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

1. What is J2EE ?(Java Enterprise Edition)

Consists of specifications only .

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

What is enterprise application ?

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

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

These specifications include ---

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

Vendor of J2EE specs -- Oracle / Sun / Eclipse

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

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

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

2. WHY J2EE
1. Can support different types of clnts --- thin client(web clnt)
thick clnt --- application clnt(eg : TCP client)
smart clnts -- mobile clnts

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

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

4. Layers involved in HTTP request-response flow (refer to day2-data\day2_help\diags\request-response-flow.png)

Web browser sends the request (URL)
 eg : http://www.abc.com:8080/day4.2
/day2.1 --- root / context path /web app name

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

5. What is a 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 including request & response structure from day1-data\day1_help\j2ee_prerequisites\HTTP Basics

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

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

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

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

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

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

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

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

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

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

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

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

value : pages.SecondServlet

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

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

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

How to read request params sent from the clnt ?

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

2. public String[] getParameterValues(String paramName)

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

WHY JDBC ? (Java Database Connectivity)
JDBC ensures (PARTIAL) DB vendor independence + platform independence to Java Applications .

HOW it grants DB independence?

1. JAR supplied by DB vendor or Driver vendor --- consists of JDBC driver -- i.e. a converter for Java Data <-----> Native DB types & implementation classes , vendor specific.
2. JDBC API (java.sql) consists of largely --- Interfaces.

Java supplies specifications or WHAT (i/fs) & leaves implementation to DB vendors or 3rd party JDBC drvr vendors.

eg : java.sql.Connection --i/f sun supplied(Java SE)
Imple class provided by MySQL -- com.mysql.cj.jdbc.ConnectionImpl.class
Imple class provided by Oracle
oracle.jdbc.OracleConnection

DB2Connection etc....

Generalized steps for DB connectivity (ref : jdbc overview)

1. Place the JDBC driver in the Java classpath
Typically JDBC drivers are in form of JAR(Java archival format : compressed bundle of pkged Java classes) :
Oracle supplies Type IV Thin Client type of the Driver :
ojdbc14.jar/classes12.jar/ojdbc6.jar/ojdbc8.jar
MySQL supplies Type IV JDBC Driver : mysql-connector-java-8.0.20.jar

How to add JDBC drvr's JAR to the classpath(w/o IDE)
```
set classpath=g:\oracle\jdbc\lib\ojdbc8.jar;
```
With IDE --- simply Add external Jar.

2. Load & register JDBC driver

2.1. Load the JDBC driver in JVM's memory.

Method of java.lang.Class<T>
```
public static Class forName(String F.Q clsName) throws
ClassNotFoundException
```

eg : `Class.forName("com.mysql.cj.jdbc.Driver");`

3. Get the fixed DB connection thro' the JDBC driver.
API : java.sql.DriverManager (class)
```
public static Connection getConnection(String dbURL,String userName,String
password) throws SQLException
```

Params : dbURL : URL to reach DB thro the drvr.
jdbc:oracle:thin:@HostDetails --- for oracle Type IV thin clnt driver

HostDetails = DBServerHost:1521:SID

eg : jdbc:oracle:thin:@localhost:1521:orcl
For mysql
jdbc:mysql://localhost:3306/test?useSSL=false&allowPublicKeyRetrieval=true
test : DB name

4. Create the JDBC statement
Connection i/f method
public Statement createStatement() throws SQLException :
creates an empty JDBC stmt to hold the query & exec.

5. Fill in the query & execute the same.
Statement i/f method
If query is : select : u must use executeQuery method.If the query is DML other than select(i.e insert,delete,update) or DDL then use the method executeUpdate

5.1. For select query :(for result set returning query)
public ResultSet executeQuery(String sql) throws SQLException
Returns the result set consisting of selected rows & cols.

5.2. For others : (queries not returning RST)
public int executeUpdate(String sql) throws SQLException
Returns the updated row count : indicating how many rows were affected.

6.1 Process the ResultSet
API : ResultSet I/f method

public boolean next() throws SQLException
Advances the RST cursor to the next row & returns true : if valid data or false if no results.(end of results)
If valid data exists : then read row data
Methods from ResultSet i/f
Type getType(int colPosition) throws SQLException(colPos : as it appears in RST)
or
Type getType(String colName) throws SQLException

Type=JDBC data type.

Mapping bet. Oracle Data types & JDBC data type

varchar/varchar2	String
number(n)	integer
number(m,n)	double/float
date	java.sql.Date
TimeStamp	java.sql.TimeStamp

7 : Insertion of a row to the table (any query returning updateCnt)
Only changes are : query & replace executeQuery by executeUpdate.

8. If Java appln is exiting : close RST,close ST & then close Cn from the finally block or finalize method.(Typically closing Cn , closes all stmts & rsts.)

What is the type of the ResultSet so far created? : Forward type only & read only
To such a RST : previous() or abs. positioning meths(absolute(n) or relative(n) positioning meths will raise SE.

How to create a scrollable RST?
Replace step 4 by the following.

6. Create the JDBC statement to support scrollable RST
Connection i/f method
public Statement createStatement(int resultSetType,int concurrencyType) throws SQLException :
resultSet type : forward type(ResultSet.TYPE_FORWARD_ONLY) or scrollable(ResultSet.TYPE_SCROLL_INSENSITIVE OR ResultSet.TYPE_SCROLL_SENSITIVE) :

Difference between these 2
A result set that is TYPE_SCROLL_INSENSITIVE does not reflect changes made while it is still open and one that is TYPE_SCROLL_SENSITIVE does. Both types of result sets will make changes visible if they are closed and then reopened:

resultset Concurrency : read only result set(ResultSet.CONCUR_READ_ONLY) or updatable result set.(i.e can make changes to RST & same changes can also be applied to the DB table. can replace thus : insert,update,delete queries)

Additional API of scrollable ResultSet :
boolean absolute(int n) throws SE : tries to place the RST cursor on the nth row.

relative,afterLast,beforeFirst,first,last,previous,getRow

Why use PreparedStatement ?

1. PST represents pre-parsed & pre-compiled Stmts. At the time of creation of the PST, 3 steps out of 4(i.e parsing ,syntax chking,compiling) take place. So when User reqs for data(eg : via a button click) the only remaining step is : fill in user data & exec query.

2. U can't pass the IN params to Statement , but can pass IN params to PST.

How to create PST?
1. Use Connection i/f method :
public PreparedStatement prepareStatement(String sql) throws SE
eg : PreparedStatememt pst=cn.prepareStatement("select * from my_emp where id=?");
? : IN param. to be filled prior to query exec.

The RST associated with above PST is of : TYPE_FORWARD_ONLY & CONCUR_READ_ONLY

How to make it scrollable?
API : Connection i/f

public PreparedStatement prepareStatement(String sql,int resultType,int concurrencyType)
throws SE

3. How to set IN params of PST?(to be invoked : in event listener : after user gives i/p or in server side code after clnt sends request)
API : PreparedStatement i/f

void setType(int placeHolderPos,Type value) throws SE
Type : JDBC data type
PlaceHolder pos : 1.....counted from left
eg : to set emp id .
pst.setInt(1,....);

4. exec the query
 rst=pst.executeQuery();

5. process rst in the standard manner.

RMI clnt : sends emp id & RMI srvr contact DB : if emp exists ---sends emp info , ow. raises exc empnot found -- via pst.

CallableStatement : i/f from java.sql
Statement <--- PST <--- CallableStatement
why CST ?
1. Use CST to execute stored procedures & stored funs exisitng on DB
2. To pass IN,OUT,IN OUT type of params

Steps to invoke & exec. the stored proc/fun
1. Create CST
API : Connection i/f
public CallableStatement prepareCall(String invocationSyntax) throws SqlException
invocationSyntax for stored proc : "{call procName(?,?.....?)}"

invocationSyntax for stored fun : "{?=call funcName(?,?.....?)}"
? : represents IN,OUT or IN OUT param
{} : represent the esc seq. for the JDBC drvr. JDBC drvr will translate this invocation to a native DB invocation form.

2. Set IN params : methods inherited from PST

void setType(int placeHolderPos,Type value) throws SE

3. Register OUT / IN OUT params (i.e specify the JDBC data type of the OUT/IN OUT param to JVM)
Method of CST
void registerOutParameter(int paramPosition,int type) throws SE

paramPosition : placeHolder pos 1....
type : java.sql.Types : class constant

3.5 For IN OUT PARAM : invoke step 2 & 3 (ie. set IN val & register out param data type)

4. Execute the stored proc or a fun

public boolean execute() throws SE

Ret val is ignored.

5. Extract the results from OUT/IN OUT
CallableStatement methods
Type getType(int paramPos) throws SE
type : JDBC data type

Objective : Using scanner : accept sid,did,amt for funds transfer, exec the st.proc & disp the results.

DB Transactions
Functionally grouped SQL stmts : representing a B.L.
Tx => all the stmts from a Tx either fail or succeed.
i. e If any stmt fails : entire Tx has to be discarded.
The changes made by the Tx will be made permanent : IFF all the stmts succeed.
eg : Purchase a product
Involves 1. Checking availability of the product
2. Customer credit/debit limit & updating the same
3. Updating stock .

How to do it from JDBC API?

1. Start a Tx
Connection i/f method
void setAutoCommit(boolean flag)
ie. unset the auto-commit flag.
eg : cn.setAutoCommit(false);

2. Wrap entire Tx within a separate try-catch block.
3. If the entire try block succeds (i.e at the end of try) ---> commit the Tx

API : cn.commit();

4. But if u reach inside the catch clause(due to system exc or custom exc) : rollback the Tx

API : cn.rollback();

5. To continue : set auto-commit to true again.

6. To rollback a transaction partially , there exists additional method for setting save points.

Connection i/f method
public Savepoint setSavepoint() throws SQLException

7. How to restore the DB state to a savepoint ?

Connection i/f method
void rollback(Savepoint savepoint) throws SQLException
Undoes all changes made after the given Savepoint object was set.

Updatable ResultSet :
How to create a PST which supports scrollable & updatable RST?

1. API : Connection i/f

public PreparedStatement prepareStatement(String sql,int resultType,int concurrencyType)
throws SE
resultSet type : TYPE_SCROLL_INSENSITIVE/SENSITIVE
concurrencyType : CONCUR_UPDATABLE

2. Alternative to update query

2.1. Get the updatable RST.(eg : via pst.executeQuery())
2.2. Place the RST cursor on the row to be updated.(via absolute/relative meths of RST)
2.3. Update the col. vals-- on the RST

ResultSet API
public void updateType(int colPosition,Type newVal) throws SE
type--- JDBC data type
OR
public void updateType(String colName,Type newVal) throws SE
type--- JDBC data type

2.4. Once all changes to a particular row are done invoke :

API : public void updateRow() throws SE
to apply these changes to the underlying DB table.

3. Alternative to insert query

3.1. Get the updatable RST.(eg : via pst.executeQuery())

3.2. Place the RST cursor on the new row to be inserted.
API
ResultSet : void moveToInsertRow() throws SE
This places the RST cursor on the newly created row.

3.3. Update the col. vals-- on the RST copy
Invoke update methods (mandatory for NOT NULL constraint) : as in step 2.3

ResultSet API
public void updateType(int colPosition,Type val) throws SE
type--- JDBC data type

3.4. Once all col vals are inserted :
API : public void insertRow() throws SE
to apply these changes to the underlying DB table. (i.e new row gets inserted in DB)
3.5. To place cursor back to original row
API : public void moveToCurrentRow() throws SE

4. Alternative to delete query
4.1. Get the updatable RST.(eg : via pst.executeQuery())
4.2. Place the RST cursor on the row to be deleted (via absolute/relative)
4.3. Delete row :
ResultSet API
void deleteRow() throws SE (NOTE : deletes row from RST & DB too!!!!! use it with care!)

For date/time handling from JDBC
classes to be used from java.sql are :
Date,Time & TimeStamp

{d 'yyyy-mm-dd'}
{t 'hh:mm:ss'}
{ts 'yyyy-mm-dd hh:mm:ss'}

steps : for handling date
1. Create a table with col. type=date
2. Create a PST
3. Use java.sql.Date API
method :
public static Date valueOf(String dateFormat)

dateFormat : yyyy-mm-dd

4. Use PST's method
public void setDate(int pos,Date val) throws SE.

Meta data associated with JDBC
1. Database meta data : holds the info like : DB version,DB drvr version, Tx are supported or not, scrollable/updateble rsts, names of all tables from DB.....,max conns available

```
To get D.M.D
API : Connection i/f
DatabaseMetaData getMetaData() throws SE
DatabaseMetaData : i/f
Has methods : getVersion(),getTables().....


How to get all the table names for the current user?
Use DMD : method
ResultSet getTables(String catalog,String schemaPattern,
                    String tableNamePattern,
                    String[] types)
                    throws SQLException

Usage
DatabaseMetadata dmd=cn.getMetaData();
ResultSet rst=dmd.getTables(null,null,null,new String[] {"TABLE"});
//to retrieve table name
invoke : rst.getString(3) ; //3 => table name


2.  ResultSetMetaData : metadata about the RST
How to get it?
Method in ResultSet API
ResultSetMetaData getMetaData() throws SE
eg :
ResultSetMetaData rmd=rst.getMetaData();

2.1. Methods of RMD
int getColumnCount() throws SE
String getColumnLabel(int colPos) throws SE
int getColumnType(int colPos) throws SE



Dirty Read --Enables un-committed tx data, to read from current tx.
Un-repeatable reads -- Enables to read committed data from concurrent tx,
may lead to un repeatable results.
Phantom reads-- Enables to read committed data from concurrent tx, may
lead to additional rows appearing in same tx.


Handling BLOBs with JDBC API
How to store BLOB data?
1.  Create DB table having blob type of column.
eg create table my_images(id number(2),name varchar2(30),snap blob);
2.  Accept bin file from user to store on DB.
3.  Use PreparedStatement API method -- to store BLOB on DB
API
public void setBinaryStream(int placeholderPos, InputStream in, int
length) throws SqlException
4.  Use executeUpdate to insert row data.

How to restore BLOB data from DB ?
```

1. Use API of PreparedStatement to read BLOB.
public Blob getBlob(int colPos) throws SqlException
2. Use java.sql.Blob i/f method
public byte[] getBytes(long pos,int length)
NOTE : pos begin with 1 .
3. Once u have byte[] , u can store the same on File(bin) using FOS or send it over sockets using Socket.getOutputStream()

Reference for MySQL connectivity
1. install MySQL

2. Clnt i/f
create database testjdbc;
use testjdbc;

create table Employee(empId int primary key, name varchar(25), deptId int, isPermanent boolean,sal double);
insert into Employee values(1,'aa',123,true,2000);
insert into Employee values(2,'ab',101,true,3000);

Driver class name : com.mysql.jdbc.Driver
To load/register driver ---- Class.forName(String F.Q className) throws ClassNotFoundExc
DB URL - jdbc:mysql://hostname:3306/databaseName
root -- user name
root -- password
example code for conn to MySQL ----
Class.forName("com.mysql.jdbc.Driver");
String dbURL="jdbc:mysql://localhost:3306/testjdbc";
//use DM.getConnection(url,username,pass)

Objective ---- RMI & JDBC integration
Func requirement --1. disp emp dtls --- if present , ow. raise cust exc.
3. Insert new emp record --- ret success msg or raise cust exc in case failure.

Server side steps
1. B.I --- method decl ---
String getEmpDtls(int empId) throws RE,EmpNotFoundExc
2. String insertEmp(emp specific dtls) throws RE,EmpInsertExc
3. Create impl class --- rem obj
constr --- cn,psts
B.M ---get ----
insert

HOW TO make JDBC applns/applets completely DB independent?

