Transactional(readOnly = true)
xix. default is false.

## Rollback behavior

i. With Spring transaction management the default behavior for automatic rollback is this: Only unchecked exceptions cause a rollback. Unchecked exceptions are RuntimeExceptions and Errors.
ii. But can be changed.
iii. eg --
iv. @Transactional(rollbackFor = IOException.class, noRollbackFor = RuntimeException.class)
v. public void doSomething(...)
vi. Implementation steps & concept for annotation based declarative transaction management.
    1. For plain JDBC implementations of PlatformTransactionManager --- use DataSourceTransactionManager --- implementation class for a single JDBC DataSource.
    2. Declare the same in spring configuration xml file.
    3. eg --
vii. p:dataSource-ref="dataSource">
    1. To enable annotated transaction support , add transaction namespace(tx) & add following
    2.
    3. Note : can even skip attribute transaction-manager, if id of Tx Mgr bean is transactionManager
viii. This completes configuration steps
    1. In service layer beans (typically annotated with @Service) , add method level annotation @Transactional along with suitable properties.

Assignment status ?

i. Vendor --HAS --BankAccounts
ii. Tested -- create new a/c
iii. get complete vendor + bank acct details
iv. Lazy Init Exc & soln.
v. Revise

# Why Spring ?

i. --simplifies development of enterprise applications.
ii. What is it ?
iii. container & a framework
iv. SC -- manage life cycle of spring beans
v. spring bean -- java objs whose life cycle managed by SC
vi. (Controller, Service, Repository , REST controller)
vii. hibernate manages POJO's life cycle(transient,persistent,detached,removed)
viii. Frmwork = readymade implementation of standard patterns.
ix. What is dependency injection & its advantages.
x. Instead of dependent objs either creating or looking up for their dependencies ---3rd party(Spring container or EJB container) can directly inject(=making dependencies available to depdent objs auto @ run time) the dependencies .

# why -- loose coupling

i. Spring IoC(inversion of control) based container --refer to diag
ii. XML based configuration ?
iii. eg :
iv. Available scope Java SE --singleton or prototype
v. Web Java -- singleton or prototype + request,session
vi. singleton -- def scope of spring bean.
vii. 1 single bean instance will be shared across multiple requests made to SC
viii. prototype -- a new bean instance will be created per request made to the SC
ix. def loading policy for singleton beans = eager (can be chaged to lazy)
x. Only supported loading policy for prototype beans = lazy (once per demand)
xi. How to get rdymade spring bean instance from SC ?
xii. Inherited method of BeanFactory i/f
xiii. T getBean(String name,Class requiredType)
xiv. throws BeansException
xv. eg : ATM atm=ctx.getBean("my_atm",ATMImpl.class);
xvi. Default scope of the spring bean ?
xvii. singleton

Default loading policy ?

i. eager (= SC will start the life cycle of spring beans @ SC startup)

ii. Life cycle ---locate --load --instantiate
iii. Setter Based D.I (optional) / wiring /collaboration dependent n dependecies
iv. init style method
v. Spring bean is rdy for servicing B.L / clnt requests
vi. end --- destroy style (called only for singleton beans)
vii. GC
viii. spring bean config file --- xml --- any name
ix. Location --- run time cls path (for creating SC from ClasspathXmlApplicationContext)
x. Which namespaces to add ? ---beans
xi. Bean tag attributes
    1. id --- unique bean id
    2. class --- F.Q bean class name
    3. 3 scope --- singleton|prototype
    4. lazy-init -- boolean
    5. def value = false
    6. applicable only for singleton beans.
    7. init-method --- name of init method in spring bean.
    8. (called by SC after setter based D.I )
    9. destroy-method -- name of destroy method( called by SC before GCing spring bean -- applicable only to singleton)
    10. autowire=no|byName|byType|constructor

Which is the child tag, within tag for setter based dependency injection?

i. eg :
ii.
iii. Discuss spring bean life cycle --(refer to diag later)
iv. Which are different modes of wiring(D.I) ?
v. Enter annotations (steps)
    1. Add context namespace in bean config file.
    2. Add ctx tags
    3. 2.1 --- enables all class internal annotations.
vi. 2.2
vii. eg :
viii. SC will auto scan these packages for stereotype annotations
ix. stereotype annotations (from the pkg -- o.s.stereotype)
    1. @Component -- spring bean
    2. @Controller -- request handling controller spring bean
    3. @Service -- B.L holder spring bean

4. @Repository -- DAO spring bean
5. @RestController -- REST server end point spring bean

x. XML Tags Vs Annotations
xi. 1. ---- @Component or its sub type ---class level annotations
xii. eg : @Component("abc")
xiii. public class MyBean {...}
    1. scope="singleton|prototype|request|session" ---- @Scope class level annotation
    2. lazy-init --- @Lazy -- class level annotation
    3. init-method ---- @PostConstruct ---method level annotation
    4. destroy-method -- @PreDestroy ---method level annotation
    5. explicit wiring --- @Required --dependency is mandatory
    6. autowire ---- @AutoWired
xiv. @AutoWired --- parameterized constr level (constr based D.I) , setter level (setter based D.I) OR field level D.I
xv. ---autowire=byType
xvi. eg : in DAO layer --- its dependency is SessionFactory
xvii. @AutoWired
xviii. private SessionFactory sf;
    1. autowire=byName ---- @AutoWired + @Qualifier("bean id")
    2. eg : In service layer
    3. @AutoWired
    4. @Qualifier("abc")

private CustomerDao dao;

# MVC(Model View Controller) --why & overview

i. Implementation using servlet / JSP & JavaBean --Shared as a readymade demo.
ii. Enter Spring MVC --concept & implementation steps
iii. 0.Open J2EE perspective
iv. 1.Create dynamic web project
    1. Add spring-all user lib in 2 places
    2. 2.1 Under build path
    3. 2.2 Under deployment assembly
    4. Configure front controller to intercept any request from any client, in web.xml.
    5. Copy welcome-file-list , servlet & servlet-mapping tag from the template present in --day10_eve_help/spring-hibernate/spring4-hib5-templates/web.xml

6. Copy master config xml file for starting SC.
7. copy spring-servlet.xml under
8. Create request handling controller.
9. Annotations used -- @Controller & @RequestMapping
10. 5.1 Test Spring MVC flow --by adding index.jsp in WebContent & welcome.jsp under /views/

## 5.2 Test Model & View

Objective

i. 1.What is ModelAndView -- holder of logical view name + model (model attributes)
    1. Understand concept of Model
ii. Concept of Model --o.s.ui.Model (i/f) = map of model attributes
iii. Type of Map -- Map
iv. who cretes empty map -- SC
v. How do u(Controller) access it --can be passed as one of the params of request handling methods. (as Dependency)
vi. who populates it -- SC / Controller(prog)
vii. why ? -- to share the results available in controller layer with View.
viii. Model attr map is sent from controller --> F.C Front Controller (Dispatcher Servlet)
ix. F.C adds it in request scope & pushes it(sends it) to JSP (view layer)
x. So this is called push MVC architecture.
    1. How to add model attr in Model map?
    2. public Model addAttribute(String nm,Object value);
xi. How to get Model map ?
xii. Using D.I -- just tell SC that your req handling method needs a model map(how ? -- by adding it in the method arg) & SC will create Model map & inject it in your code.
    1. Handling request params (w/o using form binding) -- of different types(int,string,date)
    2. eg : Test Request Params

## Annotation to use in controller method -- @RequestParam

1. Handling path variables (extremely important for RESTful web services) via
2. @PathVariable & @DateTimeFormat & JSTL fmt library

3. eg URL : Test Path Variables

eg :

i. URI Template http://www.example.com/users/{userId}
ii. Above contains the variable userId. Assigning the value "rama" to the variable yields http://www.example.com/users/rama
iii. In Spring MVC you can use the @PathVariable annotation on a method argument to bind it to the value of a URI template variable:
iv. @RequestMapping(value="/users/{userId}", method=RequestMethod.GET)
v. public String findUser(@PathVariable(name="userId") String userId123, Model model) {
vi. User user = userService.findUser(userId);
vii. model.addAttribute("user_details", user);
viii. return "display_user";
ix. }
x. The URI Template "/users/{userId}" specifies the variable name userId. When the controller handles this request, the value of userId is set to the value found in the appropriate part of the URI.
    1. How SC creates a populated Model Map ? (eg : With customer pojo)
    2. Add a parameter to req handling method (Customer)
    3. Create layered spring web application for Customer management.
    4. Controller -- @Controller , @RequestMapping
    5. @AutoWired
    6. private CustomerService service;
xi. Service -- B.L --i/f --- validate,register,list
xii. Imple class ---@Service

POJO -- Customer

i. Layers -- Controller --- Service (B.L+Tx management @Transactional) -- POJO
ii. Objective --List Customers (copy customer POJO & sample service code,copy forms)
iii. Validate Customer
iv. Flow -- index.jsp -- login form --Controller --HM based validation ---err -- login form with err mesg , success (add cust details in HS) --details page -- logout.
v. Method in CustomerController --for validating user
vi. @PostMaping("/login")
vii. public String process(c)
viii. {

ix. //invoke service
x. service.validate(em,pass);
xi. }
xii. resp.sendRedirect(resp.encodeRedirectURL("/cust/details"));
xiii. http://host:port/spring_mvc/cust_details
xiv. Register Customer --- redirect to login page with a mesg.
xv. Flow -- index.jsp -- reg form --Controller -- HM based registration --- err (duplicate email) -- login form with err mesg -- success (add cust details in HS) --details page --logout.
xvi. More on RedirectAttributes
xvii. org.springframework.web.servlet.mvc.support.RedirectAttributes
xviii. --map of flash scoped attributes

## Use case --- redirect scenario

i. To remember the attrs in the next request.
ii. API
iii. public RedirectAttributes addFlashAttributes(String nm,Object val)
    1. Add P.L (copy all annotations @Pattern etc)
    2. 7.1 Add annotations for P.L validation
    3. 7.2 Add @Valid on model attribute
    4. 7.3 along with path
    5. spring web + hib --for customer scenario
    6. Controller --- Service (@Service,@Transactional) --Hib Based DAO (@Repository) --POJO (@Entity) -- DB
    7. POJO -- @Entity,@Id
    8. POJO props ---duel role --- represents DB cols + clnt conv state
iv. eg : Customer
v. @NotEmpty
vi. @Email
vii. private String email;
viii. @Column(...)
ix. getEmail()
x. Refer to --spring hibnernate dev blocks for overview.
xi. How to auto navigate the clnt to home page after a dly ?
xii. By setting refresh header of HTTP response.
xiii. API of HttpServletResponse
xiv. public void setHeader(String name,String value)
xv. name --- refresh
xvi. value --- 10;url=home page url (root of web app)

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

## JSON format supports the following data types -

i. Type Description
ii. Number double- precision floating-point format in JavaScript
iii. String double-quoted Unicode with backslash escaping
iv. Boolean true or false
v. Array an ordered sequence of values
vi. Value it can be a string, a number, true or false, null etc
vii. Object an unordered collection of key:value pairs
viii. Whitespace can be used between any pair of tokens
ix. null empty
x. Number
xi. • It is a double precision floating-point format in JavaScript and it depends on implementation.
xii. • Octal and hexadecimal formats are not used.
xiii. • No NaN or Infinity is used in Number.
xiv. The following table shows the number types -
xv. Type Description
xvi. Integer Digits 1-9, 0 and positive or negative
xvii. Fraction Fractions like .3, .9
xviii. Exponent Exponent like e, e+, e-, E, E+, E-
xix. Syntax
xx. var json-object-name = { string : number_value, .....}
xxi. Example
xxii. Example showing Number Datatype, value should not be quoted -
xxiii. var obj = {marks: 97}
xxiv. String
xxv. • It is a sequence of zero or more double quoted Unicode characters with backslash escaping.
xxvi. • Character is a single character string i.e. a string with length 1.
xxvii. The table shows string types -
xxviii. Type Description
xxix. " double quotation
xxx. \ reverse solidus
xxxi. / solidus
xxxii. b backspace

xxxiii. f form feed
xxxiv. n new line
xxxv. r carriage return
xxxvi. t horizontal tab
xxxvii. u four hexadecimal digits
xxxviii. Syntax
xxxix. var json-object-name = { string : "string value", .....}
xl. Example
xli. Example showing String Datatype -
xlii. var obj = {name: 'Amit'}
xliii. Boolean
xliv. It includes true or false values.
xlv. Syntax
xlvi. var json-object-name = { string : true/false, .....}
xlvii. Example
xlviii. var obj = {name: 'Amit', marks: 97, distinction: true}

## Array

i. • It is an ordered collection of values.
ii. • These are enclosed in square brackets which means that array begins with .[. and ends with .]..
iii. • The values are separated by , (comma).
iv. • Array indexing can be started at 0 or 1.
v. • Arrays should be used when the key names are sequential integers.
vi. Syntax
vii. [ value, .....]
viii. Example
ix. Example showing array containing multiple objects -
x. {
xi. "books": [
xii. { "language":"Java" , "edition":"second" },
xiii. { "language":"C++" , "lastName":"fifth" },
xiv. { "language":"C" , "lastName":"third" }
xv. ]
xvi. }
xvii. Object
xviii. • It is an unordered set of name/value pairs.
xix. • Objects are enclosed in curly braces that is, it starts with '{' and ends with '}'.

xx. • Each name is followed by ':'(colon) and the name/value pairs are separated by , (comma).
xxi. • The keys must be strings and should be different from each other.
xxii. • Objects should be used when the key names are arbitrary strings.
xxiii. Syntax
xxiv. { string : value, .....}
xxv. Example
xxvi. Example showing Object -
xxvii. {
xxviii. "id": "011A",
xxix. "language": "JAVA",
xxx. "price": 500,
xxxi. }
xxxii. Whitespace
xxxiii. It can be inserted between any pair of tokens. It can be added to make a code more readable. Example shows declaration with and without whitespace -
xxxiv. Syntax
xxxv. {string:" ",....}
xxxvi. Example
xxxvii. var i = " sachin";
xxxviii. var j = " saurav"
xxxix. null
xl. It means empty type.
xli. Syntax
xlii. null
xliii. Example
xliv. var i = null;
xlv. if(i == 1){
xlvi. document.write("
value is 1
i. ");
ii. }
iii. else{
iv. document.write("
value is null
i. ");
ii. }

# JSON Overview

i. JSON
ii. JSON stands for JavaScript Object Notation.
iii. It is a lightweight text-based open standard designed for human-readable data interchange. Its light weight alternative to XML for exchanging data on the network.
iv. Conventions used by JSON are known to programmers which include C, C++, Java, Python, Perl etc.
v. This format was specified by Douglas Crockford. This was designed for human-readable data interchange
vi. Extended from the JavaScript scripting language.
vii. The filename extension is .json
viii. JSON Internet Media type is application/json
ix. Uses of JSON
    1. Used in JavaScript based application which includes websites development.
    2. JSON format is used for serializing & transmitting structured data over network connection.
    3. This is primarily used to transmit data between server and client application.
    4. Web Services use JSON format to provide public data.
    5. It can be used with modern programming languages.
x. Java JSON
xi. Support exists in different varieties
    1. Jackson json library -- JAR
    2. Google GSon
    3. Json.simple
xii. What is Maven ?
xiii. Build automation tool for overall project management.
xiv. It helps in
    1. checking a build status
    2. generating reports (basically javadocs)
    3. setting up the automated build process and monitors the same.
xv. It eases out source code compilation, distribution, documentation, collaboration with different teams .
xvi. 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.

xvii. Vendor -- Apache
xviii. Earlier build tool -- Ant
xix. 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.
3. No such support in ant.
4. There is no life cycle in Ant, where as life cycle exists in Maven.

i. Maven advantages
   1. Managing dependencies
   2. Uses Convention over configuration - configuration is very minimal
   3. Multiple/Repeated builds can be achieved.
   4. Plugin management.
   5. Testing - ability to run JUnit and other integration test suites.

## What is POM? (Project Object Model)

i. It is the core element of any maven project.
ii. Any maven project consists of one configuration file called pom.xml.
iii. Location --In the root directory of any maven project.
iv. It contains the details of the build life cycle of a project.
v. Contents
vi. Dependencies used in the projects (Jar files)
vii. Plugins used
viii. Project version
ix. Developers involved in the project
x. Build profiles etc.
xi. 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. 
7. packaging -- defines packaging type such as jar, war etc.
8. name -- defines name of the maven project.
9. plugins ---compiler plugins , eclipse plugins
10. dependencies -- collection of dependencies for this project.
11. Witin that --
12. dependency -- defines a specific dependency.(eg : hibernate dependency,spring web)
13. scope -- defines scope for this maven project. It can be compile, provided, runtime, test and system.

## Goals in Maven

i. 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.
ii. eg : clean , build ,install ,test
iii. What is a Maven Repository
iv. 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:

i. Local repository then Central repository then Remote repository.
ii. maven repositories

iii. If dependency is not found in these repositories, maven stops processing and throws an error.
    1. Maven Local Repository
iv. Maven local repository is located in the file local system. It is created by the maven when you run any maven command.
v. By default, maven local repository is HOME / .m2 directory.
vi. (Can be updated by changing the MAVEN_HOME/conf/settings.xml)
vii. 2) Maven Central Repository
viii. Maven central repository is located on the web(Created by the apache maven community)
ix. The path of central repository is: https://mvnrepository.com/repos/central
x. 3) Maven Remote Repository
xi. 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

i. What is it ?
ii. 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.
iii. Maven comes with 3 built-in build life cycles
iv. Clean - this phase involves cleaning of the project (for a fresh build & deployment)
v. Default - this phase handles the complete deployment of the project
vi. Site - this phase handles the generating the java documentation of the project.
vii. Build Profiles in Maven
viii. It is a subset of elements which allows to customize builds for particular environment. Profiles are also portable for different build environments.
ix. 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.
x. 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
xi. 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. 4.Verify maven
5. mvn -- version
6. OR use m2e plug-in (a standard part of Eclipse for J2EE)

i. Background
ii. Web services have really come a long way since its inception. 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.
iii. 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.
iv. REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.
v. eg of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.
vi. Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on.
vii. Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.

So instead , Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.

i. Refer to diag --REST 0.png

## What is Restful Web Service?

i. 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.

## Restful Methods

i. The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.
ii. Refer to -- REST 1.png

iii. Let's assume that we have a RESTful web service is defined at the location. http://www.server.com/employee . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.
iv. POST – This would be used to create a new employee using the RESTful web service
v. GET - This would be used to get a list of all employee using the RESTful web service
vi. PUT - This would be used to update all employee using the RESTful web service
vii. DELETE - This would be used to delete all employee using the RESTful web service
viii. Background
ix. Web services have really come a long way since its inception. 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.
x. 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.
xi. REST is a way to access resources which lie in a particular environment. For example, you could have a server that could be hosting important documents or pictures or videos. All of these are an example of resources. If a client, say a web browser needs any of these resources, it has to send a request to the server to access these resources. Now REST defines a way on how these resources can be accessed.
xii. eg of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.
xiii. Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on.
xiv. Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.
xv. So instead , Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.

