##

DAY 1 - JDBC

##

java database Connectivity (JDBC)

using jdbc, java application can connect to any database such as MySQL, Oracle, SQLServer etc.
jdbc provides two things:
a) **jdbc API [classes and interfaces]** which remains same no matter which database you use.
b) **jdbc drivers:** driver is a software to connect to database

There are 4 types of drivers in jdbc:

Type 1 - which requires ODBC to be installed on a machine. It has almost become obsolete.

Type 2 - which requires database client library to be installed on client machine. cannot be used for internet purpose.

Type 3 - which requires middleware (Application server) to get connected to database.

Type 4 - which is the fastest driver as it directly connects to the database.

com/mysql/jdbc/Driver.class

Steps required for jdbc application:
a) load and register the driver with DriverManager.
b) get connection with database
c) communicate with database with the help of JDBC API.

API for JDBC

1. **Connection** - interface - used to represent the connection with the database.
2. **DriverManager** - class - used to get the connection
3. **Statement** - interface - used to communicate with the database (query or update statements)
4. **ResultSet** - interface - used to store records retrieved from database.

```
Connection con=DriverManager.getConnection(ss,"root","170496")
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from dept");
```

Syntax Different between Statement & PrepareStatement

```
Statement st=con.createStatement();
ResultSet rs=st.executeQuery("select * from dept");
```

vs

```
PreparedStatement pst=con.prepareStatement("select * from dept");
ResultSet rs=pst.executeQuery();
```

PreparedStatement has 2 advantages over Statement

a) u can use placeholder so that values can be passed dynamically.
b) PreparedStatement contains precompiled sql statements which enable faster execution.

ResultSetMetaData: interface which gives complete information about ResultSet such as no. of columns, their names, types.

Scrollable and Updatable ResultSet

By default, ResultSet is read-only and forward only. It means while traversing through the ResultSet we cannot modify records and we have only one method for traversing i.e., "next".

But we can make ResultSet updatable (while traversing through the ResultSet we can modify records) and scrollable (not only "next" we can invoke "previous" or "absolute" method also)

Disconnected view of data:

A disconnected environment is one in which a user is not necessarily connected with a database. Connection is required only at the time of retrieval after that connection is close and if data needs to be updated again connection will be reestablished.

The main advantage of disconnected environment is no need to have a permanent connection with a Data Source which is advantageous especially when there are thousands of records in database which u want to traverse.

Another advantage is difference between ResultSet and CachedRowSetImpl. ResultSet is not serializable but CachedRowSetImpl is. Hence, we can always pass CachedRowSetImpl over the network.

Properties for JDBC

When we have number of application where every time, we need to make connection with database with properties – URL, user, password.
So here we made one file with .properties extension which can be use in every application while making connection with database to provide loose coupling with the database.

Note: copy "MyClass.java","SingletonCon.java" and "myproperty.properties" inside "default package" folder.

myproperty.properties contains:

```
url=jdbc:mysql://localhost:3306/mydb
user=root
password=root
```

```
ResourceBundle rb=ResourceBundle.getBundle("myproperty");
```

##

MAVEN current version = 3

##

Maven is a powerful project management tool that is based on POM (project object model). It is used for **projects build, dependency and documentation**.

Maven Repository

A maven repository is a directory of where all the project jars, library jar, plugins or any other project specific artifacts are stored with pom.xml file. Maven searches for dependencies in the repositories.
There are 3 types of maven repository:

1) Maven Local Repository
Maven local repository is located in your local system. It is created by the maven when you run any maven command.
2) Maven Central Repository
Maven central repository is located on the web. It has been created by the apache maven community itself.
3) Maven Remote Repository
Maven remote repository is located on the web. Most of libraries can be missing from the central repository such as JBoss library etc, so we need to define remote repository in pom.xml file

Maven searches for the dependencies in the following order:
Local repository then Central repository then Remote repository.
If dependency is not found in these repositories, maven stops processing and throws an error.
Everytime distributing this jar files with clients makes job very tedious.

POM.xml

It contains the names of jar files which are required for functionality of application.
It always resides in the base directory of the project as pom.xml.

Why Maven Required?

Any application required heavy libraries for functionality of application like jar files. If any functionality added in future again, we need to add jar files that makes application heavy. We need to send this jar files to client again and again if any changes made in application file which req new jars.

In case of maven has pom.xml file which contain names of jar files. In this case we have to send only this pom.xml file to the client. where we need to distribute this Application (it's a light application) to client and if we need to make any changes in the required libraries will gives us maintenance advantage.

What Maven do?

Maven's pom.xml helps to download all jar files at .m2/repository location. These jars can be used for different projects. You need not to manually download. But for the very first time you need internet connection to download all jar files. In pom.xml we have to set dependencies like given below or whatever version of jars you want to download. If later point in time you want to change the version (in following case ${spring.version} then you have to change only one place and everywhere get reflected.

```
<dependency>
  <groupId>org.springframework</groupId>
  <artifactId>spring-webmvc</artifactId>
  <version>${spring.version}</version>
</dependency>
```

If you don't use pom.xml then while transferring project you will have to transfer all jar files over the network many times.
Maven give re-usability of jar files in multiple projects. You need not to have deep copy of all jar files in WEB-INF/lib folder.

In very first attempt, any project will download all jar files from internet and will store all those jar files at .m2/repository. Second time in another project, when you try to use other pom.xml file then maven will try to find jars at local (your computer) .m2/repository whatever dependencies defined in your new project's pom.xml if those files are there in .m2/repository then maven will not try to connect to internet to download (because everything is downloaded at .m2/repository). If pom.xml file finds new entry then it will connect to internet and then download at .m2/repository

Some of the imp. Scopes for maven dependencies are:

compile	This scope indicates that dependency is available in classpath of project. It is default scope.
provided	This is much like compile, but indicates you expect the JDK or a container (sever) to provide the dependency at runtime.
runtime	This scope indicates that dependency is not required for compilation, but is required during execution.

##

Servlet

##

javax.servlet
javax.servlet.http

Request – response model

Every client gets the same page. Hence it is called as “static content”.
hear when any user sends the request then web server returns same web page to each client if more clients requesting the same request. Here web server has predefined web pages for each request which returns same page to each client.

Request-Process-Response Model

Here response is based on type of request. Hence every client will see different type of response. Hence it is called as “dynamic content”.

hear when any user sends the request it will be processed by server-side technologies and gets the data from database. server doing the smart work with help of server-side technologies. Ex. Servlet, JSP, ASP, ASP++, CGI.

What is Servlet or JSP?

It's a server-side technology.
since these technologies can be written only in java, they get all the advantages of java language.

Servlet (thread based) VS CGI (process based)

- Servlet creates one thread for each request whereas CGI creates one process for each request.
- Servlet is light wt. and CGI is heavy wt.

Servlet Hierarchy

Servlet (interface) is a parent interface from which GenericServlet is derived. From GenericServlet, HttpServlet is derived.

GenericServlet(defines only some method of Servlet) - to handle any request
HttpServlet (abstract class which defines all incomplete methods of servlet) - to handle only http request.
Nowadays mostly "HttpServlet" is used.
if u want to create a servlet, we need to derive a class from "HttpServlet" class.

servlet life cycle: - since servlet is going to run inside web container, it has life cycle. Following is the life cycle of servlet. Web server maintains the life cycle of servlet/JSP.

1. **public void init()**

- it is called only once.
- it can be used to create database connection
- lookup remote object or any ejb component.

2. **public void service(HttpServletRequest,HttpServletResponse)**

it is called whenever a new request comes to the servlet and then in turn it invokes either "doGet" or "doPost" methods depending upon the type of request.

3. **public void doGet(HttpServletRequest,HttpServletResponse)**

- request for to get the information from server
- by default HTML method is get.

4. **public void doPost(HttpServletRequest,HttpServletResponse)**

- request for addition or updatation

5. **public void destroy()**

- it is called only when we want to shutdown the server or undeploy the application.

Get vs Post

by default, HTML method is get.

1) in case of "get" we can see parameters passed on the address bar.
in "post" we don't see them.
2) in case of "get" limited amount of data can be passed to the server.
in "post" we don't have such limit.
3) Get request should be used to get information from the server.
Post request should be used to add or update information on the server.

Java enabled web server

- Java enables means it contains JRE
- It has only web container
- Web container maintains life cycle of servlet/jsp

Java enabled Application server

- It has web container as well as EJB container
- EJB maintains the life cycle of EJB components

What happen when we say?

http://localhost:8080/myapp/FirstServ , for the first time,

1) web container opens ur DD (web.xml) (deployment descripter)
2) it searches the url-pattern by the name "/FirstServ"
3) from url-pattern it finds out servlet-name .
4) from servlet-name, it finds out servlet-class.
5) once it gets the name of servlet-class, it tries to find out that .class file in "classes" folder of "myapp"
6)now it loads "FirstServ.class"
7) instantiate "FirstServ" by invoking its "public no-arg constructor".
8) it will invoke "init" method.
9) thread is created or retrieved from thread pool.
10) HttpServletRequest and HttpServletResponse are created.
11) service() method is called.
service() method checks whether request is "get" or "post". if it is "get" service() method invokes "doGet()" and if it is "post" it invokes "doPost()"

since we are running servlet directly it is "get" request , so service() method will call "doGet()" which we have overridden.

Imp. note:- when u refresh above link, i.e. for subsequent requests (without makings changes in source code)

9) 10) and 11) steps will be performed.

- Unicode based

```
PrintWriter pw = response.getWriter();
Pw.println(“”);
```

- Byte based

```
ServletOutputStream sos = response.getOutputStream();
sos.println(“”);
```

Standalone application vs server-side Application

- in case of standalone applications when u need to access class or classes from a particular jar file, u have to set the class path.

- in case of server side applications when u need to access class or classes from a particular jar file, class path won't work, u have to copy that jar file inside "lib" folder of your application and restart the server.

IMP SYNTAX

```
Class.forName("com.mysql.jdbc.Driver");
String url="jdbc:mysql://localhost:3306/mydb";
con=DriverManager.getConnection(url,"root","root");
```

In case of server-side application, we need to explicitly invoke driver whereas in standalone application it is not required.

##########

Day - 3

##########

Database connection pool

Connection pool is a cache of database connections maintained so that the connections can be reused when future requests to the database are required. Connection pools are used to enhance the performance of executing commands on a database.

In order to talk to any database, say Oracle, MySQL etc. we open up a database connection, execute the query or update and then close the connection. This opening and closing is resource intensive from CPU, Bandwidth and Memory perspective. This is especially important when there are lot of such transactions are happening in parallel.
So, in order to increase performance, we can create a pool of database connections, then each request can borrow the connection, use it for what it needed and finally return the connection back to the pool. In this way don’t keep on closing or opening the connection and reuse the already established connection from the pool.
The connection pool can be created using custom code or can be build using existing framework such as Apache Database Connection Pooling.

One of the most common issue associated with Pooling is Connection **Pool Leak**. It is important to return the borrowed connection back to the pool else it would cause Connection Pool Leak and eventually the pool would be exhausted.