1. Create text based properties file.
key & value pair.(keys --- arbitrary values---changing as per DB setting)
2. Create empty java.util.Properties<K,V> --- sub-class of HashTable
Key & values must be --- String
Can load Properties directly from any stream.
Properties API
public void load(Reader r) throws IOExc

3. Can access the Property value using API
Properties API
String getProperty(String key)
ret type=value asso with key.

eg--

```
Properties props = new Properties();

FileInputStream in = new FileInputStream("database_mysql.properties");
props.load(in);
in.close();
String drivers = props.getProperty("jdbc.drivers");
Class.forName(drivers);
String url = props.getProperty("jdbc.url");
String username = props.getProperty("jdbc.username");
String password = props.getProperty("jdbc.password");
return DriverManager.getConnection(url, username, password);
```

Regarding jar cmd line utility
0. For runnable jars --- create manifest.txt --- 1liner having Main-Class: tester.Test, new line & save file
1. cd to folder where ur classes are(eg bin)

1. From bin --- jar cvfm test.jar manifest.txt *
2. To run jar
java -jar test.jar

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

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

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

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

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

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

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

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

2 Ways
1. Client Pull
Taking the client to the next page in the NEXT request (coming all the way from client)
1.1. User takes some action --eg : clicking on a button or link & then
client browser generates new URL to take user to the next page.

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

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

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

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

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

2. Forward scenario
API of RequestDispatcher
public void forward(ServletRequest rq,ServletResponse rs)

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

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

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

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

3. Include scenario
API of RequestDispatcher

```
public void include(ServletRequest rq,ServletResponse rs)
```

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

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

What is a Session?

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

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

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

What is the need of session tracking?

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

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

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

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

Techniques

1. Plain Cookie based scenario

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

Steps :

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

2. Add the cookie/s to the resp hdr.

HttpServletResponse API :
void addCookie(Cookie c)

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

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

int getMaxAge()

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

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

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

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

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

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

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

1. Get Http Session object from WC

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

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

OR
HttpSession getSession(boolean create)

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

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

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

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

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

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

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

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

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

OR via xml tags in web.xml

```
<session-config>
  <session-timeout>5</session-timeout> : unit : min
</session-config>
```

NOTE :
What is an attribute ?
attribute = server side object(entry/mapping=key value pair)
who creates server side attrs ? -- web developer (servlet or JSP prog)
Each attribute has --- attr name(String) & attr value (java.lang.Object)
Attributes can exist in one of 3 scopes --- req. scope,session scope or application scope

1. Meaning of req scoped attr = attribute is visible for current req.
2. Meaning of session scoped attr = attribute is visible for current session.(shared across multiple reqs coming from SAME clnt)
3. Meaning of application scoped attr = attribute is visible for current web appln.(shared across multiple reqs from ANY clnt BUT for the SAME web application)

Executor Framework (a part of Java SE)

Introduced in Java 5.

What's earlier support ?

Extends Thread
Implements Runnable

Why Executor Framework?

If you have thousands of task to be executed and if you create each thread for thousands of tasks, you will get performance overheads as creation and maintenance of each thread is an overhead.

Executor framework solves this problem.

In executor framework, you can create specified number of threads and reuse them to execute more tasks once it completes its current task.

It simplifies the design of creating multithreaded application and manages thread life cycles.

The programmer does not have to create or manage threads themselves, that’s the biggest advantage of executor framework.

Important classes / interfaces for executor framework.

1. java.util.concurrent.Executor
This interface is used to submit new task.
It has a method called “execute”.

```
public interface Executor {
 void execute(Runnable task);
}
```

2. ExecutorService
It is sub-interface of Executor.
Provides methods for
Submitting / executing Callable/Runnable tasks
Shutting down service
Executing multiple tasks etc.

3. ScheduledExecutorService
It is sub-interface of executor service which provides methods for scheduling tasks at fixed intervals or with initial delay.

4. Executors
This class provides factory methods for creating thread pool based executors.

Important factory methods(=public static method rets instance of ExecutorService) of Executors are:

4.1. newFixedThreadPool: This method returns thread pool executor whose maximum size is fixed.If all n threads are busy performing the task and additional tasks are submitted, then they will have to wait in the queue until thread is available.

4.2.
newCachedThreadPool: this method returns an unbounded thread pool. It doesn’t have maximum size but if it has less number of tasks, then it will tear down unused thread. If a thread has been unused for keepAliveTime , then it will tear it down.

4.3. newSingleThreadedExecutor: this method returns an executor which is guaranteed to use the single thread.

4.4. newScheduledThreadPool: this method returns a fixed size thread pool that can schedule commands to run after a given delay, or to execute periodically.

Steps for Runnable

1. Create a thread-pool executor , using suitable factory method of Executors.

eg : For fixed no of threads

```
ExecutorService executor = Executors.newFixedThreadPool(10);
```

2. Create Runnable task

3. Use inherited method

```
public void execute(Runnable command)
```

Executes this Runnable task , in a separate thread.

4. Shutdown the service

```
public void shutdown()
```

Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.

5.
```
boolean awaitTermination(long timeout,TimeUnit unit)
              throws InterruptedException
```

Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs.

6.

```
List<Runnable> shutdownNow()
```

Attempts to stop all actively executing tasks, halts the processing of waiting tasks, and returns a list of the tasks that were awaiting execution.

BUT disadvantages with Runnable interface

1. Can't return result from the running task
2. Doesn't include throws Exception .

Better API

java.util.concurrent.Callable<V>

V : result type of call method

Represents a task that returns a result and may throw an exception.

Functional i/f
SAM :
public V call() throws Exception
Computes a result, or throws an exception if unable to do so.

Steps in using Callable i/f

1. Create a thread-pool executor , using suitable factory method of Executors.

eg : For fixed no of threads
ExecutorService executor = Executors.newFixedThreadPool(10);

2. Create Callable task , which returns a result.

3. To submit a task to executor service , use method of ExecutorService i/f :

public Future<T> submit(Callable<T> task)
Submits a value-returning task for execution and returns a Future representing the pending results of the task. It's a non blocking method (i.e rets immediately)

The Future's get method will return the task's result upon successful completion.

If you would like to immediately block waiting for a task, invoke get() on Future.
eg : result = exec.submit(aCallable).get();

OR
main thread can perform some other jobs in the mean time & then invoke get on Future , to actually get the results. (get : blocking call ,waits till the computation is completed n then rets result)

4. Other methods of ExecutorService i/f

public List<Future<T>> invokeAll(Collection<? extends Callable<T>> tasks) throws InterruptedException

It's a blocking call.(waits till all tasks are complete)
Executes the given tasks, returning a list of Futures holding their status and results when all complete. Future.isDone() is true for each element of the returned list.

5. Shutdown the service
public void shutdown()
Initiates an orderly shutdown in which previously submitted tasks are executed, but no new tasks will be accepted.

6. boolean awaitTermination(long timeout,TimeUnit unit)
                  throws InterruptedException
Blocks until all tasks have completed execution after a shutdown request, or the timeout occurs.

7.

```
List<Runnable> shutdownNow()
Attempts to stop all actively executing tasks, halts the processing of
waiting tasks, and returns a list of the tasks that were awaiting
execution.
```

Regarding SERVLET CONFIG

A servlet specific configuration object created by a servlet container to pass information to a servlet during initialization.

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

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

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

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

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

Regarding javax.servlet.ServletContext (i/f)

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

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

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

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

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

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

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

5.4 Creating request dispatcher
H.W

What is a Servlet Listener(or web application listener)?

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

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

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

```
<listener>
 <listener-class>F.Q cls name of listener</listener-class>
</listener>
```

Expression Language implicit variables(case sensitive)

1. pageContext : PageContext object (javax.servlet.jsp.PageContext) asso. with current page.
2. pageScope - a Map that contains page-scoped attribute names and their values.
3. requestScope - a Map that contains request-scoped attribute names and their values.
4. sessionScope - a Map that contains session-scoped attribute names and their values.
5. applicationScope - a Map that contains application-scoped attribute names and their values.
6. param - a Map that contains rq. parameter names to a single String parameter value (obtained by calling ServletRequest.getParameter(String name)).
7. paramValues - a Map that contains rq. param name to a String[] of all values for that parameter (similar to calling ServletRequest.getParameterValues(name)
8. initParam - a Map that contains context initialization parameter names and their String value (obtained by calling ServletContext.getInitParameter(String name)).

eg : ${initParam.db_drvr}

9. cookie : Map.Entry of cookies. (entrySet of cookies)

eg : ${cookie.cookiename.value}

key ---cookie name
value ---javax.servlet.http.Cookie

${cookie.JSESSIONID.value}
---cookie.get("JSESSIOIND").getValue()

10. To retrieve err details from Error handling page.

```
 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}
```

eg :

```
<c:set var="abc" scope="session" value="Hello User...."/>
${sessionScope.abc}
```

Session Tracking technique :
 HttpSession + URL rewriting

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

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

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

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

How to ?

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

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

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

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

WHY JSP?

1. JSP allows developer to separate presentation logic(dyn resp generation) from Business logic or data manipulation logic.
Typically JSPs -- used for P.L(presentation logic)
Java Beans or Custom Tags(actions) --- will contain Business logic.

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

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

JSP API
jsp-api.jar --- <tomcat>/lib : specs

Contains JSP API implementation classses. : jasper.jar

0. javax.servlet.Servlet -- super i/f
1. javax.servlet.jsp.JspPage -- extends Servlet i/f
1.1. public void jspInit()
1.2. public void jspDestroy()

Can be overridden by JSP page author

2. Further extended by javax.servlet.jsp.HttpJspPage
2.1. public void _jspService(HttpServletRequest rq,HttpServletResponse rs)
     throws ServletExc,IOExc.

Never override _jspService ---JSP container auto translates JSP tags (body) into _jspService.

JSP life-cycle

1. Clnt sends the 1st request to the JSP (test.jsp)
2. Web-container invokes the life cycle for JSP
3. Translation Phase : handled by the JSP container.
I/p : test.jsp O/p : test_jsp.java (name : specific to the Tomcat container)
Meaning : .jsp is translated into corresponding servlet page(.java)
Translation time errs : syntactical errs in using JSP syntax.
In case of errs : life-cycle is aborted.
4. Compilation Phase : handled by the JSP container.

I/p : Translated servlet page(.java)   O/p : Page Translation class(.class)
Meaning : servlet page auto. compiled into .class file
Compilation time errs: syntacticle errs in generated Java syntax.
5. Request processing phase / Run time phase. : typically handled by the Servlet Container.
6. S.C : will try to locate,load,instantiate the generated servlet class.
7. The 1st it calls : public void jspInit() : one time inits can be performed.(jspInit availble from javax.servlet.jsp.JspPage)
8. Then it will call follwing method using thrd created per clnt request :

public void _jspService(HttpServlet Rq,HttpServletResponse) throws ServletException,IOException(API avlble from javax.servlet.jsp.HttpJspPage)
When _jspService rets , thread's run method is over & thrd rets to the pool, where it can be used for servicing some other or same clnt's req.

9. . At the end ...(server shutting down or re-deployment of the context) : the S.C calls

public void jspDestroy()
After this : translated servlet page class inst. will be GCEd....

10. For 2nd req onwards ...... : SC will invoke step 8 onwards.

JSP 2.0/2.1/2.2/2.3 syntax
1. JSP comments

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

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

2. JSP's implicit objects (available only to _jspService) -- 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 PrintWriter in servlets)

Has the same API as PW(except printf)
usage eg : out.print("some text sent to clnt");

2.2. request : HttpServletRequest (same API)

2.3. response : HttpServletResponse

2.4. config : ServletConfig (used for passing init params)

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

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

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

2.8. application : ServletContext(used for Request dispatching, server side logging, for creating context listeners,to avail context params, to add/get context scoped attrs)

2.9. 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 . (Use only till you learn Javabeans & custom tags or JSTL,). we will use use the scriptlets to add : Req. processing logic, B.L & P.L)

syntax : <% java code...... %> : within <body> tag.
location inside the translated page : within _jspService
usage : till Java beans / JSTL or cust. tags are introduced : scriptlets used for control flow/B.L/req. proc. logic

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

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

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

Better alternative to JSP Expressions : EL syntax (Expression Language : avlble from JSP 1.2 onwards)
syntax : ${expr to evaluate} (to be added directly in ,<body> tag)

EL syntax will evaluate the expr ---to String --sends it clnt browser.

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

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

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

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

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

eg : suppose ctx scoped attr --- loan_scheme
${applicationScope.loan_scheme} ---
getServletContext().getAttribute("loan_scheme") ---to string --sent to
clnt

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

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

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

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

${pageContext.session.maxInactiveInterval}

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

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

param ----map of req parameters.

param : req. param map

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

${abc} -----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 %> (outside <body>)
Usage : 1. for creating page scoped java variables & methods (instance vars & methods/static members)
4. Also can be used for overriding life cycle methods
    (jspInit,jspDestroy)

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

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

Syntax ---
<%@ Directive name attrList %>
1. page directive
--- all commands applicable to current page only.
Syntax
<%@ page import="comma separated list of pkgs" contentType="text/html" %>
eg -- <%@ page import="java.util.*,java.text.SimpleDateFormat" contentType="text/html" %>
Imp page directive attributes
1. import --- comma separated list of pkgs
2. session --- boolean attribute. default=true.
To disable session tracking, spectify session="false"

3. errorPage="URI of err handling page" ---
tells WC to forward user to err handler page.
4. isErrorPage="true|false" def = false

If you enable this to true--- one can access 'exception' implicit object from this page.

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

Additional EL syntax

EL syntax to be used in error handling pages

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

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

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

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

Equivalent step in Servlet
Servlet class can imple. tag i/f --
javax.servlet.SingleThreadModel(DEPRECATED) -- WC ensures only 1thread (representing clnt request) can invoke service method. --NOT recommended.

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

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

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

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

OR

```
<jsp:actionName attr list />
```

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

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

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

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

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

```
eg -- public double getRegAmount(){...}

5.  Business Logic --- methods
public methods --- no other restrictions
----------------------------
Using Java Beans from JSP Via standard actions

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

default = page scope.


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

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

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


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

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

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

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

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

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

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

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

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

usage

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

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

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

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

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

${pageContext.exception.message}

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

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

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

Why JSTL ? JSP standard tag library
When JSP standard actions are in-sufficient to solve requirements ,
w/o writing scriptlets --- use additional standard actions --- supplied as JSTL actions
JSP standard Tag Library
--- has become standard part of J2EE specs from version 1.5 onwards.
---It's support exists in form of a JAR ---
1. jstl-1.2.jar
For using JSTL steps
1. Copy above JAR into your run-time classpath(copy jars either in <tomcat_home>/lib OR <web-inf>/lib
2. Use taglib directive to import JSTL tag library into JSP pages.
tag=action
tag library=collection of tags
supplier = JSTL vendor(specification vendor=Sun, JAR vendor=Sun/any J2EE compliant web/app server)
jstl.jar --- consists of Tag implementation classes
Tag libr- TLD -- Tag library descriptor -- desc of tags -- how to use tags
<%@ taglib uri="URI of JSTL tag lib" prefix="tag prefix" %>

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

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

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

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

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

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



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

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

WC invokes ---

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

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


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

<input type="submit" name="btn" value="Withdraw"

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

      formaction="transactions.jsp" /></td>

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

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

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

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

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

4.2. JSTL action --- for URL rewriting

```
<c:url var="attr Name" value="URL to be encoded"
scope="page|request|session|application"/>

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

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

How to set session tm out ?

1. programmatically --- using Java API

From HttpSession --- setMaxInactiveInterval(int secs)

2. declarativally -- either using Java annotations OR using XML config files (web.xml)

JSP Syntax (2.x)

JSP implicit objs : accessible from scriptlets(<% java code %> n expressions (<%= dynamic expr %>)
request,response, session, config,out,application,page , pageContext, exception
eg : one.jsp --- one_jsp.java
<%= page %>
o/p : org.apche.jsp.one_jsp@5375675
page => instance of translated page class

pageContext : JSP implicit obj : accessible from scriptlets n exprs

javax.servlet.jsp.PageContext class --abstract --- concrete sub class --- server supplied jar (jasper.jar)
--PageContextImpl --current page environment

It holds the references of all other implicit objects
eg : request,response, session, config,out....
Methods : getSession, getOut,getRequest.....
+ to store page scoped attr.