Refer to diag --REST 0.png

# What is Restful Web Service?

i. 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

ii. 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.

## Restful Methods

i. The below diagram shows mostly all the verbs (POST, GET, PUT, and DELETE) and an example of what they would mean.
ii. Refer to -- REST 1.png
iii. Let's assume that we have a RESTful web service is defined at the location. http://www.server.com/employee . When the client makes any request to this web service, it can specify any of the normal HTTP verbs of GET, POST, DELETE and PUT. Below is what would happen If the respective verbs were sent by the client.
iv. POST This would be used to create a new employee using the RESTful web service
v. GET - This would be used to get a list of all employee using the RESTful web service
vi. PUT - This would be used to update all employee using the RESTful web service
vii. DELETE - This would be used to delete all employee using the RESTful web service

## Annotations

1. @PathVariable --- handles URL templates. In the above code, the path variable {name} is mapped to a String object (@PathVariable("name") String name). Therefore all of the URI such as /books/xx or /books/yy will map to the methods in the controller.

i. eg : URI http://www.example.com/spring_mvc/users/123
ii. Above contains the variable userId. Assigning the value "123" to the variable yields http://www.example.com/users/123
iii. Referred to as URI template variable.
iv. In Spring MVC you can use the @PathVariable annotation on a method argument to bind it to the value of a URI template variable:
v. eg :
vi. @GetMapping(value="/users/{userId}")
vii. public User findUser(@PathVariable int userId) {
viii. User user = userService.findUser(userId);
ix. return user;
x. }

The URI Template "/users/{userId}" specifies the variable name userId. When the controller handles this request, the value of userId is set to the value found in the appropriate part of the URI.

The @ResponseBody annotation 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)

1. where -- on the ret type of request handling methods
2. eg :
3. @Controller
4. @RequestMapping("/employee")
5. public class EmpController
6. {
7. @GetMapping(....)
8. public @ResponseBody Emp fetchEmpDetails(....int empId)
9. {
10. //get emp dtls from DB
11. return e;
12. }
13. }
14. OR
15. @RestController = @Controller (at the cls level) + @ResponseBody auto added on the ret type of request handling methods

## 3.The @RequestBody annotation,

i. instead, unmarshalls the HTTP request body into a Java object injected in the method.ï»¿What is REST ?
ii. REST stands for REpresentational State Transfer.
iii. REST is web standards based architecture and uses HTTP Protocol for data communication.
iv. It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.
v. REST was first introduced by Roy Fielding in 2000.
vi. In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents the resources.
vii. Here each resource is identified by URIs
viii. 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
ix. HTTP Methods
x. Following well known HTTP methods are commonly used in REST based architecture.

xi. GET - Provides a read only access to a resource.
xii. POST - Used to create a new resource.
xiii. DELETE - Used to remove a resource.
xiv. PUT - Used to update a existing resource or create a new resource.

## RESTFul Web Services

i. A web service is a collection of open protocols and standards used for exchanging data between applications or systems.
ii. Platform & technology independent solution.
iii. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.
iv. This interoperability (e.g., between Java and Python, or Windows and Linux applications or java & .net ) is due to the use of open standards.
v. Web services based on REST Architecture are known as RESTful web services.

These web services use HTTP methods to implement the concept of REST architecture. A RESTful web service usually defines a URI, Uniform Resource Identifier a service, provides resource representation such as JSON and set of HTTP Methods.

## Annotations

1. @PathVariable --- handles URL templates. In the above code, the path variable {name} is mapped to a String object (@PathVariable("name") String name). Therefore all of the URI such as /books/xx or /books/yy will map to the methods in the controller.
2. The @ResponseBody annotation 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.

## 3.The @RequestBody annotation,

instead, unmarshalls the HTTP request body into a Java object injected in the method.

The RestTemplate is similar to other Spring templates such as JmsTemplate and JdbcTemplate in that Spring eliminates a lot of boot strap code and thus makes your code much cleaner. When applications use the RestTemplate they do not need to

worry about HTTP connections, that is all encapsulated by the template. They also get a range of APIs from the RestTemplate which correspond to the well know HTTP methods (GET, PUT, POST, DELETE, HEAD, OPTIONS). These APIs are overloaded to cater for things like different ways of passing parameters to the actual REST API.

i. Create resful service provider project & then develop its client
ii. For spring restful web service --actually only spring web mvc , hibernate validator & json jars are sufficient.
iii. Server side steps
    1. Create Spring Web MVC application
    2. The layers --service --dao--pojo --DB are the same.
    3. In controller layer , replace @Controller by @RestController annotation, at class level.
    4. Request Handling methods will respond to different HTTP methods
    5. (get/post/put/delete)
    6. For method=get
iv. Can directly return either a resource eg : Person,BankAccount or Customer or still better is can return entire HTTP response encapsulated in ResponseEntity
v. What is org.springframework.http.ResponseEntity
vi. Represents HTTP response entity, consisting of status,headers and body.
vii. Constructors
    1. public ResponseEntity(T body,HttpStatus statusCode)
    2. Create a new ResponseEntity with the given body and status code, and no headers.
    3. 2.public ResponseEntity(T body,MultiValueMap headers, HttpStatus statusCode)
    4. Create a new ResponseEntity with the given body and status code, and headers.
    5. In the controller's get handling method
    6. 1.1 In @RequestMapping(value = "add uri with template variables")
    7. Return type of the method = either a resource or resource embedded in the ResponseEntity.
    8. Use service layer to fetch a resource.
    9. Return it to the client.
    10. eg : @GetMapping(value="/cust/{id}")
    11. public Customer getCustomerById(@PathVariable int id) {
    12. invoke service layer method to get customer details
    13. }
    14. In the controller's post handling method

15. In @PostMapping(value = "uri")
16. Add @RequestBody annotated method argument
17. Return type of the method = either a resource or resource embedded in the ResponseEntity.
18. Use service layer to fetch a resource.
19. Return it to the client.

## @RestController :

Spring 4â€²s new @RestController annotation.

i. Its a combination of @Controller and @ResponseBody.
ii. @RequestBody : If a method parameter is annotated with @RequestBody, Spring will bind the incoming HTTP request body(for the URL mentioned in @RequestMapping for that method) to that parameter. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the HTTP request body into domain object [deserialize request body to domain object], based on ACCEPT or Content-Type header present in request.
iii. @ResponseBody : If a method is annotated with @ResponseBody, Spring will bind the return value to outgoing HTTP response body. While doing that, Spring will [behind the scenes] use HTTP Message converters to convert the return value to HTTP response body [serialize the object to response body], based on Content-Type present in request HTTP header. As already mentioned, in Spring 4, no need to use this annotation.
iv. ResponseEntity is a real deal. It represents the entire HTTP response. Good thing about it is that you can control anything that goes into it. You can specify status code, headers, and body. It comes with several constructors to carry the information you want to sent in HTTP Response.
v. @PathVariable This annotation indicates that a method parameter should be bound to a URI template variable [the one in '{}'].(binding between request handling method parametere & URI template variable)

MediaType : With @RequestMapping annotation, you can additionally, specify the MediaType to be produced or consumed (using produces or consumes attributes) by that particular controller method, to further narrow down the mapping.

i. URI Template Patterns
ii. URI templates can be used for convenient access to selected parts of a URL in a @RequestMapping
iii. method.
iv. A URI Template is a URI-like string, containing one or more variable names. When you substitute

v. values for these variables, the template becomes a URI. The proposed RFC for URI Templates defines
vi. how a URI is parameterized. For example, the URI Template http://www.example.com/users/
vii. {userId} contains the variable userId. Assigning the value fred to the variable yields http://
viii. www.example.com/users/fred.
ix. In Spring MVC you can use the @PathVariable annotation on a method argument to bind it to the
x. value of a URI template variable:
xi. @RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
xii. public String findOwner(@PathVariable String ownerId, Model model) {
xiii. Owner owner = ownerService.findOwner(ownerId);
xiv. model.addAttribute("owner", owner);
xv. return "displayOwner";
xvi. }
xvii. The URI Template " /owners/{ownerId}" specifies the variable name ownerId. When the controller
xviii. handles this request, the value of ownerId is set to the value found in the appropriate part of the URI.
xix. For example, when a request comes in for /owners/abc, the value of ownerId is abc.
xx. To process the @PathVariable annotation, Spring MVC needs to find the matching URI template
xxi. variable by name. You can specify it in the annotation:
xxii. @RequestMapping(value="/owners/{ownerId}", method=RequestMethod.GET)
xxiii. public String findOwner(@PathVariable("ownerId") String theOwner, Model model) {
xxiv. // some implementation
xxv. }

Or if the URI template variable name matches the method argument name you can omit that

i. detail. As long as your code is not compiled without debugging information, Spring MVC will match the method argument name to the URI template variable name:
ii. A key difference between a traditional MVC controller and the RESTful web service controller is the way that the HTTP response body is created.

Rather than relying on a view technology 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

iii. 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, but that instead the java object returned is the response body, and should be written out directly.

iv. The java object must be converted to JSON. Thanks to Springs HTTP message converter support, you dont 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)

v. API --Starting point

vi. org.springframework.http.converter.HttpMessageConverter

vii. --Interface that specifies a converter that can convert from and to HTTP requests and responses.

viii. Implementation classes

1. org.springframework.http.converter.xml.Jaxb2RootElementHttpMessageConverter
2. -- Implementation of HttpMessageConverter that can read and write XML using JAXB2.(Java architecture for XML binding)
3. org.springframework.http.converter.json.MappingJackson2HttpMessageConverter
4. --Implementation of HttpMessageConverter that can read and write JSON using Jackson 2.x's ObjectMapper class API

Good news is @RestController = @Controller + @ResponseBody added on ret types of all request handling methods

i. Revision of spring-mv-hibernate with example

ii. vendor deletion

iii. Complete Vendor Management system with

1. 2 way Form binding for --register & update
2. eg : User registration
3. Steps in 2 ways form binding
4. In show reg form phase : -- add empty POJO as method argument.
5. SC -- will create empty POJO instance & auto add it in Model map.
6. Use spring supplied form tag lib for 2 way form binding technique.

7. eg :
8. 
9. In process form phase
10. In @PostMapping method
11. Add POJO instance as method argument.
12. SC --1. Vendor v=new Vendor();
13. invokes matching setters (req param names--path
14. MUST match POJO prop names.)
15. Injects populated POJO in the req handling method.

iv. For Understanding REST --MUST go through this
v. REST --
vi. Refer to basic video tuts shared.
vii. Read regarding REST --readme & then REST simplified readme.
viii. refer to spring-restful-sequence
ix. concept & implementation steps for providing Stock as a resource ---
x. Layers ---StockController--IStockDAO , StockDaoImpl --Stock POJO
    1. Create dyn web project , add user lib , web.xml , spring-servlet.xml, --- db properties n hib persistence xml.
    2. Discard --- ViewResolver bean
    3. & view layer(JSP) (If you are converting from existing spring-mvc web application)
    4. StockController ---to test Stock server.

## What is a web service ?

i. IntegralpartofSOA(serviceorientedarchitecture)
ii. service=Businessfunctionalitytobeexportedtoremotecl nts.
iii. server--serviceprovider
iv. clnt--serviceaccessor
v. Why--ToexporttheBusinesslogic(functionallogic--banking,customerservice,paymentgateway,stockexchangeserverBSE,NSE...)toremoteclientsoverstandardsetofprotocols.
vi. ItsequivalenttoJavaRMI(remotemethodinvocation)
vii. InJavaRMI--javaclntobjectcandirectlyinvoketheremotemethod(hostedontheremotehost)&getnprocessresults.(i.eitgivesyoulocationtransperency)
viii. BUTJavaRMI---is100%javasolution.

ix. Thereisnointeroperabilityinthat(i.eitsatechnology specificsoln)
x. Howtoarriveattechnologyindesoln?
xi. CORBA---Commonobjrequestborkerarchitecture
xii. toughtosetup.(IDL---i/fdeflanguage)
xiii. Betteralternative---webservices
xiv. Earlier(J2EE1.4)--JAX-RPC
xv. JavaAPIforXMLbasedremoteprocedurecalls
xvi. TodayreplacedbyJAX-WS&JAXRS
xvii. JAXWS--JavaAPIforXMLbasedwebservices--Base dupon
xviii. Protocol--SOAP--simpleobjectaccessprotocol(runsoverHTTP)
xix. Hasadditionalheader&messageformat.
- HavetosetupNamingservice(UDDI--UniversalDescription,Discovery,andIntegration)
- HavetosetupWSDL(webservicedef.language)--xmlbasedwebservicedeflang.
- Toomuchtosetup&eatsuplargerbandwidth!!
- SoasimplesolnisJAXRS--JavaAPIforRESTfulweb service
- JAXRS---isapartofJ2EEspecifications
- KnownVendors--Apache,JBoss
- &products--RESTeasy,ApacheCXF
- BUTit'sstilldifficulttosetup.
- Sospring,beingintegrationmaster,comestotherescue.....

Referto--RegardingREST&RESTsimplified.
Createrestfulserviceproject
i. Testitwithpostman
ii. writeitsclntasanotherspringMVCwebappliation.
iii. Forspringrestfulwebserviceprovider
iv. jars--spring-web-hib+jsonjars
v. Forspringrestfulwebserviceclient--actuallyonlyspringwebmvc,hibernatevalidator&jsonjarsaresufficient.
vi. Serversidesteps
vii. 1.CreateSpringWebMVCapplication
viii. 2.Thelayers--service--dao--pojo--DBarethesame.

ix. 3.Incontrollerlayer,replace@Controllerby@RestController annotation,atclasslevel.
x. 4.RequestHandlingmethodswillrespondtodifferentHTTPmethods
xi. (get/post/put/delete)
xii. 5.Formethod=get
xiii. Candirectlyreturneitheraresourceeg:Person,BankAccountorCustomerorstillbetteriscanreturnentireHTTPresponseencapsulatedinResponseEntity<T>
xiv. Whatiso.s.http.ResponseEntity<T>
xv. RepresentsHTTPresponseentity,consistingofstatus,headersandbody.
xvi. Constructors
xvii. 1.publicResponseEntity(Tbody,HttpStatusstatusCode)
xviii. CreateanewResponseEntitywiththegivenbodyandstatuscode,andnoheaders.
xix. 2.publicResponseEntity(Tbody,MultiValueMap<String,String>headers,HttpStatusstatusCode)
xx. CreateanewResponseEntitywiththegivenbodyandstatuscode,andheaders.
xxi. 1.Inthecontroller'sgethandlingmethod
xxii. 1.In@RequestMapping(value="adduriwithtemplatevariables")
xxiii. Returntypeofthemethod=eitheraresourceorresourceembeddedintheResponseEntity.
xxiv. eg:@RequestMapping(value="/cust/{id}")

```
  public Customer getCustomer(@PathVariable int id) {
```

//accessid&invokeservicelayermethod
i. }
ii. 2.Useservicelayertofetcharesource.
iii. 3.Returnittotheclient.
iv. 2.Inthecontroller'sposthandlingmethod
v. 1.In@PostMapping(value="uri")
vi. Add@RequestBodyannotatedmethodargument
vii. Returntypeofthemethod=eitheraresourceorresourceembeddedintheResponseEntity.

viii. eg:
ix. @RequestMapping(value="/cust",method=RequestMethod.POST)

```
public ResponseEntity<Customer> createCustomer(@RequestBody Customer c) {

 //invoke service layer method
}
```

i. 3.Useservicelayertocreatearesource.
ii. 3.Returnittotheclient.

---

## ClientsideAPIforRESTfulclient

i. Startingpoint--o.s.web.client.RestTemplate
ii. Mainclassforsynchronous(BLOCKING)client-sideHTTPaccess.
iii. ItsimplifiescommunicationwithHTTPserversusingRESTfulprinciples.
iv. IthandlesHTTPconnections.
v. YouhavetoprovideURLs(withpossibletemplatevariables)andextractresults.
vi. TheRestTemplateissimilartootherSpringtemplatessuchasJmsTemplateandJdbcTemplateinthatSpringeliminatesalotofbootstrapcodeandthusmakesyourcodemuchcleaner.WhenapplicationsusetheRestTemplatetheydonotneedtoworryaboutHTTPconnections,thatisallencapsulatedbythetemplate.TheyalsogetarangeofAPIsfromtheRestTemplatewhichcorrespondtothewellknowHTTPmethods(GET,PUT,POST,DELETE,HEAD,OPTIONS).TheseAPIsareoverloadedtocaterforthingslikedifferentwaysofpassingparameterstotheactualRESTAPI.
vii. 1.Forgettingasingleormultipleresourcesfromtheserviceprovider.
viii. useHTTPmethod=get
ix. getAPIoforg.springframework.web.client.RestTemplate

x. public<T>TgetForObject(Stringurl,Class<T>responseType,Object...urlVariables)
xi. throwsRestClientException
xii. T--typeoftheresource(result/POJO/DTO)
xiii. eg:BankAccounta=template.getForObject("http://localhost:7070/day12_rest_server/bank_accts/{ac_id}/{pin}",BankAccount.class,101,"1234");
xiv. OR
xv. publicResponseEntity<T>TgetForEntity(Stringurl,Class<T>responseType,Object...urlVariables)throwsRestClientException
xvi. &thenusegetBody(),getHeaders(),getStatusCode(),togetcompletedetails
xvii. Canhandleincatchblock--o.s.web.client.HttpClientErrorException
xviii. &usegetStatusCode()&
xix. publicStringgetResponseBodyAsString()--togetcompleteerrdetails
xx. T--responsetypeexpectedfromRESTservice.
xxi. eg:template.getForObject,("http://localhost:7070/rest_server/bank/accts/{id}/{pin}",BankAccount.class,101,"1234");
xxii. 1.5getForObjectcanbereplacedbygetForEntityalso,togetentireresponseconsistingofstatuscode,headers+body.
xxiii. HowtoavoidhardcodingofRESTURLsinyourcode?
xxiv. 1.Createapropertyfile(mapofkey&valuepairs)under<resources>&addRESTURLs.Nospecificpropertyfilename.
xxv. eg:
xxvi. REST_GET_URL=http://host:port/contextpath/resource/{var1}/{var2}/....
xxvii. REST_POST_URL=http://host:port/contextpath/resource
xxviii. 2.Supplylocationofapppropertiesfileinmasterconfigurationfile(spring-servlet.xml)--addutilnamespace.
xxix. <util:propertiesid="props"location="classpath:/app.properties"/>

xxx. 3.InRestClient,useSpEL(springexpressionlanguage),toinjectvalueofthepropertyinurl.
xxxi. eg:
xxxii. @Value("#{props.REST_GET_URL}")
xxxiii. privateStringgetUrl;
xxxiv. eg:
xxxv. RestTemplateAPIusage
xxxvi. Customerc=template.getForObject(getrl,Customer.class,em,pass);

2.Forcreatingaresource
i. useHTTPmethod=post
ii. postAPIofRestTemplate
iii. 2.1public<T>TpostForObject(Stringurl,Objectrequest,Class<T>responseType,
iv. Object...uriVariables)throwsRestClientException
v. url-theURL
vi. request-theObjecttobePOSTed,maybenull
vii. responseType-thetypeofthereturnvalue
viii. uriVariables-thevariablestoexpandthetemplate
ix. Returns:
x. thecreatedobject
xi. CreatesanewresourcebyPOSTingthegivenobjecttotheURItemplate,andreturnstheresourcefoundintheresponse.
xii. url-theURL
xiii. request-theObjecttobePOSTed,maybenull
xiv. responseType-thetypeofthereturnvalue
xv. uriVariables-thevariablestoexpandthetemplate
xvi. Returns:
xvii. theconvertedobject
xviii. :
xix. eg:
xx. Customerc=template.postForObject(uri,c1,Customer.class);
xxi. 3.ForUpdatingaresource
xxii. publicvoidput(Stringurl,Objectrequest,Object...urlVariables)

```
            throws RestClientExceptio
n
```

Createsorupdatesaresource.

i. 4.Fordeletingaresource
ii. publicvoiddelete(Stringurl,Object...urlVariables)throwsRestClientException
iii. DeletestheresourcesatthespecifiedURI.