API difference

without connection pool

DriverManager.getConnection [create connection from the scratch]
con.close [close that connection]

with connection pool

DataSource ds.... [interface from javax.sql package]
ds.getConnection [retrieve connection from the connection pool]
con.close [sends the connection back to the connection pool]

JDBC using properties:

Note: SingletonCon.java and myproperty.properties must be inside "mypack"

Properities file contains following information

driver=com.mysql.jdbc.Driver
url=jdbc:mysql://localhost:3306/mydb
user=root
password=root

What is JNDI ?

JNDI is an API used to access the directory and naming services (i.e. the means by which names are associated with objects). The association of a name with an object is called a binding. A basic example of a naming service is DNS which maps machine names to IP addresses.

A naming service maintains a set of bindings. Bindings relate names to objects. Clients use the naming service to locate objects by name.

There are a number of existing naming services:

COS (Common Object Services) Naming: The naming service for CORBA applications; allows applications to store and access references to CORBA objects.

DNS (Domain Name System): The Internet's naming service; maps people-friendly names (such as www.etcee.com) into computer-friendly IP (Internet Protocol) addresses in dotted-quad notation (207.69.175.36). Interestingly, DNS is a distributed naming service, meaning that the service and its underlying database is spread across many hosts on the Internet.

LDAP (Lightweight Directory Access Protocol): Lightweight Directory Access Protocol (LDAP) is a client/server protocol used to access and manage directory information. It reads and edits directories over IP networks and runs directly over TCP/IP using simple string formats for data transfer.

NIS (Network Information System) and NIS+: Network naming services developed by Sun Microsystems. Both allow users to access files and applications on any host with a single ID and password.

In order to be a naming service, a service must at the very least provide the ability to bind names to objects and to look up objects by name.

JNDI

The **Java Naming and Directory Interface TM (JNDI)** is an application programming interface (API) that provides common way of accessing variety of directories(new, emerging, and already deployed). i.e. client can access any directory service/s using JNDI without worrying about their internal implementation.

Context and InitialContext

Here Context is an interface and InitialContext is an implementation. These two are the part of JNDI API.

Let's take a look at Context's methods:

void bind(String stringName, Object object): Binds a name to an object.

Object lookup(String stringName): Returns the specified object.

What is java_env_comp?

First you need to configure datasource implementation with some name e.g. "mypool" on the Web server or Application Server.

Now when the application server /container starts, the JNDI service will store a reference to the datasource under the java:comp/env/ namespace, so your applications can simply look up the data source implementation for obtaining a connection to the database, to which the data source implementation is pointing to. The name of the datasource implementation can be anything as far as you try to look it up using the same name by which you have configured it.

Context.xml content

Here we are binding name with object for database connection for datasource

```
    <Resource name="jdbc/mypool" auth="Container"
      type="javax.sql.DataSource" driverClassName="com.mysql.jdbc.Driver"
      url="jdbc:mysql://127.0.0.1:3306/mydb"
      username="root" password="root"/>
```

Syntax

```
Context envContext  = new InitialContext();
ds = (DataSource) envContext.lookup("java:/comp/env/jdbc/mypool");
```

Parameters

parameters are used to send some kind of information to the servlet.

types of parameters passed to servlet

1) request: these parameters are limited to only a particular request

2) init or config: these parameters are shared by all the requests to a particular servlet

3) context: shared by all the servlets in that particular context (e.g. myapp)

- request parameters are automatically set in the request.
- request parameter can be read by the method "getParameter()".
- init or config and context parameters can be read by the method "getInitParameter()".
- in order to read request parameter, you need to use "HttpServletRequest".

- in order to read init or config parameter, you need to use "ServletConfig" which is one per servlet.
- in order to read context parameter, you need to use "ServletContext" which is one per context.

Exploring init() method

1) There are two init() methods with HttpServlet

public void init()

public void init(ServletConfig config)

2) Web Container before invoking init method, creates ServletConfig and stores any init parameter set inside it.

3) Web Container invokes "init(ServletConfig config)" method of HttpServlet by passing ServletConfig implementation.

4) HttpServlet's "init(ServletConfig config)" method invokes "init()" method of its own.

To set init parameter inside web.xml file

```
<init-param>
    <param-name>user</param-name>
    <param-value>root</param-value>
</init-param>
```

To set context parameter inside web.xml file

```
<context-param>
    <param-name>database</param-name>
    <param-value>MySql5</param-value>
</context-param>
```

Difference between constructor VS init()

1. both are used for one time task
2. always it is recommended to go for init() because ServletConfig is available inside init() method not inside constructor.

ServletConfig info

When the Container initializes a servlet, it makes a unique ServletConfig for the servlet. The Container "reads" the servlet init parameters from the DD and gives them to the ServletConfig, then passes the ServletConfig to the servlet's init() method.

Your superclass [HttpServlet] includes two versions of init(), one that takes a ServletConfig and a convenient version that's a no-arg. The inherited init(ServletConfig) method calls the no-arg init() method, so the only one you need to override is the no-arg version.
There's no law that stops you from overriding the one that takes a ServletConfig, but if you DO, then you better call super.init(ServletConfig)! But there's really NO reason why you need to override the init(ServletConfig) method, since you can always get your ServletConfig by calling your inherited getServletConfig() method.

conclusion

override

a) init(ServletConfig config) - say super.init(config)
get ServletConfig and ServletContext

or

b) init(ServletConfig config)
no need to say super.init(config)
but save config somehow so that, u can get config and context parameter

or

c) init() -
get ServletConfig and ServletContext automatically from parent

d) do not override any init

still u get ServletConfig and ServletContext automatically from parent

##

Day - 4

##

What if one web component is not sufficient?

Normally when request goes to web component [servlet/jsp] , it processes the request.
what if one web component [servlet/jsp] is not sufficient? i.e. if it is not able to process the request or it is able to process it partially?

in this scenario we have to have another web component which will complete the job of processing the request.
This can be achieved by redirect or forward.

a) redirect

client request goes to "RedirectingServ".
that servlet recommends client to go for another servlet i.e. "RedirectedServ". Based on this recommendation client issues a fresh request to "RedirectedServ" servlet which processes the request.
We write inside of Redirecting class inside doGet

```
response.sendRedirect("RedirectedServ");
```

note: servlet1 of “myapp” can redirect to servlet1 or servlet2 of “myapp1” or it can even redirect to Resource outside the tomcat container.

b) forward

client request goes to "ForwardingServ".
that servlet forwards the same request to "ForwardedServ" servlet which processes the request.

```
RequestDispatcher rd=request.getRequestDispatcher("ForwardedServ");
rd.forward(request,response);
```

note: servlet1 of “myapp” can forward to servlet1 or servlet2 of same context only.

To forward it to another context we need following code:

code inside "ForwardingServ1.java"

```
ServletContext context=getServletContext();
ServletContext context1=context.getContext("/myapp1");
RequestDispatcher rd=context1.getRequestDispatcher("/FirstServ"); // must start with /
rd.forward(request, response);
```

note: servlet1 of “myapp” can forward to servlet1 or servlet2 of “myapp1” but it can-not forward to Resource outside the tomcat container

Difference between "request.getRequestDispatcher()" and "context.getRequestDispatcher()".

request.getRequestDispatcher()

cannot go beyond context.

context.getRequestDispatcher()

can go beyond context. path must start with "/".

Comparative study between Redirect VS forward

Redirect

- can go beyond web container.
- it is slower
- client clearly knows who gives the final response.
- different requests.

Forward

- within the web container.
- it is faster than redirect.
- client is under impression that first servlet/jsp has sent the response.
- same requests.

Include

```
RequestDispatcher rd=request.getRequestDispatcher("IncludedServ");
rd.include(request,response);
```

note: servlet1 of “myapp” can include the response of servlet1 or servlet2 of “myapp1” but it can not include the response of Resource outside the tomcat container.

Making HTTP statefull.

Http is a stateless protocol.
client requests a web resource (servlet/jsp).
when client requests a web resource again or to another web resource, for container it is a different client altogether.
i.e. container does not remember client after the response is given to client back.
in a practical scenario mostly we have a requirement where container should remember client. e.g. shopping cart example. i.e. we have a requirement of making Http stateful.

in java we can achieve it using:


HiddenServ2.java


HiddenServ1.java

a) hidden fields:

- here request values go from srevlet1 to servlet2
- we cannot store java objects
- we can see the hidden components via view -source


CustomCookieServ.java

b) custom cookies

- Cookies are used to store information.
- They can store only textual information (no java objects)
- Cookies can be rejected by client browser (so that's the risk of using Cookies in order to store information)
- Cookies are always added inside response and retrieved from request.

c) HttpSession

1. HttpSession interface is used to create session in java.

```
HttpSession session=request.getSession();
```

or

```
HttpSession session=request.getSession(true);
```

above statement will create a session if not exists or retrieve existing session

2. how to add values in session
   session.setAttribute(String,Object)

3. how to retrieve values from session
   session.getAttribute(String)

```
HttpSession session=request.getSession(false);
```

above statement will retrieve existing session only. if session does not exists, it will return null.

What happens when session is not there?

```
HttpSession session=request.getSession();
        or
HttpSession session=request.getSession(true);
session.setAttribute("book","CompleteReference");
```

Container will

a) create session object
b) store attribute/s
c) generate unique session id
d) create cookie
e) store session id inside cookie
f) add cookie inside response.

What happens when session is retrieved?

```
HttpSession session=request.getSession(false);
```

container will

a) retrieve cookie from request
b) retrieve session id from cookie.
c) try to match session id with the session object available on server.

```
if it matches
        return session object
else
        return null
```

SessionServ1 - doGet

```
HttpSession session=request.getSession();
session.setAttribute("book","Complete_Reference");

PrintWriter pw=response.getWriter();
pw.println("session created");
```

SessionServ2 - doGet

```
PrintWriter pw=response.getWriter();
HttpSession session=request.getSession(false);
if(session!=null)
{
        pw.println(session.getAttribute("book"));
}
```

c) URLRewriting

URLRewriting is a technique where session id can be added at the end of URL.

Why it's required?

it's required in case client does not accept cookies. Because in that case session id cannot come through cookies, they will come through URL.

```
 out.println("<html><body>");
out.println("<a href=\"" +res.encodeURL("Second")+ "\">click me for second</a>");
out.println("</body></html>");
```

How it works?

Normally container stores session id inside cookie.
when we use following statement:

```
out.println("<a href=\"" +res.encodeURL("Second")+ "\">click Go to second servlet</a>");
```

with this statement now container will not only add session id to the cookie but also append it at the end of your URL.

now even if client does not accept cookie, session id will go to server along with URL.
When the user clicks that "enhanced" link, the request goes to the container with that extra bit on the end (http://localhost:8080/myapp2/Second;jsessionid=4721261C3AF63C8625475DD090A7CC5A), and the Container simply strips off the extra part of the request URL (jsessionid=4721261C3AF63C8625475DD090A7CC5A) and uses to find the matching session.

URL Rewriting is the way to ensure that session mechanism will always works irrespective of whether client browser accepts or rejects cookies.