Which is better alternative to <%= .... %> JSP exprs ?
EL syntax
${...} : added directly in JSP body

EL implicit objs (names of maps created by WC , except pageContext)
param,pageScope,requestScope,sessionScope,applicationScope,cookie,initParam
eg : ${param.email}
WC : out.print(param.get("email"));
OR : request.getParameter("email") --> sent to clnt

Solve : ${requestScope.user_info}
WC : request.getAttribute("user_info") ---> sent to clnt

${cart} :
WC : pageContext.getAttribute("cart") --null
request.getAttribute("cart") --null
session.getAttribute("cart") --not null --to string --sent to clnt

eg : In one.jsp
```
<%
    pageContext.setAttribute("nm",val);
%>
```

How to acces this attribute from one.jsp ?
Options
1. ${nm}

```
2.  ${pageContext.nm}
3.  ${page.nm}
4.  ${pageScope.nm}
5.  <%= pageContext.getAttribute("nm") %>

Ans : 1,4,5



Q. How to get a cookie value , with a name JSESSIONID ?

JSP expression
<%= session.getId() %>


OR
via cookie : EL syntax
${session.id} : wrong(since session IS NOT EL impl obj)
${pageContext.session.id} => pageContext.getSession().getId() --> sent to
clnt
OR
${cookie.JSESSIONID.value} => cookie.get("JSESSIONID").getValue()  -->
sent to clnt



pageContext : can be accessed from
1.  declaration
2.  expression
3.  EL
4. scriptlets
Ans : 2,3,4

Various uses of pageContext
Solve

eg : ${pageContext.session.id} --
WC : pageContext.getSession().getId() --> sent to clnt

What will be o/p for
http://host:port/day8/one.jsp
In one.jsp :
${pageContext.request.contextPath} --
WC : pageContext.getRequest().getContextPath() ---> sent to clnt
o/p : /day8


How will you get the value of session time out ?
?
1.  session = HttpSession
2.  sessionScope = name of the map containing session scoped attrs
Ans : 1
HOW ? ${pageContext.session.maxInactiveInterval} =>
pageContext.getSession().getMaxInactiveInterval()
```

```
JSP Expression : <%= session.getMaxInactiveInterval() %>


How to set session scoped attribute ?

<%
   session.setAttribute("nm",val);
%>
How to retrieve using following options ?
1.  session => HttpSession (Hint : JSP expression)
<%= session.getAttribute("nm") %>

OR
2.  sessionScope = name of the map containing session scoped attrs
Better option (EL syntax) :
${sessionScope.nm)
OR
${nm}
```

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

4.
Collection Value Types :
Hibernate allows to persist collections.
But Collection value Types can be either collection of Basic value types, Composite types and custom types.
eg :
Collection mapping means mapping group of values to the single field or property. But we can’t store list of values in single table column in database. It has to be done in a separate table.

eg : Collection of embeddables
```
@ElementCollection
      @CollectionTable(name="CONTACT_ADDRESS",
joinColumns=@JoinColumn(name="USER_ID"))
      @AttributeOverride(name="streetAddress",
column=@Column(name="STREET_ADDRESS"))
      private List<ContactAddress> address;
```

eg : collection of basic type
```
      @ElementCollection
      @CollectionTable(name="Contacts",
joinColumns=@JoinColumn(name="ID"))
      @Column(name="CONTACT_NO")
      private Collection<String> contacts;
```

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 Hibernate
JPA ---standard part of J2EE specification --vendor --J2EE / Jakarta EE(sun/oracle/Eclipse)
Implementation classes -- JAR ---hibenrate core JARs(implementor of JPA)

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 / 6.x
Other popular ORM Frameworks
EclipseLink,iBATIS,Kodo 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)

4. 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
or hikari in spring boot) 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 Session? (org.hibernate.Session : interface)

Session object is persistance manager for the
hibernate application

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

Session object provides methods to perform
CRUD operations

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

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

Example of CRUD

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

What is SessionFactory? (org.hibernate.SessionFactory : interface)

It is a provider(factory) of session objects.

We use sessionfactory object to create session
object(via openSession or getCurrentSession).
It is singleton(1 instance per DB / application) ,immutable,inherently
thrd safe.
It is a heavy weight object, therefore it has to
be created only once in the beginning for an application & that too at the
very beginning.
It is associtated with L2 cache(must be explicitely enabled)

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

In earlier versions of hibernate , Configuration object was 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.
4. If there is no persistent instance currently associated with the session, it will be loaded.
5. 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.
6. 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);

There are many advantages of Hibernate Framework over JDBC

1) Opensource , Lightweight : Hibernate framework is opensource & lightweight.
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 by default.

Third type of cache is --query level cache.(not implicitely enabled)

3) Database Independent query: HQL (Hibernate Query Language) / JPQL (Java persistence 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.

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.

5) Simplifies complex join: To fetch data form multiple tables is easy in hibernate framework.

eg : To display the course names ordered by desc no of participants (many-to-many)
select c.name from dac_courses c inner join course_studs cs on c.id = cs.c_id inner join dac_students s on cs.s_id = s.stud_id group by c.id order by count(*) desc;
 JPQL -- select c from Course c join fetch c.students group by c.id order by count(*) desc

6) Provides query statistics and database status: Hibernate supports Query cache and provide statistics about query and database status.

7. 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.

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

So if we change the database, then also our application will works as HQL is database independent
HQL contains database independent commands

8. While we are inserting any record, if we dont 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 ;)
9. 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.

Hibernate supports annotations, apart from XML

10. Hibernate provided Dialect classes, so we no need to write sql queries in hibernate, instead we use the methods provided by that API.
11. Getting pagination in hibernate is quite simple.

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. 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.

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

Usage of Hibernate Session API's get()

```
int id=101;
BookPOJO b1=hibSession.get(BookPOJO.class,id);

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

3. Display all books info :

using HQL -- Hibernate Query Language/JPQL --- 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"

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

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

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

```
List<T> getResultList()

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

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

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


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

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

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

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

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

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


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

5.  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)
Objective : dec. price of all books with author=specified author.


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

set named In params
exec it (executeUpdate) ---
```

```
int updateCount= hs.createQuery(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.

6. 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.(Bulk delete)
eg

```
int deletedRows = hibSession.createQuery ("delete Subscription s  WHERE
s.subscriptionDate < :today").setParameter ("today", new Date
()).executeUpdate ();
```

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

1. Iterator iterate() throws HibernateException

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

Entities returned --- in lazy manner

Pagination

2. Query setMaxResults(int maxResults)

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

3. Query setFirstResult(int firstResult)

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

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

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

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

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

5. org.hibernate.query.Query API

<T> T getSingleResult()

Executes a SELECT query that returns a single 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("i
d", 1);

8. How to invoke native sql from hibernate?
Query q=hs.createSQLQuery("select * from
books").addEntity(BookPOJO.class);
 l1 = q.list();

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

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

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

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

1.1 Testing core api
persist ---
public void persist(Object transientRef)
---persists trasient POJO .

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.

11.
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.

12. 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)

13. get vs load
& LazyInitilalizationException.

14. 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.

4. If there is no persistent instance currently associated with the session, it will be loaded.
5. 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.

6. 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

What is Maven ?

Build automation tool for overall project management.

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

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

Maven tries 2 describe

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

Vendor -- Apache

Earlier build tool -- Ant
Vendor -- Apache.

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

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

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

Maven advantages

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

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

What is POM? (Project Object Model)

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

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

Contents
Dependencies used in the projects (Jar files)
Plugins used
Project version
Developers involved in the project
Build profiles etc.

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

Elements of maven pom.xml file

1. project It is the root element of pom.xml file.
2. modelVersion It is the sub element of project. It specifies the modelVersion.
3. groupId It is the sub element of project. It specifies the id for the project group.(typically organization name)
4. artifactId It is the sub element of project. It specifies the id for the artifact (project). An artifact is something that is either produced or used by a project. Examples of artifacts produced by Maven for a project include: JARs, WARs.
5. version It is the sub element of project. It specifies the version of the artifact under given group.
6.

packaging -- defines packaging type such as jar, war etc.

7. name -- defines name of the maven project.
8. plugins ---compiler plugins , eclipse plugins
9. dependencies -- collection of dependencies for this project.

Within that --
dependency -- defines a specific dependency.(eg : hibernate dependency,spring web)

10. scope -- defines scope for this maven project. It can be compile, provided, runtime, test and system.

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

What is a Maven Repository

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

Local Repository
Central Repository

Remote Repository

Maven searches for the dependencies in the following order:

Local repository then Central repository then Remote repository.
maven repositories

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

1. Maven Local Repository

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

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

2) Maven Central Repository

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

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

3) Maven Remote Repository

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

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

Maven comes with 3 built-in build life cycles

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

Build Profiles in Maven

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

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

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

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

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

```
  mvn -- version
```

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

Hibernate's   Automatic Dirty checking

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

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

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

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

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

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

A session gets flushed when the transaction is commited.

Advanced Hibernate
Relationship between Entity n Entity
(Inheritance , Association : HAS-A)

Types of associations
one-to-one
one-to-many
many-to-one
many-to-many

Objective --Using one-to-many & many-to-one assocition between entities

eg : Course 1 <---->* Student
Different type of relationships between entities
One To One
One To Many
Many To One
Many To Many

1. One To Many bi directional relationship between the entities
JPA Annotations : @OneToMany & @ManyToOne

eg : Course 1 <----->* Student
Table Relationship
courses Table columns : id,title, start_date , end_date , fees , capacity
students Table columns : id,name,email + Foreign Key(FK) : course_id

Since courses table has a OneToMany relationship with the students table , a single course row can be referenced by multiple student rows.

The course_id column in the students table , maps this relationship via a foreign key that references the primary key of the courses table.

Since you can't insert a student record , w/o course record
parent-side (@OneToMany) : course
child-side (@ManyToOne) : student

The @ManyToOne association is responsible for synchronizing the foreign key column with the Persistence Context (the First Level Cache).

As a thumb rule (for perfomance benefits) : DO NOT use uni directional @OneToMany associations

Owning side of the association
The side having the join column in its table is called the owning side or the owner of the relationship.

Non owning (inverse side)
The side that does not have the join column is called the non□owning or inverse side.

Entities involved :
Course Entity
Student Entity

```
Description
Course : one , parent , inverse

Student : many , child , owning side (FK)




Best Practices to code a bidirectional @OneToMany association
eg : Course 1 <----->* Student
Entity Relationships
Course POJO properties : id,title, startDate , endDate , fees +
@OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL,orphanRemoval
=true)
private List<Student> students=new ArrayList<>();
Note : Always init collection to empty one , to avoid null pointer
excpetion

Student POJO properties : id,name,email +
@ManyToOne
@JoinColumn(name="course_id")
private Course selectedCourse

Detailed explanation
1.  Add Suitable mapping annotations : @OneToMany & @ManyToOne
otherwise JPA / Hibernate throws MappingException

2.  Add mappedBy attribute in the inverse side of the association

What is mappedBy & when it's mandatory?
Mandatory only in case of bi-dir associations
It's attribute of the @OneToMany / @ManyToMany / @OneToOne annotation.

What will happen if you don't add this attribute , in case of one-to-many
 Additional table (un necessary for the relationship mapping) gets created

It MUST appear in the inverse side of the association.
What should be value of mappedBy ?
name of the association property as it appears in the owning side.

eg : In Course POJO  : inverse side
@OneToMany(mappedBy="selectedCourse")
public List<Student> getStudents() {..}

3.  Use @JoinColumn to Specify the Join Column Name (FK column)
Use it to override hibernate's default naming strategy for column names.

4.  Cascade from Parent-Side to Child-Side
If you don't add cascade option : what will happen ?
eg : When try to save Course object, with multiple students, insert query
gets fired only on courses table.
Reason -- default cascade type = none
```

Solution --Add suitable cascade type & observe.
eg : @OneToMany(mappedBy="selectedCourse",cascade=CascadeType.ALL)
public List<Student> getStudents(){...}

5. What will happen if simply add student reference into the list?
eg :
```
eg : Course newCourse=new Course(....);
newCourse.getStudents().add(newStudent1);
newCourse.getStudents().add(newStudent2);
newCourse.getStudents().add(newStudent3);
session.persist(newCourse);
```

Ans : 1 record will be inserted into courses table. Thanks to cascade option , 3 records will be inserted into students table. BUT value of FK will be null.
Why : No linking from child ----> parent &

Which is the best way to establish bi-dir linking (As per THE founder of Hibernate : Gavin King)
Add helper methods in the parent side of the POJO
eg : In Course POJO
```
public void addStudent(Student s)
{
  students.add(s);
  s.setSelectedCourse(this);
}
```

For removing bi dir link
```
public void removeStudent(Student s)
{
  students.remove(s);
  s.setSelectedCourse(null);
}
```
Above approach is recommended to keep both sides of the association in sync.

6. Set orphanRemoval on the Parent-Side
Setting orphanRemoval on the parent-side guarantees the removal of children without references.
It is good for cleaning up dependent objects that should not exist without a reference from an owner object.

eg : Cancel Student admission

Excellent reference : https://vladmihalcea.com/orphanremoval-jpa-hibernate/

Objectives :

1. Objective : Launch new course
DAO

ICourseDao
String launchCourse(Course c);

2. Admit student
I/p -- student name, email, course name
o/p -- student details inserted + linked with FK

DAO --IStudentDao
String admitNewStudent(String courseName,Student s);

3. Cancel Course
i/p : course id

4. Objective :
Launch a new course , having no of students
eg : Course : hibernate.....
s1,s2,s3,s4 : have already paid the fees for the course
Expected o/p : 1 rec should be inserted in course table & 4 recs in student tbl + linked with FK

If you don't add cascade option : problem observed
When try to save Course object, with multiple students, insert query gets fired only on courses table.
Reason -- def cascade type = none
Solution --Add suitable cascade type & observe.
eg : @OneToMany(mappedBy="selectedCourse",
cascade=CascadeType.ALL)
public List<Student> getStudents(){...}

5. Cancel Admission
String cancelAdmission(String courseName,int studentId);
Hint : use helper method.
Any problems ?????
Solution : orphanRemoval

6. Get Course Details
i/p : course name

Display course details

Display enrolled student details
Any problems observed ?

Problem associated with one to many
org.hibernate.LazyInitializationException
Trigger : GetCourseDetails : while accessing the Student details

WHY ?

Hibernate follows default fetching policies for different types of associations
one-to-one : EAGER

one-to-many : LAZY
many-to-one : EAGER
many-to-many : LAZY

one-to-many : LAZY
Meaning : If you try to fetch details of one side(eg : Course) , will it fetch auto details of many side ?
NO (i.e select query will be fired only on courses table)
Why ? : for performance

When will hibernate throw LazyInitializationException ?
Any time you are trying to access un-fetched data from DB , in a detached manner(outside the session scope)
cases : one-to-many
many-many
session's load

un fetched data : i.e student list in Course obj : represented by : proxy (substitution) : collection of proxies
proxy => un fetched data from DB

Solutions
1. Change the fetching policy of hibernate for one-to-many to : EAGER
eg :

```
@OneToMany(mappedBy = "selectedCourse",cascade = CascadeType.ALL,fetch=FetchType.EAGER)
      private List<Student> students=new ArrayList<>();
```

Is it recommneded soln : NO (since even if you just want to access one side details , hib will fire query on many side) --will lead to worst performance.

2.

```
@OneToMany(mappedBy = "selectedCourse",cascade = CascadeType.ALL)
      private List<Student> students=new ArrayList<>();
```

Solution : Access the size of the collection within session scope : soln will be applied in DAO layer

Dis Adv : Hibernate fires multiple queries to get the complete details

3. How to fetch the complete details , in a single join query ?
Using "join fetch" keyword in JPQL

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

Another trigger for lazy init exception
: Session's API
load.

More details regarding "mappedBy"
eg :
Branch 1 -----* Student
In JPA or Hibernate, entity associations are directional, either unidirectional or bidirectional. Always mappedBy attribute is used in bidirectional association to link with other side of entity.

In the above tables, BRANCH and STUDENT tables has One-To-Many association. In Hibernate association mappings, each entity plays a role of either owning-entity (the entity which is having foreign key mapping of other entityâ€™s mapped table) or non owning entity (the other side of owning entity).

In above One-To-Many relation, Student entity has owning-side role, which is mapped with foreign key of BRANCH table to the branch reference using @JoinColumn. To make the association bidirectional, the other side of entity Branch also should have Student(s) entity reference and we will be used mappedBy attribute to map to student(s) reference in Branch entity.

mappedBy attribute indicates that which entity owns the relationship (in this example, Student) and what reference is used for non-owning entity within owner entity (in this example, branch is the reference name used in Student entity to map Branch entity).