## RegardingAnnotations

i. @RestController:Spring42snew@RestControllerannotation.
ii. Itsacombinationof@Controllerand@ResponseBody.
iii. @RequestBody:Ifamethodparameterisannotatedwith@RequestBody,Springwillbindtheincoming HTTPrequestbody(fortheURLmentionedin@RequestMappingforthatmethod)tothatparameter.Whiledoingthat,Springwill[behindthescenes]useHTTPMessageconverterstoconverttheHTTPrequestbodyintodomainobject[deserializerequestbodytodomainobject],basedonACCEPTorContent-Typeheaderpresentinrequest.
iv. @ResponseBody:Ifamethodisannotatedwith@ResponseBody,Springwillbindthereturnvaluetooutgoing HTTPresponsebody.Whiledoingthat,Springwill[behindthescenes]useHTTPMessageconverterstoconvertthereturnvaluetoHTTPresponsebody[serializetheobjecttoresponsebody],basedonContent-TypepresentinrequestHTTPheader.Asalreadymentioned,inSpring4,noneedtousethisannotation.
v. org.springframework.http.ResponseEntity<T>classisareallypowerful.Itrepresentstheentire HTTPresponse.Goodthingaboutitisthatyoucancontrolanythingthatgoesintoit.Youcanspecifystatuscode,headers,andbody.Itcomeswithseveralconstructorstocarrytheinformationyouwanttosendin HTTPResponse.
vi. @PathVariableThisannotationindicatesthatamethodparametershouldbeboundtoa URItemplatevariable[theonein'{}'].(bindingbetweenrequesthandlingmethodparametere&URItemplatevariable)

vii. MediaType:With@RequestMappingannotation,youcanadditionally,specifytheMediaTypetobeproducedorconsumed(usingproducesorconsumesattributes)bythatparticularcontrollermethod,tofurthernarrowdownthemapping.
viii. Spring 4 DB Transactions
ix. Required JARS ---org.springframework.transaction & aop jars
x. Basics
xi. Benefits of Spring Transaction Management

```
* Very easy to use, does not require any underlying transaction API knowledge
* Your transaction management code will be independent of the transaction technology
* Both annotation- and XML-based configuration
* It does not require to run on a server - no server needed
```

## What is a Transaction?

i. A Transaction is a unit of work performed on the database and treated in a reliable way independent of other transaction. In database transaction processing ACID property refers to the Atomicity, Consistency, Isolation, Durability respectively.
ii. Atomicity- This property says that all the changes to the data is performed as if they form single operation. For example suppose in a bank application if a fund transfer from one account to another account the atomicity property ensures that is a debit is made successfully in one account the corresponding credit would be made in other account.
iii. Consistency- The consistency property of transaction says that the data remains in the consistence state when the transaction starts and ends. for example suppose in the same bank account, the fund transfer from one account to another account, the consistency property ensures that the total value(sum of both account ) value remains the same after the transaction ends.
iv. Isolation- This property says that, the intermediate state of transaction are hidden/ invisible to another transaction process.
v. Durability- The Durability says that when the transaction is completed successfully, the changes to the data persist and are not un-done, even in the event of system failure.A transaction is not considered durable until it commits. A system failure entails a database recovery, which includes a

rollback procedure for all uncommitted transactions, ultimately leaving the database in a consistent state.

vi. Transaction Handling
vii. Now, in Java you can handle transactions with plain SQL, with plain JDBC (a bit higher level), using Hibernate (or any other ORM library), or on an even higher level - with EJB or, finally, Spring!
viii. EJBs require an application server, but spring based jdbc application doesn't.
ix. Ways of Transaction Handling
x. Programmatic vs. Declarative
xi. Spring offers two ways of handling transactions: programmatic and declarative. If you are familiar with EJB transaction handling, this corresponds to bean-managed and container-managed transaction management.
xii. Programmatic means you have transaction management code surrounding your business code. That gives you extreme flexibility, but is difficult to maintain and too much of boilerplate code.
xiii. Declarative means you separate transaction management from the business code. You only use annotations or XML based configuration.
xiv. As a summary

```
* programmatic management is more flexible during development ti
me but less flexible during application life
* declarative management is less flexible during development tim
e but more flexible during application life
```

## Global transactions Vs Local Transactions

i. Global transactions enable you to work with multiple transactional resources, typically multiple relational databases . The application server manages global transactions through the JTA. (complex to use through UserTransaction object)
ii. Local Transactions
iii. Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have significant disadvantages as they cannot work across multiple transactional resources.
iv. Spring's solution
v. Spring resolves the disadvantages of global and local transactions. It enables application developers to use a consistent programming model in

any environment. You write your code once, and it can benefit from different transaction management strategies in different environments. It supports both declarative and programmatic transaction management. Most users prefer declarative transaction management.

vi. API details
    1. The key to the Spring transaction abstraction is the notion of a transaction strategy. Its the central interface in Spring's transaction infrastructure. A transaction strategy is defined by the org.springframework.transaction.PlatformTransactionManager
    2. interface: which has TransactionStatus getTransaction(TransactionDefinition td) throws TransactionExc
    3. TransactionException --- As in Spring's philosophy, the TransactionException that is thrown by any of the PlatformTransactionManager interface's methods , is unchecked Transaction infrastructure failures are generally fatal , managed by spring Tx frmwork & developer is NOT forced to handle this.

vii. The TransactionDefinition interface specifies:

viii. i.e @Transactional annotation supports 5 parameters.

ix. Isolation: The degree to which this transaction is isolated from the work of other transactions.

x. Concurrent transactions cause problems that might be difficult to investigate.

```
* Lost update – if two transactions are updating different colum
ns of the same row, then there is no conflict. The second update
blocks until the first transaction is committed and the final re
sult reflects both update changes.
```

BUT if the two transactions want to change the same columns, the second transaction will overwrite the first one, therefore loosing the first transaction update.

i. * Dirty read - reading changes that are not yet committed

ii. * Unrepeatable read - a transactions reads twice the same row, getting different data each time

iii. * Phantom read - similar to the previous one, except that the number of rows changed

iv. Now, the perfect solution to these problems is maximum isolation, but in reality this would cost too much resources and could lead to deadlocks. So, instead, you will set one of five isolation levels (where the fifth one is actually the maximum isolation level):

v. Supported levels

vi. ISOLATION_DEFAULT -- Use the default isolation level of the underlying datastore.
vii. ISOLATION_READ_UNCOMMITTED --- Indicates that dirty reads, non-repeatable reads and phantom reads can occur.
viii. ISOLATION_READ_COMMITTED --- Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
ix. ISOLATION_REPEATABLE_READ -- Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
x. ISOLATION_SERIALIZABLE -- Indicates that dirty reads, non-repeatable reads and phantom reads are prevented.

## Transaction Propagation

i. Whenever a one transactional method calls other transactional mehod , a decision is made - what to do with the transaction. Create a new one? Use an existing one if it exists, otherwise create a new one? Use an existing one only if it exists, otherwise fail?
ii. Supported behaviors ---
iii. MANDATORY --Supports a current transaction; throws an exception if no current transaction exists.
iv. REQUIRED -- default behavior.
v. Supports a current transaction; creates a new one if none exists.
vi. NESTED ---Executes within a nested transaction if a current transaction exists, otherwise same as REQUIRED
vii. SUPPORTS
viii. Supports a current transaction; executes non-transactionally if none exists.
ix. REQUIRES_NEW
x. Creates a new transaction, suspending the current transaction if one exists.
xi. NEVER
xii. Does not support a current transaction; throws an exception if a current transaction exists.
xiii. NOT_SUPPORTED
xiv. Does not support a current transaction; always executes non-transactionally.
xv. Timeout: in seconds .How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.
xvi. Default value = -1 , indefinite w/o time out
xvii. Otherwise specify value

xviii. eg
xix. @Transactional(timeout=100)
xx. Read-only status: A read-only transaction can be used when your code reads but does not modify data.
xxi. Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.
xxii. eg -- @Transactional(readOnly = true)
xxiii. default is false.
xxiv. Rollback behavior
xxv. With Spring transaction management the default behavior for automatic rollback is this: Only unchecked exceptions cause a rollback. Unchecked exceptions are RuntimeExceptions and Errors.
xxvi. But can be changed.
xxvii. eg --
@Transactional(rollbackFor = IOException.class, noRollbackFor = RuntimeException.class)
i. public void doSomething(...)

Implementation steps & concept for annotation based declarative transaction management.

1. For plain JDBC implementations of PlatformTransactionManager --- use DataSourceTransactionManager --- implementation class for a single JDBC DataSource.
2. Declare the same in spring configuration xml file.
3. eg --

i. p:dataSource-ref="dataSource">

1. To enable annotated transaction support , add transaction namespace(tx) & add following
2. 
3. Note : can even skip attribute transaction-manager, if id of Tx Mgr bean is transactionManager

ii. This completes configuration steps

1. In service layer beans (typically annotated with @Service) , add method level annotation @Transactional along with suitable properties

For Maven web app with hibernate(POM.xml support for ---Spring MVC/ AOP/Core/REST/Hibernate)

# 1.Create Maven Project

i. New -- Maven Project ---

ii. Check Create Simple Project (skip archetype selection)
iii. Choose WAR
iv. Check Use default workspace location
v. Next
vi. Group ID -- reversed domain name (eg : if domain name is www.serverside.com , then it can be com.serverside )
vii. Artifact ID -- Name of the WAR File
viii. Name -- Testing web app with hibernate
ix. Finish
    1. Modify pom.xml , to add
    2. 2.1 Properties --
    3. 2.2 Build plugins -- 2 plugins (maven-compiler-plugin & maven-eclipse-plugin)
    4. 2.3 Dependencies for --mysql , JUnit , spring , hibernate & json.
    5. Project -- R Click -- Maven --Update Project
    6. Check project structure for simple Java web application project.
    7. src/main/java -- Java sources
    8. src/test/java -- JUnit Test Cases
    9. src/main/resources --configuration files
    10. src/test/resources --configuration files for testing
    11. src/main/webapp --root of web application(equivalent to WebContent)
    12. Choose Java EE perspective
    13. Project -- R Click --Java EE Tools --Generate dep desc stub
    14. This will create src/main/webapp/WEB-INF/web.xml
x. Add welcome page as "index.jsp" in web.xml
    1. Project -- R Click --Properties --Targeted Runtimes -- Choose Tomcat 8
    2. Create packaged classes under src/main/java.
    3. utils,pojos,dao,beans,listeners
xi. 7.5 Copy log4j.properties,database.properties,hibernate-ersistence.xml(i.e all configuration files) under src/main/resources
    1. Project -- R Click -- Maven Build --goals --clean install
xii. 8.5 In case of any errors(red cross!)
xiii. Project -- R Click -- Maven --Update Project
    1. Project -- R Click --- Run on server ---run
xiv. mysql
xv. mysql-connector-java
xvi. 8.0.11
xvii.

## What is Maven ?

i. Build automation tool for overall project management.
ii. It helps in
    1. checking a build status
    2. generating reports (basically javadocs)
    3. setting up the automated build process and monitors the same.
iii. It eases out source code compilation, distribution, documentation, collaboration with different teams .
iv. 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.
v. Vendor -- Apache
vi. Earlier build tool -- Ant
vii. 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.
3. No such support in ant.
4. There is no life cycle in Ant, where as life cycle exists in Maven.

## Maven advantages

1. Managing dependencies
2. Uses Convention over configuration - configuration is very minimal
3. Multiple/Repeated builds can be achieved.
4. Plugin management.
5. Testing - ability to run JUnit and other integration test suites.

i. What is POM? (Project Object Model)
ii. It is the core element of any maven project.
iii. Any maven project consists of one configuration file called pom.xml.
iv. Location --In the root directory of any maven project.
v. It contains the details of the build life cycle of a project.
vi. Contents
vii. Dependencies used in the projects (Jar files)
viii. Plugins used

ix. Project version
x. Developers involved in the project
xi. Build profiles etc.
xii. 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. 
7. packaging -- defines packaging type such as jar, war etc.
8. name -- defines name of the maven project.
9. plugins ---compiler plugins , eclipse plugins
10. dependencies -- collection of dependencies for this project.
11. Witin that --
12. dependency -- defines a specific dependency.(eg : hibernate dependency,spring web)
13. scope -- defines scope for this maven project. It can be compile, provided, runtime, test and system.

i. Goals in Maven
ii. 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.
iii. eg : clean , build ,install ,test
iv. What is a Maven Repository
v. 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:

i. Local repository then Central repository then Remote repository.
ii. maven repositories
iii. If dependency is not found in these repositories, maven stops processing and throws an error.
    1. Maven Local Repository
iv. Maven local repository is located in the file local system. It is created by the maven when you run any maven command.
v. By default, maven local repository is HOME / .m2 directory.
vi. (Can be updated by changing the MAVEN_HOME/conf/settings.xml)
vii. 2) Maven Central Repository
viii. Maven central repository is located on the web(Created by the apache maven community)
ix. The path of central repository is: https://mvnrepository.com/repos/central
x. 3) Maven Remote Repository
xi. 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

i. What is it ?
ii. 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.
iii. Maven comes with 3 built-in build life cycles
iv. Clean - this phase involves cleaning of the project (for a fresh build & deployment)
v. Default - this phase handles the complete deployment of the project
vi. Site - this phase handles the generating the java documentation of the project.
vii. Build Profiles in Maven
viii. It is a subset of elements which allows to customize builds for particular environment. Profiles are also portable for different build environments.
ix. 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.

x. 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

xi. 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. 4.Verify maven
5. mvn -- version
6. OR use m2e plug-in (a standard part of Eclipse for J2EE)
7. If there multiple request params(use case -- register/update) --- bind POJO directly to a form.
8. How ?
9. 1.1 For loading the form (in showForm method of the controller) , bind empty POJO (using def constr) in model map
10. How ?
11. Explicit --add Model as dependency & u add
12. map.addAttribute(nm,val)
13. OR better way
14. implicit -- add POJO as a dependency
15. eg : User registration
16. @GetMapping("/reg")
17. public String showForm(User u) {...}

## What will SC do ?

i. SC --- User u=new User();

ii. chks --- Are there any req params coming from client ? --- typically --no --- getter --

iii. adds pojo as model attr (in Model map)

iv. map.addAttribute("User",new User());

v. 1.2 In form (view ---jsp) -use spring form tags along with modelAttribute

vi. 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)

vii. 1.3 Upon form submission (clnt pull l)
viii. clnt sends a new req --- containing req params
ix. @PostMapping("/reg")
x. public String processForm(User u,RedirectAttributes flashMap,HttpSession hs) {
xi. //SC calls
xii. User u=new User();
xiii. SC invokes MATCHING (req param names --POJO prop setters)
xiv. setters. -- conversational state is transferred to Controller.
xv. adds pojo as model attr (in Model map)
xvi. map.addAttribute(nm,val)
xvii. Thus you get a populated POJO directly in controller w/o calling & w/o using any java bean.

## Transactions & Locking

1. Change tx isolation level --to REPEATABLE_READ
2. OR
3. 2 Use "select for update" Query
4. Both of above approach applies a write lock

i. Better approach --Optismistic Locking
ii. JPA 2 supports both optimistic locking and pessimistic locking. Locking is essential to avoid update collisions resulting from simultaneous updates to the same data by two concurrent users. Locking in JPA) is always at the database object level, i.e. each database object is locked separately.
iii. Optimistic locking is applied on transaction commit. Any database object that has to be updated or deleted is checked. An exception is thrown if it is found out that an update is being performed on an old version of a database object, for which another update has already been committed by another transaction.
iv. Optimistic locking should be the first choice for most applications, since compared to pessimistic locking it is easier to use and more efficient.
v. In the rare cases in which update collision must be revealed earlier (before transaction commit) pessimistic locking can be used. When using pessimistic locking, database objects are locked during the transaction and lock conflicts, if they happen, are detected earlier.

## Optismistic Locking

i. Add @Version annotated property in hibernate POJO.(data type Integer)

ii. The initial version of a new entity object (when it is stored in the database for the first time) is 1. In every transaction in which an entity object is modified its version number is automatically increased by one.
iii. During commit , hibernate checks every database object that has to be updated or deleted, and compares the version number of that object in the database to the version number of the in-memory object being updated. The transaction fails and an OptimisticLockException is thrown if the version numbers do not match, indicating that the object has been modified by another user (using another transaction) since it was retrieved by the current updater.
iv. Pessimistic Locking
v. The main supported pessimistic lock modes are:
vi. PESSIMISTIC_READ - which represents a shared lock.
vii. PESSIMISTIC_WRITE - which represents an exclusive lock.
viii. Setting a Pessimistic Lock
ix. An entity object can be locked explicitly by the lock method:
x. org.hibernate.Session API
xi. void lock(Object object,LockMode lockMode)
xii. Obtain the specified lock level upon the given object. This may be used to perform a version check (LockMode.READ) or to upgrade to a pessimistic lock (LockMode.PESSIMISTIC_WRITE)
xiii. eg :

sf.getCurrentSession().lock(employee, LockMode.PESSIMISTIC_WRITE);

1. Revise REST & complete CRUD operations.
2. refer to --spring-restful-sequence
3. refer to --for SOAP vs REST
4. Get existing resource
5. --@GetMapping & can add @PathVariable ---for adding URI variables.
6. Create new resource
7. Use @PostMapping as method level annotation + @RequestBody on method argument --for SC to auto perform un marshalling from xml/json ---> Java object(POJO)

i. 3.Better practise :
ii. Instead of just returning response data , return ResponseEntity -- to encapsulate -- sts code, hdr/s , response data

1. Update existing resource.
2. Delete existing stock details
3. i/p stock id

rets -- string --- stock not found / stock deleted successfully.