To kill session use following API

```
session.invalidate();
```

parameters vs attributes

a) parameters are strings, whereas attributes can be java objects.
b) parameters can be set in DD only (except request parameter which is set implicitly). attributes are set programmatically only.
c) **parameters** are of
request, config and context

attributes in servlet

a) request: request attributes can be accessible only till the request is available.
b) session: session attributes are accessible till the session is alive.
c) context: context attributes can be accessible to all the servlets/jsps of the context (e.g. myapp).

##

DAY 5

##

Filters

Filters are pluggable components i.e. they can be easily plugged in or out (using DD).
The main advantage of filters is maintenance. i.e., you don't have to make changes inside your servlet for every new requirement/s given by client.
There are 2 types of filters:

request filters: - they are called before request goes to servlet.
response filters: - they are called after servlet execution but before response goes to client.

how do you create filter?
derive a class from "Filter" interface and define 3 methods of it:

public void init(FilterConfig)
public void doFilter(ServletRequest,ServletResponse,FilterChain)
public void destroy()

here "doFilter()" is the method where u will write a logic for which filter has been created.

FilterChain represents the filter chain i.e. number of filters to be invoked along with Servlet or JSP.
Imp: - The last component inside filter chain is always a servlet or jsp.

when filter invokes "chain.doFilter(req,res)" it indicates web container that
I have done my job, next component (another filter/servlet/jsp) should be invoked.

What is FilterChain?
FilterChain is the thing that knows what comes next. We already mentioned that the filters (not to mention the servlet) shouldn't know anything about the other filters involved in the request... but someone needs to know the order, and that someone is the FilterChain, driven by the filter elements you specify in the DD.

ThreadSafety

Thread safety and servlet

service() or doGet() or doPost()- are always called by threads and all the threads share same instance of servlet class.
you need to take care of avoiding race condition inside "service() or doGet() or doPost()" methods.

how will you do that?
synchronized service() or doGet() or doPost() is one of the ways.
initially we had interface "SingleThreadModel" in JavaEE.
whenever programmer wanted to make sure that servlet should be executed by one thread at a time, he used to say

```
public class MyServ exends HttpServlet implements SingleThreadModel
```

SingleThreadModel is a marker interface which ensures that your servlet will be executed by one thread at a time.
Nowadays idea of implementing SingleThreadModel has been deprecated.
but it's not recommended at all as it will definitely degrade the servlet performance.

at the most u need

```
service() or doGet() or doPost()
{
        synchronized(this)
        {
                critical code
        }
}
```

Note: try to avoid member variables inside servlet.
if at all you need access them inside synchronized block.
you need not worry about request and response as they both are thread-safe because they both are local.

Load Balancing

Load balancing refers to efficiently distributing incoming network traffic across a group of backend servers, also known as a *server farm* or *server pool*.
Modern high-traffic websites must serve thousands of concurrent requests from users or clients and return the correct text, images, videos, or application data, all in a fast and reliable manner. In order to achieve this more than one server need to be added.
A load balancer acts as the “traffic cop” sitting in front of your servers and routing client requests across all servers capable of fulfilling those requests in a manner that maximizes speed and capacity utilization and ensures that no one server is overworked, which could degrade performance. If a single server goes down, the load balancer redirects traffic to the remaining online servers. When a new server is added to the server group, the load balancer automatically starts to send requests to it.

In case of Load Balancing

Only HttpSession objects (and their attributes) move from one VM to another.
There is one
Servlet instance per VM, there is one ServletContext per VM. There is one ServletConfig per servlet, per VM. But there is only one HttpSession object for a given session ID per web app, regardless of how many VM’s the app is distributed across.
Each servlet has its own ServletConfig, and both servlets in the web app share a ServletContext.
Everything except the HttpSession is duplicated on the other VM.

POJO (plain old java object)

if we need business logic inside servlet, can we write that business logic inside servlet?
yes. but that will have two drawbacks:

a) maintenance
b) no reusability

if we write business logic separately
it will have two advantages
a) maintenance
b) reusability

How to Upload File?

For uploading a file to the server, we have **used MultipartRequest class provided by oreilly**. For using this class, you must have maven dependency for **cos.jar** file.

```
import com.oreilly.servlet.MultipartRequest;
```

command to add file

```
MultipartRequest m=new MultipartRequest(request,"d:/new");
```

Load on Start up

what to load?
Servlet

what does that mean?
it means load servlet/s when application starts.

So, when exactly servlet gets loaded by default?
servlet gets loaded only when first request comes. Not before that.

so load on startup means you would like a particular servlet/s to get loaded when application starts i.e. even before first request.

why do you want that?

It's because that servlet/s performes some very very imp task from the point of view of entire web application. That task needs to performed even before first request comes.

How to do it?

Depending on your application design, you may want to specify that one or more servlets be loaded during the startup of the web application. Furthermore, these servlets may need to be loaded in a particular order. The optional
<load-on-startup> element is used for this purpose.
Loading servlets on startup

```
<web-app>
<servlet>
<servlet-name>MyServ3</servlet-name>
<servlet-class>MyServ3</servlet-class>
<load-on-startup>1</load-on-startup>
</servlet>
…..
</web-app>
```

##

DAY 6 – Java Server Page (JSP)

##

JSP:

- Server-side technology in java.
- middle-tier technology like Servlet.
- Advanced version of Servlet.

Difference bet'n JSP and Servlet

a) Jsp allows you to mix static html and elements which can be processed during runtime. Whereas in case of Servlet, html can be generated dynamically.
b) jsp is faster than servlet.

Does that mean Servlet has become outdated?

No – in case of MVC architecture Servlet has a very Imp. Role to play as a Controller.
JSP acts as a View inside MVC application as it has got lots of powerful built-in tags.

In order to create Jsp page, you have to save file by .jsp extension.
When jsp is run on web server, it is converted into Servlet.

When jsp is requested for the first time :

1. jsp is converted into .java (servlet)
2. servlet is compiled by a web container.
3. .class is loaded
4. Instantiated using “public no-arg constructor”.
5. Initialized by calling “init() “ method.
6. Thread is created or retrieved from a thread pool.
7. request and response are created
8. service() method is called.

For subsequent requests to same jsp , provided jsp source code is not modified :

6. Thread is created or retrieved from a thread pool.
7. request and response are created
8. service() method is called.

LifeCycle methods of Jsp

```
public void jspInit(){}
public void _jspService(HttpServletRequest request, HttpServletResponse response) {}
public void jspDestroy(){}
```

out of above methods , if you want to override, you can override only “jspInit()” and “jspDestroy()”. You can not override “_jspService()”. But you can write a code in your jsp, which can go to generated “_jspService()” method of servlet.

Implicit objects in jsp

API	Implicit Object
JSPWriter	out
HttpServletRequest	request
HttpServletResponse	response
HttpSession	session
ServletContext	application
ServletConfig	config
JspException	exception (available to only error pages.)
PageContext	pageContext
Object	page

Jsp can contain

1. static html
2. elements which can be processed on server

elements which can be processed on server:

1. jsp declaration
2. jsp expression
3. jsp scriptlet
4. jsp directive
5. jsp standard action

jsp declaration :- <%! %>

it is used to declare instance variables and define member functions.

Jsp expression :- <%= %>

It is used to write java expressions.

You can even call a method in jsp expression, provided it has some return type other than “void”.

Whatever u write in jsp expression, it goes directly to out.println() method of generated servlet.

Jsp scriptlet:- <% %>

It is used to write pure java code because whatever u write in the scriptlet, that goes as it is in the generated servlet’s service method.

Jsp directives:- <%@ directive name attribute/s=”value” %>

Directives are the special messages which jsp developer can convey to the web container.

Three types of directives are there.

a) **page directive**

page directive tells web container about the requirements of jsp page . e.g. if you want to import any package etc.

b) **include directive**

include directive tells web container to include the content of other page inside current page at the translation time.

c) **taglib directive**

Taglib directive is used to create custom tags.

Jsp standard actions :-

<jsp:standard action attribute/s=”value” />

They are used to perform some java action behind a particular tag.

Following are some of the standard actions

a) **useBean**
b) **param**
c) **include**
d) **forward**

isThreadSafe info

one of the attributes of "page" directive is isThreadSafe.

It has by default value "true". That means your generated servlet is thread-safe.

if you set the value of "isThreadSafe" to "false". it means your generated servlet is not thread-safe and that also means that web container will make your generated servlet implement "SingleThreadModel"

What is SingleThreadModel interface?

initially when programmer didn't want to make servlet accessible from multiple threads, he used to make his servlet implement "SingleThreadModel" (it is a marker interface). But now this interface has become obsolete.

page directive's two more attributes are

errorPage and isErrorPage

errorPage is used to inform web container about error page you are going to use for this jsp.

isErrorPage is used to make jsp as error page.

isErrorPage is "false" by default. That is JSP page is not an error page by default.
in order to make JSP as an error page , inside it you have to say isErrorPage="true"
"exception" implicit object can be accessed inside error jsp only.

Different bet'n include directive and include standard action

- include directive is used to include the content of other page inside the current page at translation time (when jsp is converted to java).
- include standard action is used to include the response of other jsp inside the current jsp.

different scopes in jsp
there are 4 scopes in jsp
page, request, session and application.

	scope	object to represent
1.	page	pageContext
2.	request	request
3.	session	session
4.	application	application

##

DAY 7 – Java Server Page (JSP)

##

useBin (standard action)

useBean standard action is used to invoke or access "POJO class" or "JavaBean" from a JSP.
<jsp:useBean id="ob1" class="beanpack.MyClass"/>
Mainly used to instantiate a pojo class.
when container encounters above statement, it will search "MyClass.class" inside "beanpack".

if found it will load the class.
instantiate class using no-arg constructor and the instance will be referred by "ob1" reference.

useBean has 3 attributes

1. id
2. class
3. scope - page/request/session/application (by default scope is page)

you can access "POJO class" or "JavaBean" from JSP using java code.
e.g.

```
package pack1;
public class MyClass
{
	public void disp()
	{
		//code
	}
}
```

JSP code to invoke "disp"
Caller.jsp

```
<%@page import="pack1.*"%>

<%
MyClass m1=new MyClass();

m1.disp();
%>
```

Even though above code works without any problem, it is not recommended to use java code inside JSP (JSP should contain only tags and not java code). So that's why we have to have some mechanism whereby without using Java we should be able to instantiate "POJO class" or "JavaBean" and invoke its method. This can be achieved with the help of "useBean" standard action.

syntax for "useBean" standard action

<jsp:useBean id=" " scope=" " class=" "/>

here "class" means .class file of POJO (has to be a part of package)

"id" means a reference which refers to the object of java class (POJO).

"scope" means scope of the above-mentioned object.
	"scope" attribute is optional.
"scope" can be "page", "request", "session" or "application".

When we don't mention a scope, it is by default "page".

EL (Expression Language)

required to get the values from various attributes as well as parameters. it is null friendly that is if it encounters null, it will not throw NullPointerException.

syntax
${expression}

EL provides various implicit objects

- pageScope- to read page attributes
- requestScope - to read request attributes
- sessionScope- to read session attributes
- applicationScope- to read context attributes

- param- to read request parameters
- paramValues- to read request parameters returning more than one value
- cookie- to read cookies
- initParam- to read context parameters

All the EL implicit objects are implicitly maps.