Model 1 and Model 2 (MVC) Architecture

Before developing any web application, we need to have idea about design models.
There are two types of programming models (design models)

Model 1 Architecture
Model 2 (MVC) Architecture

Model 1 Architecture
Servlet and JSP are the main technologies to develop the web applications.

Servlet is considered faster alternative to CGI. Servlet technology doesn't create process, rather it uses threads from a thread pool to handle request. The advantage of creating thread over process is that it doesn't allocate separate memory area , leading to better performance. Thus many subsequent requests can be easily handled by servlet.

Problem in Servlet technology Servlet needs to recompile if any designing code is modified. It doesn't provide separation of concern. Presentation and Business logic are mixed up.

JSP overcomes almost all the problems of Servlet. It provides better separation of concern, now presentation and business logic can be easily separated. You don't need to redeploy the application if JSP page is modified. JSP provides support to develop web application using JavaBean, custom tags and JSTL so that we can put the business logic separate from our JSP that will be easier to test and debug.

Advantage of Model 1 Architecture
Easy and Quick to develop web application

Disadvantage of Model 1 Architecture

Navigation control is decentralized since every page contains the logic to determine the next page.

If JSPs names are modified, we need to change it in all the pages leadining to the maintenance problem.

Time consuming.
Hard to extend It is better for small applications but not for large applications.

Enter Model 2 (MVC) Architecture
Model 2 is based on the MVC (Model View Controller) design pattern. The MVC design pattern consists of three modules model, view and controller.

Model The model represents the state (data) and business logic of the application. (JavaBean , POJOs)

View The view module is responsible to display data i.e. it represents the presentation.(JSP)

Controller : The Front controller module(Servlet/Filter) acts as an interface between view and model. It intercepts all the requests i.e. receives input and sends commands to Model / View to change accordingly.

Advantage of Model 2 (MVC) Architecture

Navigation control is centralized .
Now only controller contains the logic to determine the next page.
Easy to maintain
Easy to extend
Easy to test
Better separation of concerns

Disadvantage of Model 2 (MVC) Architecture
Developing centralized dispatcher (a navigation controller) is tedius, especially when web app size grows. That's the reason , Spring MVC is so popular , which implements Servlet based MVC pattern n supplies readymade components : Front Controller , Handler mapping , View Resolver ...

Problems with traditional/legacy spring framework

We use different modules from spring such as core module, to do dependency injection.The MVC module to develop the web layer for our application or even the restful web services layer.And then the DAO layer where we use the spring JDBC/ORM which makes our life easy to develop a data access layer for our application. When we are using ORM tools like Hibernate, we can use spring data JPA and we use these modules and more that are valuable from Spring.

Initially we used XML based configuration or annotations based configuration,This configuration can get difficult and hard to maintain over time.And also we need to make sure that each of these modules is available for our application by defining all the dependencies in the Maven pom xml.And at runtime we have to be sure that these versions of various Modules that we use are compatible with each other.But it's our responsibility to do all that, and once we have all that in place we will build our application and will have to deploy to an external web container to test it .

Spring boot will automate all this for us.

What are Spring Boot Features ?

1. The first of those super cool features is auto configuration - Spring Boot automatically configures everything that is required for our application. We don't have to use XML or Java configuration anymore.

For example if you are using Spring MVC or the web to develop a web application or a restful web service application spring boot will automatically configure the dispatcher servlet and does all the request mapping for us. We don't have to use any xml or annotation based configuration to configure this servlet.

Similarly if you are using spring data or object relational mapping while working with tools like Hibernate to perform database crud we no longer have to configure the data source or even the transaction manager. Spring boot will automatically configure these for our application.

2. The next super cool feature is spring boot starters .With the spring boot starters spring boot takes the problem off module availability we need. Before Spring Boot we had to make sure that a particular library required for our project is available and also the versions of different libraries are compatible.But we don't have to do that anymore thanks to spring boot starters every spring boot project will have a parent project.This project has all the version information of various libraries that we will be using in our project so we need not worry about version compatibility. The spring developers have already done it for us and put all that information into this spring boot starter parent .

Secondly we have starters for different types of projects if we are developing a web project then we simply need to include the starter web . We don't have to include any other libraries or dependencies. Spring boot will automatically pull all the other libraries that are required because this project here or this or dependency transitively depends on other libraries that are required. Maven will automatically pull those libraries. The spring boot developers have given us starter dependencies which when we use in our projects will not have the Modular availability problem and the version compatibility problem.

Another example is the spring boot starter data jpa .When you want to work with Hibernate you simply include the single dependency in your maven pom xml and all the other libraries will be pulled accordingly. And also the correct versions of those libraries will be included because all that version information is available in this spring boot starter parent which is a parent for every spring boot project.

3. The third super cool feature we don't have to worry about deploying our applications to external container spring boot comes with an embedded servlet container and these containers.

By default it is Tomcat but you can also use Jetty and undertow or any other external server. So no longer external deployment you can simply right click on your project and run it and your application will be launched on its embedded Tomcat server by default.

4. Last and very important spring boot gives us a lot of health checks of our application for free through the spring boot actuators.We can use different types of health checks that come for free and we can use these even on production when our application is running. These can be activated easily and will display all the auto configuration reports and everything that is automatically configured for our application.

What is Spring Boot
=Spring Framework + Embedded web server (Tomcat) -(complex config setting / complex dependency management in pom.xml)

Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

      public static void main(String[] args) {
                 SpringApplication.run(HelloSpringBootApplication.class,
args);
      }

}
```
About : org.springframework.boot.SpringApplication

It's Class used to bootstrap and launch a Spring application from a Java main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC)
2. Manages life cycle of spring beans
3. Launches the embedded Tomcat Container

@SpringBootApplication - This is where all the spring boot magic happens. It consists of following 3 annotations.

1. @SpringBootConfiguration
It tells spring boot that this class here can have several bean definitions(@Bean annotation configured methods , equivalent to <bean> tag in xml based configuration)
We can define various spring beans here and those beans will be available at run time .

2. @EnableAutoConfiguration
It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source,using the properties in application.properties file.

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

3. @ComponentScan (equivalent to xml tag : context:component-scan)

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans .
Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :
@ComponentScan(basePackages = "com")
For scanning entities : (equivalent to packagesToScan)
@EntityScan(basePackages = "com.app.pojos")

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication

public class HellospringbootApplication { p.s.v.m(...) {...}}
```

@SpringBootApplication - This is where all the spring boot magic happens.

The SpringBootApplication is a key annotation which is a top level annotation which contains several other annotations on it.

@SpringBootConfiguration

@EnableAutoConfiguration

@ComponentScan

The first one @SpringBootConfiguration tells spring boot or the container that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time so you define a method here .

The second annotation @EnableAutoConfiguration is a very important annotation at enable Auto configuration.This annotation tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml as automatically Spring Boot will create a data source. We will also provide other information like username password etc. but spring boot will scan through all these dependencies and it will automatically configure the data source required for us.

Another example is spring web, If we have spring web in your dependencies.

Then spring boot will automatically create the dispatcher servlet on all that configuration file you for free.

All the xml, all the java based configuration is now gone. We as developers need not do all that configuration it all comes for free thanks to spring boots to enable auto configuration annotation.

@ComponentScan
So this tells us that spring boot or spring should scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans . Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :
@ComponentScan(basePackages = "com")
For scanning entities :
@EntityScan(basePackages = "com.app.pojos")

Enter Spring boot

1. What is Spring Boot?
Spring Boot is a Framework from "The Spring Team" to ease the bootstrapping and development of new Spring Applications.

It provides defaults for code and annotation configuration to quick-start new Spring projects within no time.

It follows "Opinionated Defaults Configuration" an approach to avoid lot of boilerplate code and configuration to improve Development, Unit Test and Integration Test Process.

2. What is NOT Spring Boot?
Spring Boot Framework is not implemented from the scratch by The Spring Team
It's implemented on top of existing Spring Framework (Spring IO Platform).
It is not used for solving any new problems. It is used to solve same problems like Spring Framework.
(i.e to help in writing enterprise applications)

3. Advantages of Spring Boot:

It is very easy to develop Spring Based applications with Java
It reduces lots of development time and increases productivity.
It avoids writing lots of boilerplate Code, Annotations and XML Configuration.
It is very easy to integrate Spring Boot Application with its Spring Ecosystem like Spring JDBC, Spring ORM, Spring Data, Spring Security etc.

It follows "Opinionated Defaults Configuration" Approach to reduce Developer effort

It provides Embedded HTTP servers like Tomcat, Jetty etc. to develop and test our web applications very easily.

It provides CLI (Command Line Interface) tool to develop and test Spring Boot(Java or Groovy)
Applications from command prompt very easily and quickly.

It provides lots of plugins to develop and test Spring Boot Applications very easily using Build Tools like Maven and Gradle

It provides lots of plugins to work with embedded and in-memory Databases very easily.

In short
Spring Boot = Spring Framework + Embedded HTTP Server(eg Tomcat) - XML Based configuration - efforts in identifying dependencies in pom.xml

4. What is that "Opinionated Defaults Configuration" ?

When we use Hibernate/JPA, we would need to configure a datasource, a session factory, a transaction manager among lot of other things.
Refer to our hibernate-persistence.xml , to recall what we did earlier .

Spring Boot says can we look at it differently ?
Can we auto-configure a Data Source(connection pool) / session factory / Tx manager if Hibernate jar is on the classpath?

It says :
When a spring mvc jar is added into an application, can we auto configure some beans automatically?
(eg HandlerMapping , ViewResolver n configure DispatcherServlet)

By the way :
There would be of course provisions to override the default auto configuration.

5. How does it work ?
Spring Boot looks at
1. Frameworks available on the CLASSPATH
2. Existing configuration for the application.
Based on these, Spring Boot provides basic configuration needed to configure the application with these frameworks. This is called Auto Configuration.

6. What is Spring Boot Starter ?
Starters are a set of convenient dependency descriptors that you can include in your application's pom.xml
.
eg : Suppose you want to develop a web application.

W/o Spring boot , we would need to identify the frameworks we want to use, which versions of frameworks to use and how to connect them together.

BUT all web application have similar needs.
These include Spring MVC, Jackson Databind (for data binding), Hibernate-Validator (for server side validation using Java Validation API) and Log4j (for logging). Earlier while creating any web app, we had to choose the compatible versions of all these frameworks.

With Spring boot : You just add Spring Boot Starter Web.

Dependency for Spring Boot Starter Web
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
```

Just by adding above starter , it will add lot of JARs under maven dependencies

Another eg : If you want to use Spring and JPA for database access, just include the spring-boot-starter-data-jpa dependency in your project, and you are good to go.

7. Another cool feature of Spring boot is : we don't have to worry about deploying our applications to external container. It comes with an embedded servlet container.

8.Important components of a Spring Boot Application

Below is the starting point of a Spring Boot Application

```
@SpringBootApplication
public class HelloSpringBootApplication {

      public static void main(String[] args) {
                 SpringApplication.run(HelloSpringBootApplication.class,
args);
      }

}
```

About : org.springframework.boot.SpringApplication
It's Class used to bootstrap and launch a Spring application from a Java main method.

By default class will perform the following steps to bootstrap the application
1. Create an ApplicationContext instance (representing SC)
2. Manages life cycle of spring beans

@SpringBootApplication - This is where all the spring boot magic happens. It consists of following 3 annotations.

1. @SpringBootConfiguration
It tells spring boot that this class here can have several bean definitions. We can define various spring beans here and those beans will be available at run time .

2. @EnableAutoConfiguration
It tells spring boot to automatically configure the spring application based on the dependencies that it sees on the classpath.

eg:
If we have a MySql dependency in our pom.xml , Spring Boot will automatically create a data source,using the properties in application.properties file.

If we have spring web in pom.xml , then spring boot will automatically create the dispatcher servlet n other beans (HandlerMapping , ViewResolver)

All the xml, all the java based configuration is now gone.It all comes for free thanks to spring boot to enable auto configuration annotation.

3. @ComponentScan (equivalent to xml tag : context:component-scan)

So this tells us that spring boot to scan through the classes and see which all classes are marked with the stereotype annotations like @Component Or @Service @Repository and manage these spring beans .
Default base-pkg is the pkg in which main class is defined.
Can be overridden by
eg :
@ComponentScan(basePackages = "com")
For scanning entities : (equivalent to packagesToScan)
@EntityScan(basePackages = "com.app.pojos")

Steps

1. File --New --Spring starter project -- add project name , group id ,artifact id ,pkg names , keep packaging as JAR for Spring MVC web application.

2. Add dependencies
web -- web
sql -- Spring Data JPA, MYSQL
Core -- DevTools
Lombok
validation

3. Copy the entries from supplied application.properties & edit DB configuration.

4. For Spring MVC (with JSP view layer demo) using spring boot project

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

```
      <!-- Additional dependencies for Spring MVC -->
                  <dependency>
                              <groupId>org.apache.tomcat.embed</groupId>
                              <artifactId>tomcat-embed-jasper</artifactId>
                  </dependency>

                  <dependency>
                              <groupId>javax.servlet</groupId>
                              <artifactId>jstl</artifactId>
                  </dependency>
```

5. Create under src/main/webapp : WEB-INF folder

6. Create `r n test it.

7. Port earlier spring MVC app , observe the problems.

& fix it.

Problem observed : ??????

Reason : Could not find org.hibernate.SessionFactory (since Spring boot
DOES NOT support any native hibenrate implementationj directly)

Solution : Replace hibernate's native API (org.hibernate) by JPA
(refer : day17-data\day17_help\diagrams\jpa-entitymgr-session-layers.png)

In DAO layer : replace native hibernate API by JPA
i. e instead of auto wiring SF in DAO layer : inject JPA' EntityManager
    directly in DAO.

How ?

```
@PersistenceContext
//OR can continue to use @AutoWired : spring supplied annotation
private EntityManager mgr;
//uses def persistence unit , created auto by spring boot using db setting
added //in application.properties file , 1 per app / DB
```

Use directly EntityManager API (refer :)
OR
Unwrap hibernate Session from EntityManager

```
Session session = mgr.unwrap(Session.class);
```

Which one is preferred ? 1st soln.

8. Test Entire application.

1. @SpringBootApplication : is added on the main class , in spring boot application , to run it as a standalone JAR / WAR

@SpringBootApplication =

1.1. @Configuration – Designates this main class as a configuration class for Java configuration. In addition to beans configured via component scanning, an application may want to configure some additional beans via the @Bean annotation as demonstrated here. Thus, the return value of methods having the @Bean annotation in this class are registered as beans.

1.2. @EnableAutoConfiguration – This enables Spring Boot’s autoconfiguration mechanism. Auto-configuration refers to creating beans automatically by scanning the classpath.

1.3. @ComponentScan – Typically, in a Spring application, annotations like @Component, @Configuration, @Service, @Repository are specified on classes to mark them as Spring beans. The @ComponentScan annotation basically tells Spring Boot to scan the current package and its sub-packages in order to identify annotated classes and configure them as Spring beans. Thus, it treats the current package as the root package for component scanning.

Annotation added automatically :
@EnableWebMvc: Marks the application as a web application and activates setting up a DispatcherServlet , HandlerMapping , ViewResolvers . Spring Boot adds it automatically when it sees spring-webmvc on the classpath.

2. How to use the @SpringBootApplication annotation
In order to run a Spring Boot application, it needs to have a class with the @SpringBootApplication annotation.

eg :
```
package com.app;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication
public class Main {
    public static void main(String[] args) {
          SpringApplication.run(Main.class, args);
      }
}
```
The Main class has the @SpringBootApplication annotation

It simply invokes the SpringApplication.run method.
This starts the Spring application as a standalone application, runs the embedded servers and loads the beans.

Normally, such a main class is placed in a root package above other packages. This enables component scanning to scan all the sub-packages for beans.

1. What is a model attribute?
It's the attribute(server side entry=k n value pair) : String , Object
Purpose : to store the results of B.L
Who creates ---Request handling Controller(Prog)
Who sends it to whom : Controller ---> D.S
After D.S gets actual view name from V.R :
D.S chks : are there any model attrs :
Yes : D.S saves model attrs under Request scope & then forwards the clnt to view layer .
NO : D.S forwards the clnt to view layer .
How to access these model attrs from JSP ?
${requestScope.attrName}

What are different ways for Controller to add model attrs ?
1. Via is o.s.w.s.ModelAndView?

o.s.w.s.ModelAndView : class => holder for model attrs + logical view name

ModelAndView(String viewName,String modelAttrName,Object modelAttrVal)
eg : what can be valid ret type of req handling method
String OR ModelAndView

2. Is there any Simpler way to add Model Attributes ?

Use o.s.ui.Model : i/f ---holder (Map) of model attributes
How do u add model attributes ?
public Model addAttribute(String modelAttrName,Object modelAttrVal)
eg : How to add 2 model attrs? : method chaining

Who will supply empty Model map ? : SC
IoC : DI
How to tell SC that request handling method needs a model map ? :Simply add it as the arg of req handling method
When controller rets string : logical view name (controller impl. rets all model map)

3. How to hide index.jsp under WEB-INF (i.e how to ensure that index page is served to client via D.S)
Add HomePageController to serve index.jsp

4. Handling request parameters in Controller ?
eg : @RequestParam("price") double productPrice
SC : double productPrice
=Double.parseDouble(request.getParameter("price"))
Reco : Match req param names with method arg names.
URL :
http://localhost:8080/day13.1/test/product?nm=pen&qty=10&price=40.5&manuDate=2020-1-1
SC : def date time format : mon/day/year

Problem : HTTP 400 : Bad request => some request data coming from client is invalid
Def dat format : mon/day/yr
How to tell SC : about exact Date time format : annotation.

5. What's the meaning of HTTP status 400 : Bad Client Error
Typically it represents some request parameter conversion error.

Default date format used by SC : MM/dd/yyyy
In order to change it use : @DateTimeFormat annotation.

