freelance_Project available to buy contact on 8007592194

SR.NO	Project NAME	Technology
1	Online E-Learning Platform Hub	React+Springboot+MySql
2	PG Mates / RoomSharing / Flat Mates	React+Springboot+MySql
3	Tour and Travel management System	React+Springboot+MySql
4	Election commition of India (online Voting System)	React+Springboot+MySql
5	HomeRental Booking System	React+Springboot+MySql
6	Event Management System	React+Springboot+MySql
7	Hotel Management System	React+Springboot+MySql
8	Agriculture web Project	React+Springboot+MySql
9	AirLine Reservation System / Flight booking System	React+Springboot+MySql
10	E-commerce web Project	React+Springboot+MySql
11	Hospital Management System	React+Springboot+MySql
12	E-RTO Driving licence portal	React+Springboot+MySql
13	Transpotation Services portal	React+Springboot+MySql
14	Courier Services Portal / Courier Management System	React+Springboot+MySql
15	Online Food Delivery Portal	React+Springboot+MySql
16	Muncipal Corporation Management	React+Springboot+MySql
17	Gym Management System	React+Springboot+MySql
18	Bike/Car ental System Portal	React+Springboot+MySql
19	CharityDonation web project	React+Springboot+MySql
20	Movie Booking System	React+Springboot+MySql

freelance_Project available to buy contact on 8007592194

21	Job Portal web project	React+Springboot+MySql
22	LIC Insurance Portal	React+Springboot+MySql
23	Employee Management System	React+Springboot+MySql
24	Payroll Management System	React+Springboot+MySql
25	RealEstate Property Project	React+Springboot+MySql
26	Marriage Hall Booking Project	React+Springboot+MySql
27	Online Student Management portal	React+Springboot+MySql
28	Resturant management System	React+Springboot+MySql
29	Solar Management Project	React+Springboot+MySql
30	OneStepService LinkLabourContractor	React+Springboot+MySql
31	Vehical Service Center Portal	React+Springboot+MySql
32	E-wallet Banking Project	React+Springboot+MySql
33	Blogg Application Project	React+Springboot+MySql
34	Car Parking booking Project	React+Springboot+MySql
35	OLA Cab Booking Portal	React+Springboot+MySql
36	Society management Portal	React+Springboot+MySql
37	E-College Portal	React+Springboot+MySql
38	FoodWaste Management Donate System	React+Springboot+MySql
39	Sports Ground Booking	React+Springboot+MySql
40	BloodBank mangement System	React+Springboot+MySql
41	Bus Tickit Booking Project	React+Springboot+MySql
42	Fruite Delivery Project	React+Springboot+MySql
43	Woodworks Bed Shop	React+Springboot+MySql
44	Online Dairy Product sell Project	React+Springboot+MySql
45	Online E-Pharma medicine sell Project	React+Springboot+MySql
46	FarmerMarketplace Web Project	React+Springboot+MySql
47	Online Cloth Store Project	React+Springboot+MySql
48		React+Springboot+MySql
49		React+Springboot+MySql
50		React+Springboot+MySql

Java EE

Trainer: Nilesh Ghule

Java EE

Sunbeam Infotech

Java EE

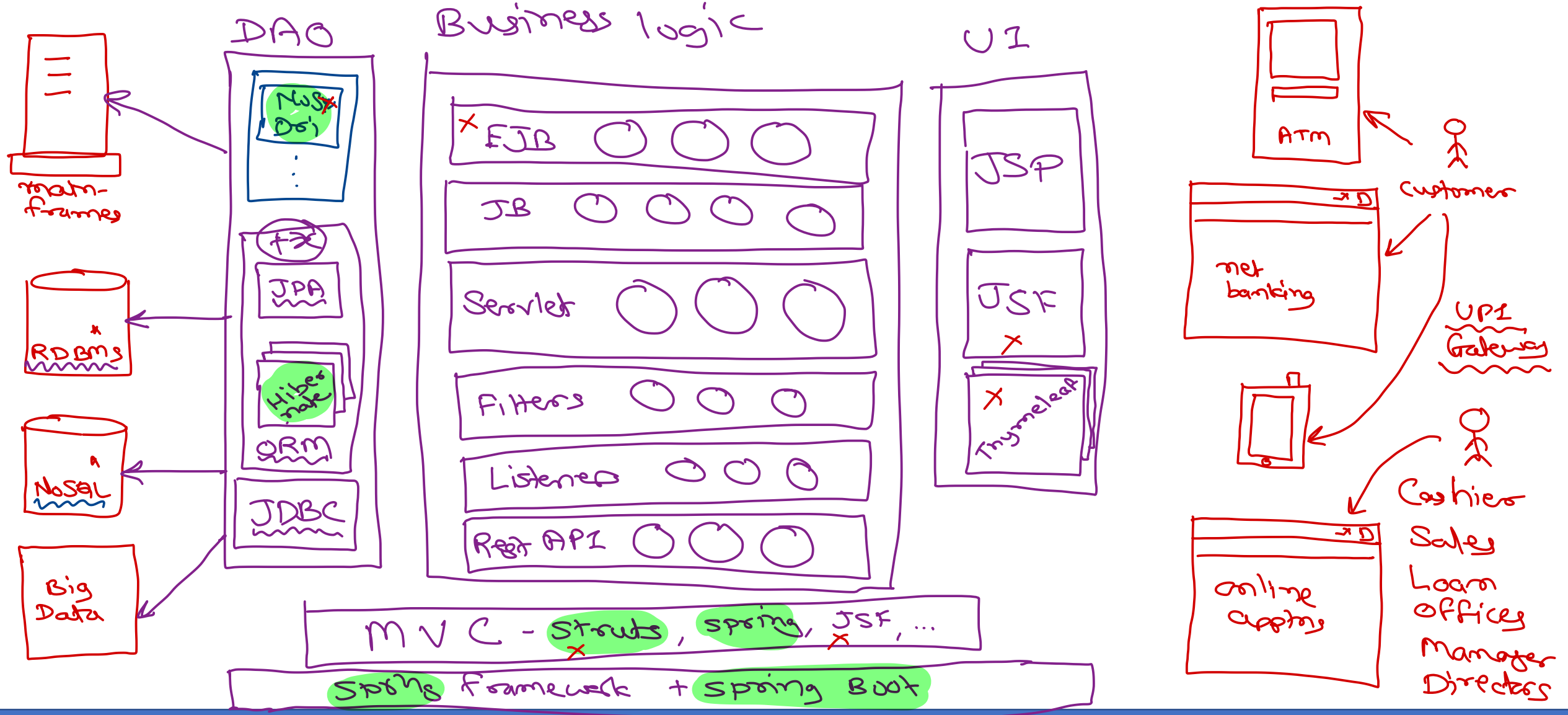
- Java EE is a specification to build enterprise applications.
- In fact it is suite of specifications.
  - Servlet/JSP specs
  - JSF specs
  - JPA specs
  - JTA specs
  - JNDI specs
  - JMS specs
  - EJB specs
- Java EE builds web based solutions.