Jvm standard tag library (JSTL)

taglib directive is used to access tags (in-built or user defined)
taglib directive has got two attributes

1. **uri** - is a location/address where entire information about that tag is present.
2. **prefix** - it's like a namespace and is used to access the tag.

##

DAY 8 – Java Server Page (JSP)

##

User Defined Tags

We need 3 files

1. JSP (where we use that tag) - WEB-INF
2. Java code (which contain main logic behind particular tag) - tagpack
3. Tld file - web content

What happen when this command encounters?

```
<%@taglib prefix="abc" uri="/WEB-INF/hello.tld"%>
<abc:first/>
```

how does it work?

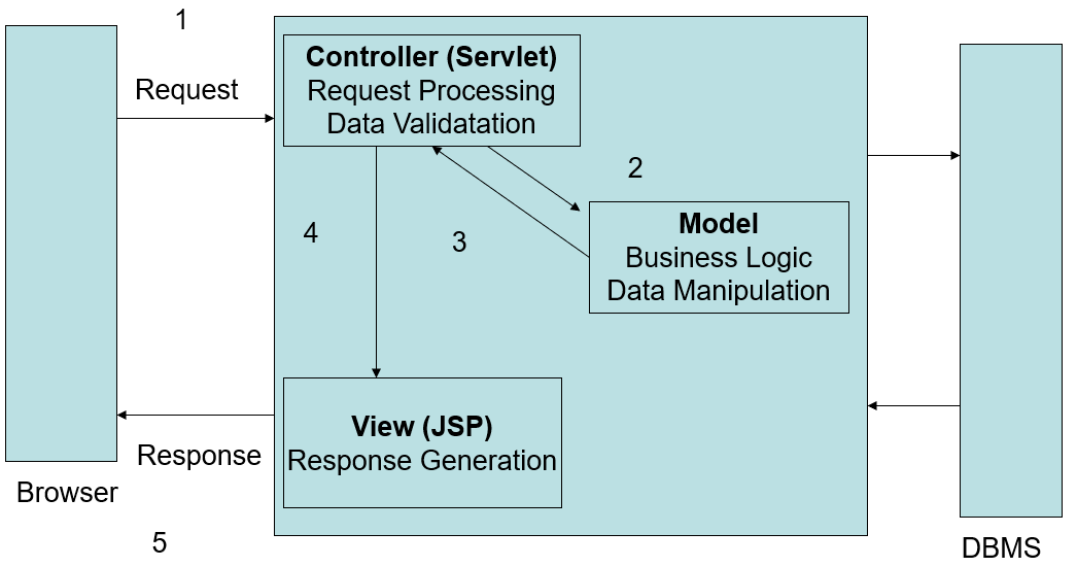
when container encounters "<abc:first/>" tag for the first time, it will
a) go to "hello.tld"
b) check which java class is designed for this tag (TagCode1.class)
c) it will load the class
d) instantiate the class
e) invoke "doTag()" method inside "TagCode1".

when container encounters "<abc:first/>" tag for the subsequent time,
a) instantiate the class
b) invoke "doTag()" method inside "TagCode1".

MVC (Model-View-controller)

Model – pojo/bean/helper classes
View- JSP
Controller – Servlet

In this Architecture, a central servlet, known as the ***Controller***, receives all requests for the application. The controller then processes the request and works with the *Model* to prepare any data needed by the *View* (which is usually JSP) and forwards the data to a JSP. The JSP then uses data prepared by the Controller to generate a response to the browser. In this architecture, the business and presentation logic are separated from each other. This separation mainly provides excellent reuse of code



Controller

- At the core of the MVC architecture are the controller components.
- The Controller is typically a servlet that receives requests for the application and manages the flow of data between the Model layer and the View layer.
- Thus, it controls the way that the Model and View layers interact.
- Controller often uses helper classes for delegating control over specific requests or processes.

Model

- In the MVC architecture, model components provide a service used by an application.
- This way, controller components don't unnecessarily embed code for manipulating an application's data. Instead, they communicate with the model components that perform the data access and manipulation.
- Thus, the model component provides the business logic.
- Model components come in many different forms and can be as simple as a basic Java bean or as complex as Enterprise JavaBeans (EJBs) or Web Services.

View

- View components are used in the MVC architecture to generate the response to the browser.
- Thus, a view component provides what the user sees.
- Often times the view components are simple JSPs or HTML pages.

##

DAY 9 - Hibernate

##

ORM (Object-Relational mapping)

- The object-oriented model use classes whereas the relational databases use tables.
- Getting the data and associations from objects into relational table structure and vice-versa requires a lot of tedious programming due to the difference between the two. This difference is called The Impedance Mismatch.
- Developers need something simple to covert from one to the other automatically.
- Bridging the gap between the object model and the relational model is known as Object –Relational Mapping. (ORM)

Advantages

- Eliminates writing SQL to load and persist object state, leaving the developer free to concentrate on the business logic.
- Enables creating an appropriate DOMAIN model, after which, the developer only needs to think in terms of objects, rather than tables, rows and columns.
- Reduces dependence on database specific SQL and thus provides portability across databases.
- Reduces more than 30% of the amount of Java Code spec that needs to be written by adopting an ORM

Different ORM tools available in market:

1. Hibernate

2. TopLink
3. EclipseLink
4. OpenJPA

What is hibernate?

Hibernate is java-based middleware designed to complete the Object Relational (O/R) mapping model. One of the most complex parts of many O/R mapping mechanisms- writing SQL code- can be simplified significantly with Hibernate
Hibernate not only takes care of the mapping from Java classes to database tables (and from Java data types to SQL data types), but also provides data query and retrieval facilities and can significantly reduce development time otherwise spent with manual data handling in SQL and JDBC.

What is Hibernate dialect?

hibernate.dialect
This property makes Hibernate generate the appropriate SQL for the chosen database.

Relationship flow

1. **Hibernate.cgf.xml**

URL, user, password, entity class/classes

2. **Session Factory**

Creates various session objects

3. **Session**

[save, load, query, update, delete etc.]
encapsulates database connection
provides "First-Level cache"

4. **Transaction**

commit/rollback

Session Factory

The main contract here is the creation of Session instances. Usually, an application has a single SessionFactory instance and threads servicing client requests obtain Session instances from this factory. The internal state of a SessionFactory is immutable. Once it is created this internal state is set. This internal state includes all of the metadata about Object/Relational Mapping.
Implementors must be thread safe.

- SessionFactory is a factory to hibernate Session objects.
- SessionFactory is often built during start-up and used by application code to get session object.
- Java JEE application has just one SessionFactory, and individual threads, which are servicing client's request obtain hibernate Session instances from this factory.
- The internal state of a SessionFactory is immutable.
- Most problems with concurrency occur due to sharing of objects with mutable state.
- Once the object is immutable, its internal state is settled on creation and cannot be changed. So many threads can access it concurrently and request for sessions. hence the SessionFactory is thread-safe.

- Session represents a small unit of work in Hibernate, they maintain a connection with the database.
- Session is the primary interface for the persistence service.
- session opens a single database connection when it is created, and holds onto it until the session is closed.
- Every object that is loaded by Hibernate from the database is associated with the session, allowing Hibernate to automatically persist objects that are modified, and allowing Hibernate to implement functionality such as lazy-loading.
- A Session instance is serializable if its persistent classes are serializable.
- The Session object should only by used by a single Thread.

Imp for setter method and auto generation of id

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
```

even after we mention this for auto generation of id, "setter" method of id is compulsory because hibernate will generate id and then assigned to the corresponding instance through setter methods.

Annotation @id importance

```
@Id
@GeneratedValue(strategy=GenerationType.AUTO)
@Column(name = "id", nullable = false)
public int getId() {
        return this.id;}
```

@Id attribute must be there on getter method and not on setter method.
if u give it on setter method u get exception.

@Id can be either with "getter" method or with variable declaration.

other attributes e.g., @Columns can have on either getter or setter methods.


Transient_Persistent_ Detached Info.doc

Kinds of Objects

transient: never persistent, not associated with any Session
persistent: associated with a unique Session
detached: previously persistent, not associated with any Session

Difference between Transient and Detached States:

Transient objects do not have association with the databases and session objects. They are simple objects and not persisted to the database. Once the last reference is lost, that means the object itself is lost. And of course, garbage collected. The commits and rollbacks will have no effects on these objects. They can become into persistent objects through the save method calls of Session object.

The detached object has corresponding entries in the database. These are persistent and not connected to the Session object. These objects have the synchronized data with the database when the session was closed. Since then, the change may be done in the database which makes this object stale. The detached object can be reattached after certain time to another object in order to become persistent again.

Refreshing objects using Refresh()

Hibernate provides a method called refresh() which when invoked refreshes the persistent object from their database representation. The most ideal scenario where a refresh would be used is to undo changes that were made to a persistent object in memory. In such a situation, refresh() when called, reloads the in-memory objects from their database representation.

The update() method

If an object is modified after the session is closed, it is not persisted in the database. When the session is closed, the object is detached. It needs to be re-associated with a new Session by calling the update() method. The update() method forces an update to the persistent state of the object in the database, scheduling an SQL Update.
it does not matter , if the object is modified before or after it is passed to update().
Update() when invoked re-associates the detached object to the new Session [session2] and informs Hibernate to treat that object as dirty. Hibernate propagates the modifications to the database when "trans.commit()" is invoked.

Dirty Checking

Automatic dirty checking is a feature that saves us the effort of explicitly asking Hibernate to update the database when we modify the state of an object inside a transaction.
In dirty checking, hibernate automatically detects whether an object is modified (or) not and need to be updated. As long as the object is in persistent state i.e., bound to a particular Session(org.hibernate.Session). Hibernate monitors any changes to the objects and executes sql.

Note:- For dirty checking to work, the object must exist in cache.

##

DAY -10 HQL (hibernate query language)

##

HQL is much like SQL. The only difference is that SQL operates on relations, whereas HQL operates on objects. HQL is an extremely powerful query language and is considered the most preferred way.

Comparative study of queries