# What is Maven & why Maven ?

1. refer to readme(exam objective)
2. Maven demo -- To create spring-mvc-hibernate based maven project (from scratch)
3. Explain configuration files.
4. hibernate-persistence.xml
5. Understanding Transaction Management in Spring

i. How to automate Tx management in spring?
   1. Add spring supplied tx manager bean in config file
   2. Enable tx annotation support
   3. Use @Transactional attribute typically in Service or DAO Layer.

ii. 3.5 Refer to diag --spring-tx-management
   1. How to customize tx management -- using @Transactional attributes
   2. 4.1 timeout
   3. eg : @Transactional(timeout=100)
   4. service/dao layer method
   5. 4.2 readOnly --
   6. def value --false;
   7. eg : @Transactional(readOnly=true)
   8. 4.3
   9. @Transactional(rollbackFor = IOException.class, noRollbackFor = RuntimeException.class)
   10. public void doSomething(...)

iii. 4.4 Tx propagation level

iv. 4.5 Tx isolation level

v. later --refer to shared article & "regarding transactions locking" readme & diag

vi. tx-isolation levels.
   1. (Post-Redirect-Get pattern --double submit guard --ref : PostRedirectGet_DoubleSubmitProblem.png)

vii. How to replace default forward view by redirect view?

viii. In servlet -- you have to write --

ix. response.sendRedirect(response.encodeRedirectURL("/day12.2/user/details"));

x. In spring MVC --- return "redirect:/user/details";
   1. HttpSession tracking in spring MVC
   2. ---simply add HttpSession as one of the dependencies(as method arg) in req handling method

3. How to add link href or form action ?
4. Use spring:url tag (by importing spring supplied JSP tag library)
5. response.encodeUrl("/user/logout")

URL http://host:port/day12/user/logout;jsessionid=135436543.

1. How to redirect client to next page after some delay ?
2. API of HttpServletResponse
3. void setHeader(String name,String value)
4. eg : response.setHeader("refresh","10;url="+request.getContextPath());
5. Refer to a readme --"regarding form binding"
6. What is RedirectAttributes ?
7. o.s.w.s.mvc.support.RedirectAttributes --i/f
8. --map of flash scoped attributes
9. Use case --- redirect scenario
10. To remember the attrs till the next request only
11. API
12. public RedirectAttributes addFlashAttributes(String nm,Object val)

i. How to get this flash map (=holder of redirect attrs/flash scoped attrs)?
ii. As a dependency from SC (add it as method argument in req handling method of the controller)

1. P.L Validations in spring MVC hibernate -- vendors project. (day 13.1)
2. Hibernate -- remaining Session API.
3. ref project -- hibernate_session_api
4. Many-to-many demo
5. ref project --spring_hib_javase_many_to_many_bi_dir
6. Inheritance in hibernate
7. ref project -- spring_hib_javase_inheritance
8. Enter Spring Boot

## Project Tip

How to avoid writing multiple of such forward to view type methods?(global mapping)

i. @GetMapping("/{viewName}")
ii. public String forwardToView(@PathVariable viewName)
iii. {
iv. return "/cust/"+viewName;
v. }

Here , request url pattern MUST MATCH with view page name.

i. SOAP vs REST web services

ii. SOAP stands for simple object access protocol
iii. REST stands for REpresentational State Transfer
iv. Protocol vs Architectural style
v. SOAP is a standard protocol to create web services
vi. Rest is architectural style to create web services.
vii. Contract
viii. Client and Server are bind with WSDL contract in SOAP
ix. There is no contract between client and Server in REST
x. Format Support
xi. SOAP supports only XML format
xii. REST web services supports XML, json and plain text etc.
xiii. Maintainability
xiv. SOAP web services are hard to maintain as if we do any changes in WSDL , we need to create client stub again
xv. REST web services are generally easy to maintain.
xvi. Service interfaces vs URI
xvii. SOAP uses Service interfaces to expose business logic
xviii. Rest uses URI to expose business logic
xix. Security
xx. SOAP has its own security : WS-security
xxi. Rest inherits its security from underlying transport layer.
xxii. Bandwidth
xxiii. SOAP requires more bandwidth and resources as it uses XML messages to exchange information
xxiv. REST requires less bandwith and resources. It can use JSON also.
xxv. Learning curve
xxvi. SOAP web services are hard to learn as you need to understand WSDL , client stub.
xxvii. REST web services are easy to understand as you need to annotate plain java class with JAX-RS annotations to use various HTTP methods .
xxviii. Spring 4 DB Transactions
xxix. Required JARS ---org.springframework.transaction & aop jars
xxx. Basics
xxxi. Benefits of Spring Transaction Management

```
* Very easy to use, does not require any underlying transaction API knowledge
* Your transaction management code will be independent of the transaction technology
```

```
* Both annotation- and XML-based configuration
* It does not require to run on a server - no server needed
```

## What is a Transaction?

i. A Transaction is a unit of work performed on the database and treated in a reliable way independent of other transaction. In database transaction processing ACID property refers to the Atomicity, Consistency, Isolation, Durability respectively.
ii. Atomicity- This property says that all the changes to the data is performed as if they form single operation. For example suppose in a bank application if a fund transfer from one account to another account the atomicity property ensures that is a debit is made successfully in one account the corresponding credit would be made in other account.
iii. Consistency- The consistency property of transaction says that the data remains in the consistence state when the transaction starts and ends. for example suppose in the same bank account, the fund transfer from one account to another account, the consistency property ensures that the total value(sum of both account ) value remains the same after the transaction ends.
iv. Isolation- This property says that, the intermediate state of transaction are hidden/ invisible to another transaction process.
v. Durability- The Durability says that when the transaction is completed successfully, the changes to the data persist and are not un-done, even in the event of system failure.A transaction is not considered durable until it commits. A system failure entails a database recovery, which includes a rollback procedure for all uncommitted transactions, ultimately leaving the database in a consistent state.
vi. Transaction Handling
vii. Now, in Java you can handle transactions with plain SQL, with plain JDBC (a bit higher level), using Hibernate (or any other ORM library), or on an even higher level - with EJB or, finally, Spring!
viii. EJBs require an application server, but spring based jdbc application doesn't.
ix. Ways of Transaction Handling
x. Programmatic vs. Declarative
xi. Spring offers two ways of handling transactions: programmatic and declarative. If you are familiar with EJB transaction handling, this

corresponds to bean-managed and container-managed transaction management.

xii. Programmatic means you have transaction management code surrounding your business code. That gives you extreme flexibility, but is difficult to maintain and too much of boilerplate code.

xiii. Declarative means you separate transaction management from the business code. You only use annotations or XML based configuration.

xiv. As a summary

```
* programmatic management is more flexible during development time but less flexible during application life
* declarative management is less flexible during development time but more flexible during application life
```

## Global transactions Vs Local Transactions

i. Global transactions enable you to work with multiple transactional resources, typically multiple relational databases . The application server manages global transactions through the JTA. (complex to use through UserTransaction object)

ii. Local Transactions

iii. Local transactions are resource-specific, such as a transaction associated with a JDBC connection. Local transactions may be easier to use, but have significant disadvantages as they cannot work across multiple transactional resources.

iv. Spring's solution

v. Spring resolves the disadvantages of global and local transactions. It enables application developers to use a consistent programming model in any environment. You write your code once, and it can benefit from different transaction management strategies in different environments. It supports both declarative and programmatic transaction management. Most users prefer declarative transaction management.

vi. API details

1. The key to the Spring transaction abstraction is the notion of a transaction strategy. Its the central interface in Spring's transaction infrastructure. A transaction strategy is defined by the org.springframework.transaction.PlatformTransactionManager
2. interface: which has TransactionStatus getTransaction(TransactionDefinition td) throws TransactionExc

3. TransactionException --- As in Spring's philosophy, the TransactionException that is thrown by any of the PlatformTransactionManager interface's methods , is unchecked Transaction infrastructure failures are generally fatal , managed by spring Tx frmwork & developer is NOT forced to handle this.

vii. The TransactionDefinition interface specifies:
viii. i.e @Transactional annotation supports 5 parameters.
ix. Isolation: The degree to which this transaction is isolated from the work of other transactions.
x. Concurrent transactions cause problems that might be difficult to investigate.

```
* Lost update – if two transactions are updating different colum
ns of the same row, then there is no conflict. The second update
blocks until the first transaction is committed and the final re
sult reflects both update changes.
```

BUT if the two transactions want to change the same columns, the second transaction will overwrite the first one, therefore loosing the first transaction update.

i. * Dirty read - reading changes that are not yet committed
ii. * Unrepeatable read - a transactions reads twice the same row, getting different data each time
iii. * Phantom read - similar to the previous one, except that the number of rows changed
iv. Now, the perfect solution to these problems is maximum isolation, but in reality this would cost too much resources and could lead to deadlocks. So, instead, you will set one of five isolation levels (where the fifth one is actually the maximum isolation level):
v. Supported levels
vi. ISOLATION_DEFAULT -- Use the default isolation level of the underlying datastore.
vii. ISOLATION_READ_UNCOMMITTED --- Indicates that dirty reads, non-repeatable reads and phantom reads can occur.
viii. ISOLATION_READ_COMMITTED --- Indicates that dirty reads are prevented; non-repeatable reads and phantom reads can occur.
ix. ISOLATION_REPEATABLE_READ -- Indicates that dirty reads and non-repeatable reads are prevented; phantom reads can occur.
x. ISOLATION_SERIALIZABLE -- Indicates that dirty reads, non-repeatable reads and phantom reads are prevented.

## Transaction Propagation

i. Whenever a one transactional method calls other transactional mehod , a decision is made - what to do with the transaction. Create a new one? Use an existing one if it exists, otherwise create a new one? Use an existing one only if it exists, otherwise fail?
ii. Supported behaviors ---
iii. MANDATORY --Supports a current transaction; throws an exception if no current transaction exists.
iv. REQUIRED -- default behavior.
v. Supports a current transaction; creates a new one if none exists.
vi. NESTED ---Executes within a nested transaction if a current transaction exists, otherwise same as REQUIRED
vii. SUPPORTS
viii. Supports a current transaction; executes non-transactionally if none exists.
ix. REQUIRES_NEW
x. Creates a new transaction, suspending the current transaction if one exists.
xi. NEVER
xii. Does not support a current transaction; throws an exception if a current transaction exists.
xiii. NOT_SUPPORTED
xiv. Does not support a current transaction; always executes non-transactionally.
xv. Timeout: in seconds .How long this transaction runs before timing out and being rolled back automatically by the underlying transaction infrastructure.
xvi. Default value = -1 , indefinite w/o time out
xvii. Otherwise specify value
xviii. eg
xix. @Transactional(timeout=100)
xx. Read-only status: A read-only transaction can be used when your code reads but does not modify data.
xxi. Read-only transactions can be a useful optimization in some cases, such as when you are using Hibernate.
xxii. eg -- @Transactional(readOnly = true)
xxiii. default is false.
xxiv. Rollback behavior
xxv. With Spring transaction management the default behavior for automatic rollback is this: Only unchecked exceptions cause a rollback. Unchecked

exceptions are RuntimeExceptions and Errors.

xxvi. But can be changed.

xxvii. eg --

xxviii. @Transactional(rollbackFor = IOException.class, noRollbackFor = RuntimeException.class)

xxix. public void doSomething(...)

Implementation steps & concept for annotation based declarative transaction management.

1. For plain JDBC implementations of PlatformTransactionManager --- use DataSourceTransactionManager --- implementation class for a single JDBC DataSource.
2. Declare the same in spring configuration xml file.
3. eg --

i. p:dataSource-ref="dataSource">

1. To enable annotated transaction support , add transaction namespace(tx) & add following
2. 
3. Note : can even skip attribute transaction-manager, if id of Tx Mgr bean is transactionManager

ii. This completes configuration steps

1. In service layer beans (typically annotated with @Service) , add method level annotation @Transactional along with suitable properties.

iii. In many cases, the host that serves the JS --front end (e.g. example.com) is different from the host that serves the data --back end (e.g. api.example.com). In such a case, CORS(Cross origin resource sharing) enables the cross-domain communication.

iv. Spring provides excellent support for CORS, as an easy and powerful way of configuring it in any Spring or Spring Boot web application.

v. The support can be enabled by simply adding @CrossOrigin annotation at RestController class level or can be added at method level.

vi. eg : @CrossOrigin(origins = "http://localhost:4200")

vii. @RestController

viii. @RequestMapping("/flights")

ix. public class FlightController {...}

x. By def -- all origins are allowed.

xi. Interview question

xii. Why have you used spring framework in your project ?

xiii. Spring : Design Patterns Used in Java Spring Framework

1.

i. Dependency injection/ or IoC (inversion of control) – Is the main principle behind loose coupling of layers.
    1. Factory – Spring uses factory pattern to create objects of beans using Application Context reference
    2. eg : refer to eg code in spring & Java SE (ApplicationContext & its getBean method)
    3. Proxy – used heavily in AOP, and remoting.
    4. eg : @Transactional class is proxied by spring as an example of AOP proxy
    5. Singleton – By default, beans defined in spring config file (xml) are only created once. No matter how many calls were made using getBean() method, it will always have only one bean. This is because, by default all beans in spring are singletons.
    6. This can be overridden by using Prototype bean scope.Then spring will create a new bean object for every request.
    7. Model View Controller – The advantage with Spring MVC is that your controllers are POJOs as opposed to being servlets. This makes for easier testing of controllers. One thing to note is that the controller is only required to return a logical view name, and the view selection is left to a separate ViewResolver. This makes it easier to reuse controllers for different view technologies.
    8. Front Controller – Spring provides DispatcherServlet to ensure an incoming request gets dispatched to your controllers.
    9. View Helper – Spring has a number of custom JSP tags, and view resolvers, to assist in separating code(B.L) from presentation in views.
    10. Template method – used extensively to deal with boilerplate repeated code (such as closing connections cleanly, etc..). For example JdbcTemplate, JmsTemplate,JdbcTemplate,JpaTemplate,RestTemplateHibernate API
    11. SessionFactory API
    12. getCurrentSession vs openSession
ii. public Session openSession() throws HibernateExc
iii. opens new session from SF,which has to be explicitely closed by prog.
iv. public Session getCurrentSession() throws HibernateExc
v. Opens new session , if one doesn't exist , otherwise continues with the exisitng one.
vi. 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
2. persist ---
3. public void persist(Object transientRef)

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

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

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

1. public Serializable save(Object ref)
2. save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exc.
3. Ignores ur passed id & creates its own id & inserts a row.
4. saveOrUpdate
5. public void saveOrUpdate(Object ref)
6. --either inserts/updates or throws exc.
7. null id -- fires insert (works as save)
8. non-null BUT existing id -- fires update (works as update)
9. 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

i. merge

ii. public Object merge(Object ref)

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

iv. O/P --Rets PERSISTENT POJO ref.

v. null id -- fires insert (works as save)

vi. non-null BUT existing id -- fires update (select , update)

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

1. get vs load
2. & LazyInitilalizationException.
3. update
4. Session API
5. public void update(Object object)
6. Update the persistent instance with the identifier of the given detached instance.
7. I/P --detached POJO containing updated state.
8. Same POJO becomes persistent.

viii. Exception associated :
    1. org.hibernate.TransientObjectException: The given object has a null identifier:
    2. i.e while calling update if u give null id. (transient ----X ---persistent via update)
    3. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.
    4. 3.
    5. org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session
    6. public Object merge(Object ref)
    7. Can Transition from transient -->persistent & detached --->persistent.
    8. Regarding Hibernate merge
    9. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().
    10. API of Session
    11. Object merge(Object object)
    12. 3.
    13. Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
    14. 3.If there is no persistent instance currently associated with the session, it will be loaded.
    15. 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.
    16. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.
ix. 7.public void evict(Object persistentPojoRef)
x. It detaches a particular persistent object
xi. detaches or disassociates from the session level cache(L1 cache)
xii. (Remove this instance from the session cache. Changes to the instance will not be synchronized with the database. )
    1.
    2. void clear()
xiii. When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.
xiv. But Databse Connection is not returned to connection pool.
xv. (Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)

1. void close()

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

1. void flush()

xviii. When the object is in persistent state , whatever changes we made to the object
xix. state will be reflected in the databse only at the end of transaction.
xx. BUT If we want to reflect the changes before the end of transaction
xxi. (i.e before commiting the transaction )
xxii. call the flush method.
xxiii. (Flushing is the process of synchronizing the underlying DB state with persistable state of session cache )

1. boolean contains(Object ref)

xxiv. The method indicates whether the object is
xxv. associated with session or not.(i.e is it a part of l1 cache ?)
xxvi. 12.
xxvii. void refresh(Object ref) -- ref --persistent or detached
xxviii. This method is used to get the latest data from database and make
xxix. corresponding modifications to the persistent object state.
xxx. (Re-reads the state of the given instance from the underlying database

## Hibernate API

1. SessionFactory API
2. getCurrentSession vs openSession

i. public Session openSession() throws HibernateExc
ii. opens new session from SF,which has to be explicitely closed by prog.
iii. public Session getCurrentSession() throws HibernateExc
iv. Opens new session , if one doesn't exist , otherwise continues with the exisitng one.
v. 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)
2. API (method) of org.hibernate.Session
3. public Serializable save(Object o) throws HibernateException

vi. I/P ---transient POJO ref.
vii. save() method auto persists transient POJO on the DB(upon committing tx) & returns unique serializable ID generated by (currently) hib frmwork.

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

viii. Usage of Hibernate Session API's get()
ix. int id=101;
x. BookPOJO b1=hibSession.get(BookPOJO.class,id);
xi. BookPOJO b1=
(BookPOJO)hibSession.get(Class.forName("pojos.BookPOJO"),id);

1. Display all books info :

xii. 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.
xiii. (JPA--- Java Persistence API compliant syntax --- JPQL )
xiv. eg --- HQL --- "from BookPOJO"
xv. eg JPQL -- "select b from BookPOJO b"
xvi. 2.1 Create Query Object --- from Session i/f
xvii. org.hibernate.query.Query createQuery(String queryString,Class resultType)
xviii. eg : Query q=hs.createQuery(hql/jpql,Book.class);
xix. 2.2. Execute query to get List of selected PERSISTENT POJOs
xx. API of org.hibernate.query.Query i/f
xxi. (Taken from javax.persistence.TypedQuery)
xxii. List getResultList()
xxiii. Execute a SELECT query and return the query results as a generic List.
xxiv. T -- type of POJO / Result
xxv. eg : hs,tx
xxvi. String jpql="select b from Book b";
xxvii. try {
xxviii. List l1=hs.createQuery(jpql,Book.class).getResultList();
xxix. }
xxx. Usage ---
xxxi. String hql="select b from BookPOJO b";
xxxii. List l1=hibSession.createQuery(hql).getResultList();

1. Passing IN params to query. & execute it.
2. Objective : Display all books from specified author , with price < specified price.