implemented by Java web servers & appln servers

Servlet interface

JDBC

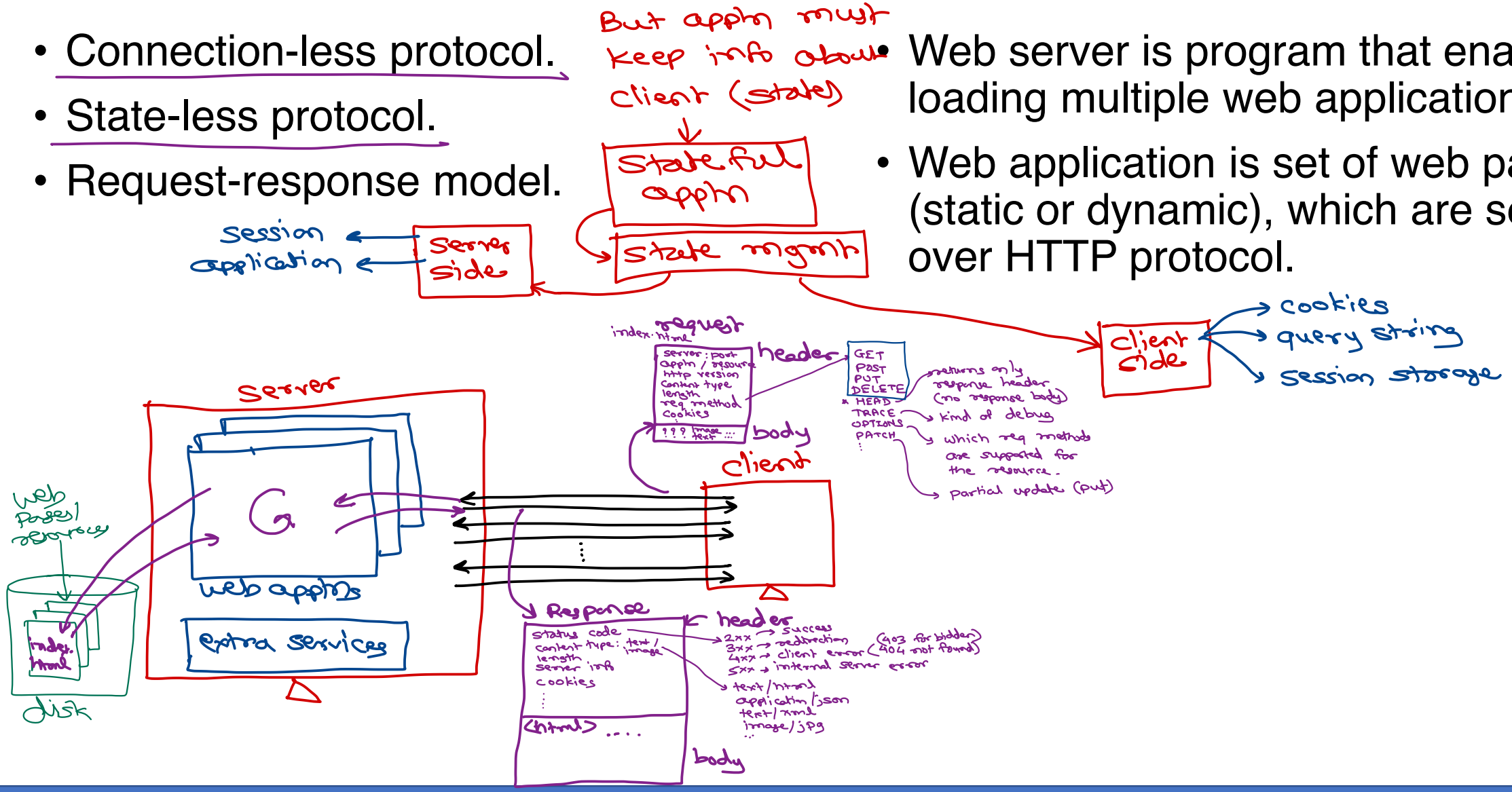
Java EE application (web applm



HTTP protocol

- Connection-less protocol.
- State-less protocol.
- Request-response model.

- Web server is program that enable loading multiple web applications in it.
- Web application is set of web pages (static or dynamic), which are served over HTTP protocol.



Java Web Servers

- There are many web servers from different vendors. But all implement the same Java EE specifications.
- Java web server = Servlet container + Extra services.
  - e.g. Tomcat, Lotus, …
- Java application server = Servlet container + EJB container + Extra services.
  - e.g. JBoss, WebSphere, WebLogic, …
- Extra services includes security (HTTPS), JNDI, Connection pool, …

Apache tomcat

- Apache tomcat is Java web server (Servlet container & extra services).
- Apache tomcat 9.x implements Java EE 8 specs.
  - Servlet 4.0 specs
  - JSP 2.3 specs
  - JSF 2.3 specs
- Tomcat directory structure
  - bin
  - conf
  - lib
  - webapps
  - work
  - logs

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE (Advanced Java)

Agenda

- Module Introduction
- STS IDE
- Java EE
- HTTP Protocol
- Java Web Server
- Tomcat Web Server