```
Query<?> query=session.createQuery("from Person");
```

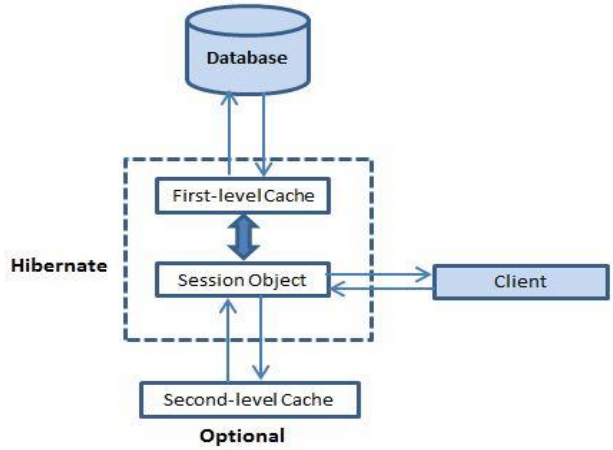
```
List<Person> mylist1=(List<Person>) query.list();

query=session.createQuery("select c.name from Person c");
List<String> mylist2=(List<String>) query.list();

query=session.createQuery("select c.name,c.age from Person c");
List<Object[]>mylist3=(List<Object[]>) query.list();
```

What is Chaching?

One of the primary concerns of mappings between a database and your Java application is performance. Caching is all about application performance optimization and it sits between your application and the database to avoid the number of database hits as many as possible to give a better performance for performance critical applications.



Caching is widely used for optimizing database applications. A cache is designed to reduce traffic between your application and the database by conserving data already loaded from the database. Database access is necessary only when retrieving data that is not currently available in the cache. The application may need to empty (invalidate) the cache from time to time if the database is updated or modified in some way, because it has no way of knowing whether the cache is up to date.

Hibernate Caching

Hibernate uses three different caches for objects:

1. first-level cache = belongs to session
2. second-level cache = belongs to session factory
3. Query cache = belongs to query

First-Level cache

compulsory for an application, because it is related to hibernate session. it's available as long we don't say "session.evict()" or "session.close()"

Second-Level cache

Need Depndency :

```
<dependency>
   <groupId>org.hibernate</groupId>
```

```
    <artifactId>hibernate-ehcache</artifactId>
    <version>5.3.7.Final</version>
</dependency>
```

not compulsory. If application would like to have Second-Level cache, it has to get it explicitly. Two points are imp. here:

1) Second Level cache is common for all the hibernate sessions. [it's because session level cache belongs to SessionFactory]
2) how the data is stored inside Second Level Cache?

key = value
primary key field = [set of other members]

need to use this annotation

@Cache(usage = CacheConcurrencyStrategy.READ_ONLY)
or
@Cache(usage = CacheConcurrencyStrategy.READ_WRITE)

Query cache

it is required to fire same query again and again with the same parameters.
There are two points imp:
1) how does quey cache store the data?

key = value
query + parameter/s = result which comes from database

2) it requires Second-Level cache.

What is Dialect?

The dialect specifies the type of database used in hibernate so that hibernate generate appropriate type of SQL statements. For connecting any hibernate application with the database, it is required to provide the configuration of SQL dialect.

##

DAY -11

##

Associations:

Need to add both entities inside .cfg file

1. one to one

```
@OneToOne(cascade = CascadeType.ALL)
        public Account getAcc() {
                return acc;
        }
```

2. one to many

```
@OneToMany(cascade = CascadeType.ALL)
@JoinColumn(name = "sid", referencedColumnName="STUDENT_ID")
public Set<Phone> getStudentPhoneNumbers() {
return this.studentPhoneNumbers;
}
```

above annotation will create additional column inside "Phone" table as "sid" which will store students id.

3. One to many delete

```
session.delete(s1);
```

to delete all records from database related to s1 object

Lazy and Eager fetch

Note: By default, hibernate5 has got "Lazy Fetching Strategy".

Eager Fetch

```
@OneToMany(cascade = CascadeType.ALL,fetch=FetchType.EAGER)
```

In case of Eager +get fetch hibernate will raise queries (Fat queries) for loading parent as well child data even if we want only data of parent. Eager fetch will give N+1 problem.

Lazy fetch

```
@OneToMany(cascade = CascadeType.ALL)  //by default lazy fetch
```

Hibernate now can "lazy-load" the children, which means that it does not actually load all the children (Employees) when loading the parent (Company). Instead, it loads them when requested to do so. This is known as “Lazy_Load”. If we closed the session during trying to fetch child data it will give **LazyInitializationException**

What is hibernate proxy?

A proxy is a subclass implemented at runtime. Hibernate creates a proxy (a subclass of the class being fetched) instead of querying the database directly, and this proxy will load the "real" object from the database whenever one of its methods is called.

By default, hibernate creates a proxy for each of the class you map in mapping file. This class contain the code to invoke JDBC. This class is created by hibernate using CGLIB.

Proxies are created dynamically by subclassing your object at runtime. The subclass has all the methods of the parent, and when any of the methods are accessed, the proxy loads up the real object from the DB and calls the method for you.

Mapping of classes can be made into a proxy instead of a table. A proxy is returned when actually a load is called on a session. The proxy contains actual method to load the data. The proxy is created by default by Hibernate, for mapping a class to a file. The code to invoke Jdbc is contained in this class.

N+1 problem?

While giving command for getting list of companies only hibernate will give whole object (company + employees) (one to many relationship). hibernate will raise queries for company as well as for each employee.

Hibernate writes fat queries unnecessarily. It’s called N+1 problem.

Solution for N+1 problem is to make lazy initialization as “true”.

Get vs Load

Here are few differences between get and load method in Hibernate.

1. Behavior when Object is not found in Session Cache
get method of Hibernate Session class returns null if object is not found in cache as well as on database while load() method throws ObjectNotFoundException if object is not found on cache as well as on database but never return null.

2. Database hit
Get method always hit database while load() method may not always hit the database.

3. Proxy
Get method never returns a proxy, it either returns null or fully initialized Object, while load() method may return proxy, which is the object with ID but without initializing other properties, which is lazily initialized. If you are just using returned object for creating relationship and only need Id then load() is the way to go.

4. Performance
get method will return a completely initialized object if Object is not on the cache but exists on Database, which may involve multiple round-trips to database based upon object relational mappings while load() method of Hibernate can return a proxy which can be initialized on demand (lazy initialization) when a non identifier method is accessed. Due to above reason use of load method will result in slightly better performance, but there is a possibility that proxy object will throw ObjectNotFoundException later if corresponding row doesn't exists in database.
.

When to use Session get() and load() in Hibernate

1. Use get method to determine if an instance exists or not because it can return null if instance doesn't exists in cache and database and use load method to retrieve instance only if you think that instance should exists and non availability is an error condition.

2. get() method could suffer performance penalty if only identifier method like getId() is accessed. So consider using load method if your code doesn't access any method other than identifier or you are OK with lazy initialization of object, if persistent object is not in Session Cache because load() can return proxy.

Hibernate Named Query

The hibernate named query is way to use any query by some meaningful name. It is like using alias names. The Hibernate framework provides the concept of named queries so that application programmer need not to scatter queries to all the java code.
It will give maintenance advantage.

```
@NamedQueries(
        {
          @NamedQuery(
          name = "findEmployeeByName",
          query = "from Employee e where e.name = :name"
          ),
```

```
        @NamedQuery(
            name = "findEmployeeBySalary",
            query = "from Employee e where e.salary>:amount"
           )
      }
    )
```

##

DAY -12

##

What is DAO (Data Access Object)?

DAO (Data Access Object) = design pattern
DAO decouples business layer from a particular persistence mechanism [file, database etc.,]. So, business layer can concentrate only on business logic.

HCQL (Hibernate criterion Query Language)

import javax.persistence.criteria.CriteriaQuery;
The Hibernate Criteria Query Language (HCQL) is used to fetch the records based on the specific criteria.

HCQL provides methods to add different criteria on a query.

It not only enables us to write queries without doing raw SQL, but also gives us some Object-Oriented control over the queries, which is one of the main features of Hibernate. The Criteria API allows us to build up a criteria query object programmatically, where we can apply different kinds of filtration rules and logical conditions.

```
CriteriaBuilder cb=session.getCriteriaBuilder();
CriteriaQuery<Employee> criteria=cb.createQuery(Employee.class);
criteria.from(Employee.class);
Query<?> q=session.createQuery(criteria);
List<?> results = q.getResultList();
System.out.println(results);


System.out.println("find out only desig");
CriteriaQuery<String>criteria1=cb.createQuery(String.class);
Root<Employee> root = criteria1.from(Employee.class);
criteria1.select(root.get("desig"));
q=session.createQuery(criteria1);
 results = q.getResultList();
System.out.println(results);
```

##

DAY -12 (spring)

##

Spring has feature of dependency injection

In java we do:

1. Pojo based programming
2. jdbc
3. ORM tools
4. Enterprise Applications [RMI, EJB]
5. based Applications

Spring framework was released by Rod Johnson in 2005 which provides a great support for all the above-mentioned Applications.

What is Framework?

- Framework consist of classes and interfaces.
- These classes and interfaces can work together in order to solve a particular problem.
- Framework components are reusable.

A framework is similar to an application programming interface (API), though technically a framework includes an API. As the name suggests, a framework serves as a foundation for programming, while an API provides access to the elements supported by the framework. A framework may also include code libraries, a compiler, and other programs used in the software development process.

Core parts of spring framework:

- ✓ POJO: Development with Plain Old Java Objects
- ✓ Dependency Injection [DI]: Loose coupling, Classes are no longer required to be coupled to any specific implementation, which is the key benefit of DI, Loose Coupling.
- ✓ Aspect-Oriented Programming [AOP]: Declarative programming
- ✓ Templates: code reduction

Different types of spring bean scopes

In the spring bean configurations, bean attribute called 'scope' defines what kind of object has to created and returned. There are 5 types of bean scopes available, they are:

1) singleton: Returns a single bean instance per Spring IoC container.

2) prototype: Returns a new bean instance each time when requested.

3) request: Returns a single instance for every HTTP request call.

4) session: Returns a single instance for every HTTP session.

5) global session: global session scope is equal as session scope on portlet-based web applications.

If no bean scope is specified in bean configuration file, then it will be by default 'singleton'.

In Singleton scope ioc container is create object of class in eager manner means when we make object of ioc container while in prototype scope ioc container creates object in lazy manner means when we say getBean().

Autowiring in Spring

Autowiring feature of spring framework enables you to inject the object dependency implicitly. It internally uses setter or constructor injection.
Autowiring can't be used to inject primitive and string values. It works with reference only.

Advantage of Autowiring

It requires the less code because we don't need to write the code to inject the dependency explicitly.

Disadvantage of Autowiring

It can't be used for primitive and string values.

XML Comparisons

Autowiring Modes

1. **no** - It is the default autowiring mode. It means no autowiring bydefault.
2. **byname** - The byName mode injects the object dependency according to name of the bean. In such case, property name and bean name must be same. It internally calls setter method.
3. **byType** - The byType mode injects the object dependency according to type. So property name and bean name can be different. It internally calls setter method.
4. **Constructor** - The constructor mode injects the dependency by calling the constructor of the class. It calls the constructor having large number of parameters.

----------**byname**----------

```
 <bean id="ob" class="mypack.MyClass1"></bean>
<bean id="x" class="mypack.MyClass2" autowire="byName"></bean>
</beans>
```

condition= in myclass2 ref of myclass1 should have the same name as in xml i.e., "ob"
then we take benefit of autowire = "byname". it will automatocally call setter method of myclass2

----------**bytype**----------

<bean id="ob1" class="mypack.MyClass1"></bean>
<bean id="x" class="mypack.MyClass2" autowire="byType"></bean>
</beans>

condition= in myclass2 ref of myclass1 should have the type as in xml i.e., class="mypack.MyClass1"
here we dont have restrictions as like byName . hare we are taking benefit of autowire = "byType".
it will automatocally call setter method of myclass2

----------**constructor**----------

<bean id="ob" class="mypack.MyClass1"></bean>

<bean id="x" class="mypack.MyClass2" autowire="constructor"></bean>
</beans>

no need to write constructor injection below bean myclass1 - constructor-arg ref="a2"

AutoScanning

we don't have to provide "bean" tags inside spring bean configuration file.
IOC container will automatically find out about beans based on @Component annotation which u need to apply on the class which u treat as a bean and instantiates it.

when u use <context:component-scan> u don't use "<bean>" tag in spring bean configuration file, but u must mention
@Component for each class, u want to treat it as a bean.

<context:component-scan base-package="mypack"/>
it means automatically scan all the classes of mypack where @Component annotation has been applied.
i.e. treat those classes as beans.

@AutoWired on setter method