```
eg : @RequestParam("exp_date") @DateTimeFormat(pattern="yyyy-MM-dd")  Date
expDate
SC invokes : SimpleDateFormat sdf=new SDF("yyyy-MM-dd");
Date expDate=sdf.parse(request.getParameter("exp_date"));
```

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

How to replace default forward view by redirect view in spring MVC ?
Ans -- use redirect keyword.
eg : return "redirect:/customer/topics";
Internals
D.S skips the V.R & sends temp redirect response to the clnt browser.
How ?
D.S invokes ---
response.sendRedirec(response.encodeRedirectURL("/customer/topics");
So clnt browser will send a next request ---with method=GET
URL --
http://host:port/spring_mvc/customer/topics

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

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

Method
public RedirectAttributes addFlashAttribute(String attrName,Object value)

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

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

9. How to take care of links(href)/form actions + add URL rewriting support ?
1. Import spring supplied JSP tag lib.
(via taglib directive)
prefix ="spring"

2. Use the tag.

```
<a href="<spring:url value='/user/logout'/>">Log Out</a>
```

/ --- root of curnt web app.

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

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

OR form action example

```
<spring:url var="url" value='/admin/list'/>
eg : <form action="${var}">
form inputs
</form>
```

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

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

name --- refresh
value --- 10;url=any url you want to redirect to (eg : home page url (root of web app))
10 : delay in seconds

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

```
1.  If there multiple request params(use case -- register/update) --- bind
    POJO directly to a form.(2 way form binding technique)
How ?
1.1. For  loading the form (in showForm method of the controller)  , bind
     empty POJO (using def constr) in model map
How  ?
Explicit --add Model as dependency & u add
map.addAttribute(nm,val)
OR better way
implicit -- add POJO as a dependency
eg : User registration
@GetMapping("/reg")
public String showForm(User u) {...}

What will SC do ?
SC --- User u=new User();
chks --- Are there any req params coming from client ? --- typically --no
--- only getters will be called --
adds pojo as model attr (in Model map)
map.addAttribute("user",new User());

1.2. In form (view ---jsp)  -use spring form tags along with
     modelAttribute
Steps
1.  import spring supplied form tag lib
2.  Specify the name of modelAttribute under which form data will be
    exposed.(name of model attr mentioned in the controller)
<s:form method="post" modelAttribute="user">
  <s:input path="email"/>.....
</s:form>


1.3 Upon form submission (clnt pull I)
clnt sends a new req --- containing req params
@PostMapping("/reg")
public String processForm(User u,RedirectAttributes flashMap,HttpSession
hs) {
//SC calls
User u=new User();
SC invokes MATCHING (req param names --POJO prop setters)
setters. -- conversational state is transferred to Controller.
adds pojo as model attr (in Model map)
map.addAttribute("user",u)
Thus you get a populated POJO directly in controller w/o calling
<jsp:setProperty> & w/o using any java bean.


Spring form tags
1.  <form:input path="name" />

2.  <form:input type="email" path="email" />

3.  <form:password path="password" />
```

```
4.  <form:textarea path="notes" rows="3" cols="20"/>

5.  <form:checkboxes items="${languages}" path="favouriteLanguage" />

6.  Male: <form:radiobutton path="sex" value="M"/>
Female: <form:radiobutton path="sex" value="F"/>

7.  <form:radiobuttons items="${jobItem}" path="job" />

8.
<form:select path="book">
    <form:option value="-" label="--Please Select--"/>
    <form:options items="${books}" />
</form:select>

9.  <form:hidden path="id" value="12345" />

10.  <form:errors path="name" cssClass="error" />
```

What Is Spring?

In simple words , Spring framework provides comprehensive infrastructure support for developing Java applications.

It's comes with some nice features like Dependency Injection, and ready made modules like:

Spring JDBC
Spring MVC
Spring Security
Spring AOP
Spring ORM
Spring Test ....

These modules can drastically reduce the development time of an application.

eg : in the early days of Java web development, we needed to write a lot of boilerplate code to set up MVC based web app. With the help of Spring MVC module , it gets easier.

Now ...What Is Spring Boot?

Spring Boot is basically an extension of the Spring framework, which eliminates the boilerplate configurations required for setting up a Spring application.

In short
Spring Boot =Existing Spring Framework + Embedded HTTP Server(eg Tomcat) - XML Based configuration - efforts in identifying dependencies in pom.xml

It takes an "opinionated" view of the Spring platform, which helps programmers for a faster and more efficient development ecosystem.

Important features in Spring Boot:

Opinionated ‘starter' dependencies to simplify the build and application configuration
Embedded server to avoid complexity in application deployment
Health check, and externalized configuration
Automatic config for Spring functionality – whenever possible

Comparison points

1. Maven Dependencies
1.1. Look at the minimum dependencies required to create a web application using Spring:

```
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-web</artifactId>
    <version>5.3.5</version>
```

```
</dependency>
<dependency>
    <groupId>org.springframework</groupId>
    <artifactId>spring-webmvc</artifactId>
    <version>5.3.5</version>
</dependency>
+ jackson dependencies + ....
```

Unlike Spring, Spring Boot requires only one dependency to get a web application up and running:

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
    <version>2.7.2</version>
</dependency>
```

All other dependencies are added automatically to the final project during build time.

1.2.
In a Spring project, we have to explicitly add all of testing related dependencies

In Spring Boot , spring boot starter automatically pulls in testing related libraries.

Spring Boot provides a number of starter dependencies for different Spring modules.

Some of the most commonly used ones are:

spring-boot-starter-data-jpa
spring-boot-starter-security
spring-boot-starter-test
spring-boot-starter-web
spring-boot-starter-thymeleaf

2. MVC Configuration
Now , Compare the configuration required to create a JSP web application using both Spring and Spring Boot.

Spring requires defining the dispatcher servlet, mappings, and other supporting configurations. We can do this using either the web.xml file or java config classes.
(recall web.xml , spring-servlet.xml : configuration files)

By comparison, Spring Boot only needs a couple of properties to make things work once we add the web starter:

```
spring.mvc.view.prefix=/WEB-INF/views
spring.mvc.view.suffix=.jsp
```

All of the Spring configuration above(D.S , HandlerMapping , View Resolver) is automatically included by adding the Boot web starter through a process called auto-configuration.

What this means is that Spring Boot will look at the dependencies, properties, and beans that exist in the application and enable configuration based on these.

On the other hand , if we want to add our own custom configuration, then the Spring Boot auto-configuration can be overridden(BUT with additional config steps)

3. Configuring Template Engine
To configure a Thymeleaf template engine in both Spring and Spring Boot.

In Spring, we need to add the thymeleaf-spring5 dependency and some configurations for the view resolver:

Spring Boot only requires the dependency of spring-boot-starter-thymeleaf to enable Thymeleaf support in a web application.

Once the dependency is added , we can add the template(.html) to the src/main/resources/templates folder and the Spring Boot will display them automatically.

4. Spring Security Configuration

Spring requires both the standard spring-security-web and spring-security-config dependencies to set up Security in an application.

In Spring Boot , we only need to define the dependency of spring-boot-starter-security as this will automatically add all the necessary dependencies to the classpath.

The security configuration in Spring Boot is the same as in Spring framework(no additional support!)

5. The JPA configuration can be achieved in both Spring and Spring Boot.

But without spring boot , we have to add all datasource related dependencies in pom.xml n have to configure DataSource(connection pool), Session Factory (for hibernate) , Transaction manager using either xml or java config classes.
eg : hibernate-persistence.xml

Where as , To enable JPA in a Spring Boot application , you just need to add single dependency : spring-boot-starter-data-jpa

The spring-boot-starter contains the necessary auto-configuration for Spring JPA.
The spring-boot-starter-jpa project references all the necessary dependencies such as hibernate-core.

5.1. Configuration

Spring Boot configures Hibernate as the default JPA provider, so it's not necessary to define the entityManagerFactory bean unless we want to customize it.

Spring Boot can also auto-configure the dataSource bean, depending on the database we're using. In the case of an in-memory database eg : H2 , Boot automatically configures the DataSource if the corresponding database dependency is present on the classpath.

If we want to use JPA with MySQL database, we need the mysql-connector-java dependency.

6. Application Bootstrap

The basic difference in bootstrapping an application in Spring and Spring Boot

Spring uses typically the legacy web.xml as its bootstrap entry point(i.e configure DispatcherServlet there)

Details :
 WC   reads web.xml.
The DispatcherServlet defined in the web.xml is instantiated by the WC
DispatcherServlet creates WebApplicationContext by reading WEB-INF/{servletName}-servlet.xml.
DispatcherServlet with help of SC registers the beans defined in the application context.

How Spring Boot Bootstraps?
The entry point of a Spring Boot application is the class which is annotated with @SpringBootApplication:

```
@SpringBootApplication
public class Application {
    public static void main(String[] args) {
        SpringApplication.run(Application.class, args);
    }
}
```

By default, Spring Boot uses an embedded container to run the application. In this case, Spring Boot uses the public static void main entry point to launch an embedded web server.

It also takes care of the binding of the Servlet, spring beans from the application context to the embedded servlet container.

It automatically scans all the classes in the same package or sub packages of the Main-class for components.

Additionally, Spring Boot provides the option of deploying it as a web archive in an external container.

7. Packaging and Deployment

Both of these frameworks support common package managing technologies like Maven and Gradle; BUT for deployment, these frameworks differ a lot.

The Spring Boot Maven Plugin provides Spring Boot support in Maven. It also allows packaging executable jar or war archives and running an application “in-place.”

Advantages of Spring Boot over Spring as far as deployment is concerned :

Provides embedded container support
Provision to run the jars independently using the command java -jar
Option to exclude dependencies to avoid potential jar conflicts when deploying in an external container
Option to specify active profiles when deploying
Random port generation for integration tests

If the JPA specification and its implementations provide most of the features that you use with Spring Data JPA, do you really need the additional layer? Can't you just use the Hibernate directly ?

You can, of course, do that. That's what a lot of Java applications do. Spring ORM provides a good integration for JPA (eg : Spring native Hibernate Integration or Spring JPA)

But the Spring Data team took the extra step to make your job a little bit easier. The additional layer on top of JPA enables them to integrate JPA into the Spring stack easily.
 They also provide a lot of functionality that you otherwise would need to implement yourself.

Why Spring Data JPA

1. No-code Repositories
The repository pattern is one of the most popular persistence-related patterns. It hides the DB specific implementation details and enables you to implement your business logic with higher abstraction level.

eg : For Author Entity
How ?
to persist, update and remove one or multiple Author entities,
to find one or more Authors by their primary keys,
to count, get and remove all Authors and
to check if an Author with a given primary key exists.

All you need to do is :

```
public interface AuthorRepository extends JpaRepository<Author, Integer>
{}
```

2. Reduced boilerplate code
To make it even easier, Spring Data JPA provides a default implementation for each method defined by one of its repository interfaces. You don't need to implement these operations.

3. Generated queries
Another cool feature of Spring Data JPA is the generation of database queries based on method names.(finder method pattern)

eg : Write a method that gets a Book entity with a given title.
Internally, Spring generates a JPQL query based on the method name, sets the provided method parameters as bind parameter values, executes the query and returns the result.

```
//JpaRepository<T,ID>
//T : entity type
//ID : Data type of id property(PK)
public interface BookRepository extends JpaRepository<Book, Integer> {

    Optional<Book> findByTitle(String title123);
}
```

Assumption : title : property of Book POJO

Using Spring Data JPA with Spring Boot

You only need to add the spring-boot-starter-data-jpa and your JDBC driver to your maven build. The Spring Boot Starter includes all required dependencies and activates the default configuration.
Add DB config properties in application.properties file

By default, Spring Boot expects that all repositories are located in a sub-packages of the class annotated with @SpringBootApplication.
If your application doesn’t follow this default, you need to configure the packages of your repositories using an @EnableJpaRepositories annotation.

Repositories(API) in Spring Data JPA
package : o.s.data.repository
CrudRepository
PagingAndSortingRepository
JpaRepository
The CrudRepository interface defines a repository that offers standard create, read, update and delete operations.
The PagingAndSortingRepository extends the CrudRepository and adds findAll methods that enable you to sort the result and to retrieve it in a paginated way.
The JpaRepository adds JPA-specific methods, like flush() to trigger a flush on the persistence context or findAll(Example<S> example) to find entities

Defining an entity-specific repository
eg :
Book entity is a normal JPA entity with a generated primary key of type Long, a title and a many-to-many association to the Author entity.

```
@Entity
@Table(name="books")
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    @Version
    private int version; //for optimistic locking

    private String title;

    @ManyToMany
    @JoinTable(name = "book_author",
                joinColumns = { @JoinColumn(name = "fk_book") },
                inverseJoinColumns = { @JoinColumn(name = "fk_author") })
    private Set<Author> authors = new HashSet<>();

    ...
}
```

If you want to define a JPA repository for this entity, you need to extend Spring Data JpaRepository interface and type it to Book and Long.

```
public interface BookRepository extends JpaRepository<Book, Long> {

    Book findByTitle(String title);
}
```

Working with Repositories

After you defined your repository interface, you can use the @Autowired annotation to inject it into your service implementation. Spring Data will then provide you with a proxy implementation of your repository interface. This proxy provides default implementations for all methods defined in the interface.

For More API Details
refer " regarding Spring Data JPA"

In entire web application ,the DAO layer usually consists of a lot of boilerplate code that can be simplified.

The DB journey from JDBC ---> Hibernate/EclipseLink/Kodo ---> JPA ---> finally Spring Data JPA

JDBC is the basic most API for communicating to DB , at the closest to DB level.

Hibernate/EclipseLink/Kodo ,as ORM tools(=auto persistence providers) solve the problem of impedence mismatch (i.e acts as the intermediary between Object world n DB world n translate OOP concepts in DB . eg inheritance n associations.

JPA is a part of Java EE/Jakarta EE specification that defines an API for ORM and for managing persistent objects. Hibernate and EclipseLink are popular implementations of this specification.

Spring Data JPA adds a layer on top of JPA. That means it uses all features defined by the JPA specification, especially the entity and association mappings, the entity lifecycle management, and JPA’s query capabilities. On top of that, Spring Data JPA adds its own features like a no-code implementation of the repository pattern and the creation of database queries from method names and custom queries.