3. API from org.hibernate.query.Query i/f

xxxiii. Query setParameter(String name,Object value)

xxxiv. Bind a named query parameter using its inferred Type.

xxxv. name -- query param name

xxxvi. value -- param value.I

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

xxxviii. How to set IN params ?

xxxix. org.hibernate.query.Query API

xl. public Query setPrameter(String pName,Object val)

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

xlii. Objective --Offer discount on all old books

xliii. i/p -- date , disc amt

xliv. 4.Updating POJOs --- Can be done either with select followed by update or ONLY with update queries(following is eg of 2nd option--Bulk update scenario)

xlv. Objective : dec. price of all books with author=specified author.

xlvi. String jpql = "update BookPOJO b set b.price = b.price - :disc where b.author = :au and b.publishDate < :dt ";

xlvii. set named In params

xlviii. exec it (executeUpdate) ---

xlix. int updateCount= hs.createQuery(hql).setParameter("disc", disc).setParameter("dt", d1).executeUpdate();

l. ---This approach is typically NOT recommended often, since it bypasses L1 cache . Cascading is not supported. Doesn't support optismistic locking directly.

1. Delete operations.
2. API of org.hibernate.Session
3. --void delete(Object object) throws HibernateException
4. ---POJO is marked for removal , corresponding row from DB will be deleted after comitting tx & closing of session.

li. OR

## 5.5

i. One can use directly "delete HQL" & perform deletions.(Bulk delete)

ii. eg

iii. int deletedRows = hibSession.createQuery ("delete Subscription s WHERE s.subscriptionDate < :today").setParameter ("today", new Date ()).executeUpdate ();
iv. API of org.hibernate.query.Query
    1. Iterator iterate() throws HibernateException
v. Return the query results as an Iterator. If the query contains multiple results per row, the results are returned Object[].
vi. Entities returned --- in lazy manner

## Pagination

1. Query setMaxResults(int maxResults)
2. 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)
4. Set the first row to retrieve. If not set, rows will be retrieved beginnning from row 0. (NOTE row num starts from 0)

i. eg --- List l1=sess.createQuery("select c from Customer c",Customer.class).setFirstResult(30).setMaxResults(10).getResultList();
ii. 4.How to count rows & use it in pagination techniques?
iii. int pageSize = 10;
iv. String countQ = "Select count (f.id) from Foo f";
v. Query countQuery = session.createQuery(countQ);
vi. Long countResults = (Long) countQuery.uniqueResult();
vii. 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();
```

1. org.hibernate.query.Query API

T getSingleResult()

i. Executes a SELECT query that returns a single typed result.
ii. Returns: Returns a single instance(persistent) that matches the query.
iii. Throws:
iv. NoResultException - if there is no result
v. NonUniqueResultException - if more than one result
vi. IllegalStateException - if called for a Java Persistence query language UPDATE or DELETE statement

1. How to get Scrollable Result from Query?

vii. ScrollableResults scroll(ScrollMode scrollMode) throws HibernateException
viii. Return the query results as ScrollableResults. The scrollability of the returned results depends upon JDBC driver support for scrollable ResultSets.
ix. Then can use methods of ScrollableResults ---first,next,last,scroll(n) .
    1. How to create Named query from Session i/f?
    2. What is a named query ?
    3. 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.
x. Fail fast: Their syntax is checked when the session factory is created, making the application fail fast in case of an error.
xi. Reusable: They can be accessed and used from several places which increase re-usability.
xii. eg : In POJO class, at class level , one can declare Named Queries
xiii. @Entity
xiv. @NamedQueries
xv. ({@NamedQuery(name=DepartmentEntity.GET_DEPARTMENT_BY_ID, query="select d from DepartmentEntity d where d.id = :id")}
xvi. public class Department{....}
xvii. Usgae
xviii. Department d1 = (Department) session.getNamedQuery(DepartmentEntity.GET_DEPARTMENT_BY_ID).setInteger("id", 1);
    1. How to invoke native sql from hibernate?
    2. Query q=hs.createSQLQuery("select * from books").addEntity(BookPOJO.class);
    3. l1 = q.list();
    4. Hibernate Criteria API
    5. A powerful and elegent alternative to HQL
    6. Well adapted for dynamic search functionalities where complex Hibernate queries have to be generated 'on-the-fly'.
xix. Typical steps are -- Create a criteria for POJO, add restrictions , projections ,add order & then fire query(via list() or uniqueResult())
    1. For composite primary key
    2. Rules on prim key class
    3. Annotation -- @Embeddable (& NOT @Entity)

4. Must be Serializable.
5. Must implement hashCode & equals as per general contract.

xx. In Owning Entity class
xxi. Add usual annotation -- @Id.
xxii. 1.1 Testing core api
xxiii. persist ---
xxiv. public void persist(Object transientRef)
xxv. ---persists trasient POJO .
xxvi. if u give some non-null id (existing or non-existing) while calling persist(ref) --gives exc
xxvii. org.hibernate.PersistentObjectException: detached entity passed to persist:

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

1. public Serializable save(Object ref)
2. save --- if u give some non-null id(existing or non-existing) while calling save(ref) --doesn't give any exc.
3. Ignores ur passed id & creates its own id & inserts a row.
4. saveOrUpdate
5. public void saveOrUpdate(Object ref)
6. --either inserts/updates or throws exc.
7. null id -- fires insert (works as save)
8. non-null BUT existing id -- fires update (works as update)
9. 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

i. public Object merge(Object ref)
ii. I/P -- either transient or detached POJO ref.
iii. O/P --Rets PERSISTENT POJO ref.
iv. null id -- fires insert (works as save)
v. non-null BUT existing id -- fires update (select , update)
vi. non-null BUT non existing id -- no exc thrown --Ignores ur passed id & creates its own id & inserts a row.(select,insert)
    1. get vs load
    2. & LazyInitilalizationException.
    3. update
    4. Session API
    5. public void update(Object object)

6. Update the persistent instance with the identifier of the given detached instance.
7. I/P --detached POJO containing updated state.
8. Same POJO becomes persistent.

vii. Exception associated :

1. org.hibernate.TransientObjectException: The given object has a null identifier:
2. i.e while calling update if u give null id. (transient ----X ---persistent via update)
3. org.hibernate.StaleStateException --to indicate that we are trying to delete or update a row that does not exist.
4. 3.
5. org.hibernate.NonUniqueObjectException: a different object with the same identifier value was already associated with the session
6. public Object merge(Object ref)
7. Can Transition from transient -->persistent & detached --->persistent.
8. Regarding Hibernate merge
9. The state of a transient or detached instance may also be made persistent as a new persistent instance by calling merge().
10. API of Session
11. Object merge(Object object)

3.

1. Copies the state of the given object(can be passed as transient or detached) onto the persistent object with the same identifier.
2. 3.If there is no persistent instance currently associated with the session, it will be loaded.
3. 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.
4. will not throw NonUniqueObjectException --Even If there is already persistence instance with same id in session.

i. 7.public void evict(Object persistentPojoRef)
ii. It detaches a particular persistent object
iii. detaches or disassociates from the session level cache(L1 cache)
iv. (Remove this instance from the session cache. Changes to the instance will not be synchronized with the database. )

1.
2. void clear()

v. When clear() is called on session object all the objects associated with the session object(L1 cache) become detached.
vi. But Databse Connection is not returned to connection pool.
vii. (Completely clears the session. Evicts all loaded instances and cancel all pending saves, updates and deletions)
    1. void close()
viii. When close() is called on session object all
ix. the persistent objects associated with the session object become detached(l1 cache is cleared) and also closes the Database Connection.
    1. void flush()
x. When the object is in persistent state , whatever changes we made to the object
xi. state will be reflected in the databse only at the end of transaction.
xii. BUT If we want to reflect the changes before the end of transaction
xiii. (i.e before commiting the transaction )
xiv. call the flush method.
xv. (Flushing is the process of synchronizing the underlying DB state with persistable state of session cache )
    1. boolean contains(Object ref)
xvi. The method indicates whether the object is
xvii. associated with session or not.(i.e is it a part of l1 cache ?)
xviii. 12.

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

i. This method is used to get the latest data from database and make
ii. corresponding modifications to the persistent object state.
iii. (Re-reads the state of the given instance from the underlying database
iv. There are three inheritance mapping strategies defined in the hibernate:
v. 1.Table Per Hierarchy
    1. Table Per Concrete class
    2. Table Per Subclass
    3. Table Per Hierarchy
    4. In table per hierarchy mapping, single table is required to map the whole hierarchy, an extra column (known as discriminator column) is added to identify the class. But nullable values are stored in the table .
    5. Table Per Concrete class
    6. In case of table per concrete class, tables are created as per class. But duplicate column is added in subclass tables.
vi. 3.

## Table Per Subclass

i. In this strategy, tables are created as per class but related by foreign key. So there are no duplicate columns.Basics of java mail
ii. The JavaMail is an API that is used to compose, write and read electronic messages (emails).
iii. The JavaMail API provides protocol-independent and platform independent framework for sending and receiving mails.
iv. The javax.mail and javax.mail.activation packages contains the core classes of JavaMail API.
v. The JavaMail facility can be applied to many events. It can be used at the time of registering the user (sending notification such as thanks for your interest to my site), forgot password (sending password to the users email id), sending notifications for important updates etc. So there can be various usage of java mail api.
vi. Protocols used in JavaMail API
vii. There are some protocols that are used in JavaMail API.

```
SMTP
POP
IMAP
MIME
NNTP and others
```

## SMTP

i. SMTP is an acronym for Simple Mail Transfer Protocol. It provides a mechanism to deliver the email. We can use Apache James server, Postcast server, cmail server etc. as an SMTP server. But if we purchase the host space, an SMTP server is bydefault provided by the host provider. For example, my smtp server is mail.abc.com. If we use the SMTP server provided by the host provider, authentication is required for sending and receiving emails.
ii. POP
iii. POP is an acronym for Post Office Protocol, also known as POP3. It provides a mechanism to receive the email. It provides support for single mail box for each user. We can use Apache James server, cmail server etc. as an POP server. But if we purchase the host space, an POP server is bydefault provided by the host provider. For example, the pop server

provided by the host provider for my site is mail.abc.com. This protocol is defined in RFC 1939.

iv. IMAP

v. IMAP is an acronym for Internet Message Access Protocol. IMAP is an advanced protocol for receiving messages. It provides support for multiple mail box for each user ,in addition to, mailbox can be shared by multiple users. It is defined in RFC 2060.

vi. MIME

vii. Multiple Internet Mail Extension (MIME) tells the browser what is being sent e.g. attachment, format of the messages etc. It is not known as mail transfer protocol but it is used by your mail program.

viii. NNTP and Others

ix. There are many protocols that are provided by third-party providers. Some of them are Network News Transfer Protocol (NNTP), Secure Multipurpose Internet Mail Extensions (S/MIME) etc.

x. JavaMail Architecture

xi. The java application uses JavaMail API to compose, send and receive emails. The JavaMail API uses SPI (Service Provider Interfaces) that provides the intermediatory services to the java application to deal with the different protocols. Let's understand it with the figure given below:

xii. JavaMail API Architecture

xiii. JavaMail API Core Classes

xiv. There are two packages that are used in Java Mail API: javax.mail and javax.mail.internet package. These packages contains many classes for Java Mail API. They are:

```
javax.mail.Session class
javax.mail.Message class
javax.mail.internet.MimeMessage class
javax.mail.Address class
javax.mail.internet.InternetAddress class
javax.mail.Authenticator class
javax.mail.PasswordAuthentication class
javax.mail.Transport class
javax.mail.Store class
javax.mail.Folder class etc.
```

java send mail explaination

i. Before explaining the Java program to send email, I will explain the classes and methods used here from Java Mail API, that will provide some background knowledge about the stuff here. This code is configured to send email from Gmail to any email service provider by using Java programming language. If you wish to send email from any other email service provider, then you need to change host (d_host) and port number (d_port) corresponding to the service provider. This requires Mail.jar to be downloaded which can be done here
ii. Authenticator
iii. This class obtains authentication for network connection. Authentication can be done by means of providing username, password. When authentication is required, the system will invoke a method on the subclass getPasswordAuthentication() and this will query about the authentication to number of inherited methods and returns the result.
iv. MIME Message

```
            This class represents MIME style email message.  Thi
s class provides methods to set various stuffs for sending email
s. Some basic methods required for sending emails are setSubjec
t, setFrom, addRecipient, setText these are self-explanatory by
name.  Even bigger emails with HTML content can be sent through
setContent method. Emails can also be sent to multiple people th
rough addRecipients method. There are lot more interesting metho
ds inside MIME Message.
```

Transport.send

```
            Transport is an abstract class that that models a me
ssage transport.  Send is a static method that sends the message
to specified address along with data stored in MIME Message. If
any invalid email is found it returns SendFailedException
```

JavaMail.java - Download

i. package javamail;
ii. import javax.mail.*;
iii. *import javax.mail.internet.*;*
iv. import java.util.*;
v. public class JavaMail {
vi. String d_email = "fromAddress@gmail.com",
vii. d_password = "password", //your email password
viii. d_host = "smtp.gmail.com",

ix. d_port = "465",
x. m_to = "ToAddress", // Target email address
xi. m_subject = "Testing",
xii. m_text = "Hey, this is a test email.";

```
public JavaMail() {
    Properties props = new Properties();
    props.put("mail.smtp.user", d_email);
    props.put("mail.smtp.host", d_host);
    props.put("mail.smtp.port", d_port);
    props.put("mail.smtp.starttls.enable", "true");
    props.put("mail.smtp.auth", "true");
    //props.put("mail.smtp.debug", "true");
    props.put("mail.smtp.socketFactory.port", d_port);
    props.put("mail.smtp.socketFactory.class", "javax.net.ssl.SSLSocketFactory");
    props.put("mail.smtp.socketFactory.fallback", "false");
    try {
        Authenticator auth = new SMTPAuthenticator();
        Session session = Session.getInstance(props, auth);
        MimeMessage msg = new MimeMessage(session);
        msg.setText(m_text);
        msg.setSubject(m_subject);
        msg.setFrom(new InternetAddress(d_email));
        msg.addRecipient(Message.RecipientType.TO, new InternetAddress(m_to));
        Transport.send(msg);
    } catch (Exception mex) {
        mex.printStackTrace();
    }
}

```

```
26 public static void main(String[] args) {
27     JavaMail blah = new JavaMail();
28 }
29
30 private class SMTPAuthenticator extends javax.mail.Authenticator
   {
31     public PasswordAuthentication getPasswordAuthentication() {
32         return new PasswordAuthentication(d_email, d_password);
33     }
34 }
```