```
@Autowired
public void setAuthor(Author author) {
    this.author = author;
}
```

here @Autowired annotation tells IOC container to invoke setAuthor method by considering "byType" autowiring.

3 important Annotations

1. @Component or @Component(value="accountbean")
2. @Autowired
3. @Qualifier("caccount")

##

DAY -13 (spring-AOP, Templates)

##

Spring DAO supports some templates e.g. JdbcTemplate, HibernateTemplate etc.

The main purpose of template is Boilerplate reduction.
another point is all these templates hit database, in case if there is any SQLException(checked exception) they convert it into "DataAccessException" (unchecked exception) and throw it to the client application. Advantage here is client need not handle or declare "DataAccessException" as it is unchecked exception.

Hibernate Template

HibernateTemplate is the class of org.springframework.orm.hibernate3.
HibernateTemplate provides the integration of hibernate and spring. Spring manages database connection DML, DDL etc commands by itself. HibernateTemplate has the methods like save, bulkUpdate, delete, evict etc.

How to pass named parameter in case of HibernateTemplate

```
public List<?> getDepartments(String location) {
                return template.findByNamedParam("from Dept where loc=:str", "str", location);
        }
```

##

DAY -14 (spring-MVC)

##

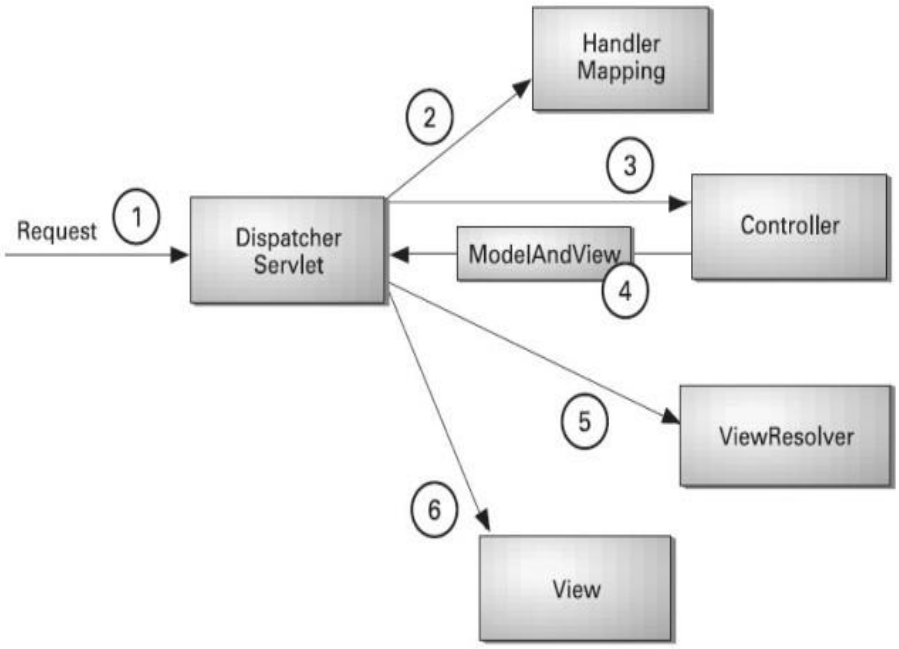
Why MVC framework?

core MVC solves most of the problems u have faced in Model 1 architecture. i.e. high maintenance and no reusability. But when volume of application is large, i.e. u have so many forms, javabeans and view components (jsps), your controller will become overburdened.
moreover if in future application workflow is changed, u will face the problem of maintenance in controller.

Solution:
MVC-based Frameworks: struts, springMVC, JSF

How these frameworks will solve your problem?
They will provide u a Controller and that Controller will work according to Application Workflow u have mentioned inside some xml configuration file or through annotation.

MVC Architecture



##

DAY -15 (validation, service layer, JPA repository)

##

Validations

for uname

```
@NotEmpty(message="name can not be empty")
	@Pattern(regexp = "^[a-zA-Z]+$", message = "name must contain characters")
	@Size(max = 6)
```

for password