Java EE Module

Contents

- Servlet
- JSP
- Spring
- Spring Boot
- Hibernate
- JPA
- Spring Data JPA

Pre-requisites

- Java
  - JDBC (DAO approach)
  - Collections (ArrayList and HashMap)
  - Reflection, Annotations, Proxy -- To understand internals
    - https://dzone.com/articles/java-dynamic-proxy
  - OOP (Class, Object, Inheritance, Overriding, etc).
  - Stream programming (map(), filter(), etc.) -- Optional
- Web Programming
  - HTML -- html forms
  - React

Schedule

- 3 Weeks
- Week 1:
  - Foundations: Servlets & JSP
- Week 2 + 3:
  - Spring & Hibernate
- Week 3 (End):
  - Exam oriented topics
- Lectures:
  - 9:00 am to 1:30 pm -- Lecture

        - 30 mins break
    - 3:00 pm to 7:00 pm -- Lab

Evaluation

- Theory: 40 marks -- CCEE
- Lab: 40 marks -- Spring Boot + JPA
- Internals: 20 marks
    - Quiz
    - Case study

Software Setup

- Apache Tomcat 9.x
- Spring Tool Suite (Eclipse)

Java EE Introduction

- Java SE - Standard Edition
    - Java platform
    - Applications: Console, Desktop (AWT, Swing, JavaFX, ...)
- Java EE - Java Enterprise Edition
    - Enterprise -- Business, Organization, ...
        - SBI Bank
        - LIC
        - Indian Rail
        - Amazon
        - Netflix
- Java EE versions
    - Java EE 1.2 -- 1999
    - Java EE 1.3 -- 2001
    - Java EE 1.4 -- 2003
    - Java EE 5 -- 2006
    - Java EE 6 -- 2009
    - Java EE 7 -- 2013
    - ** Java EE 8 -- 2017 -- Last release of Java EE (by Oracle)
    - Jakarta EE 8 -- 2018 -- First release of Jakarta EE (by Eclipse)
- Due to copyright issues Java EE is renamed to Jakarta EE.
- https://javaee.github.io/javaee-spec/javadocs/

HTTP Protocol

- Web Server: Program/Application that enable deploying/executing one/more web applications in it.
    - e.g. IIS (Microsoft), Apache Web Server (PHP), NGINX (PHP).
    - Java Web Server: Java (JVM based) application that enable deploying one/more Java web applications in it.
        - e.g. Tomcat, Lotus, JBoss, WebSphere, etc.
- Web Clients: These web applications will be consumed by the clients over HTTP protocol.

    - e.g. Browser, Mobile applns, Embedded applns, etc.
- Web application/site: Set of web pages/resources served over the internet (using HTTP protocol).
- HTTP protocol
    - Application layer protocol (in ISO-OSI layers) depending on TCP protcol.
    - TCP protocol
        - Connection oriented protocol
        - Stateful protocol
        - Reliable (Acknowlegement)
    - HTTP protocol
        - Connection-less protocol
        - State-less protocol
        - Request Response model

Java Server

- Java (JVM based) application that enable deploying one/more Java web applications in it.
- Java Web Server implements Java EE (suite of) specifications.
    - e.g. javax.servlet.Servlet interface (spec) implemented into the classes
        - javax.servlet.http.HttpServlet (class implemented in web server)
        - javax.servlet.ftp.FtpServlet (class implemented in web server)
- Two types of Java servers
    - Java Web Server
    - Java Application Server

Java Web Server

- Java Web Server = Web container + Extra services
    - Web container -- Required for executing Servlet & JSP.
    - Extra servies -- SSL, JNDI, Connection pool, etc.
    - e.g. Tomcat, Lotus, ...

Java Application Server

- Java Application Server = Web container + Extra services + EJB container
    - Web container -- Required for executing Servlet & JSP.
    - Extra servies -- SSL, JNDI, Connection pool, etc.
    - EJB container -- Required for executing EJB, Message Beans (Heavy-weight).
    - e.g. JBoss, WebSphere, WebLogic, ...

Apache Tomcat

- Java Web Server
- Directory structure
    - bin: Contains tomcat binaries and scripts (e.g. tomcat9.exe, startup.bat, shutdown.bat, etc.)
    - conf: Contains tomcat config files (e.g. server.xml, users.xml, etc.)
        - server.xml for tomcat port setting (Check port number for tomcat downloaded on your computer)
    - lib: Contains tomcat spec implementation jars (e.g. servlet-api.jar, jsp-api.jar, etc.)

- webapps: Web applications are deployed here (Hot deployment folder)
- work: Contains temp files (We will discuss while JSP programming)
- logs: Contains execution logs