Benefits of simplification
1. Decrease in the number of layers that we need to define and maintain
2. Consistency of data access patterns and consistency of configuration.

Spring Data JPA framework takes this simplification one step forward and makes it possible to remove the DAO implementations entirely. The interface of the DAO is now the only artifact that we need to explicitly define.

For this , a DAO interface needs to extend the JPA specific Repository interface – JpaRepository or its super i/f CrudRepository. This will enable Spring Data to find this interface and automatically create an implementation for it.

By extending the interface we get the most required CRUD methods for standard data access available in a standard DAO.

eg : CRUDRepository methods
long count()
Returns the number of entities available.
void delete(T entity)
Deletes a given entity.
void deleteAll()
Deletes all entities managed by the repository.
void deleteAll(Iterable<? extends T> entities)
Deletes the given entities.
void deleteById(ID id)
Deletes the entity with the given id.

```
boolean    existsById(ID id)
Returns whether an entity with the given id exists.
Iterable<T>      findAll()
Returns all instances of the type.
Iterable<T>      findAllById(Iterable<ID> ids)
Returns all instances of the type with the given IDs.
Optional<T>      findById(ID id)
Retrieves an entity by its id.
<S extends T>
S     save(S entity)
Saves a given entity.
<S extends T>
Iterable<S>      saveAll(Iterable<S> entities)
Saves all given entities.

Method of JpaRepository
void  deleteAllInBatch()
Deletes all entities in a batch call.
void  deleteInBatch(Iterable<T> entities)
Deletes the given entities in a batch which means it will create a single
Query.
List<T>    findAll()
<S extends T>
List<S>    findAll(Example<S> example)
<S extends T>
List<S>    findAll(Example<S> example, Sort sort)
List<T>    findAll(Sort sort)
List<T>    findAllById(Iterable<ID> ids)
void  flush()
Flushes all pending changes to the database.
T     getOne(ID id)
Returns a reference to the entity with the given identifier.
<S extends T>
List<S>    saveAll(Iterable<S> entities)
```

3. Custom Access Method and Queries

By extending one of the Repository interfaces, the DAO will already have some basic CRUD methods (and queries) defined and implemented.

To define more specific access methods, Spring JPA supports quite a few options:

1. simply define a new method in the interface

provide the actual JPQL query by using the @Query annotation

When Spring Data creates a new Repository implementation, it analyses all the methods defined by the interfaces and tries to automatically generate queries from the method names. While this has some limitations, it’s a very powerful and elegant way of defining new custom access methods with very little effort.

eg :

```
   Customer findByName(String name);
   List<Person> findByAddressAndLastname(String address, String lastname);
```

```
  // Enables the distinct flag for the query
  List<Person> findDistinctPeopleByLastnameOrFirstname(String lastname,
String firstname);


  // Enabling ignoring case for an individual property
  List<Person> findByLastnameIgnoreCase(String lastname);


  // Enabling static ORDER BY for a query
  List<Person> findByLastnameOrderByFirstnameAsc(String lastname);

List<Person> findByAddressZipCode(String zipCode);
```

Assuming a Person p has an Address with a String zipCode. In that case, the method creates the property traversal p.address.zipCode.

Limiting the result size of a query with Top and First

```
User findFirstByOrderByLastnameAsc();

User findTopByOrderByAgeDesc();

Page<User> queryFirst10ByLastname(String lastname, Pageable pageable);

Slice<User> findTop3ByLastname(String lastname, Pageable pageable);

List<User> findFirst10ByLastname(String lastname, Sort sort);

List<User> findTop10ByLastname(String lastname, Pageable pageable);
```

2. Or in case of custom queries , can add directly in DAO i/f.
eg :

```
@Query("select u from User u where u.emailAddress = :em")
  User findByEmailAddress(@Param("em")String emailAddress);

@Query("SELECT p FROM Person p WHERE LOWER(p.name) = LOWER(:nm)")
Foo retrieveByName(@Param("nm") String name);
```

3. Transaction Configuration
The actual implementation of the Spring Data managed DAO is hidden since we don’t work with it directly. It's implemented by – the SimpleJpaRepository – which defines default transaction mechanism using annotations.

These can be easily overridden manually per method.

4. Exception Translation is still supplied
Exception translation is still enabled by the use of the @Repository annotation internally applied on the DAO implementation class.

The six architectural constraints of REST APIs

1. Client-server architecture
An API’s job is to connect two pieces of software without limiting their own functionalities. This objective is one of the core restrictions of REST: the client (that makes requests) and the server (that gives responses) stay separate and independent.

When done properly, the client and server can update and evolve in different directions without having an impact on the quality of their data exchange. This is especially important in various cases where there are plenty of different clients a server has to cater to. Think about weather APIs — they have to send data to tons of different clients (all types of mobile devices are good examples) from a single database.

2. Statelessness
For an API to be stateless, it has to handle calls independently of each other. Each API call has to contain the data and commands necessary to complete the desired action.

An example of a non-stateless API would be if, during a session, only the first call has to contain the API key, which is then stored server-side. The following API calls depend on that first one since it provides the client’s credentials.

In the same case, a stateless API will ensure that each call contains the API key and the server expects to see proof of access each time.

Stateless APIs have the advantage that one bad or failed call doesn’t derail the ones that follow.

3. Uniform Interface
While the client and the server change in different ways, it’s important that the API can still facilitate communication. To that end, REST APIs impose a uniform interface that can easily accommodate all connected software.

In most cases, that interface is based on the HTTP protocols. Besides the fact that it sets rules as to how the clients and the server may interact, it also has the advantage of being widely known and used on the Internet. Data is stored and exchanged through JSON files because of their versatility.

4. Layered system
To keep the API easy to understand and scale, RESTful architecture dictates that the design is structured into layers that operate together.

With a clear hierarchy for these layers, executing a command means that each layer does its function and then sends the data to the next one. Connected layers communicate with each other, but not with every component of the program. This way, the overall security of the API is also improved.

If the scope of the API changes, layers can be added, modified, or taken out without compromising other components of the interface.

5. Cacheability
It’s not uncommon for a stateless API’s requests to have large overhead. In some cases, that’s unavoidable, but for repeated requests that need the same data, caching said information can make a huge difference.

The concept is simple: the client has the option to locally store certain pieces of data for a predetermined period of time. When they make a request for that data, instead of the server sending it again, they use the stored version.

The result is simple: instead of the client sending several difficult or costly requests in a short span of time, they only have to do it once.

6. Code on Demand
Unlike the other constraints we talked about up to this point, the last one is optional. The reason for making “code on demand” optional is simple: it can be a large security risk.

The concept is to allow code or applets(now obsolete!) to be sent through the API and used for the application. As you can imagine, unknown code from a shady source could do some damage, so this constraint is best left for internal APIs where you have less to fear from hackers and people with bad intentions. Another drawback is that the code has to be in the appropriate programming language for the application, which isn’t always the case.

The upside is that “code on demand” can help the client implement their own features on the go, with less work being necessary on the API or server. In essence, it permits the whole system to be much more scalable and agile.

What is REST ?

REST stands for REpresentational State Transfer.

REST is web standards based architecture and uses HTTP Protocol for data communication.

It revolves around resource where every component is a resource and a resource is accessed by a common interface using HTTP standard methods.

REST was first introduced by Roy Fielding in 2000.

In REST architecture, a REST Server simply provides access to resources and REST client accesses and presents the resources.

Here each resource is identified by URIs

REST uses various representations to represent a resource like text, JSON and XML. Most popular light weight data exchange format used in web services = JSON

HTTP Methods
Following well known HTTP methods are commonly used in REST based architecture.

GET - Provides a read only access to a resource.

POST - Used to create a new resource.

DELETE - Used to remove a resource.

PUT - Used to update a existing resource or create a new resource.

PATCH -- Used to perform partial updates to the resource.

What is Restful Web Service?

REST is used to build Web services that are lightweight, maintainable, and scalable in nature. A service which is built on the REST architecture is called a RESTful service. The underlying protocol for REST is HTTP, which is the basic web protocol. REST stands for REpresentational State Transfer

The key elements of a RESTful implementation are as follows:

1. Resources The first key element is the resource itself. Let assume that a web application on a server has records of several employees. Let's assume the URL of the web application is http://www.server.com. Now in order to access an employee record resource via REST, one can issue the command http://www.server.com/employee/1 - This command tells the web server to please provide the details of the employee whose employee number is 1.

2. Request Verbs - These describe what you want to do with the resource. A browser issues a GET verb to instruct the endpoint it wants to get data. However, there are many other verbs available including things like POST, PUT, and DELETE. So in the case of the example http://www.server.com/employee/1 , the web browser is actually issuing a GET Verb because it wants to get the details of the employee record.

3. Request Headers These are additional instructions sent with the request. These might define the type of response required or the authorization details.

4. Request Body - Data is sent with the request. Data is normally sent in the request when a POST request is made to the REST web service. In a POST call, the client actually tells the web service that it wants to add a resource to the server. Hence, the request body would have the details of the resource which is required to be added to the server.

5. Response Body This is the main body of the response. So in our example, if we were to query the web server via the request http://www.server.com/employee/1 , the web server might return an XML document with all the details of the employee in the Response Body.

6. Response Status codes These codes are the general codes which are returned along with the response from the web server. An example is the code 200 which is normally returned if there is no error when returning a response to the client.

A key difference between a traditional MVC controller and the RESTful web service controller is the way that the HTTP response body is created.Instead of relying on a view technology(JSP / Thymeleaf) to perform server-side rendering of the data to HTML, typically a RESTful web service controller simply populates and returns a java object. The object data will be written directly to the HTTP response as JSON/XML/Text

To do this, the @ResponseBody annotation on the ret type of the request handling method tells Spring MVC that it does not need to render the java object through a server-side view layer.

Instead it tells that the java object returned is the response body, and should be written out directly.

The java object must be converted to JSON. Thanks to Spring's HTTP message converter support, you don't need to do this conversion manually. Because Jackson Jar is on the classpath, SC can automatically convert the java object to JSON & vice versa (using 2 annotations @ReponseBody & @RequestBody)

API --Starting point
o.s.http.converter.HttpMessageConverter<T>
--Interface that specifies a converter that can convert from and to HTTP requests and responses.
T --type of request/response body.

Implementation classes
1. o.s.http.converter.xml.Jaxb2RootElementHttpMessageConverter
-- Implementation of HttpMessageConverter that can read and write XML using JAXB2.(Java architecture for XML binding)

2. o.s.http.converter.json.
MappingJackson2HttpMessageConverter
--Implementation of HttpMessageConverter that can read and write JSON using Jackson 2.x's ObjectMapper class API

Important Annotations
1. @ResponseBody

Applied at : return value of the request handling method , annotated with @RequestMapping or @GetMapping / @PostMapping / @PutMapping / @DeleteMapping

It is used to marshall(serialize) the return value into the HTTP response body. Spring comes with converters that convert the Java object into a format understandable for a client(text/xml/json)

eg :
```
@Controller
@RequestMapping("/employees")
public class EmpController
{
   @GetMapping(....)
   public @ResponseBody Emp fetchEmpDetails(int empId)
   {
      //get emp dtls from DB through layers
      return e;
   }
}
```

2. @RestController
Class level annotation

Good news is @RestController = @Controller(at the class level) +
@ResponseBody added on ret types of ALL request handling methods

eg :

```
@RestController
@RequestMapping("/employee")
public class EmpController
{
   @GetMapping(....)
   public Emp fetchEmpDetails(int empId)
   {
      //get emp dtls from DB through layers
      return e;
   }.....
}
```

3. @PathVariable --- handles URI templates.(URI variables or path variables)
Where to apply : on the method argument
Purpose : to access a path variable

eg : URL -- http://host:port/products/1234

Method of ProductController

```
@RestController
@RequestMapping("/products") {
@GetMapping("/{pid}")
public Product getDetails(@PathVariable(name="pid") int pid1234)
{...}
}
```

In the above URL , the path variable {pid} is mapped to an int . Therefore all of the URIs such as /products/1 or /products/10 will map to the same method in the controller.

4. The @RequestBody annotation, unmarshalls the HTTP request body into a Java object injected in the method.
Applied on the method argument of the reuqest handling methods , containing request body
eg : Typically in POST , PUT , PATCH

@RequestBody must be still added on a method argument of request handling method , for un marshaling(de serialization)

5. @CrossOrigin
Class/Method level annotation

What is CORS ?
Cross-Origin Resource Sharing (CORS)

CORS is an HTTP-header based mechanism that allows a server to indicate any origins (domain, scheme, or port) other than its own from which a browser should permit loading of resources.

A cross-origin HTTP request is a request to a specific resource, which is located at a different origin, namely a domain, protocol and port, than the one of the client performing the request.

For security reasons, browsers can request several cross-origin resources, including images, CSS, JavaScript files etc. By default, a browserâ€™ security model will deny any cross-origin HTTP request performed by client-side scripts.

While this behavior is desired, to prevent different types of Ajax-based attacks, sometimes we need to instruct the browser to allow cross-origin HTTP requests from JavaScript clients with CORS.

eg : React client running on http://localhost:3000, and a Spring Boot RESTful web service API listening at http://host:port/products

In such a case, the client should be able to consume the REST API, which by default would be forbidden.

To make this work , enable CORS by simply annotating the class / methods of the RESTful web service API responsible for handling client requests with the @CrossOrigin annotation

eg : @CrossOrigin(origins = "http://localhost:3000")
```
@RestController
public class ProductController{....}
```

What is REST?

This term "REST" was first defined by Roy Fielding in 2000.
It stands for Representational State Transfer(REST).
Actually, REST is an architectural model and design for server network applications.

The most common application of REST is the World Wide Web itself, which used REST as a basis for HTTP 1.1 development.

What is REST Architecture and REST Constraints

What is the REST API?
A RESTful API is an application program interface (API) that uses HTTP requests to GET, PUT, POST and DELETE data.

Representational state transfer (REST), which is used by browsers or front end app , can be thought of as the language of the Internet.

REST architectural Model

REST-REpresentational State Transfer

Resource-based
REST is resource based API because RESTful API is as below points

Nouns vs Verbs
Identified by URIs
Multiple URIs may refer to the same resource as like CRUD operation on student resource using HTTP verbs.
Separate from their representations-resource may represent as per as request content type either JSON or XML etc.

Representation
How resources get manipulated
Part of the resource state-transferred between client and server
Typically JSON or XML

For Example-

Resource– Author (Rama)
Service– Contact information about Rama(GET)
Representation-name,mobile, address as JSON or XML format

REST Constraints
REST architecture style describes six constraints for REST APIs.

1. Uniform Interface
The uniform interface constraint defines the interface between clients and servers. It simplifies and decouples the architecture, which enables each part to evolve independently.

For us this means

HTTP Verbs (GET,POST,PUT,DELETE)
URIs (resource name)
HTTP response (status and body)
The four guiding principles of the uniform interface are:

1. 1) Identifying the resource – Each resource must have a specific and cohesive URI to be made available, for example, to bring a particular user registered on the site:

```
HTTP/1.1 GET http://www.abc.com:8080/user/Rama
```

1. 2) Resource representation – Is how the resource will return to the client. This representation can be in HTML, XML, JSON, TXT, and more. Example of how it would be a simple return of the above call:

```
{
"name": "Rama Parekh",
"job": "REST API Developer",
"hobbies": ["blogging", "coding", "music"]
}
```

1. 3) Self-descriptive Messages – Each message includes enough information to describe how to process the message. Beyond what we have seen so far, the passage of meta information is needed (metadata) in the request and response. Some of this information are HTTP response code, Host, Content-Type etc. Taking as an example the same URI as we have just seen:

```
GET /user/Rama HTTP/1.1
User-Agent: Chrome/37.0.2062.94
Accept: application/json
Host: abc.com
```

1. 4) Hypermedia as the Engine of Application State (HATEOAS)– Clients deliver state via body contents, query-string parameters, request headers and the requested URI (the resource name). Services deliver the state to clients via body content, response codes, and response headers. This part is often overlooked when talking about REST. It returns all the necessary information in response to the client knows how to navigate and have access to all application resources.