```
@NotEmpty(message="password can not be empty")
	@Pattern(regexp = "^[A-Za-z]+[0-9]+$", message = "password must contain characters as well as digits")
	@Size(min=4,max = 7)
```

Service Layer

@Component vs @Repository and @Service in Spring

@Component is a generic stereotype for any Spring-managed component

We can use @Component across the application to mark the beans as Spring's managed components. Spring only pick up and registers beans with @Component and doesn't look for @Service and @Repository in general.

@Service annotates classes at the service layer
@Repository annotates classes at the persistence layer, which will act as a database repository

What's the exact use of Service Layer?

Service layer is used for loose coupling:
Let's say you have 100 controller classes and 20 DAO methods serving them.
Now if you directly call DAO methods in the controllers, and at later point of time you want to have different DAO methods serving these controllers.
You have to change all your 100 controllers.
So, have 20 service methods calling those 20 DAO methods.
Now if you want to change your DAO methods serving these 100 controllers, you can just change the service methods (i.e. 20 methods) to point to new DAO methods rather than changing your 100 controller classes.
That's how you achieve loose coupling and is a better way of programming.


Microsoft Word Document

JPA Repository

CrudeRepository

The Java Persistence API (JPA) is the standard way of persisting Java objects into relational databases. ...
The Spring Data JPA module implements the Spring Data Common repository abstraction to ease the repository implementations even more, making a manual implementation of a repository obsolete in most cases.
Spring Data 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.
In order to start leveraging the Spring Data programming model with JPA, a DAO interface needs to extend the JPA specific *Repository* interface – *JpaRepository*. This will enable Spring Data to find this interface and automatically create an implementation for it.
By extending the interface we get the most relevant **CRUD** methods for standard data access available in a standard DAO.

Interface CrudRepository<T,ID>

##

DAY -16 (Web service)

##

What are Web Services?

It means services available on web.
Web service is a way of communication that allows interoperability between different applications on different platforms, for example, a java based application on Windows can communicate with a .Net based one on Linux.
Thus Web services are browsers and operating system independent service, which means it can run on any browser without the need of making any changes.

What are features of web services?

1. Interoperability
2. Reusability
3. Loosely Coupled
4. Extensibility

What are different types of web services?

1. SOAP - specification JAX-WS
2. Restful web services - specification JAX-RS

SOAP Web Services

SOAP stands for Simple Object Access Protocol. It is a XML-based protocol for accessing web services.
SOAP is a W3C recommendation for communication between two applications.
SOAP is XML based protocol. It is platform independent and language independent. By using SOAP, you will be able to interact with other programming language applications
Advantages of Soap Web Services
WS Security: SOAP defines its own security known as WS Security.
Language and Platform independent: SOAP web services can be written in any programming language and executed in any platform.

Disadvantages of Soap Web Services
Slow: SOAP uses XML format that must be parsed to be read. It defines many standards that must be followed while developing the SOAP applications. So it is slow and consumes more bandwidth and resource.
WSDL dependent: SOAP uses WSDL and doesn't have any other mechanism to discover the service.

Some jargons used in Web services:

Simple Object Access Protocol(SOAP):
SOAP is a protocol specification for exchanging structured information in the implementation of Web services. It relies on XML as its message format.
Web Service Description Language(WSDL):
WSDL stands for Web Service Description Language. It is an XML file that describes the technical details of how to implement a web service, more specifically the URI, port, method names, arguments, and data types. Since WSDL is XML, it is both human-readable and machine-consumable, which aids in the ability to call and bind to services dynamically.

Universal Description, Discovery and Integration(UDDI):

UDDI stands for Universal Description, Discovery and Integration. It is a directory service. Web services can register with a UDDI and make themselves available for discovery.

Service provider register its WSDL to UDDI and client can access it from UDDI:UDDI stands for Universal Description, Discovery and Integration.It is a directory service. Web services can register with a UDDI and make themselves available through it for discovery.So following steps are involved.
Service provider registers with UDDI.
Client searches for service in UDDI.
UDDI returns all service providers offering that service.
Client chooses service provider
UDDI returns WSDL of chosen service provider.
Using WSDL of service provider,client accesses web service.

RMI vs SOAP

Soap (Simple Object Access Protocol) and RMI are entirely different technologies. RMI is Java-centric, whereas SOAP, which uses XML, is language independent. However there are some similarities. SOAP, like RMI, allows you to make an RPC on another machine over HTTP or SMTP. Soap works on the request/response model of making remote procedure calls. The client program forms a request which consists of a valid SOAP XML document and sends it over HTTP or SMTP to a server. The server picks up the SOAP request, parses and validates it, invokes the requested method with any supplied parameters, forms a valid SOAP XML response and sends it back over HTTP or SMTP.

REST vs SOAP

1. In case of soap client need to call methods which are defined in developers sides programs while in case of REST client no need to know about methods he just hit the URI to get services
2. SOAP is a **protocol**. REST is an **architectural style**.
3. SOAP stands for **Simple Object Access Protocol**. REST stands for **REpresentational State Transfer**.
4. SOAP **can't use REST** because it is a protocol. REST **can use SOAP** web services because it is a concept and can use any protocol like HTTP, SOAP.
5. SOAP **uses services interfaces to expose the business logic**. REST **uses URI to expose business logic**.
6. **JAX-WS** is the java API for SOAP web services. **JAX-RS** is the java API for RESTful web services.
7. SOAP **defines standards** to be strictly followed. REST does not define too many standards like SOAP.

8. SOAP **requires more bandwidth** and resources than REST. REST **requires less bandwidth** and resources than SOAP.
9. SOAP **permits XML** data format only. REST **permits different** data format such as Plain text, HTML, XML, JSON etc.

Rest (Representational State Transfer API)

It is a software that allows two applications to communicate with each other over the internet and through various devices. Every time you access an app like Facebook or check the weather on you smartphone, an API is used.

Advantages of RESTful Web Services

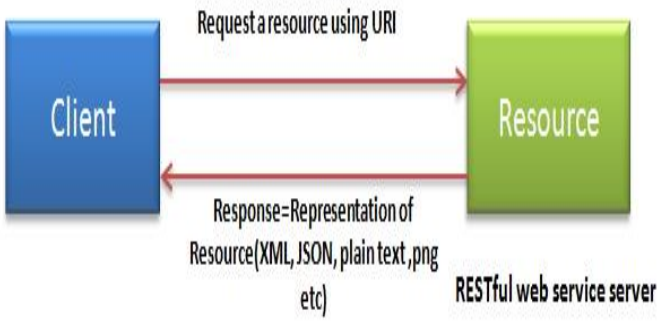
Fast: RESTful Web Services are fast because there is no strict specification like SOAP. It consumes less bandwidth and resource.
Language and Platform independent: RESTful web services can be written in any programming language and executed in any platform.
Can use SOAP: RESTful web services can use SOAP web services as the implementation.
Permits different data format: RESTful web service permits different data format such as Plain Text, HTML, XML and JSON.

What is restful webservices?

In the web services terms, REpresentational State Transfer (REST) is a stateless client-server architecture in which the web services are viewed as resources and can be identified by their URIs. Web services client uses that URI to access the resource.

It consists of two components REST server which provides access to the resources and a REST client which accesses and modify the REST resources.

In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol. REST isn't protocol specific, but when people talk about REST they usually mean REST over HTTP.
The response from server is considered as the representation of the resources.



REST allows that resources have different representations, e.g.xml, json etc. The rest client can ask for specific representation via the HTTP protocol.

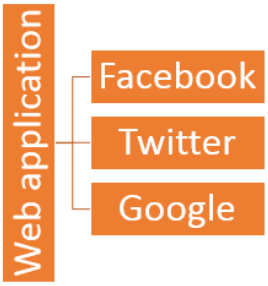
HTTP methods : RESTful web services use HTTP protocol methods for the operations they perform. Methods are:

GET: It defines a reading access of the resource without side-effects. This operation is idempotent i.e. they can be applied multiple times without changing the result
PUT : It is generally used for updating resource. It must also be idempotent.
DELETE : It removes the resources. The operations are idempotent i.e. they can get repeated without leading to different results.
POST : It is used for creating a new resource. It is not idempotent.
Idempotent means result of multiple successful request will not change state of resource after initial application
For example:
Delete is idempotent method because when you first time use delete, it will delete the resource (initial application) but after that, all other request will have no result because resource is already deleted.

Post is not idempotent method because when you use post to create resource, it will keep creating resource for each new request, so result of multiple successful request will not be same.

Why Restful?

Restful mostly came into popularity due to the following reasons:
Heterogeneous languages and environments – This is one of the fundamental reasons which is the same as we have seen for SOAP as well.
It enables web applications that are built on various programming languages to communicate with each other
With the help of Restful services, these web applications can reside on different environments, some could be on Windows, and others could be on Linux.
But in the end, no matter what the environment is, the end result should always be the same that they should be able to talk to each other. Restful web services offer this flexibility to applications built on various programming languages and platforms to talk to each other.
The below picture gives an example of a web application which has a requirement to talk to other applications such Facebook, Twitter, and Google.
Now if a client application had to work with sites such as Facebook, Twitter, etc. they would probably have to know what is the language Facebook, Google and Twitter are built on, and also on what platform they are built on.
Based on this, we can write the interfacing code for our web application, but this could prove to be a nightmare.
Facebook, Twitter, and Google expose their functionality in the form of Restful web services. This allows any client application to call these web services via REST.



The event of Devices – Nowadays, everything needs to work on Mobile devices, whether it be the mobile device, the notebooks, or even car systems.

Can you imagine the amount of effort to try and code applications on these devices to talk with normal web applications? Again Restful API's can make this job simpler because as mentioned in point no 1, you really don't need to know what is the underlying layer for the device.

Finally is the event of the Cloud – Everything is moving to the cloud. Applications are slowly moving to cloud-based systems such as in Azure or Amazon. Azure and Amazon provide a lot of API's based on the Restful architecture. Hence, applications now need to be developed in such a way that they are made compatible with the Cloud. So since all Cloud-based architectures work on the REST principle, it makes more sense for web services to be programmed on the REST based architecture to make the best use of Cloud-based services.

REST API Examples

You can find REST APIs all over the web — you've likely used some today without realizing it. Here are a few examples:

Twitter

The Twitter API lets third-party applications read and write data. Use it to write and post tweets, share tweets, and read profiles. This API is especially effective for downloading and analyzing large amounts of tweets about specific topics. Learn more about using Twitter's API in our guide.

Instagram

The Instagram Basic Display API offers access to profile information, photos, and videos. You can utilize this API and others to build apps that pull this user information and integrate it into your own product. Instagram also has a Graph API available for professional Instagram accounts to manage their online activities.

Spotify

Spotify's web API allows clients to request information about artists, songs, albums, and playlists on its platform. You can also use it to add songs to playlists, pause and play music, shuffle songs, and a lot more.

HubSpot

All of HubSpot's APIs are made with REST conventions and designed for robust integrations that help businesses get the most value out of HubSpot's tools. You can add advanced functionality to HubSpot's powerful marketing software and sync your HubSpot account with other useful tools.

For more REST API examples that you can use for your business, check out our list of our favorite free and open APIs for marketers.

URI, URL, and URN

All three URI, URL, and URN are used to identify any resource or name on the internet, but there is a subtle difference between them. URI is the superset of both URL and URN. By the way, the main difference between URL and URI is protocol to retrieve the resource. URL always include a network protocol e.g. HTTP, HTTPS, FTP etc to retrieve a resource from its location. While URI, in case of URN just uniquely identifies the resource e.g. ISBN numbers which are a good example of URN is used to identify any book uniquely.

What is bandwidth?

In computer networks, bandwidth is used as a synonym for data transfer rate, the amount of data that can be carried from one point to another in a given time period (usually a second). Network bandwidth is usually expressed in bits per second (bps); modern networks typically have speeds measured in the millions of bits per second (megabits per second, or Mbps) or billions of bits per second (gigabits per second, or Gbps).

Difference between Controller and RestController

- Controller is the one which returns control to "View" i.e. JSP.

That's why Controller's method can either return "ModelAndView" or "String". Both ModelAndView and String consist of JSP name.

- RestController is the one which returns the response directly to the client.

@RequestBody, @ResponseBody & @RestController

@RequestBody : Annotation indicating a method parameter should be bound to the body of the HTTP request.

@ResponseBody annotation can be put on a method and indicates that the return type should be written straight to the HTTP response body

Alternatively, we can use @RestController annotation in place of @Controller annotation. This will remove the need to using **@ResponseBody.**

i.e. **@RestController= @Controller + @ResponseBody**

JSON (JavaScript Object Notation)

JSON (JavaScript Object Notation) is a lightweight format that is used for data interchanging. It is based on a subset of JavaScript language (the way objects are built in JavaScript).

An example of where this is used is web services responses. In the 'old' days, web services used XML as their primary data format for transmitting back data, but since JSON appeared, it has been the preferred format because it is much more lightweight

JSON is built on two structures:

A collection of name/value pairs. In various languages, this is realized as an object, record, struct, dictionary, hash table, keyed list, or associative array.

An ordered list of values. In most languages, this is realized as an array, vector, list, or sequence.

##

DAY 17 - Spring Boot, I18N, Annotation

##

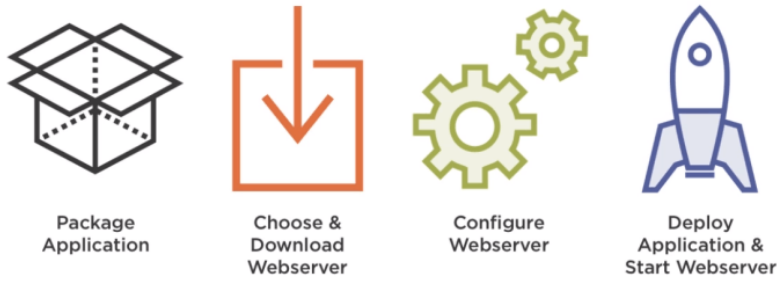
What is Spring Boot?

Spring Boot makes it easy to create stand-alone, production grade spring based applications that you can "just run".

- Spring Boot is not a framework to write application, it helps you to build, package and deploy the spring application with minimal or absolutely no configurations.
- Creates standalone spring applications.
- Web application can be delivered as a FAT-JAR file to run as standalone application.

The Process of Starting a Java-Based Web Application

- First of all, you need to package your application.
- Choose which type of web servers you want to use and download it. They are lot of different solutions out there.
- You need to configure that specific web server.
- After that, you must organize the deployment process and start your web server.



With Spring Boot, you need the following process:

- Package your application
- Run it with some simple command like java -jar my-application.jar

Really, it's that simple.

Spring Boot takes care of the rest by starting and configuring an embedded web server and deploys your application there.

@SpringBootApplication should be applied to that class which is going to start the server and deploy your web application.

SpringApplication's **"run()"** method starts embedded Tomcat Server and deploys your web application on it.

```
@SpringBootApplication
@ImportResource("classpath:applicationContext.xml")
```

```
public class Application {
   public static void main(String[] args) {
      ApplicationContext ctx = SpringApplication.run(Application.class);
      System.out.println("Done");

     }
   }
```

Internationalization (I18N)

```
Locale locales[]=DateFormat.getAvailableLocales();
for(Locale loc:locales)
{
          System.out.println(loc.getDisplayName()+"\t"+loc.getLanguage()+"\t"+loc.getCountry());
}
          }
}
```

Importance of Locale class in Internationalization

An object of Locale class represents a geographical or cultural region. This object can be used to get the locale specific information such as country name, language, variant etc.

Here DateFormat is calss has method getAvailableLocales() method it returns an array of available locales.

Internationalization and Localization in Java

Internationalization is also abbreviated as I18N because there are total 18 characters between the first letter 'I' and the last letter 'N'.
Internationalization is a mechanism to create such an application that can be adapted to different languages and regions.
Internationalization is one of the powerful concept of java if you are developing an application and want to display messages, currencies, date, time etc. according to the specific region or language.

Localization is also abbreviated as L10N because there are total 10 characters between the first letter 'L' and last letter 'N'. Localization is the mechanism to create such an application that can be adapted to a specific language and region by adding locale-specific text and component.

ResourceBundle class in Java

The **ResourceBundle** class is used to internationalize the messages. In other words, we can say that it provides a mechanism to globalize the messages.
The hardcoded message is not considered good in terms of programming, because it differs from one country to another. So we use the ResourceBundle class to globalize the massages. The ResourceBundle class loads these information's from the properties file that contains the messages.

Conventionally, the name of the properties file should be **filename_languagecode_country code** for example **MyMessage_en_US.properties.**

Java Annotations

Java Annotation is a tag that represents the metadata i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler or JVM.
Annotations in java are used to provide additional information, so it is also an alternative option for XML.

Types of Annotation

There are three types of annotations.

1. Marker Annotation
2. Single-Value Annotation
3. Multi-Value Annotation

1) Marker Annotation

An annotation that has no method, is called marker annotation. For example:

```
@interface MyAnnotation{}
```

The @Override and @Deprecated are marker annotations.

2) Single-Value Annotation

An annotation that has one method, is called single-value annotation. For example:

```
@interface MyAnnotation{
int value();
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{
int value() default 0;
}
```

How to apply Single-Value Annotation

Let's see the code to apply the single value annotation.