```
    }
```

i. Possible Exceptions:
ii. 1) Unknown SMTP host: smtp.gmail.com - This error might occur if you do not have a valid internet connection.
iii. 2) Could not connect to SMTP host: smtp.gmail.com - If you face this error, first disable your firewall and try running the code, still if you face this issue, then check your Antivirus settings.

Java receive mail explaination

i. Lets see how to read/retreive email in your inbox using Javamail API. Its very simple, this tutorial explains this step by step. If you have not configured Javamail API in your IDE, please download it from here and get it configured only then you will be able to run this program.
ii. As explained in the previous tutorial let me explain the classes and method used here, that will make us to understand the program in a better way. Below are the steps to read email :
iii. Step 1 - Define Protocol
iv. Step 2 - Get a session instance to read email
v. Step 3 - Access emails through store class
vi. Step 4 - Read Inbox
vii. Step 1 : Define Protocol
viii. props.setProperty("mail.store.protocol", "imaps")
ix. First we need to define the protocol for processing emails.
x. SMTP - is the protocol to send email
xi. POP3 - is the protocol to receive emails
xii. IMAP- IMAP is an acronym for Internet Message Access Protocol. Its an advanced protocol for receiving messages.

xiii. This property takes two parameters (key, Value) key is "mail.store.protocol" and its value is "imaps" since we are going to read email protocol is defined as "imaps"
xiv. Step 2 : Get a session instance to read email
xv. This property is used to get a session instance for reading email and its done as shown below in the code.
xvi. Session session = Session.getInstance(props, null);
xvii. Step 3 : Access emails through store class
xviii. Store - An abstract class that models a message store and its access protocol, for storing and retrieving messages. Store provides many common methods for naming stores, connecting to stores, and listening to connection events.
xix. We will be making use of connect(String host,String user,String password) method to connect to specified host and get access to Inbox.
xx. Store store = session.getStore();
xxi. store.connect("imap.gmail.com", "yourEmailId@gmail.com", "password");
xxii. Folder inbox = store.getFolder("INBOX");
xxiii. Then we need to open the required folder in Read mode.
xxiv. inbox.open(Folder.READ_ONLY);
xxv. Summary: So far we have created a session and connected to gmail host with our username and password and got read access to Inbox.
xxvi. Step 4 : Read Inbox
xxvii. We are almost done, now get access to your email using 'Message' class as shown below and typecast the content of the mail to Multipart to read the body of the email.
xxviii. Message msg = inbox.getMessage(1);
xxix. Here 1 indicates the first email received in your inbox and getMessageCount() will give you the number of emails in your inbox. So read the latest email use,
xxx. Message msg = inbox.getMessage(inbox.getMessageCount());
xxxi. import java.util.*;
xxxii. *import javax.mail.*;
xxxiii. public class ReadingEmail {
xxxiv. public static void main(String[] args) {
xxxv. Properties props = new Properties();
xxxvi. props.setProperty("mail.store.protocol", "imaps");
xxxvii. try {
xxxviii. Session session = Session.getInstance(props, null);
xxxix. Store store = session.getStore();

xl. store.connect("imap.gmail.com", "yourEmailId@gmail.com", "password");
xli. Folder inbox = store.getFolder("INBOX");
xlii. inbox.open(Folder.READ_ONLY);
xliii. Message msg = inbox.getMessage(inbox.getMessageCount());
xliv. Address[] in = msg.getFrom();
xlv. for (Address address : in) {
xlvi. System.out.println("FROM:" + address.toString());
xlvii. }
xlviii. Multipart mp = (Multipart) msg.getContent();
xlix. BodyPart bp = mp.getBodyPart(0);
l. System.out.println("SENT DATE:" + msg.getSentDate());
li. System.out.println("SUBJECT:" + msg.getSubject());
lii. System.out.println("CONTENT:" + bp.getContent());
liii. } catch (Exception mex) {
liv. mex.printStackTrace();
lv. }
lvi. }
lvii. }

## What is JNDI ?

i. Java Naming and Directory Interface (JNDI)
ii. J2EE API --- that provides naming and directory functionality to applications written using the Java. It is independent of any specific directory service implementation.
iii. Thus variety of directories(new, emerging, and already deployed) can be accessed in a common way.
iv. What is its basic use?
v. JNDI allows distributed applications to look up services in an abstract, resource-independent way.
vi. When it is used?
vii. The most common use case is to set up a database connection pool on a Java EE application server. Any application that's deployed on that server can gain access to the connections they need using the JNDI name java:comp/env/myPool without having to know the details about the connection.
viii. Advantage
ix. Applications don't have to change as they migrate between environments.
x. Refer to image
xi. Some of the common service providers

xii. Lightweight Directory Access Protocol (LDAP)
xiii. Common Object Request Broker Architecture (CORBA) Common Object Services (COS) name service
xiv. Java Remote Method Invocation (RMI) Registry
xv. So basically you create objects and register them on the directory services which you can later do lookup and execute operation on.

## JNDI Overview

i. JNDI is an API specified in Java technology that provides naming and directory functionality to applications written in the Java programming language. It is designed especially for the Java platform using Java's object model. Using JNDI, applications based on Java technology can store and retrieve named Java objects of any type. In addition, JNDI provides methods for performing standard directory operations, such as associating attributes with objects and searching for objects using their attributes.
ii. JNDI is also defined independent of any specific naming or directory service implementation. It enables applications to access different, possibly multiple, naming and directory services using a common API. Different naming and directory service providers can be plugged in seamlessly behind this common API. This enables Java technology-based applications to take advantage of information in a variety of existing naming and directory services, such as LDAP, NDS, DNS, and NIS(YP), as well as enabling the applications to coexist with legacy software and systems.
iii. Java Annotation
iv. Annotation is a tag that represents the metadata.
v. Meta data meant for java tools
vi. It can be present at different levels.
vii. class, interface, methods or fields
viii. It indicates some additional information that can be used by java compiler and JVM.
ix. 2 Types
    1. Built-In Annotations 2. Custom Annotations
x. There are several built-in annoations.
xi. Built in annotations meant for java code
xii. Eg:
xiii. @Override
xiv. @SuppressWarnings
xv. @Deprecated

xvi. @FunctionalInterface
xvii. Built-In Annotations that are applied to other annotations (to supply more information of the annotation itself)
    1. @Target -- target of the annotation
    2. Can have typically these values -- CONSTRUCTOR,LOCAL_VARIABLE,FIELD,METHOD,PARAMETER etc.
    3. @Retention -- Specifies Retention policy of the annotation.
    4. Can have typically these values
    5. SOURCE
    6. Annotations are to be discarded by the compiler (i.e don't appear in .class file).
    7. eg : @Override
xviii. CLASS --Annotations are kept in the class file by the compiler but not retained by the VM at run time. This is the default behavior.
xix. RUNTIME -- Annotations kept in the class file by the compiler and retained by the JVM at run time, so as to read using reflection
xx. eg : @Deprecated,@FunctionalInterface
xxi. @Inherited --
xxii. Indicates that an annotation type is automatically inherited.
xxiii. When you apply this annotation to any other annotation i.e. @MyCustomAnnotation; and @MyCustomAnnotation is applied of any class MySuperClass then @MyCustomAnnotation will be available to all sub classes of MySuperClass as well.
xxiv. @Documented
xxv. This annotation indicates that new annotation should be included into java documents generated by java document generator tools.
xxvi. Meaning of annotations
xxvii. @Override
xxviii. @Override annotation assures that the subclass or implementation class method is overriding/implementing the parent class / interface method. If it is not so, compile time error occurs.
xxix. @SuppressWarnings
xxx. @SuppressWarnings annotation: is used to suppress warnings issued by the compiler.
xxxi. eg : @SuppressWarnings("serial")
xxxii. or
xxxiii. @SuppressWarnings("unchecked")
xxxiv. @Deprecated

xxxv. @Deprecated annoation marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

## Custom Annotations

i. Java Custom Annotation
ii. Java Custom annotations or Java User-defined annotations are easy to create and use. The @interface element is used to declare an annotation. For example:
iii. @interface MyAnnotation{}
iv. Here, MyAnnotation is the custom annotation name.
v. Points to remember for java custom annotation signature (methods in i/f)
vi. Method should not have any throws clauses
vii. Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
viii. Method should not have any parameter.
ix. We should attach @ just before interface keyword to define annotation.
x. It may assign a default value to the method.
xi. Types of Annotation
xii. There are three types of annotations.
xiii. Marker Annotation
xiv. Single-Value Annotation
xv. Multi-Value Annotation
xvi. 1) Marker Annotation
xvii. An annotation that has no method, is called marker annotation. For example:
xviii. @interface MyAnnotation{}
xix. eg : @Override , @Deprecated are marker annotations.
xx. 2) Single-Value Annotation
xxi. An annotation that has one method, is called single-value annotation. For example:
xxii. @interface MyAnnotation{
xxiii. int value();
xxiv. }
xxv. We can provide the default value also. For example:
xxvi. @interface MyAnnotation{
xxvii. int value() default 0;
xxviii. }
xxix. How to apply Single-Value Annotation

xxx. eg :
xxxi. @MyAnnotation(value=10)
xxxii. The value can be anything.
xxxiii. 3) Mulit-Value Annotation
xxxiv. An annotation that has more than one method, is called Multi-Value annotation.
xxxv. For example:
xxxvi. @interface MyAnnotation{
xxxvii. int value1();
xxxviii. String value2();
xxxix. String value3();
xl. }
xli. We can provide the default value also. For example:
xlii. @interface MyAnnotation{
xliii. int value1() default 1;
xliv. String value2() default "";
xlv. String value3() default "xyz";
xlvi. }
xlvii. How to apply Multi-Value Annotation
@MyAnnotation(value1=10,value2="abc",value3="xyz")
i. Built-in Annotations used in custom annotations in java
ii. @Target
iii. @Retention
iv. @Inherited
v. @Documented
vi. Example to specify annoation for a class (i.e before a class definition)
vii. @Target(ElementType.TYPE)
viii. @interface MyAnnotation{
ix. int value1();
x. String value2();
xi. }
xii. Example to specify annoation for a class, methods or fields
xiii. @Target({ElementType.TYPE, ElementType.FIELD, ElementType.METHOD})
xiv. @interface MyAnnotation{
xv. int value1();
xvi. String value2();
xvii. }

## @Retention

i. @Retention annotation is used to specify to what level annotation will be available.
ii. RetentionPolicy Availability
iii. RetentionPolicy.SOURCE refers to the source code, discarded during compilation. It will not be available in the compiled class.
iv. RetentionPolicy.CLASS refers to the .class file, available to java compiler but not to JVM . It is included in the class file.
v. RetentionPolicy.RUNTIME refers to the runtime, available to java compiler and JVM .
vi. Example to specify the RetentionPolicy
vii. @Retention(RetentionPolicy.RUNTIME)
viii. @Target(ElementType.TYPE)
ix. @interface MyAnnotation{
x. int value1();
xi. String value2();
xii. }

## How built-in annotations are used in real scenario?

i. In real scenario, java programmer only needs to apply annotation. You don't need to create and access annotation. Creating and Accessing annotation is performed by the implementation provider. On behalf of the annotation, java compiler or JVM performs some additional operations.
ii. Use Cases for Annotations
iii. Annotations are very powerful and Frameworks like spring and Hibernate use Annotations very extensively for logging and validations. Annotations can be used in places where marker interfaces are used. Marker interfaces are for the complete class but you can define annotation which could be used on individual methods for example whether a certain method is exposed as service method or not.
iv. Design patterns
v. There are common problems faced by programmers all over.
vi. Not a domain specific problem --asso with banking / insurance or manufacturing or pharma etc.
vii. BUT if its a generic problem(eg : writing a web app , separating cross cutting concerns ) --then there are standard solutions to these common problems -- these are nothing but the design patterns---best practises.

viii. Design Patterns -- Started with Gang Of Four in 1994 --but being modified/enhanced since then.

## Summary was

1. Always prefer composition over inheritance
2. Always code for i/f & not imple.

i. In a nut shell
ii. Good programming practises --to be followed by prog community.
iii. Should u know about all of them ? -- 100 %
iv. Should u apply all of them -- NO ---nobody does that--you should only use those patterns which your appln demands.
v. They are categorised into 3 parts
    1. Creational design patterns -- best practises for object creation
    2. eg : singleton , factory , builder
    3. Structural -- best practises for composition of objects to create a larger application
    4. eg : adapter
    5. Behavioural -- best practises for communication between the objects (w/o any composition)
vi. You can think of additional category as
    1. J2EE design patterns -- MVC , Front Controller , DAO , DTO
vii. Pattern examples
    1. singleton
    2. Typically all spring beans , used are singleton.
    3. Factory
    4. eg : o.s.bean.factory.BeanFactory <----- ApplicationContext <----- ClassPathXmlApplicationContext
    5. In above case , SC is following factory pattern --provider of the rdymade spring beans (via ctx.getBean method)
    6. Another eg : Hibernate's session factory

## 2.5 Builder --

i. Use case --When there are too many parameters needed to be passed to a class constructor, instead use builder pattern.
ii. Structural Patterns
    1. Adapter design pattern is one of the structural design pattern and its used so that two unrelated interfaces can work together. The object that joins these unrelated interface is called an Adapter.

2. eg : Mobile charger works as an adapter between mobile charging socket and the wall socket.
3. We will have two classes – Volt (to measure voltage) and Socket (producing constant voltage of 230V).
4. Now build an adapter that can produce 3 volts, 12 volts and default 230 volts. So create an adapter interface with these methods.
5. Object adapter implementation --based on composition

## 4. A tester

1. Facade Design Pattern -- structural design pattern

Suppose we have an application with set of interfaces to use MySql/Oracle database and to generate different types of reports, such as HTML report, PDF report etc.

i. So we will have different set of interfaces to work with different types of database. Now a client application can use these interfaces to get the required database connection and generate reports.
ii. But when the complexity increases or the interface behavior names are confusing, client application will find it difficult to manage it.
iii. So we can apply Facade design pattern here and provide a wrapper interface on top of the existing interface to help client application.
iv. eg :
    1. Two helper interfaces, -- namely MySqlHelper and OracleHelper.
    2. Facade pattern interface
    3. Tester

Proxy Design pattern is one of the Structural design pattern

i. Provide a surrogate(substitute) or placeholder for another object to control access to it.
ii. eg : AOP proxies.
iii. Refer to example of tx manager in spring framework

## eclipse project --test_aop_simple

Behavioral Design patterns

1. Template Method is a behavioral design pattern.
2. Template Method design pattern is used to create a method stub and deferring some of the steps of implementation to the subclasses.
3. eg : Spring example -- RestTemplate

A Command pattern is an object behavioral pattern

Allows us to achieve complete decoupling between the sender and the receiver.

i. A sender -- is an object that invokes an operation
ii. A receiver is an object that receives the request to execute a certain operation.
iii. With decoupling, the sender has no prior knowledge of the Receiver's interface.
iv. Here request = command that is to be executed.
v. The Command pattern also allows us to vary when and how a request is fulfilled. Thus : a Command pattern provides us flexibility as well as extensibility.
vi. eg :
    1. 2 electrical instruments to operate --fan & light bulb
    2. Command i/f
    3. Command imple classes
    4. Switch -- invokes command on the receiver object
    5. tester

## Aspect Oriented Programming(AOP)

i. WHY
ii. Separating cross-cutting concerns(=repeatative tasks) from the business logic .
iii. IMPORTANT
iv. 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)
v. eg scenario --
vi. 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.
vii. 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.
viii. 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.
ix. Thats separation of responsibilities. --- This is what spring offers u thro AOP.
x. It is NOT an alternative to OOP BUT it complements OOP.

xi. The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.
xii. 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)
xiii. Enables the modularization of cross cutting concerns(=task)
xiv. Eg : Logging,Authentication,Transactions.
xv. Similar in functionality to ---In EJB framework -- EJBObject
xvi. Struts 2 -- interceptors
xvii. Servlet -- filters.
xviii. RMI -- stubs
xix. Hibernate --- proxy (hib frmwork -- lazy --- load or any--many associations --rets typically un-inited proxy/proxies)
xx. AOP with Spring Framework
xxi. One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework.
xxii. Like DI, AOP supports loose coupling of application objects.
xxiii. The functionalities that span multiple points of an application are called cross-cutting concerns.
xxiv. With AOP, applicationwide concerns(common concerns-responsibilities or cross-cutting concerns like eg - declarative transactions , security,logging,monitoring,auditing,authentication....)
xxv. are decoupled from the objects to which they are applied.
xxvi. Its better for application objects to focus on the business domain problems theyre designed for and leave certain ASPECTS to be handled by someone else.
xxvii. Job of AOP framework is --- Separating these cross-cutting concerns(repeatative tasks) from the core business logic
xxviii. AOP is like triggers in programming languages such as Perl, .NET.
xxix. 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

i. Advice : Action(=cross cutting concern) to take either before/after or around the method (B.L) execution.
ii. Advice WHAT & WHEN

iii. 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?
iv. Join Point : Place in application where advice can be applied.(Spring AOP, supports only method execution join point )
v. Pointcut : Collection of join points.
vi. Advisor Group of 'Advice and 'Pointcut into a single unit.
vii. Aspect : class representing advisor(advice logic + point cut definition) -- @Asepct cls level annotation.
viii. Target : Application Object containing Business logic.(To which advice gets applied at specified join points)
ix. Proxy : Object created after applying advice to the target object(created by SC dynamically by implementing typically service layer i/f) ---cross cutting concern
x. Weaving -- meshing(integration) cross cutting concern around B.L
xi. (3 ways --- compile time, class loading time or spring supported --method exec time or run time)
xii. Examples of readymade aspects :
xiii. Transaction management & security.
xiv. Types of Advice --appear in Aspect class
xv. @Before Executes only before method execution.
xvi. @AfterReturning Executes only after method returns in successful manner
xvii. @AfterThrowing - Executes only after method throws exception
xviii. @After Executes always after method execution(in case of success or failure)
xix. @Around Most powerful, executes before & after.
xx. Regarding pointcuts
xxi. 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.
xxii. eg of PointCut annotation syntax
xxiii. @Before("execution (* com.app.bank..(..))")
xxiv. @Pointcut("execution (* com.app.bank..(double))")
xxv. // point cut expression
xxvi. @Pointcut("execution (* com.app.service..*add*(..))")

xxvii. // point cut signature -- empty method .
xxviii. public void test() {
xxix. }
xxx. eg of Applying point cut
    1. @Before(value = "test()")
    2. public void logBefore(JoinPoint p) {.....}
    3. @Pointcut("within(com.app.service.*)")
    4. public void allMethodsPointcut(){}
xxxi. @Before("allMethodsPointcut()")
xxxii. public void allServiceMethodsAdvice(){...}
xxxiii. 3.
xxxiv. @Before("execution(public void com.app.model..set*())")
xxxv. public void loggingAdvice(JoinPoint joinPoint){pre processing logic ....}
    1. //Advice arguments, will be applied to bean methods with single String argument
    2. @Before("args(name)")
    3. public void logStringArguments(String name){....}
    4. //Pointcut to execute on all the methods of classes in a package
    5. @Pointcut("within(com.app.service.*)")
    6. public void allMethodsPointcut(){}
xxxvi. 6.@Pointcut("execution(* com.core.app.service..(..))") // expression
xxxvii. private void meth1() {} // signature
xxxviii. 7.@Pointcut("execution(* com.app.core.Student.getName(..))")
xxxix. private void test() {}
xl. 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
    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
    8. Eg of Point cut definition.
    9. @PointCut("execution (* com.aop.service.Account.add*(..))") *public void test() {} OR @Before("execution (* com.aop.service.Account.add*(..))")*
    10. public void logIt()
    11. {
    12. //logging advice code

13. }

xli. Use such point cut to define suitable type of advice.
xlii. Test the application.
xliii. execution --- exec of B.L method
xliv. eg : @Before("execution (* com.app.bank..(..))")
xlv. public void logIt() {...}
xlvi. Above tell SC ---- to intercept ---ANY B.L method ---
xlvii. having ANY ret type, from ANY class from pkg -- com.app.bank
xlviii. having ANY args
xlix. Before its execution.
l. Access to the current JoinPoint
li. 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.)
lii. The org.aspectj.lang.JoinPointJoinPoint interface methods
    1. Object[] getArgs() -- returns the method arguments.
    2. Object getThis() --returns the proxy object
    3. 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

## Why Custom Tags.....

i. When JSP 2.x std actions or JSTL actions are in-sufficient to solve B.L w/o writing scriptlets --- create your own tags & use them
ii. For separation ---
iii. Supply B.L in custom tags & JSP can invoke the custom tags using WC
iv. Steps for creation of Custom Tags
    1. Identify the Business logic(tag execution logic) & encapsulate the same in TagHandler class.
    2. Custom Tag Handlers can be created using javax.servlet.jsp.tagext.SimpleTag ---i/f
    3. API gives u the impl. class : SimpleTagSupport .
v. As a tag dev : extend from SimpleTagSupport .
vi. Must override :
vii. public void doTag() throws IOExc,JSPExc : represents the tag exec. logic.
viii. Current objective : Generate a Hello msg from the custom tag. (1st tag will be w/o attrs & without body)

ix. How to get the JSPWriter inst : connected to clnt browser : from inside the Tag class?
x. API : inside : doTag....
xi. getJspContext() ---> JsPContext (env. of the JSP : impl. objs avlbl to JSP , whichever has invoked this tag)
xii. On JSPContext : to get JSPWriter
xiii. JspWriter getOut()
xiv. & then invoke : out.println(dyn cont.)
    1. Describe the tag to W.C : so that W.C can manage the life-cycle of the tag.
xv. Creating : TLD (Tag library descriptor : .tld : xml syntax)
xvi. copy the template : ur web-appln's : web-inf\tlds\example.tld
xvii. eg :
xviii. welcome : tag suffix
xix. cust_tags.MytagHandler : F.Q class name
xx. empty : no body supported by the tag + no attrs.
    1. Invoking the custom tag from the JSP
    2. 3.1 Import the TLD : which contains the description of custom tags
    3. <%@ taglib uri="URI of the TLD" prefix="tag prefix" %>
    4. 3.2 Invoke the tag
    5. eg :Life-cycle of the custom Tag
xxi. Ref to JEE -- apidoc---javax.servlet.jsp.tagext.SimpleTag ---i/f
xxii. SimpleTag : i/f ----contains the desc of the life cycle.
    1. W.C will invoke tag life cycle -- when JSP invoke tag.(eg :
    2. 2.WC locates TLD (using taglib directive)
    3. From TLD --- searches for matching tag suffix (under tag name)
    4. From tag name -- gets tag class --- locates/loads/instantiates tag class in WC's mem.
    5. WC invokes setJspContext(JspContext ctx) --- to pass the entire JSP page env(PageContext obj containing all impl objs) to the tag handler class.(mandatory)
    6. 6.WC will invoke the setters for attributes (optional)
    7. WC will invoke setJspBody(JspFragment jspf) iff body content is non-empty.(optional)
    8. 8.Finally WC invokes --- doTag()
    9. Upon returninng from doTag() --- WC GCs T.H class inst.
xxiii. body-content = empty =>W.C will skip handling of body-content.
xxiv. body-content = scriptless => WC will invoke setJspBody(JspFragment jspf) to pass tag body content to T.H class.

xxv. allowed content types ---- plain text/HTML/std action/custom action
xxvi. How to retrieve & invoke body content from T.H class?
xxvii. From doTag ---
    1. get JspFragment (API --- getJspBody())
    2. To invoke JspFragment ----use API
    3. public abstract void invoke(Writer w) throws JspException,IOException
    4. if w=null --- o/p(contents) will be auto. sent to clnt browser.
xxviii. Objective : Create & test cust tag : to support body + attr
xxix. HTML/Plain Text/custom action
xxx.
xxxi. Expected o/p : Body -content should be displayed count no of times.
xxxii. Expected o/p : Body -content should be upper-cased & then displayed count no of times.
xxxiii. Help for solving Custom Tag assignment --
    1. Create QueryTagHandler class ---extends SimpleTagSupport.
    2. 2 Create private data members -- same name & type as attributes(eg - 2props -- dept & salary. ---provide setters.
xxxiv. 3.Override doTag
xxxv. --JspFragment jspf=getJspBody();
xxxvi. StringWriter sw=new StringWriter();
xxxvii. jspf.invoke(sw);
xxxviii. String query=sw.getBuffer().toString();
xxxix. s.o.p(query);//confirm if u have got till body-content as query.
xl. //append to the query -- where dept=? and salary=?
xli. //invoke DAO class --- openCn,invokeQuery(query,dept,sal)
xlii. // DAO rets to cust tag -- AL
xliii. //use for-each from doTag --- & display using out.println(empPojo) to clnt browser.
xliv. //dao.closeConnection();
xlv. ---doTag() rets.
xlvi. Objective : remove scriptlet : for discarding session from logout.jsp
xlvii. Invoke JavaBean method from custTag
xlviii. Why Filters ?
    1. Provides re-usability.
    2. Meaning --- They provide the ability to encapsulate recurring tasks(=cross cutting concerns) in reusable units.
    3. Can dynamically intercept req/resp to dyn or static content

## What is Filter?

i. Dynamic web component just like servlet or
ii. JSP. Resides within web-appln.(WC)
iii. Filter life-cycle managed by WC
iv. 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.
v. It can dynamically intercepts requests and responses .
vi. Usage of Filters
    1. Authentication Filters
    2. Logging Filters
    3. Image conversion Filters
    4. Data compression Filters
    5. Encryption Filters
    6. Session Check filter
vii. How to create Filter Component?
    1. Create Java class imple. javax.servlet.Filter i/f
    2. Implements 3 life-cycle methods
    3. 2.1 public void init(FilterConfig filterConfig)
    4. throws ServletException
viii. Above called by WC --- only once during filter creation & initialization. (@appln start up time)
ix. 2.2
x. void doFilter(ServletRequest request,ServletResponse response, FilterChain chain) throws IOException, ServletException
xi. Invoked by WC -- per every rq & resp processing time.
xii. 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.
xiii. 2.3
xiv. public void destroy() ---invoked by WC at the end of filter life cycle.
xv. Triggers --- server shut down/re-dploy/un deploy
xvi. How to deploy a Filter component ?
    1. Annotation -- @WebFilter (class level annotation)
    2. OR
    3. XML tags (in web.xml)
    4. abc filters.AuthenticatioFilter nm1 val1
    5. abc /*

---

## Detailed Description ---

i. Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.
ii. 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.
iii. 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.
iv. Upload / download Images
v. Its a complete spring web mvc project.(eclipse project : spring_mvc_image_upload2 ) for uploading images from clnt & storing it NOT under web application folder structure BUT elsewhere on server folder structure.
vi. (eg : e:\uploaded_contents)
vii. You can import it directly & test.
viii. Instructions to run
    1. First upload few images , using the link.
    2. Then list images , then choose one .
ix. Note
    1. No index.jsp , under WebContent. Its kept under itself.
    2. Check abc-servlet.xml , is the tag added for that.
    3. Meaning --Maps a simple (w/o actually writing a controller) view controller to a specific URL pattern -- to render a response with a pre configured view.
    4. Upload images & store it under (as per standard recommendation) on the server side folder .
x. Even if you clean your project , images will NOT be deleted.

xi. Flow :
    1. abc-servlet.xml
    2.
    3. Meaning --
    4. For incoming url pattern / -- use view name "index" --a logical view name --which gets translated to /web-inf/views/index.jsp
xii. 1.1 In abc-servlet.xml
xiii. --imports xml from
xiv. In file-upload.xml --
xv. class="org.springframework.web.multipart.commons.CommonsMultipartResolver">
xvi.
xvii. Above configures multipart resolver bean , with max upload size of 8MB.
xviii. You can change it as per your requirement
xix. 1.2
xx.
xxi. Supplies the location of property file.
xxii. In property file
xxiii. image_path=file:/e:/uploaded_contents/
xxiv. 1.3 Entry ---for handling static resources.
xxv.
xxvi. Above is to tell SC that if incoming url pattern is matching with /upload_images/ & anything below that , then its a static resource to be served from the location of the specidied ${image_path} i.e some server side folder.
xxvii. 1.4
xxviii. Above is to tell SC that if incoming url pattern is matching with / & anything below that , then its a static resource to be served from the location "/" i.e the root of web appln.
xxix. If u are wondering --what if /list_images comes --not to worry --we alrdy have a controller assgined to it.
xxx. Order of above 2 (1.3 & 1.4) tag is important, otherwise it won't work since the mapping /upload_images doesn't get mapped at all.
    1. index.jsp -- File Upload link.
    2. href -- file_upload
    3. URL generated - .../file_upload
    4.
    5. FileUploadController

6. @RequestMapping(value = "/{path}") -- using path variable.
7. public String showForm(@PathVariable String path) {
8. return path;
9. }
10. Matches ANY pattern /path , currently path=file_upload --so FORWARDS user to /web-inf/views/file_upload.jsp
11. NOTE : not added any model attribute to the model map.
12. file_upload.jsp
13. No spring form , just ordinary HTML form (spring not needed since no model attribute is bound to the form)

xxxi. Choose File to Upload :
xxxii. Important : Must choose enctype as multipart/form-data , to upload the content , method must be post & then use input type="file"
    1. When form is submitted URL -- .../file_upload , method=post, so uploadFile of FileUploadController gets called.
xxxiii. FileUploadController
xxxiv. Dependency Injection
xxxv. @Value("${upload_image_path}")
xxxvi. private String imagePath;
xxxvii. @RequestMapping(value = "file_upload", method = RequestMethod.POST)
xxxviii. public String uploadFile(@RequestParam("contents") MultipartFile file, Model map, HttpServletRequest request) {
xxxix. String uploadLocation = imagePath;
xl. System.out.println(
xli. "in upload file " + file.getOriginalFilename() + " size " + file.getSize() + " " + uploadLocation);
xlii. try {
xliii. File dest=new File(uploadLocation, file.getOriginalFilename());
xliv. //file transferred to server side folder
xlv. file.transferTo(dest);