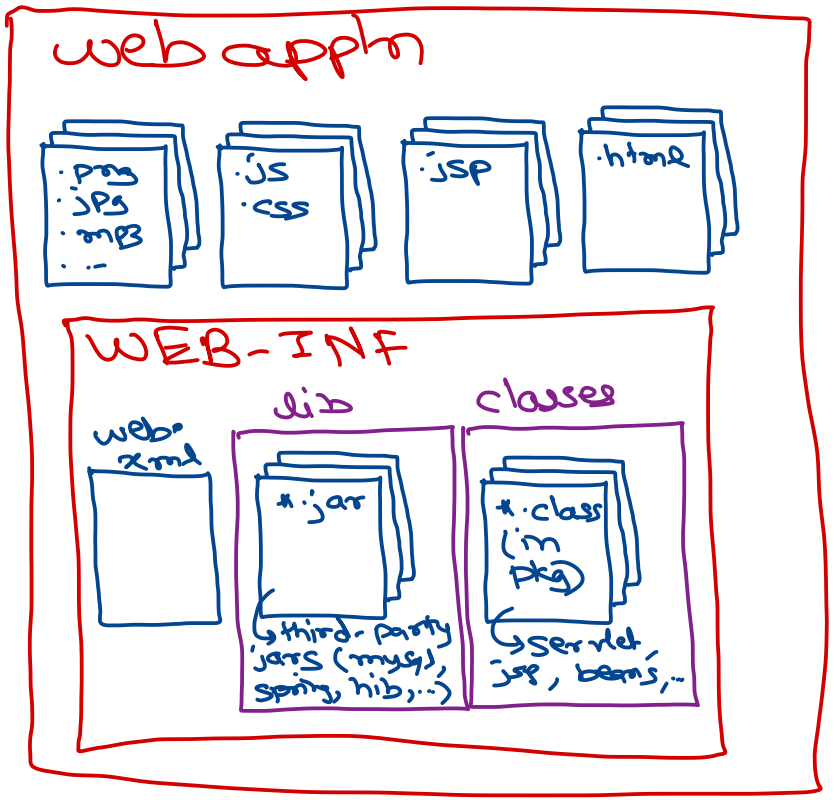
Java EE

Trainer: Nilesh Ghule

Java EE

Sunbeam Infotech

Java EE Dir Structure

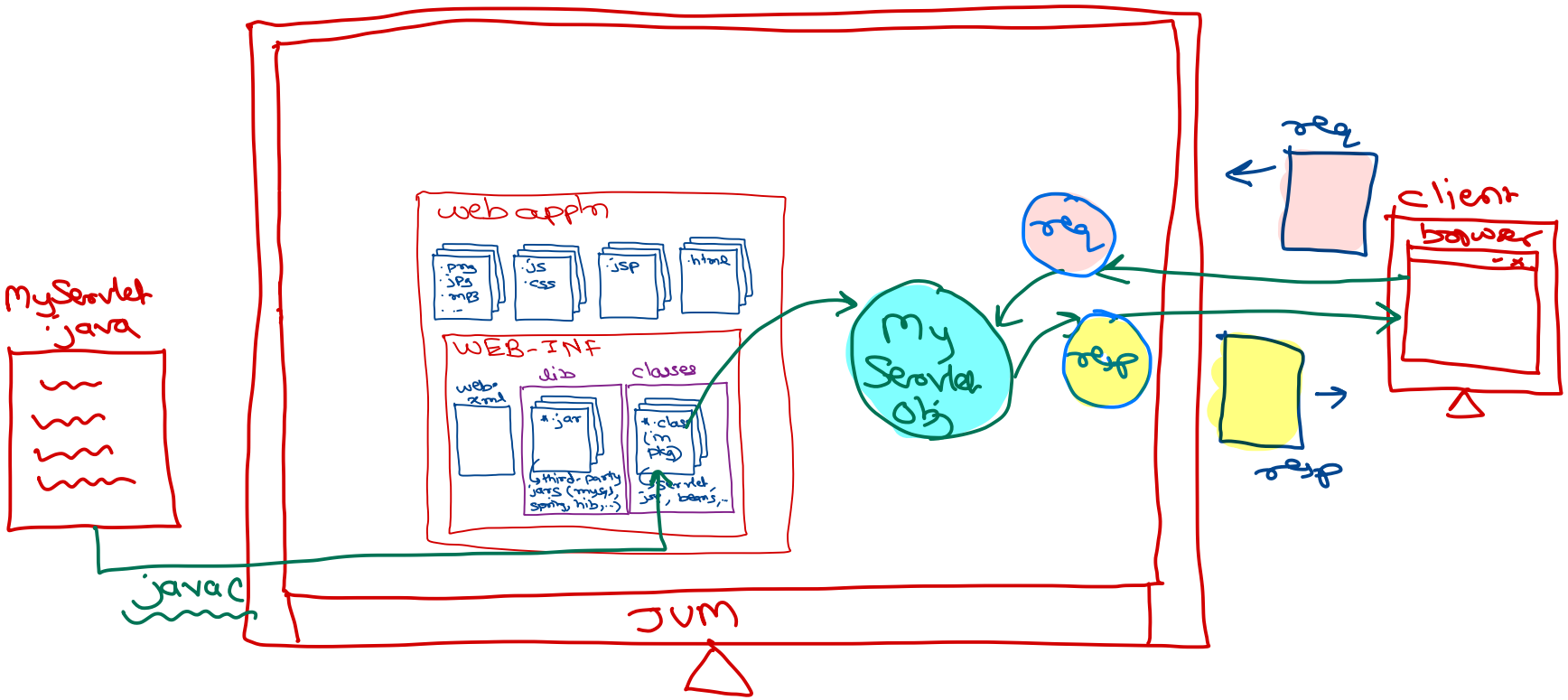


Java Servlets

Sunbeam Infotech

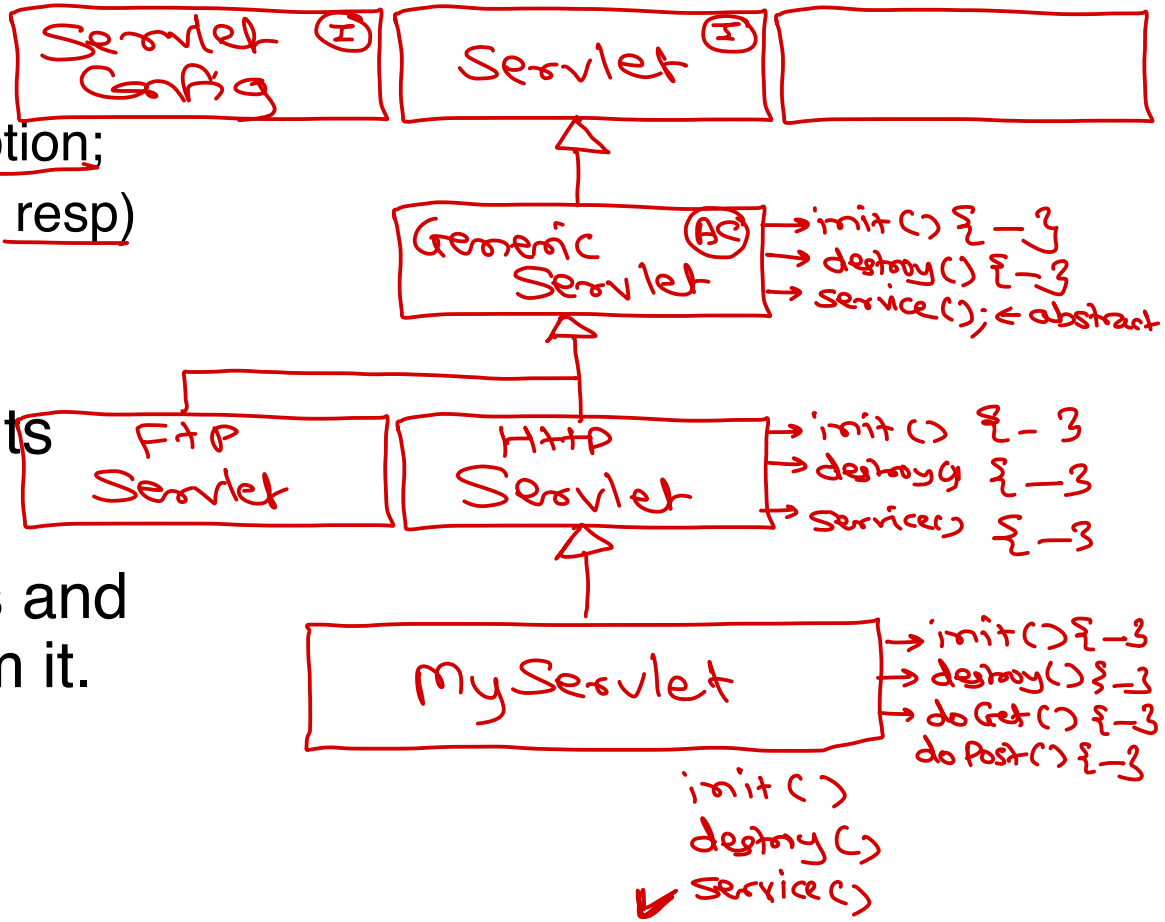
Servlets

- Servlet is java class that is executed on server side, when request is done by client and produces result, which is sent to the client as response.
- Servlet specs include multiple interfaces like Servlet, ServletRequest, ServletResponse, Filter, RequestDispatcher, …



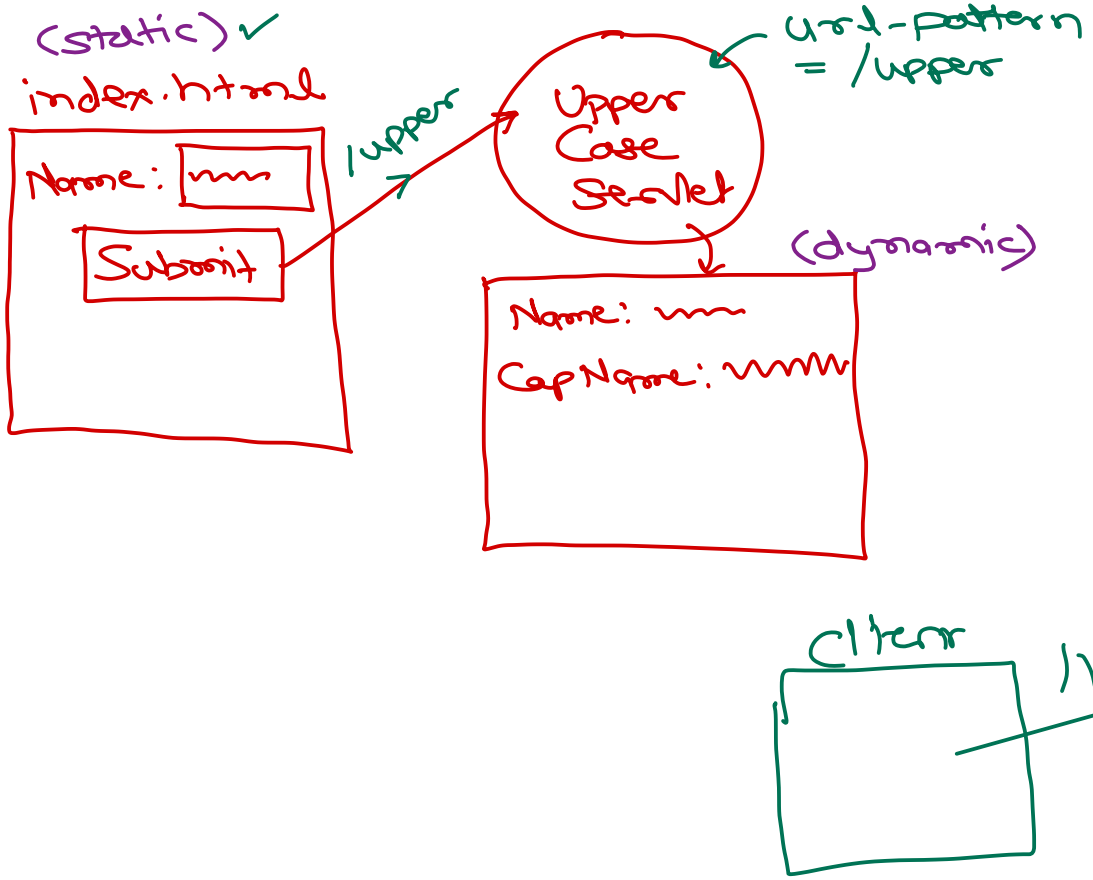
Servlets

- javax.servlet.Servlet interface
  - void init(ServletConfig config) throws ServletException;
  - void service(ServletRequest req, ServletResponse resp) throws IOException, ServletException;
  - void destroy();
- GenericServlet is abstract class that represents protocol-independent servlet.
- HttpServlet represent http based servlet class and user defined servlet classes are inherited from it.
  - Overrides service() method.
  - Provide doGet(), doPost(), doPut(), doDelete(), doHead(), doTrace(), doOptions()