```
@MyAnnotation(value=10)
```

The value can be anything.

3) Mulit-Value Annotation

An annotation that has more than one method, is called Multi-Value annotation. For example:

```
@interface MyAnnotation{
int value1();
String value2();
String value3();
}
}
```

We can provide the default value also. For example:

```
@interface MyAnnotation{
int value1() default 1;
String value2() default "";
String value3() default "xyz";
}
```

How to apply Multi-Value Annotation

Let's see the code to apply the multi-value annotation.

```
@MyAnnotation(value1=10,value2="Arun Kumar",value3="Ghaziabad")
```

Java Custom Annotation

Java Custom annotations or Java User-defined annotations are easy to create and use. The @interface element is used to declare an annotation. For example:

```
@interface MyAnnotation{}
```

Here, MyAnnotation is the custom annotation name.

There are few points that should be remembered by the programmer.

- Method should not have any throws clauses

- Method should return one of the following: primitive data types, String, Class, enum or array of these data types.
- Method should not have any parameter.
- We should attach @ just before interface keyword to define annotation.
- It may assign a default value to the method.

Return type of annotation method must be one of:

- Primitive
- String
- Class
- an Enum
- another Annotation
- an array of any of the above

Where exactly annotation can be declare?

ANNOTATION_TYPE

Annotation type declaration

CONSTRUCTOR

Constructor declaration

FIELD

Field declaration (includes enum constants)

LOCAL_VARIABLE

Local variable declaration

METHOD

Method declaration

PARAMETER

Parameter declaration

TYPE

Class, interface (including annotation type), or enum declaratio

What is Retention policy in java annotations?

- A retention policy determines at what point annotation should be discarded.
- Java defined 3 types of retention policies through java.lang.annotation.RetentionPolicy enumeration. It has SOURCE, CLASS and RUNTIME.
- **RetentionPolicy.SOURCE**: Discard during the compile. These annotations don't make any sense after the compiler has completed, so they aren't written to the bytecode.
  Example: @Override, @SuppressWarnings
- **RetentionPolicy.CLASS**: RetentionPolicy.CLASS means that the annotation is stored in the .class file, but not available at runtime. This is the default retention policy, if you do not specify any retention policy at all.
- **RetentionPolicy.RUNTIME**: Do not discard. The annotation should be available for reflection at runtime. Example: @Entity, @Autowired etc.

Note:

- @Target(ElementType.TYPE)
  it means annotation can be applied to class, interface,or enum declaration.

- @Retention(RetentionPolicy.RUNTIME)
  it means the annotation should be available for reflection at runtime.

- @Inherited
  Applied annotations can be available to child class also.

##

DAY 18 – Log4J, Junit

##

```
final static Logger logger = Logger.getLogger(HelloExample.class);
logger.warn("This is warn : " + parameter);
```

Log4j

While developing Java/JavaEE applications, for debugging an application that is to know the status of a java application at its execution time, in general we use system.out.println statements.

Log4j is a Java library that specializes in logging. At its most basic level, you can think of it as a replacement for System.out.println's in your code. Why is it better than System.out.println's?
To begin with, System.out.println outputs to standard output, which typically is a console window. The output from Log4j can go to the console, but it can also go to an email server, a database table, a log file, or various other destinations.

Another great benefit of Log4j is that different levels of logging can be set. The levels are hierarchical and are as follows: DEBUG, INFO, WARN, ERROR, and FATAL.

If you set a particular log level, messages will get logged for that level and all levels above it, and not for any log levels below that. As an example, if your log level is set to ERROR, you will log messages that are errors and fatals. If your log level is set to INFO, you will log messages that are infos, warns, errors, and fatals.
Typically, when you develop on your local machine, it's good to set the log level to DEBUG, and when you deploy a web application, you should set the log level to INFO or higher so that you don't fill up your error logs with debug messages.
Log4j can do other great things. For instance, you can set levels for particular Java classes, so that if a particular class spits out lots of warnings, you can set the log level for that class to ERROR to suppress all the warning messages.

Typically, you can configure Log4j via a configuration file. This means that you can change your logging configuration without requiring code updates. If you need to do something like set the log level to DEBUG for your deployed application in order to track down a particular bug, you can do this without redeploying your application. Instead, you can change your log configuration file, reread the configuration file, and your logging configuration gets updated.

Logging is very important for the development and testing of an application. Every application needs a good logging API because it is essential for debugging, testing and troubleshooting the application.
Log4j is such a complete logging package, one which allows better and faster application development.

Log4j logger contains three main components namely **logger, appender and layout.** Logger takes care of the logging mechanism and deals with level of logging. Log4j provides five standard levels of logging. There are two more special levels given by log4j. Above all these, log4j allows you to create custom levels.

Setting a Log Level

Setting a level causes a Logger to ignore messages below that level. For example, the following statement causes the Logger to ignore messages below the WARNING level.
logger.setLevel(Level.WARNING);

What are different logging level in Java
Anybody who is using logging in java must be familiar with basic java logging level
e.g. DEBUG, INFO, WARN and ERROR.

DEBUG is the lowest restricted java logging level and we should write everything we need to debug an application, this java logging mode should only be used on Development and Testing environment and must not be used in production environment.

INFO is more restricted than DEBUG java logging level and we should log messages which are informative purpose like Server has been started, Incoming messages, outgoing messages etc in INFO level logging in Java.

WARN is more restricted than INFO java logging level and used to log warning sort of messages e.g. Connection lost between client and server. Database connection lost, Socket reaching to its limit. These messages and java logging level are almost important because you can setup alert on these logging messages in Java and let your support team monitor health of your java application and react on this warning messages. In Summary WARN level is used to log warning message for logging in Java.

ERROR is the more restricted java logging level than WARN and used to log Errors and Exception, you can also setup alert on this java logging level and alert monitoring team to react on this messages. ERROR is serious for logging in Java and you should always print it.

FATAL java logging level designates very severe error events that will probably lead the application to abort. After this mostly your application crashes and stopped.

OFF java logging level has the highest possible rank and is intended to turn off logging in Java.

ALL Level: This log4j level is used to turn on all levels of logging. Once this is configured and the levels are not considered.

When a logger is created, generally you assign a level. The logger outputs all those messages equal to that level and also to all greater levels than it. So the order for the standard log4j levels are:

Log4J Levels	*DEBUG Level*	*INFO Level*	*WARN Level*	*ERROR Level*	*FATAL Level*
DEBUG Level	Y	Y	Y	Y	Y
INFO Level	N	Y	Y	Y	Y
WARN Level	N	N	Y	Y	Y
ERROR Level	N	N	N	Y	Y
FATAL Level	N	N	N	N	Y
ALL Level	Y	Y	Y	Y	Y
OFF Level	N	N	N	N	N

Log4j Level Inheritance Rule

Suppose , you have a package as com.foo.bar and you haven't allocated a level, then it will inherit the level from com.foo package. Still in that package also if the level is not available, then it will inherit from the log4j root level. The log4j's root logger is instantiated and available always. **Log4j's root logger by default has DEBUG level.**

We have mainly 3 components to work with Log4j

1. **Logger**
2. **Appender**
3. **Layout**

Logger

Logger is a class, in org.apache.log4j.*
We need to create Logger object one per java class.
This component enables Log4j in our java class.
Logger methods are used to generate log statements in a java class instead of sopls.
So in order to get an object of Logger class, we need to call a static factory method [factory method will gives an object as return type]
Getting Logger Object
static Logger log = Logger.getLogger(YourClassName.class.getName())
Note: while creating a Logger object we need to pass either fully qualified class name or class object as a parameter, class means current class for which we are going to use Log4j.

Logger object has some methods, actually we used to print the status of our application by using these methods only
We have totally 5 methods in Logger class.

- debug()
- info()
- warn()

- error()
- fatal()

Priority Order

debug < info < warn < error < fatal

I mean, fatal is the highest error like some database down/closed

Appender

Appender job is to write the messages into the external file or database.

Where Logger classes generates some statements under different levels , Appender takes these log statements and stores in some files or database.

Appender is an interface

In log4j we have different Appender implementation classes

- FileAppender [writing into a file]
- ConsoleAppender [Writing into console]
- [For Databases]
- SMTPAppender [Mails]
- SocketAppender [For remote storage]

Layout

This component specifies the format in which the log statements are written into the destination repository by the appender

We have different type of layout classes in log4j

- SimpleLayout
- PatternLayout
- HTMLLayout
- XMLLayout

Junit / unit testing

Unit testing means testing the smaller units of your application, like classes and methods.

The reason you test your code is to prove to yourself, and perhaps to the users / clients / customers, that your code works.

JUnit is an open source framework designed by Kent Beck, Erich Gamma for the purpose of writing and running test cases for java programs. In case of web applications, JUnit is used to test the application without server. This framework builds a relationship between development and testing process.

@**Test** This annotation is a replacement of org.junit.TestCase which indicates that public void method to which it is attached can be executed as a test Case.

Assert Methods

1. **assertArrayEquals()**
2. **assertEquals()**
3. **assertTrue() + assertFalse()**
4. **assertNull() + assertNotNull()**

There are two ways to run test cases

SampleTest
we can run this by
1) right click on SampleTest - run as - JUnit Test case
2) JUnitCore.runClasses(SampleTest.class);

JUnit core class

JUnitCore is an inbuilt class in JUnit package and it is based on facade design pattern.

JUnitCore class is used to run only specified test classes.org.junit.runner.JUnitCore class is responsible for executing tests. runClasses() method of JUnitCore class enables running the one or more test classes which yield Result Object (org.junit.runner.Result).

The test results will be extracted from the Result Object.

JUnit - Suite Test

Test suite

Test suite means bundle a few unit test cases and run it together. In JUnit,
both @RunWith and @Suite annotation are used to run the suite test. Here we've two TestJunit1 & TestJunit2 test classes to run together using Test Suite.

Mockito

What is mocking?

Mocking is a testing technique where real components are replaced with objects that have a predefined behaviour (mock objects) only for the test/tests that have been created for. In other words, a mock object is an object that is configured to return a specific output for a specific input, without performing any real action.

Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

What is Mocking?

Mocking is a way to test the functionality of a class in isolation. Mocking does not require a database connection or properties file read or file server read to test a functionality. Mock objects do the mocking of the real service. A mock object returns a dummy data corresponding to some dummy input passed to it.

Mockito

Mockito facilitates creating mock objects seamlessly. It uses Java Reflection in order to create mock objects for a given interface. Mock objects are nothing but proxy for actual implementations.

Why should we mock?

There are several scenarios where we should use mocks:

1. When we want to **test a component that depends on other component, but which has not yet been developed.** This happens often when working in team, and component development is divided between several developers. If mocking weren't exist, we would have to wait until the other developer/developers end the required component/component for testing ours.

2. When the **real component performs slow operations**, usual with dealing with database connections or other intense disk read/write operations. It is not weird to face database queries that can take 10, 20 or more seconds in production environments. Forcing our tests to wait till that time would be a considerable waste of useful time that can be spent in other important parts of the development.
3. When there are **infrastructure concerns that would make impossible the testing**. This is similar in same way to the first scenario described when, for example, our development connects to a database, but the server where it is hosted is not configured or accessible for some reason.