Just one example:

Request:
```
HTTP/1.1 POST http://abc.com/Rama/posts
```
Response:

```
{
"post": {
"id": 42,
"title": "concepts REST",
"decription": "a little about the concept of architecture of REST",
"_links": [
{
```

```
"href": "/Rama/post/42",
"method": "GET",
"rel": "self"
},
{
"href": "/Rama/post/42",
"method": "DELETE",
"rel": "remove"
},
{
"href": "/Rama/post/42/comments",
"method": "GET",
"rel": "comments"
},
{
"href": "/dinesh/posts/42/comments",
"method": "POST",
"rel": "new_comment"
},
{...}
]
},
"_links": {
"href": "/post",
"method": "GET",
"rel": "list"
}
}
```

2. Stateless

One client can send multiple requests to the server; however, each of them must be independent, that is, every request must contain all the necessary information so that the server can understand it and process it accordingly. In this case, the server must not hold any information about the client state. Any information status must stay on the client – such as sessions.

3. Cacheable

Because many clients access the same server, and often requesting the same resources, it is necessary that these responses might be cached, avoiding unnecessary processing and significantly increasing performance.

4. Client-Server

The uniform interface separates clients from servers. This separation of concerns means that, for example, clients are not concerned with data storage, which remains internal to each server, so that the portability of client code is improved. Servers are not concerned with the user interface or user state so that servers can be simpler and more scalable. Servers and clients may also be replaced and developed independently, as long as the interface is not altered.

5. Layered System

A client cannot ordinarily tell whether it is connected directly to the end server, or to an intermediary along the way. Intermediary servers may improve system scalability by enabling load-balancing and by providing shared caches. Layers may also enforce security policies.

6. Code-On-Demand (Optional)
This condition allows the customer to run some code on demand, that is, extend part of server logic to the client, either through an applet or scripts. Thus, different customers may behave in specific ways even using exactly the same services provided by the server. As this item is not part of the architecture itself, it is considered optional. It can be used when performing some of the client-side services which are more efficient or faster.

Ref : https://www.javaguides.net/2018/06/rest-architectural-constraints.html

What is a web service ?
A solution to distributed computing.

Integral part of SOA (service oriented architecture)
service = Business functionality to be exported to remote clnts via standard protocols(HTTP/s)
server -- service provider
clnt --service accessor

Why -- To export the Business logic (functional logic --banking, customer service, payment gateway , stock exchange server BSE , NSE ...) to remote clients over standard set of protocols.

Its equivalent to Java RMI (remote method invocation)
In Java RMI -- java clnt object can directly invoke the remote method (hosted on the remote host) & get n process results. (i.e it gives you location transperency)
BUT Java RMI ---is 100% java solution.
There is no inter operability in that(i.e its a technologoy specific soln)
How to arrive at technology inde soln ?

CORBA --- Common obj request borker architecture
tough to set up. (IDL ---i/f def language)
Better alternative --- web services
In 2002, the World Wide Web consortium(W3C) had released the definition of WSDL(web service definition language) and SOAP web services. This formed the standard of how web services are implemented.
Earlier (J2EE 1.4) -- JAX-RPC
Java API for XML based remote procedure calls
Today replaced by JAX-WS

In 2004, the web consortium also released the definition of an additional standard called RESTful. Over the past couple of years, this standard has become quite popular. And is being used by many of the popular websites around the world which include Facebook and Twitter.

Corresponding J2EE specification JAX RS

JAX WS -- Java API for XML based web services -- Based upon
Protocol --SOAP -- simple object access protocol (runs over HTTP)
Has additional header & message format .
+ Have to set up Naming service (UDDI --Universal Description, Discovery, and Integration)
+ Have to set up WSDL (web service def. language)-- xml based web service def lang.
--supports only XML as data exchange format.

Too much to set up & eats up larger bandwidth !!
So a simple soln is JAX RS -- Java API for RESTful web service
JAX RS --- is a part of J2EE specifications
Known Vendors --Apache , JBoss
& products --RESTeasy,Apache CXF
BUT it's still difficult to set up.
So spring , being integration master , comes to the rescue.....

Aspect Oriented Programming(AOP)

WHY

Separating cross-cutting concerns(=repeatative tasks) from the business logic/ request handling /persistence

IMPORTANT

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect. Dependency Injection helps you decouple your application objects from each other(eg : Controller separate from Service or DAO layers) and AOP helps you decouple cross-cutting concerns from the objects that they affect.(eg : transactional code or security related code can be kept separate from B.L or exception handling code from request handling method)
eg of ready made aspects provided by SC : tx management,security , exc handling

eg scenario --

Think of a situation Your manager tells you to do your normal development work(eg - write stock trading appln) + write down everything you do and how long it takes you.

A better situation would be you do your normal work, but another person observes what youre doing and records it and measures how long it took.

Even better would be if you were totally unaware of that other person and that other person was able to also observe and record , not just yours but any other peoples work and time.

That's separation of responsibilities.   --- This is what spring offers you through AOP.

It is NOT an alternative to OOP BUT it complements OOP.

The key unit of modularity in OOP is the class, whereas in AOP the unit of modularity is the aspect.
Aspects enable the modularization of concerns.(concern=task/responsibility) such as transaction management,logging,security --- that cut across multiple types and objects. (Such concerns are often termed crosscutting concerns in AOP jargon)

Enables the modularization of cross cutting concerns(=task)

Eg : Logging,Security,Transaction management, Exception Handling

Similar in functionality to ---In EJB framework -- EJBObject
Struts 2 -- interceptors
Servlet -- filters.
RMI -- stubs
Hibernate --- proxy (hib frmwork -- lazy --- load or any--many associations --rets typically un-inited proxy/proxies)

AOP with Spring Framework

One of the key components of Spring Framework is the Aspect oriented programming (AOP) framework.

Like DI, AOP supports loose coupling of application objects.

The functionalities that span multiple points of an application are called cross-cutting concerns.

With AOP, applicationwide concerns(common concerns-responsibilities or cross-cutting concerns like eg - declarative transactions , security,logging,monitoring,auditing,exception handling....)
are decoupled from the objects to which they are applied.

Its better for application objects(service layer/controller/rest controller/DAO) to focus on the business domain problems that they are designed for and leave certain ASPECTS to be handled by someone else.

Job of AOP framework is --- Separating these cross-cutting concerns(repeatative tasks) from the core business logic

AOP is like triggers in programming languages such as Perl, .NET.

Spring AOP module provides interceptors to intercept an application, for example, when a method is executed, you can add extra functionality before or after the method execution.

Key Terms of AOP

Advice : Action(=cross cutting concern) to take either before/after or around the method (req handling logic ,OR B.L OR data access logic) execution. eg: transactional logic(begin tx,commit,rollback,session closing)

Advice describes WHAT is to be done & WHEN it's to be done.

eg : logging. It is sure that each object will be using the logging framework to log the event happenings , by calling the log methods. So, each object will have its own code for logging. i.e the logging functionality requirement is spread across multiple objects (call this as Cross cutting Concern, since it cuts across multiple objects). Wont it be nice to have some mechanism to automatically execute the logging code, before executing the methods of several objects?

2. Join Point : Place in application WHERE advice should be applied.(i.e which B.L methods should be advised)
(Spring AOP, supports only method execution join point)

3. Pointcut : Collection of join points.
It is the expression used to define when a call to a method should be intercepted.
eg :

```
@Pointcut("execution (Vendor com.app.bank.*.*(double))")
advice logic{....}
```

4. Advisor Group of Advice and Pointcut into a single unit.

5. Aspect : class representing advisor(advice logic + point cut definition)-- @Asepct -- class level annotation.

6. Target : Application Object containing Core domain logic.(To which advice gets applied at specified join points) --supplied by Prog

7. Proxy : Object created after applying advice to the target object(created by SC dynamically by implementing typically service layer i/f) ---consists of cross cutting concern(repeatative jobs , eg : tx management,security, exc handling)
Understanding proxies
The most basic concept that we need to understand how AOP works in Spring is that of a Proxy. A proxy is an object that wraps another object maintaining its interface and optionally providing additional features. Proxies usually delegate behavior on the real object they are proxying but can execute code around the call to the wrapped object.
Spring uses proxies under the hood to support some of its magic features including AOP.

8.Weaving -- meshing(integration) cross cutting concern around B.L
(3 ways --- compile time, class loading time or spring supported --dynamic --method exec time or run time)

Examples of readymade aspects :
Transaction management & security.

Types of Advice --appear in Aspect class

@Before : This advice (cross cutting concern) logic gets Executed only before B.L method execution.
@AfterReturning Executes only after method returns in successful manner
@AfterThrowing - Executes only after method throws exception
@After -- Executes always after method execution(in case of success or failure)

@Around -- Most powerful, executes before & after.

Regarding pointcuts
Sometimes we have to use same Pointcut expression at multiple places, we can create an empty method with @Pointcut annotation and then use it as expression in advices.

eg of PointCut annotation syntax

```
@Before("execution (* com.app.bank.*.*(..))")

@Pointcut("execution (* com.app.bank.*.*(double))")

// point cut expression
@Pointcut("execution (* com.app.service.*.add*(..))")
      // point cut signature -- empty method .
      public void test() {
      }
```

eg of Applying point cut

```
1.  @Before(value = "test()")
public void logBefore(JoinPoint p) {.........}

2.
@Pointcut("within(com.app.service.*)")
    public void allMethodsPointcut(){}

@Before("allMethodsPointcut()")
    public void allServiceMethodsAdvice(){...}

3.
@Before("execution(public void com.app.model..set*(*))")
 public void loggingAdvice(JoinPoint joinPoint){pre processing logic ....}

4.  //Advice arguments, will be applied to bean methods with single String
    argument
    @Before("args(name)")
    public void logStringArguments(String name){....}

5.  //Pointcut to execute on all the methods of classes in a package
    @Pointcut("within(com.app.service.*)")
    public void allMethodsPointcut(){}

6.  @Pointcut("execution(* com.core.app.service.*.*(..))") // expression
private void meth1() {}  // signature

7.  @Pointcut("execution(* com.app.core.Student.getName(..))")
private void test() {}
```

Steps in AOP Implementation
1. Create core java project.
2. Add AOP jars to runtime classpath.
3. Add aop namespace to spring config xml.

4. To Enable the use of the @AspectJ style of Spring AOP & automatic proxy generation, add <aop:aspectj-autoproxy/>
5. Create Business object class. (using stereotype anotations)
6. Create Aspect class, annotated with @Aspect & @Component
7. Define one or more point cuts as per requirement

Eg of Point cut definition.

```
@PointCut("execution (* com.aop.service.Account.add*(..))")
public void test() {}
OR
@Before("execution (* com.aop.service.Account.add*(..))")
public void logIt()
{
   //logging advice code
}
```

Use such point cut to define suitable type of advice.

Test the application.

execution --- exec of B.L method

```
eg : @Before("execution (* com.app.bank.*.*(..))")
public void logIt()  {...}
```

Above tell SC ---- to intercept ---ANY B.L method ---
having ANY ret type, from ANY class from pkg -- com.app.bank
having ANY args
Before its execution.

Access to the current JoinPoint

Any advice method may declare as its first parameter, a parameter of type org.aspectj.lang.JoinPoint (In around advice this is replaced by ProceedingJoinPoint, which is a subclass of JoinPoint.)

The org.aspectj.lang.JoinPointJoinPoint interface methods
1. Object[] getArgs() -- returns the method arguments.
2. Object getThis() --returns the proxy object
3. Object getTarget() --returns the target object
4. Signature getSignature() -- returns a description of the method that is being advised
5. String toString() -- description of the method being advised

Objective : Invoking one REST API from another REST API
Request URL from front end (postman):
http://host:port/api/employees/{empId}/accounts/{acctNo}

Resource : employees (available on EMS backend : which will acts as REST clnt to NetBanking Server)
Sub Resource : accounts (available on NetBanking Server)

1.1 NetBanking REST Server
Ready made Spring boot project : NetBankingRESTServer

DML
```
insert into bank_customers values('hdfc-00001','Rama','12345');
insert into bank_accounts values(default,0,'SAVING',234567,'hdfc-00001');
insert into bank_accounts values(default,0,'CURRENT',24567,'hdfc-00001');
```

REST API : /bank/accounts/{acctNo}

1.2 Employee App REST Server for React App & client to NetBanking
1.3 Front end Postman (later add it in React app)

Details
2. Objective : Testing E-R with REST API + REST Client(RestTemplate)
Test setup : Postman -- Emp Management API invoking REST Banking API

Get Account summary for a bank customer.
Resource : /accounts
I/P : acct no

O/P : In case of success : Account DTO
or in case of invalid credentials : Send Error resp code : HTTP 404 (not found)

Get acct details
Method =GET (/bank/accounts/{acctNo})

Layers :
REST Controller --Service --Repository--POJO --DB
Customer 1<------* BankAccount
Customer : customer id(eg of assigned id here) ,name, password

BankAccount : acct id (auto generation) AcctType(enum) ,balance + Customer owner

For Data Transfer : DTOs
LoginRequest : customerId , password
LoginResponse : customer name & List<BankAccount>

How to make a REST call from one web app to another ?
Use : org.springframework.web.client.RestTemplate

The RestTemplate class in Spring Framework is a synchronous HTTP client for making HTTP requests to consume RESTful web services.

It exposes a simple and easy-to-use template method API for sending an HTTP request and also handling the HTTP response.

The RestTemplate class also provides aliases for all supported HTTP request methods, such as GET, POST, PUT, PATCH , DELETE, and OPTIONS.

In a service layer : inject

```
public class ClntService {
private RestTemplate template;

      @Autowired //autowire=constructor
      public ClntService(RestTemplateBuilder builder) {
                  template = builder.build();
      }


}
      // SpEL : spring expression language
      @Value("${REST_GET_URL}")
      private String authUrl;
```

Use Method of o.s.w.c.RestTemplate public <T> ResponseEntity<T>

1. public <T> ResponseEntity<T> getForEntity(String url,Class<T> responseType,Object... uriVariables) throws RestClientException

2. public <T> ResponseEntity<T> postForEntity(String url,@Nullable Object request, Class<T> responseType, Object... uriVariables) throws RestClientException

Invoke REST end point (post)
URL : http://localhost:8080/api/employees
raw json body :

```
{
}
```

What will happen ? No ERROR !!!!!!!!!!!!!!
Since no validation rules added in the back end

1. Objective Add request data Validation rules back end

Steps
1.1. Add validation dependency
We Have already added , "spring-boot-starter-validation" in Spring boot project.

1.2. Identify validation rules , add these annotations on POJO properties.
eg : @NotBlank, @Pattern, @Min, @Max...
Imported form javax.validation , org.hibernate.validator
(refer to templates.txt under <ready code>)

1.3. For validating RequestBody : add @Valid annotation in addition to
     @RequestBody
SC performs un marshalling + validation

1.4. Invoke REST end point (post)
URL : http://localhost:8080/emps/
raw json body :

```
{
}
```

Observation : HTTP status 400 , BUT entire error stack trace sent to clnt.
Exception : MethodArgumentNotValidException

Can it be avoided by catching the exception in RestController (ans : in some cases yes eg : custom exc) , but then will have to supply multiple repeatative exc handling code.

How to add global(centralized) exception handler ?
Using 2 Annotations

@ControllerAdvice & @ExceptionHandler

@ControllerAdvice is a specialization of the @Component annotation which allows to handle exceptions across the whole application in one global handling component.

It's interceptor of exceptions thrown by methods annotated with @RequestMapping and similar(eg : @GetMapping,@PostMapping....)

Add in this class , @ExceptionHandler methods to be shared across multiple @Controller classes.

Typically extend this global exc handler by ResponseEntityExceptionHandler

What is it ?
A convenient base class for @ControllerAdvice classes to provide centralized exception handling across all @RequestMapping methods through @ExceptionHandler methods.
This base class provides an @ExceptionHandler method for handling internal Spring MVC exceptions. This method returns a ResponseEntity for writing to the response with a message converter,

@ExceptionHandler : method level annotation.

Steps :
1. Create a class that extends from ResponseEntityExceptionHandler
2. Add class level annotation @ControllerAdvice
3. Add @ExceptionHandler methods , to handle different type of exceptions.