Servlets

- Servlet can be configured using @WebServlet or web.xml

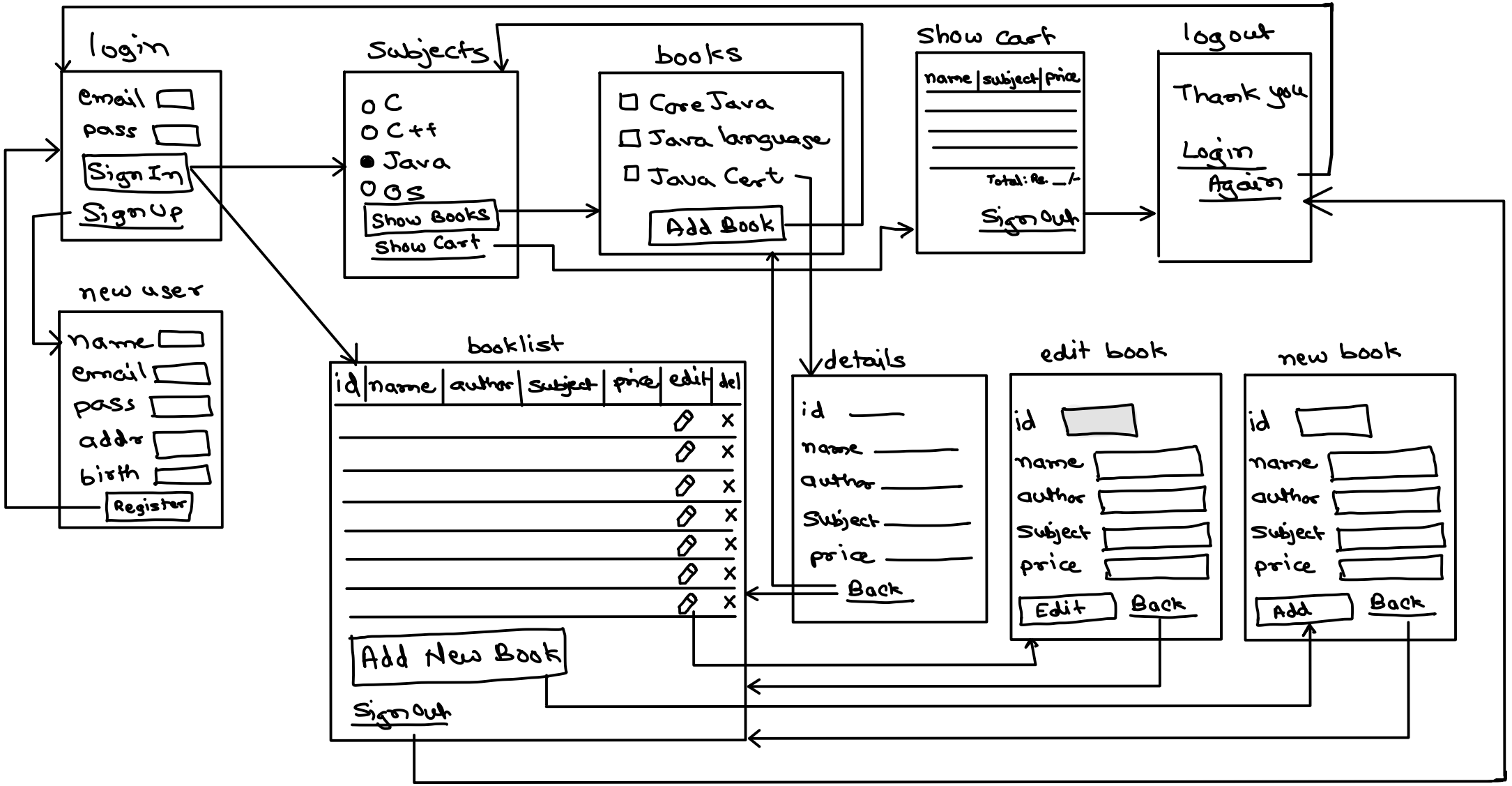



```
<!-- web.xml -->
<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>pkg.Hello</servlet-class>
</servlet>
<servlet-mapping>
  <servlet-name>Hello</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

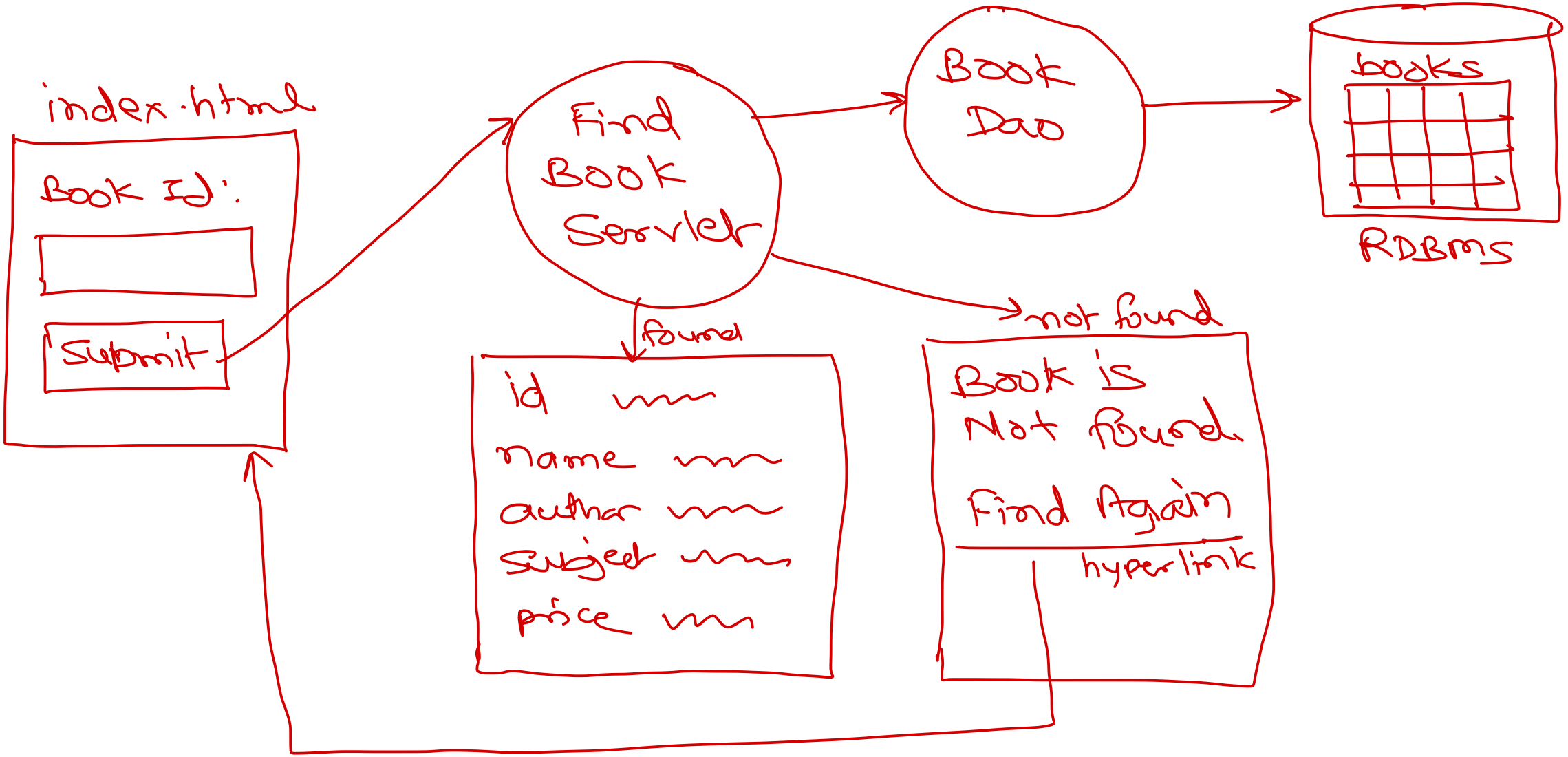
Servlets – HttpServletRequest & HttpServletResponse

- Request Parameters
  - req.getParameter()
  - req.getParameterValues()
- Request Headers
  - req.getHeader()
  - req.getHeaderValues()
- Request upload
  - req.openInputStream()
- Response content type
  - resp.setContentType()
- Response download
  - resp.openOutputStream()

Project requirement - online book shop



Assignment 1 (Servlets)



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE (Advanced Java)

Agenda

- Java EE application directory structure
- Java Servlets
- HttpServletRequest & HttpServletResponse
- ~~Inter-servlet communication~~
- ~~State Management~~

HTTP protocol

- GET request is generated in following places
  - Submit HTML form with method="GET"
  - When user enter an URL in browser
  - When user click on a Hyperlink `<a href="...">`
- POST request is generated in following places
  - Submit HTML form with method="POST"

Apache Tomcat

- Download Tomcat zip/rar.
- Extract Tomcat.
- Set Environment variable: JAVA_HOME
  - This PC --> Properties --> Advanced System Settings --> Advanced --> Environment Variables
  - New (User or System variable)
    - JAVA_HOME = "C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot"
    - PATH add "C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot\bin"
- Run tomcat/bin/startup.bat.
- Browser: http://localhost:8080/
- Stop tomcat/bin/shutdown.bat.

Java EE application directory structure

- Web application: Set of web pages/resources
  - Web Pages
    - Static web pages: Static contents e.g. HTML pages.
      - No code execution at server side.
    - Dynamic web pages: Dynamic contents produced when request is done
      - Java code executed on server side to produce HTML output.
      - Most commonly Business logic, Database connectivity, ...
- Java EE Web application directory structure (part of specification -- applicable for all web servers)

Hello Web Application

- Create directory "helloweb" in tomcat/webappps.
- helloweb

    - index.html
    - WEB-INF
        - web.xml
        - lib
        - classes
- Start the tomcat (bin/startup.bat)
- Browser: http://localhost:8080/helloweb/index.html
    - index.html is visible on browser
- Browser: http://localhost:8080/helloweb/WEB-INF/web.xml
    - 404 - Not found
    - All files under WEB-INF are private to the application and not accessible to end user "directly".
- .jar --> Java archive (Zip format)