```
    } catch (Exception e) {

        map.addAttribute("mesg", "File upload failed : " + e.get
Message());
        return "file_upload";
    }
    return "upload_succ";
```

```
}
```

## Meaning :

i. 4.1
ii. API
iii. org.springframework.web.multipart.MultipartFile --i/f
iv. Implementing Class
v. CommonsMultipartFile --based on Apache Commons File upload (jar -- commons file upload)
vi. MultipartFile
vii. Represents an uploaded file received in a multipart request.
viii. The file contents are either stored in memory or temporarily on disk. In either case, the user(controller method) is responsible for copying file contents in persisten manner(either within webapp under webcontent or server side folder or db) . The temporary storages will be cleared at the end of request processing.
ix. 4.2
x. String uploadLocation = imagePath
xi. This sets the upload location to -- server side folder
xii. 4.3
xiii. file.transferTo(new File(uploadLocation, file.getOriginalFilename()));
xiv. API of MultiPartFile
xv. void transferTo(File dest) throws IOException,IllegalStateException
xvi. Transfers the received file to the given destination file.
xvii. If the destination file already exists, it will be deleted first.
xviii. If the multipart file doesn't exist (i.e its alrdy moved) --throws -- IllegalStateException
xix. In case of exceptions(failures) ,catch it --add exception mesg to model map & forward user to "file_upload.jsp"
xx. In case of success -- forward user to "upload_succ.jsp" , to display success mesg.
    1. Now for displaying images part (pre requisite : image has to be uploaded under the server side folder first)
    2. index.jsp -- link -- "List Images"
    3. href -- list_images
    4. @RequestMapping(value = "/list_images")
    5. public String listImages(Model map, HttpServletRequest request) {
    6. System.out.println("in list images " + imagePath);
    7. File uploadLocationDir = new File(imagePath);

8. System.out.println(uploadLocationDir.exists() + " " +
   uploadLocationDir.isDirectory() + " "
9. + uploadLocationDir.getAbsolutePath());