For handling Spring MVC validation errors : extend exception handler class from ResponseEntityExceptionHandler & override
protected ResponseEntity<Object> handleMethodArgumentNotValid(MethodArgumentNotValidException ex,HttpHeaders headers,HttpStatus status,WebRequest request)

4. For Path Variables/request params : @Validated , at class level on controller class.

Exception raised : ConstraintViolationException

5. Any remaining exceptions , handle it by a common handler method.

Objective 2. : Adding global / centralized exception handler
Problem : Access REST end point : http://localhost:8080/emps/123
(i.e non existing emp id)
Observation : Complete error stack on front end .
Fix it

Basic Authentication is a method(or scheme) for an HTTP client (eg a web browser or Front end app) to provide a username and password when making a request.

In Our REST API example :

When a client makes a request to protected(secured) resource , REST API sends back error code HTTP 401 (un authenticated)
If it's a web browser client , user will be prompted with a login form.

You can see the in postman , resp header
WWW-Authenticate : Basic Realm

If it's a postman or a react app , client will have to send the "Authorization" header . If it succeeds , then client can access the secured resource.

When using Basic Authentication, clients will include an encoded string(not encrypted) in the "Authorization" header of each request they make. The string is used by the request’s recipient(i.e Spring security filter) to verify users’ identity and permissions to access a resource.

The Authorization header follows this format:

Authorization: Basic <credentials>

Base64(username:password)

Credentials are constructed like explained below :
The user’s username and password are combined with a colon.
The resulting string is base64 encoded.
So if your username and password are abc and 1234 , the combination is abc:1234 , and when base64 encoded, this becomes YWJjOjEyMzQ=

So requests made by this client would be sent with the following header:

eg : Authorization: Basic YWJjOjEyMzQ=

Security issues : Since above info is just a text value , can be decoded easily. So should be sent over HTTPS , for encryption

For base64 encoding :
Ref : https://www.base64encode.org/
For decoding :
https://www.base64decode.org/

Why Filters ?

1. Provides re-usability.
Meaning --- They provide the ability to encapsulate recurring tasks(=cross cutting concerns) in reusable units.
They provide clear cut separation between B.L & cross cutting concerns.

2. Can dynamically intercept req/resp to dyn or static content

What is Filter?

Dynamic web component just like servlet or
JSP. Resides within web-appln.(WC)
Filter life-cycle managed by WC

It performs filtering tasks on either the request to a resource (a servlet,JSP or static content), or on the response from a resource, or both.

It can dynamically intercepts requests and responses .

Usage of Filters
1. Authentication Filters
2. Logging Filters
3. Image conversion Filters
4. Data compression Filters
5. Encryption Filters
6. Session Check filter

How to create Filter Component?
1. Create Java class imple. javax.servlet.Filter i/f
2. Implements 3 life-cycle methods
2.1. public void init(FilterConfig filterConfig)
          throws ServletException

Above called by WC --- only once during filter creation & initialization.(@appln start up time)
2.2.
void doFilter(ServletRequest request,ServletResponse response,FilterChain chain) throws IOException, ServletException

Invoked by WC -- per every rq & resp processing time.

Here u can do pre-processing of req, then invoke chain.doFilter -- to invoke next component of filter chain --- finally it invokes service method of JSP/Servlet --- on its ret path visits filter chain in opposite manner & finally renders response to clnt browser.

2.3.
public void destroy() ---invoked by WC at the end of filter life cycle.
Triggers --- server shut down/re-deploy/un deploy

How to deploy a Filter component ?

1. Annotation -- @WebFilter (class level annotation)
OR
2. XML tags (in web.xml)

```
<filter>
 <filter-name>abc</..>
 <filter-class>filters.AuthenticatioFilter</...>
 <init-param>
  <param-name>nm1</..>
  <param-value>val1</..>
 </...>
</filter>
<filter-mapping>
 <filter-name>abc</..>
 <url-pattern>/*</...>
</filter-mapping>
```

Detailed Description ---
Filters typically do not themselves create responses, but instead provide universal functions that can be "attached" to any type of servlet or JSP page.

Filters are important for a number of reasons. First, they provide the ability to encapsulate recurring tasks in reusable units. Organized developers are constantly on the lookout for ways to modularize their code. Modular code is more manageable and documentable, is easier to debug, and if done well, can be reused in another setting.

Second, filters can be used to transform the response from a servlet or a JSP page. A common task for the web application is to format data sent back to the client. Increasingly the clients require formats (for example, WML) other than just HTML. To accommodate these clients, there is usually a strong component of transformation or filtering in a fully featured web application. Many servlet and JSP containers have introduced proprietary filter mechanisms, resulting in a gain for the developer that deploys on that container, but reducing the reusability of such code. With the introduction of filters as part of the Java Servlet specification, developers now have the opportunity to write reusable transformation components that are portable across containers.

Spring Security

Spring Security is a powerful and highly customizable authentication and access-control framework.
It is supplied as a "ready made aspect" , from spring security framework , that can be easily plugged in spring MVC application.

It is "THE" standard for securing Spring-based applications. Spring Security is a framework that focuses on providing both authentication and authorization to Java applications.

Like all Spring projects, the real power of Spring Security is found in how easily it can be extended to meet custom requirements.

Features

1. Comprehensive and extensible support for both Authentication and Authorization
2. Protection against attacks like session fixation, clickjacking, cross site request forgery, etc..
3. Servlet API integration (Uses Servlet Filter chain)
4. Integration with Spring Web MVC.

Common Security Terms

Credentials : Way of confirming the identity of the user (email /username , password typically)
Principal: Currently logged in user.
Authentication: Confirming truth of credentials(i.e confirming who you are)
Authorisation: Defines access policy for the Principal.(i.e confirming your permissions, i.e what you can do)
GrantedAuthority: Permission granted to the principal.
AuthenticationManager (i/f): Controller in the authentication process.
Authenticates user details via authenticate() method.

Steps

1. Create a new spring boot project (RESTful web service) , adding usual dependenices . DO NOT add spring security yet. Copy earlier working application.properties.

NOTE : spring boot version downgrade : 2.6.7

2. Add a ProductController (/products), with 3 end points

/view : any one should be able view the products
/purchase : customer should be able to purchase products
/add : admin should be able to add the products
/categories : only authenticated user (any role !) can browse the categories

Currently : respond to GET method , with simple string response.

3. Test it .(using browser/postman)

Did it work ????

4. Add spring security dependency
 . Test the end points again.
Did it work ? NO

Observation : Suddenly n automatically all end points are now protected.
So on browser it will prompt you to login form (spring security supplied)
On postman it will give you HTTP 401 (Un authorized error)

We have not yet supplied any credentials .
Def credentials are : user n password(UUID : universally unique ID : 128 bit) from server console.

So w/o configuring anything , the moment spring security JARs are added , all your end points are secured automatically .

Thus Spring Boot(running on the top of the Spring Framework) , provides a ready made aspect(solution to cross cutting concern like authetication n authorization) in form of spring security

After supplying correct credentials(i.e after authentication) , spring security will redirect you to the resource : http://localhost:8080/home ,and you will be able to access it.
Supplies you automatically with a logout page (test it on the browser)

Observe on postman(w/o setting authorization header)
Response : (HTTP 401)

From authorization , choose Basic Authentication (referred as Basic Auth) ,
Add user name n password.

It will be encoded using base64 encoding.

Basic authentication, or “basic auth” is formally defined in the HTTP standard. When a client (your browser) connects to a web server, it sends a “WWW-Authenticate: Basic” message in the HTTP header. After that, it sends your login credentials to the server using a mild concealment technique called base64 encoding.

Not desirable , to use such credentials , so continue to next step.

5. Can you configure username n password , in a property file ?
YES .

Add these 2 properties in application.properties file

```
spring.security.user.name=
spring.security.user.password=
```

So now instead of spring security generated user name n pwd , these will be used for authentication.

6. Ultimate goal is using DB to store the authentication details .
BUT immediate next goal , to understand spring security is : Basic In memory authentication

The credentials will be stored in memory.
Comment earlier properties from app property file.

6.1. Add security config class, extending
     org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter
It's a convenient base class , to customize security configuration

6.2. Class level annotations
@EnableWebSecurity
@Configuration (annotation based approach equivalent to bean config xml file containing <bean id xml..../>)

6.3. Override
protected void configure(AuthenticationManagerBuilder auth) throws Exception
for supplying authentication details

6.4. Refer to diag : spring security architecture
Refer to readme : "spring sec auth flow"
Diagram : detailed flow.png

Add in memory authentication to the AuthenticationManagerBuilder , which will allow customization of the same.

API Methods
inMemoryAuthentication
withUser
password
roles
and

eg :
```
auth.inMemoryAuthentication().withUser("kiran").password(encoder().encode("1234")).roles("USER")
.and().withUser("rama").password(encoder().encode("3456")).roles("ADMIN");
```

6.5. For supplying authorization details :
Objective :
/home : accessible to all
/admin : only to admin user
/user : accessible to user n admin role

Override
protected void configure(HttpSecurity http) throws Exception

By extending from WebSecurityConfigurerAdapter and only a few lines of code we can do the following:

1. Require the user to be authenticated prior to accessing any URL within our application
2. Create a user with the username “user”, password “password”, and role of “USER”
3. Enables HTTP Basic and Form based authentication
4. Spring Security will automatically render a login page and logout success page for you

Refer to it's super class 's implementation n use it for overriding
Methods :
authorizeRequests()
antMatchers(String matchers...)
hasRole(String roleName) : no ROLE prefix
httpBasic()
formLogin

eg :
```
http.authorizeRequests().antMatchers("/admin").
                 hasRole("ADMIN").
                 antMatchers("/user").hasAnyRole("USER", "ADMIN")
                                  .antMatchers("/", "/home").permitAll()
                                   .and().httpBasic()
                                  .and().formLogin();
```

6.6 Run the application.
Problem : java.lang.IllegalArgumentException: There is no PasswordEncoder mapped for the id "null"

Reason -- Prior to Spring Security 5.0 the default PasswordEncoder was NoOpPasswordEncoder which required plain text passwords.
From Spring Security 5, the default is DelegatingPasswordEncoder, which requires Password Storage Format.

Solution : provide Password encoder bean

```
@Bean // equivalent to <bean id ...> tag in xml)
      public PasswordEncoder encoder() {
                 return new BCryptPasswordEncoder();
      }
```

Test the application.

7. Replace in memory authentication by DB based authentication.
Using Spring Data JPA.

7.1 Edit application.properties file with DB settings.

Can optionally add these for debugging.

```
debug=true
logging.level.org.springframework.security=DEBUG
```

7.2 In Security config class
replace in memory authentication , by UserDetailsService based auth mgr builder
refer to diag : detailed flow.png

API of AuthenticationManagerBuilder

public DaoAuthenticationConfigurer userDetailsService(UserDetailsService service) throws Exception

Add authentication based upon the custom UserDetailsService that is passed in. It then returns a DaoAuthenticationConfigurer to allow customization of the authentication.

So auto wire UserDetailsService n use it. Set password encoder.

eg :

```
@Autowired
private UserDetailsService userDetailsService;
protected void configure(AuthenticationManagerBuilder auth) throws
Exception {
                 //since there is no out-of-box imple for JPA based auth ,
u have to create a custom class imple UserDetailsService n inject it here,
n set password encoder

      auth.userDetailsService(userDetailsService).passwordEncoder(encoder(
));

      }
```

7.3
The org.springframework.security.core.userdetails.UserDetailsService interface is used to retrieve user-related data. It has one method named loadUserByUsername() which can be overridden to customize the process of finding the user.

It is used by the DaoAuthenticationProvider to load details about the user during authentication.
It is used throughout the framework as a user DAO and is the strategy used by the DaoAuthenticationProvider.

class DaoAuthenticationProvider :
Represents an AuthenticationProvider implementation that retrieves user details from a UserDetailsService.

Method

```
UserDetails loadUserByUsername(java.lang.String username)
                        throws UsernameNotFoundException
```

7.4 How to load user by user name ?

1. Create POJOs User n Role with many-many (UserEntity *---->* Role)
   EAGER or can later replace it by UserEntity *---->1 Role

UserEntity extending from BaseEntity
Properties : userName,email, password,active, roles : Set<Role>
Role : id , enum UserRole (ROLE_USER....)

2. DAO layer : UserRepository -- findByUserName

RoleRepository

3. Create custom implementation of
   org.springframework.security.core.userdetails.UserDetailsService

n implement

```
UserDetails loadUserByUsername(String username)
                        throws UsernameNotFoundException
```

In case , user entity not found , raise UsernameNotFoundException , with suitable error message.

4. In case of success , create custom implementation of ,
   org.springframework.security.core.userdetails.UserDetails i/f

, by passing to it's constructor , User entity details , lifted from DB

o.s.s.c.userdetails.UserDetails : represents core user information. It stores
user information which is later encapsulated into Authentication object. This
allows non-security related additional user information (eg : email
acct expiry, user enabled ...) in addition to user name n password to be stored in a convenient location.

One important method in above i/f to implement is

```
     public Collection<? extends GrantedAuthority> getAuthorities() ,
```

which should return , granted authorities (role based) for the loaded user.

eg : user => UserEntity

```
user.getRoles().stream().map(role -> new
SimpleGrantedAuthority(role.getUserRole().name()))
.collect(Collectors.toList());
```

5. Implement all other methods , suitably .

How to run ?

1. Write dao layer test case : to add 2 roles : ROLE_ADMIN n ROLE_USER

2. Write dao layer test case : to add 2 users , with admin n user role
   each.

3. For hashing the password :

use :
https://bcrypt-generator.com/

For more details : https://dzone.com/articles/hashing-passwords-in-java-with-bcrypt#:~:text=One%20way%20hashing%20%2D%20BCrypt%20is,hashes%20across%20each%20user's%20password.

Project Tip :
Later to test it with React/Angular front end :
use below for authorization.

```
      http.csrf().disable().
                  cors().and().
                  authorizeRequests().

                  antMatchers(HttpMethod.OPTIONS, "/**").permitAll().
                  antMatchers("/", "/home", "/api/signup").permitAll().
                  antMatchers("/admin").hasRole("ADMIN").
                  antMatchers("/user").hasAnyRole("USER", "ADMIN").
                  and().httpBasic();
```

How does spring security works internally?

Spring security is enabled automatically , by just adding the spring security starter jar. But, what happens internally and how does it make our application secure?

Common Terms

Principal: Currently logged in user.
Authentication: Confirming truth of credentials.
Authorisation: Defines access policy for the Principal.
GrantedAuthority: Permission granted to the principal.
AuthenticationManager (i/f): Controller in the authentication process. Authenticates user details via authenticate() method.

AuthenticationManager i/f implemented by : ProviderManager class .
Diagram : detailed flow .png
It Iterates an Authentication request through a list of AuthenticationProviders.
AuthenticationProviders are usually tried in order until one provides a non-null response. A non-null response indicates the provider had authority to decide on the authentication request and no further providers are tried.

AuthenticationProvider: Interface that maps to a data store that stores your data.
Authentication Object: Object that is created upon authentication. It holds the login credentials. It is an internal spring security interface.
UserDetails: Data object that contains the user credentials but also the role of that user.
UserDetailsService: Collects the user credentials, authorities (roles) and build an UserDetails object.

The Spring Security Architecture

When we add the spring security starter jar, it internally adds Filter to the application. A Filter is an object that is invoked at pre-processing and post-processing of a request. It can manipulate a request or even can stop it from reaching a servlet. There are multiple filters in spring security out of which one is the Authentication Filter, which initiates the process of authentication.

Once the request passes through the authentication filter, the credentials of the user are stored in the Authentication object. Now, what actually is responsible for authentication is AuthenticationProvider (Interface that has method authenticate()). A spring app can have multiple authentication providers, one may be using Dao based , JWT based , OAuth, LDAP ... To manage all of them, there is an AuthenticationManager.

The authentication manager finds the appropriate authentication provider by calling the supports() method of each authentication provider. The supports() method returns a boolean value. If true is returned, then the authentication manager calls its authenticate() method.

After the credentials are passed to the authentication provider, it looks for the existing user in the system by UserDetailsService. It returns a UserDetails instance which the authentication provider verifies and authenticates. If success, the Authentication object is returned with the Principal and Authorities otherwise AuthenticationException is thrown.