- .war --> Web archive (Zip format)
    - Java EE web application directory structure
- To deploy any Java EE web application it must be packaged as .war file and deploy in tomcat/webapps.

Java Servlets

- Servlet is user-defined java class inherited from javax.servlet.Servlet interface.

- javax.servlet.Servlet interface

    - init():
        - Called only once by web container when first request is made for the servlet.
        - For the first request to the servlet
            - Servlet class (.class file) is loaded
            - Object is created (by web container)
            - Constructor is executed (by JVM)
            - init() is called (by web container) -- can access web server facilities like connection pool, JNDI, ...
        - Programmer override this method to perform one-time initialization e.g. getting init params, create one-time database connection, etc.
        - Receives a param -- ServletConfig
            - ServletConfig object is created by web container
            - Stores information about current servlet
        - If initialization fails, must throw ServletException (if exception is of other type, it should be nested/chained in ServletException).
        - Then web container will not process the servlet further.
    - destroy():
        - Called only once by web container when server shutdown.
        - While server shutdown:
            - Call destroy() method of all servlet objects
            - Objects are ready for garbage collection.
        - Programmer override this method to perform de-initialization. Typically resources acquired during init() are released in destroy().
        - destroy() method should not fail. Handle exception if any.
    - service():

        - Called once by web container for each request.
        - For each request:
            - Web container create HttpServletRequest object (representing HTTP request).
            - Web container create HttpServletResponse object (to represent HTTP response).
            - Calls service() method of the servlet.
            - Convert response object into HTTP response.
        - Programmer should override this method to implement Business logic and response generation logic.

- javax.servlet.GenericServlet class (abstract)
    - Default implementation of init()
        - Attach ServletConfig object with current Servlet object.
    - Default implementation of destroy()
    - Do not provide implementation of service() -- remains abstract
        - GenericServlet is protocol independent servlet
        - service() handling is always protocol dependent.

- javax.servlet.HttpServlet class (abstract)
    - init() implementation
    - destroy() implementation
    - service() implementation -- to handle HTTP request