```
String[] files = uploadLocationDir.list();
System.out.println("files " + Arrays.toString(files));
map.addAttribute("image_list", files);
return "list_images";
```

10. }

## Meaning

1. 5.1 Create File instance(representing a folder) --pointing to server side folder
2. 5.2 Store file names under that folder in model map & forward user to "list_images.jsp"
3. list_images.jsp

i. ${img}

ii. Meaning : ordinary HTML form (since no model attributes are bound). Attached for each to image list , user chooses a radio btn & submit the form.

1. FileUploadController
2. @RequestMapping(value = "/choose")
3. public String chooseImage(Model map, @RequestParam String imgName) {
4. System.out.println("in choose img name " + imgName);
5. map.addAttribute("img_name", imgName);
6. return "show_image";
7. }
8. Meaning : req param contains the image file name . so adding that in model map & forwarding user to "show_image.jsp"
9. In abc-servlet.xml
10. there are already entries.
11. 
12. show_image.jsp

Config. steps for the pooled connectivity

i. MUST Copy JDBC drvr jar(mysql connector jar) to tomcat-home/lib

1. stop the srvr.
2. Ref Doc : \webapps\docs\jndi-datasource-examples-howto.html
3. Open context.xml from folder

4. & add Resource tag under Context tag.

ii. initialSize="1" maxActive="2" maxIdle="1" maxWait="-1" username="root"
iii. password="root" driverClassName="com.mysql.jdbc.Driver" url="jdbc:mysql://localhost:3306/test"
iv. removeAbandoned="true" />
    1. Open ur web-appln's web.xml(WebContent\web-inf\web.xml)
v. copy the Resource - ref. tag in web.xml
vi. eg :
vii. Oracle Datasource example
viii. jdbc/ora_pool
ix. javax.sql.DataSource
x. Container
xi. ENSURE : res-ref-name matches with Resource name added in server.xml
xii. Info about resource settings
    1. maxActive: Maximum number of database connections in pool.
    2. maxIdle: Maximum number of idle database connections to retain in pool.
    3. maxWait: Maximum time to wait for a database connection to become available
    4. in ms, An Exception is thrown if
    5. this timeout is exceeded. Set to -1 to wait indefinitely.
    6. initialSize -- initial size of the pool at start up
    7. removeAbondened -- true(def=false) -- if there are any abondened cns --- then they are re-cycled to pool after tmout(def val=300 sec)If you consider a spring mvc web app configuration where all the requests are mapped to a DispatcherServlet. You can categorize those requests as requests for static and dynamic resources.
xiii. The requests for dynamic resources are matched by what you program inside your controller methods, and they are routed via handler mapping , view resolver etc.
xiv. The requests for static resources are the requests for .js, .css, images or some other resources that are not getting created rather already exist deployed with your application. They are not handled by the controller methods rather by the ResourceHttpRequestHandler, simply because they have a completly different set of processing actions comparing to dynamic request (apart from path matching). You can define the location of static files for the given mapping
xv. Reference project -- spring_mvc_hibernate_bootstrap

1. Copy bootstrap(css)/jquery/.... & js under or better answer --- under root of web application(webapp in maven)
2. Check entry under spring-servlet.xml
3. This entry tells spring container about the location of your static resources.
4. Check test_bootstrap.jsp under
5. spring_mvc_hibernate_bootstrap\WebContent\WEB-INF\viewsInternationalization -- i18n -- Technique for developing applications that support multiple languages, data formats , currency formats , date formats etc without having to re-write view generation logic.

xvi. Localization --- l10n --- technique for adapting an internationalized appln to support a specific locale.
xvii. Locale = specific geographical,political or cultural region.
xviii. Steps for i18n
    1. Identify the locales

## 2. Create text-based property file -- one per each locale. ---store it under or

For making your application support different locales, we need to create locale specific properties file. The file names follow the pattern of bundle name with language code and country code, for example ApplicationMessages_en_US.properties.

i. Once the property files for specific locales are ready, all you need to do is initialize the resource bundle with correct Locale.
ii. API
iii. Java provides two classes java.util.ResourceBundle and java.util.Locale that are used for this purpose.
iv. ResourceBundle reads the locale specific property file and you can get the locale specific value for any key.
v. Use case
vi. This is very helpful in making your web application texts locale specific, you can get the locale information from the HTTP request and generate the dynamic page with that locale resource bundle files. You can also provide option to user to chose the locale and update the labels dynamically.
vii. Spring MVC i18n steps (ref : eclipse project : spring_demo)
viii. 1.. Add locale resolver bean definition in spring-servlet.xml file.
ix. SessionLocaleResolver

x. SessionLocaleResolver resolves locales by inspecting a predefined attribute in a user's session. If the session attribute doesn't exist, this locale resolver determines the default locale from the accept-language HTTP header.
    1. Add mvc interceptors for detecting change in locale, in spring-servlet.xml
xi. LocaleChangeInterceptor interceptor detects if a special parameter is present in the current HTTP request. The parameter name can be customized with the paramName property of this interceptor. If such a parameter is present in the current request, this interceptor changes the user's locale according to the parameter value.
xii. p:paramName="locale123" />
xiii. Above 2 can be either declared in spring-servlet.xml or also in imported xml file.
    1. Create copies of message based property files
    2. Create JSP with links to add support for various locales -- using same param name
    3. en
    4. English
xiv. mr_IN
xv.
xvi. Marathi
    1. Use
    2. eg :
xvii. Spring Boot demo for RESTful server
xviii. Steps
    1. File --New --Spring starter project -- add project name , artifact id ,pkg names , keep jar only.
    2. (pic1)
    3. Add dependencies
    4. web -- web (pic2)
    5. sql -- JPA, MYSQL
    6. Core -- DevTools
xix. 2.4 If you are creating Spring based REST server , you don't need to add any more dependencies in pom.xml
    1. Add following in application.properties file
    2. #Tomcat server port number
    3. server.port=7070
    4. #context path default value /

5. server.servlet.context-path=/test_boot
6. #DB properties
7. spring.datasource.url=jdbc:mysql://localhost:3306/test
8. spring.datasource.username=root
9. spring.datasource.password=root

# JPA properties

spring.jpa.show-sql = true

i. spring.jpa.hibernate.ddl-auto = update
ii. #logging.level.org.springframework.orm.hibernate5=DEBUG

Spring MVC (with JSP view layer demo) (ref project : demo2) --- spring boot project

Add following dependencies ONLY for Spring MVC with JSP as View Layer in pom.xml

1. Changed pkging from jar -->war
2. war
3. 1.

i. org.springframework.boot
ii. spring-boot-starter-tomcat
    1. For JSP working
    2. org.apache.tomcat.embed tomcat-embed-jasper
    3. JSTL library
iii. javax.servlet
iv. jstl
v. 2.5 Copied application.properties entries & added 2 more (for view layer)

# Spring MVC ViewResolver related

spring.mvc.view.prefix=/WEB-INF/views/

i. spring.mvc.view.suffix=.jsp
ii. 2.6 Created this folder structure under src/main (fig :spring-boot-mvc-folders.png)
    1. Added DAO i/f & imple class --with @Transaction & @Repo
    2. Autowired EntityMgr
    3. eg : @PersistenceContext
    4. private EntityManager mgr;
iii. 4.Added CustomerCRUDController & test it for list customers with browser

iv. NOTE : since there isn't any web.xml --no way to specify to WC that index.jsp is welcome page.
v. So have to give this url from web browser
vi. http://localhost:7070/test_boot/index.jsp

Either write IndexController or below approach

i. (via java config approach)
ii. @Configuration
iii. @EnableWebMvc
iv. public class WebConfig extends WebMvcConfigurerAdapter {
v. @Override
vi. public void addViewControllers(ViewControllerRegistry registry) {
vii. registry.addViewController("/").setViewName("home");
viii. }
ix. }Steps
    1. Create spring web mvc project.
    2. Create separate spring bean config file to add spring mail sender bean settings.-- / email-settings.xml
    3. Import this cofig xml file in master config file(spring-servlet.xml)
    4. Inject Mail sender bean in your Controller to send the email.
    5. HOW ?
    6. In your Controller , inject dependency.
    7. @AutoWired
    8. private JavaMailSender sender; ---- injects byType --- JavaMailSenderImpl class instance.
    9. Its an abstraction to hide the complex details of actually sending/receving the mail using javax.mail API
x. Java mail demo tip
xi. In spring sending mail -- in case of AuthenticationException
xii. For security purposes -By default, gmail does not allow less secure apps to get authenticated. You need to turn on the option in you gmail account to allow less secure apps to get authenticated.
xiii. Solution
xiv. login to gmail
xv. then goto this url --- https://www.google.com/settings/security/lesssecureapps
xvi. & turn on(radio btn) access allowed for less secure apps
xvii. & then test send mail demo of spring
    1. MVC overview
    2. Demo ---

3. Create New Dyn web project, create user library(add lib in build path & dep assembly) ,configure DispatcherServlet in web.xml --- to ensure all or any req coming from clnt will be intercepted by this servlet
4. spring org.springframework.web.servlet.DispatcherServlet 1
5. spring /
6. create master spring configuration file ---- Name ---- servletName-servlet.xml under ---- add beans,context , mvc & p namespaces.
7. 2.1 Add tags -- context:annotation-config,context:component-scan,mvc:annotation-driven(to enable annotated MVC controller support)
8. 2.2 declare view resolver bean --- Can use InternalResourceViewResolver or its super-class --- UrlBasedViewResolver --- Props are same for both beans --- viewClass(choose JSTL view for JSP view templates using JSTL actions), prefix -- typically under web-inf & suffix -- .jsp

xviii. 2.3 Write Hello Controller

xix. Annotations used --- @Controller(class level) & @RequestMapping(add at method level ---& pass only URL value)

xx. eg -- @RequestMapping("/hello") . Ret type -string --represents logical view name --- will get resolved by viewResolver bean.

xxi. 2.4 Add to same controller , one more req handling method -- @RequestMapping("/welcome1") & ret type as ModelAndView.

xxii. Using M&V constructor --- set model obj & view name --- test it.

xxiii. 2.5 Add 1 more method --- ret Type String BUT annotated with @ResponseBody

-- tells SC -- not to pass it to view (JSP) layer, but directly add ret val as resp data & commit response.

1. Objective ---In Contact management Utility --- Contact authentication using Login form.
2. Layers used ---
3. 3.1 -- POJO (Contact) --email,password,name,regDate,regAmt
4. 3.2 -- Sevice layer bean -- @Service annotation -- i/f & implmenation class --
5. constr -- create & populate HM with contacts. B.L method ---- validateContact -- i/p -- contact with email & password , o/p --- validated pojo or null
6. 3.3 -- Controller bean -- @Controller annotated methods
7. For displaying form --- create empty model instance

8. API of org.springframework.ui.Model --represents request scoped attribute map
9. How to add attribute--
10. public Model addAttribute(Object attributeValue) -- attr name is derived from attribute value class type eg --- UserPOJO ---userPOJO
11. OR
12. addAttribute(String attributeName,Object attributeValue)

i. After adding model attribute --- return logical name of jsp page --- to navigate user to view layer.
ii. 3.4 Create login page using spring form tag lib & for displaying validation errs add style tag -- under
iii.
iv. eg of spring form tags
v. --- name of the model attr=request scoped attr name(similar to model driven approach in sturts2) -- def name is command
vi. Enter User Email --label as with ui comp
vii. --- path --name of the rq param name
viii. 3.5 Since action is typically not mentioned -- its submitted to same controller
ix. supply different req mapping method --- use method=RequestMethod.POST -- for processing the form.
x. min method args --POJO ref, BindingResult(org.springframework.validation.BindingResult), HttpSession hs -- if validated pojo needs to be stored in session scope.
xi. -- in method --always check first for P.L errors --
xii. API of BindingResult
xiii. boolean hasErrors() -- rets true in case of P.L errors. -- in this navigate user to input page.
xiv. In absence of P.L erros -- invoke service layer method -- in case of success --store dtls under session scope & navigate user to success page. --
xv. 3.6 Give logout link from success page -- in logout controller method --
xvi. min args required ---org.springframework.web.bind.support.SessionStatus
xvii. API of SessionStatus --
xviii. void setComplete() --- to discard session.

1. How to add P.L validations?
2. Annotations based approach ---
3. 4.1 -- Add validation annotations in POJO class.
4. (ref : from templates.txt)

5. 4.2 -- in the request handling method of Controller --- in the form processing --- (3.5)
6. simply add @Valid annotation for POJO class reference -- which will force SC to apply validation rules mentioned in annotated POJO class & BindingResult i/f is populated with validation results .

xix. 4.3 Problem in above scenarios is -- in case of validation failures --- page will display spring's default err mesgs.

xx. How to add custom err messages & perform validation using annotations?

1. Add entry in xml config file (spring-servlet.xml) --define name of message resource bundle.

xxi.

xxii. NOTE -- bean id must be messageSource & can be declared either in spring-servlet.xml (i.e root xml for dispatcher servlet) or imported xml file.

1. Add property file & see error msgs derived from property file.
2. How to ?
3. syntax for property name = constraint name.model attr name.model prop name
4. eg
5. NotEmpty.user.name=Name is required
6. or in case of invalid date format ---
7. typeMismatch.java.util.Date=Invalid date format
8. How to add automatic exc handling support in spring MVC ?
9. Add following bean definition in spring-servlet.xml

xxiii. exc-handler

xxiv. Above describes centralized exc handler page .(exc-handler.jsp)

Spring From Tags examples -- import spring form tag library

1. 1.
2. 
3. 3.
4. 
5. where countryList -- is model attribute of type List & Country has
6. countryId & countryName -- as properties.
7. 6
8. 7
9. How to add I18N support?

Internationalization is the process of designing a software application so that it can potentially be adapted to various languages and regions without engineering changes.

Localization is the process of adapting internationalized software for a specific region or language by adding locale-specific components and translating text .
Spring frameworkgives you LocaleResolver bean to support the internationalization and thus localization as well.
6.1. Add locale resolver bean definition in spring-servlet.xml file.
SessionLocaleResolver
SessionLocaleResolver resolves locales by inspecting a predefined attribute in a user's session. If the session attribute doesn't exist, this locale resolver determines the default locale from the accept-language HTTP header.
6.1.5 Add mvc interceptors for detecting change in locale
LocaleChangeInterceptor interceptor detects if a special parameter is present in the current HTTP request. The parameter name can be customized with the paramName property of this interceptor. If such a parameter is present in the current request, this interceptor changes the user's locale according to the parameter value.
p:paramName="locale123" />
Above 2 can be either declared in spring-servlet.xml or also in imported xml file.
6.2 Create copies of message based property files
6.3 Create JSP with links to add support for various locales -- using same param name
en

English
6.4 Use

1. How to serve static resources?
2. Eg : CSS,JavaScript or images.

Configure a handler for serving static resources in spring-servlet.xml.
For applying CSS
8 How to add content(file) uploading feature?
Spring supports multipart i.e file uploading. Spring has built-in multipart support for file uploads in web applications.
A multipart content is the content with enctype="multipart/form-data".
(def HTML form encoding type : application/x-www-form-urlencoded)
API
Interface : org.springframework.web.multipart.MultipartResolver
One implementation of the MultipartResolver i/f is Commons FileUpload

Spring CommonsMultipartResolver is a MultipartResolver implementation for use with Apache Commons FileUpload.
Additional jar required -- apache commons-fileupload.jar
Steps to enable the Spring multipart handling:

1. Add a multipart resolver bean to the web application's context(either directly in spring-servlet.xml or import from separate file)
2. Spring provides org.springframework.web.multipart.MultipartFile which is a representation of an uploaded file received in a multipart request. It provides handy methods like getName(), getContentType(), getBytes(), getInputStream() etc.. which make it easier while retrieving information about file being uploaded.
3. Ref : javadocs.

Create a wrapper POJO having MutipartFile as a state(data member)

i. (not mandatory)

1. Write a controller .
2. 3.1 Add req handling method to show upload form
3. Add POJO instance to the Model map (model attribute)
4. Create JSP view with a form (form:form) having same model attribute & encType="multipart/form-data"

Add
OR can be done also by a simple form.
In the same controller add req handling method to process upload form
Use MultipartFile API to transfer file from temp location or memeory to permanent file on server.
API of MultipartFile
String getContentType()
String getOriginalFileName()
byte[] getBytes()
long getSize()
public static void copy(byte[] in,File out) throws IOException
Meaning
Each request is inspected to see if it contains a multipart.
If no multipart is found, the request continues as expected.
If a multipart is found in the request, the MultipartResolver that has been declared in your context is used. The multipart attribute in your request is treated like any other attribute.
Java mail demo tip
In spring sending mail -- in case of AuthenticationException

For security purposes -By default, gmail does not allow less secure apps to get authenticated. You need to turn on the option in you gmail account to allow less secure apps to get authenticated.
HOW ??
Ans --
login to gmail
then goto this url --- https://www.google.com/settings/security/lesssecureapps
& turn on(radio btn) access allowed for less secure apps
& then test send mail demo of spring
Spring Boot Tutorial
Spring Boot uses completely new development model to make Java Development very easy by avoiding some tedious development steps and boilerplate code and configuration.
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” Approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.
What is NOT Spring Boot?
Spring Boot Framework is not implemented from the scratch by The Spring Team, rather than 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.

## Why Spring Boot?

To ease the Java-based applications Development, Unit Test and Integration Test Process.
To reduce Development, Unit Test and Integration Test time by providing some defaults.
To increase Productivity.
Don’t worry about what is “Opinionated Defaults Configuration” Approach at this stage. We will explain this in detail with some examples in coming posts.
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 Simple Terminology, What Spring Boot means
ref : WhatIsSpringBoot1.png
What Is Spring Boot, Spring Boot Tutorial
That means Spring Boot is nothing but existing Spring Framework + Some Embedded HTTP Servers (Tomcat/Jetty etc.) – XML or Annotations Configurations.
Here minus means we don’t need to write any XML Configuration and few Annotations only.
Main Goal of Spring Boot:
The main goal of Spring Boot Framework is to reduce Development, Unit Test and Integration Test time and to ease the development of Production ready web applications very easily compared to existing Spring Framework, which really takes more time.
To avoid XML Configuration completely
To avoid defining more Annotation Configuration(It combined some existing Spring Framework Annotations to a simple and single Annotation)
To avoid writing lots of import statements
To provide some defaults to quick start new projects within no time.
To provide Opinionated Development approach.
By providing or avoiding these things, Spring Boot Framework reduces Development time, Developer Effort and increases productivity.
Limitation/Drawback of Spring Boot:
Spring Boot Framework has one limitation.
It is some what bit time consuming process to convert existing or legacy Spring Framework projects into Spring Boot Applications but we can convert

all kinds of projects into Spring Boot Applications. It is very easy to create brand new/Greenfield Projects using Spring Boot.

To Start Opinionated Approach to create Spring Boot Applications, The Spring Team (The Pivotal Team) has provided the following three approaches.

Using Spring Boot CLI Tool

Using Spring STS IDE

Using Spring Initializr Website

We can use Spring STS IDE or Spring Initializr Website to develop Spring Boot Java Applications.

Spring Boot Framework also combined existing Spring Framework annotations into some simple or single annotations. We will explore those annotations one by one in coming posts with some real-time examples.

In 28 miniutes ....

The most important feature of Spring Framework is Dependency Injection. At the core of all Spring Modules is Dependency Injection or IOC Inversion of Control.

Example Without Dependency Injection

Consider the example below: WelcomeController depends on WelcomeService to get the welcome message. What is it doing to get an instance of WelcomeService?

WelcomeService service = new WelcomeService();

It's creating an instance of it. And that means they are tightly coupled. For example: If I create a mock for WelcomeService in a unit test for WelcomeController, how do I make WelcomeController use the mock? Not easy!

```
@RestController
public class WelcomeController {
private WelcomeService service = new WelcomeService();
@RequestMapping("/welcome")
public String welcome() {
return service.retrieveWelcomeMessage();
}
}
```

Same Example with Dependency Injection

The world looks much simpler with dependency injection. You let the Spring Framework do the hard work. We just use two simple annotations: @Component and @Autowired.

Using @Component, we tell Spring Framework: Hey there, this is a bean that you need to manage.

Using @Autowired, we tell Spring Framework: Hey find the correct match for this specific type and autowire it in.
In the example below, Spring framework would create a bean for WelcomeService and autowire it into WelcomeController.
In a unit test, I can ask the Spring framework to auto-wire the mock of WelcomeService into WelcomeController. (Spring Boot makes things easy to do this with @MockBean. But, that's a different story altogether!)

```
@Component
public class WelcomeService {
//some B.L
}
@RestController
public class WelcomeController {
@Autowired
private WelcomeService service;
@RequestMapping("/welcome")
public String welcome() {
return service.retrieveWelcomeMessage();
}
}
```

What Else Does Spring Framework Solve?
Problem 1: Duplication/Plumbing Code
Does Spring Framework stop with Dependency Injection? No. It builds on the core concept of Dependency Injection with a number of Spring Modules
Spring JDBC
Spring MVC
Spring AOP
Spring ORM
Spring JMS
Spring Test
Consider Spring JMS and Spring JDBC for a moment.
Do these modules bring in any new functionality? No. We can do all this with J2EE or Java EE. So, what do these bring in? They bring in simple abstractions. The aim of these abstractions is to
Reduce Boilerplate Code/Reduce Duplication
Promote Decoupling/Increase Unit Testability
For example, you need much less code to use a JDBCTemplate or a JMSTemplate compared to a traditional JDBC or JMS.
Problem 2: Good Integration With Other Frameworks

The great thing about Spring Framework is that it does not try to solve problems that are already solved. All that it does is to provide a great integration with frameworks which provide great solutions.

## Hibernate for ORM

iBatis for Object Mapping
JUnit and Mockito for Unit Testing
What Is the Core Problem That Spring MVC Framework Solves?
Spring MVC Framework provides decoupled way of developing web applications. With simple concepts like Dispatcher Servlet, ModelAndView and View Resolver, it makes it easy to develop web applications.
When things are this good , Why Do We Need Spring Boot?
Spring based applications need a lot of configuration.
When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars(for delivering static content) among other things.
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
/WEB-INF/views/
.jsp

The code snippet below shows the typical configuration of a dispatcher servlet in a web application.

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>
        org.springframework.web.servlet.DispatcherServlet
    </servlet-class>
    <init-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>/WEB-INF/todo-servlet.xml</param-value>
    </init-param>
    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>dispatcher</servlet-name>
```

```
    <url-pattern>/</url-pattern>
</servlet-mapping>
```

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

Problem #1: Spring Boot Auto Configuration: Can We Think Different?

Spring Boot brings a new thought process around this.

Can we bring more intelligence into this? When a spring mvc jar is added into an application, can we auto configure some beans automatically?

How about auto-configuring a Data Source if Hibernate jar is on the classpath?

How about auto-configuring a Dispatcher Servlet if Spring MVC jar is on the classpath?

There would be ofcourse provisions to override the default auto configuration.

Spring Boot looks at a) Frameworks available on the CLASSPATH b) 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.

When we use Spring MVC, we need to configure component scan, dispatcher servlet, a view resolver, web jars(for delivering static content) among other things.

Problem #2: Spring Boot Starter Projects: Built Around Well-Known Patterns

Let's say we want to develop a web application.

First of all, we would need to identify the frameworks we want to use, which versions of frameworks to use and how to connect them together.

All web application have similar needs. Listed below are some of the dependencies we use in our Spring MVC Course. These include Spring MVC, Jackson Databind (for data binding), Hibernate-Validator (for server side validation using Java Validation API) and Log4j (for logging). Whenever creating any web app, we had to choose the compatible versions of all these frameworks.

org.springframework
spring-webmvc
4.3.2.RELEASE

com.fasterxml.jackson.core
jackson-databind
2.5.3

org.hibernate
hibernate-validator
5.2.6.Final

log4j
log4j
1.2.17
Here's what the Spring Boot documentations says about starters.
Starters are a set of convenient dependency descriptors that you can include in your application. You get a one-stop-shop for all the Spring and related technology that you need, without having to hunt through sample code and copy paste loads of dependency descriptors. For example, if you want to get started using Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.
Let's consider an example starter: Spring Boot Starter Web.
If you want to develop a web application or an application to expose restful services, Spring Boot Start Web is the starter to pick. Let's create a quick project with Spring Boot Starter Web using Spring Initializr.
Dependency for Spring Boot Starter Web
org.springframework.boot
spring-boot-starter-web
Just by adding above starter , it will add lot of JARs under maven dependencies.
These Dependencies can be classified into:
Spring: core, beans, context, aop
Web MVC: (Spring MVC)
Jackson: for JSON Binding
Validation: Hibernate Validator, Validation API
Embedded Servlet Container: Tomcat
Logging: logback, slf4j
Any typical web application would use all these dependencies. Spring Boot Starter Web comes pre-packaged with these. As a developer, I would not need to worry about either these dependencies or their compatible versions.
Spring Boot Starter Project Options
As we see from Spring Boot Starter Web, starter projects help us in quickly getting started with developing specific types of applications.
spring-boot-starter-web-services: SOAP Web Services
spring-boot-starter-web: Web and RESTful applications

spring-boot-starter-test: Unit testing and Integration Testing
spring-boot-starter-jdbc: Traditional JDBC
spring-boot-starter-hateoas: Add HATEOAS features to your services
spring-boot-starter-security: Authentication and Authorization using Spring Security
spring-boot-starter-data-jpa: Spring Data JPA with Hibernate
spring-boot-starter-cache: Enabling Spring Framework's caching support
spring-boot-starter-data-rest: Expose Simple REST Services using Spring Data REST
Other Goals of Spring Boot
There are a few starters for technical stuff as well
spring-boot-starter-actuator: To use advanced features like monitoring and tracing to your application out of the box
spring-boot-starter-undertow, spring-boot-starter-jetty, spring-boot-starter-tomcat: To pick your specific choice of Embedded Servlet Container
spring-boot-starter-logging: For Logging using logback
spring-boot-starter-log4j2: Logging using Log4j2
Spring Boot aims to enable production ready applications in quick time.
Actuator: Enables Advanced Monitoring and Tracing of applications.
Embedded Server Integrations: Since the server is integrated into the application, I would NOT need to have a separate application server installed on the server.
Provides Default Error Handling

Regarding annotation used on main class

1. What is @SpringBootApplication annotation in spring boot?

i. Many Spring Boot developers always have their main class annotated with @Configuration, @EnableAutoConfiguration and @ComponentScan. Since these annotations are so frequently used together (especially if you follow the best practices above), Spring Boot provides a convenient @SpringBootApplication alternative.
ii. The @SpringBootApplication annotation is equivalent to using @Configuration, @EnableAutoConfiguration and @ComponentScan with their default attributes:

Here are some useful benefits of using Spring Boot:

i. Automatic configuration of an application uses intelligent defaults based on the classpath and the application context, but they can be overridden to suit the developer's requirements as needed.
ii. When creating a Spring Boot Starter project, you select the features that your application needs and Spring Boot will manage the dependencies for

you.

iii. A Spring Boot application can be packaged as a JAR file. The application can be run as a standalone Java application from the command line using the java -jar command.
iv. When developing a web application, Spring Boot configures an embedded Tomcat server so that it can be run as a standalone application. (Tomcat is the default, but you can configure Jetty or Undertow instead.) You can package the application as a WAR file and deploy it to an external servlet container if you prefer
v. Spring Boot includes many useful non-functional features (such as security and health checks) right out of the box.
vi. Although Spring Boot will autoconfigure the application for you, it also gives you a lot of flexibility in terms of giving you leeway to make customizations. Consequently, Spring Boot gives you the best of both worlds.