```
abstract class HttpServlet extends GenericServlet ... {
    @Override
    public void init(ServletConfig config) ... {
        super.init(config);
        // ...
    }
    @Override
    public void destroy() {
        // ...
        super.destroy();
    }
    @Override
    public void service(ServletRequest req, ServletResponse resp) ... {
        // get data from HTTP request and load into req object.
        this.service((HttpServletRequest)req, (HttpServletResponse) resp);
        // get data from resp object and convert into HTTP response.
    }
    protected void service(HttpServletRequest req, HttpServletResponse resp)
... {
        String method = req.getMethod(); // GET, POST, PUT, DELETE, HEAD,
TRACE, OPTIONS
        if(method.equals("GET"))
            this.doGet(req, resp); // if overridden by user-defined servlet,
it's doGet() will be called
        else if(method.equals("POST"))
            this.doPost(req, resp); // if overridden by user-defined
servlet, it's doPost() will be called
```

```
        else if(method.equals("PUT"))
            this.doPut(req, resp); // ...
        else if(method.equals("DELETE"))
            this.doDelete(req, resp); // ...
        else if(method.equals("HEAD"))
            this.doHead(req, resp);
        else if(method.equals("TRACE"))
            this.doTrace(req, resp);
        else if(method.equals("OPTIONS"))
            this.doOptions(req, resp);
        // ...
    }
    protected void doGet(HttpServletRequest req, HttpServletResponse resp)
... {
        throw new RuntimeException(...);
    }
    protected void doPost(HttpServletRequest req, HttpServletResponse resp)
... {
        throw new RuntimeException(...);
    }
    // ...
}
```

- User-defined Servlet class

```
package sunbeam;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;
import javax.servlet.annotation.*;

@WebServlet("/hello") // url-pattern
public class MyServlet extends HttpServlet {
    @Override
    public void init(ServletConfig config) throws ServletException {
        super.init(config);
        System.out.println("MyServlet.init() called.");
    }
    @Override
    public void destroy() {
        System.out.println("MyServlet.destroy() called.");
    }
    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp)
throws IOException, ServletException {
        System.out.println("MyServlet.doGet() called.");
        processRequest(req, resp);
    }
    @Override
    public void doPost(HttpServletRequest req, HttpServletResponse resp)
throws IOException, ServletException {
```

```
        System.out.println("MyServlet.doPost() called.");
        processRequest(req, resp);
    }
    public void processRequest(HttpServletRequest req, HttpServletResponse
resp) throws IOException, ServletException {
        // business logic
        // presentation logic
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Java EE</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h2>Hello Servlet</h2>");
        out.println("<h5>Response generated at: " + new Date() + "</h5>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Command Line Compilation of Servlet

- Continue with helloweb application (directory struct already created in tomcat/webapps).
- Create directory WEB-INF/src.
- Save MyServlet.java into WEB-INF/src.
- Compile MyServlet.java into MyServlet.class

```
cmd> cd tomcat\webapps\helloweb\WEB-INF\src
cmd> set CLASSPATH=tomcat\lib\servlet-api.jar;%CLASSPATH%
cmd> javac -d ..\classes MyServlet.java
```

- Start tomcat.
- Browser: http://localhost:8080/helloweb/hello

Hello Servlet on Eclipse

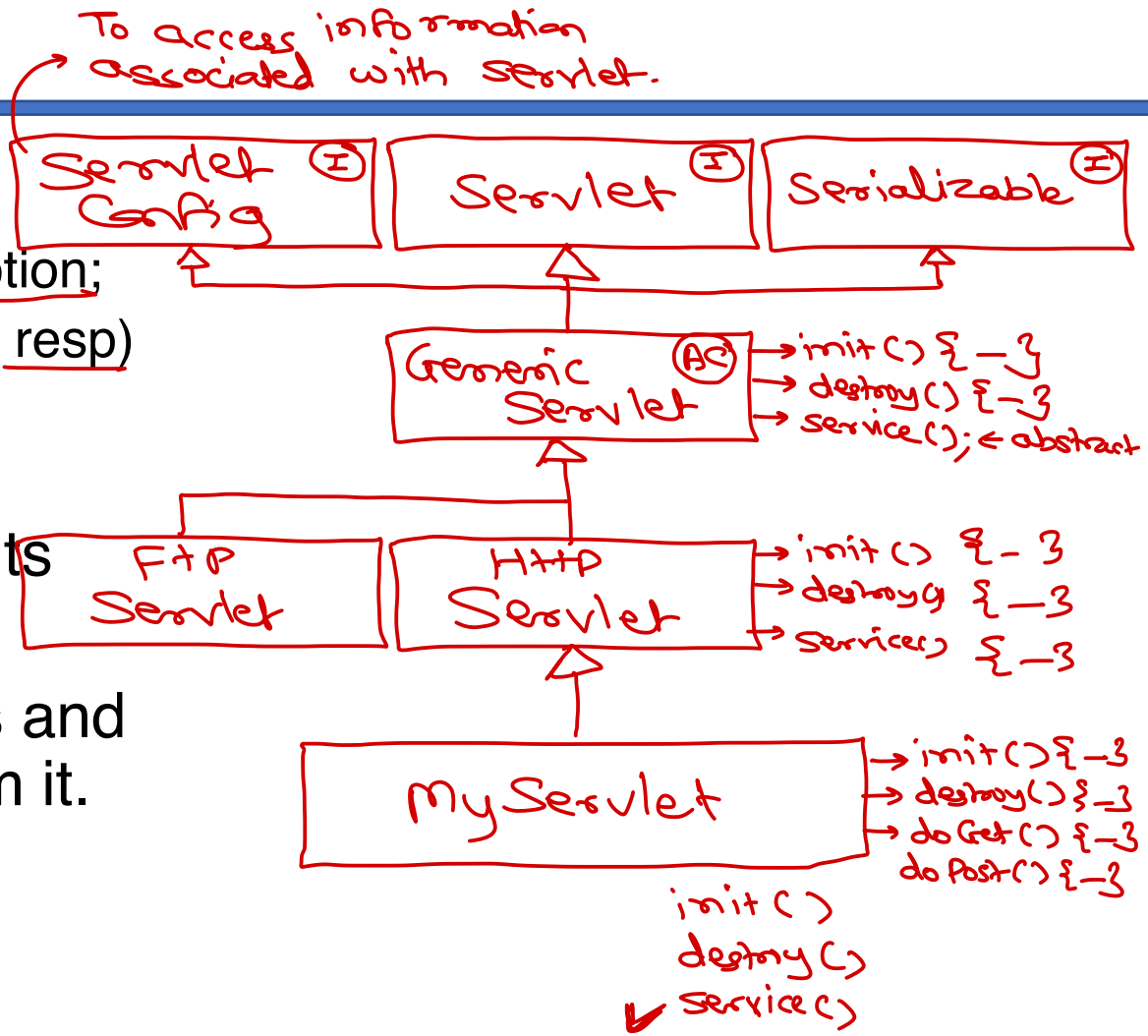
- Configure Eclipse with Apache Tomcat (once per workspace).
  - Window --> Server --> Runtime --> Add New Tomcat 9.0 (as given in PDF)
- Create new project --> Web --> Dynamic Web Project --> demo01
  - Check "Generate web.xml deployment descriptor"
- Create MyServlet.java under src.
- Run As --> Run on Server --> Select Tomcat (Check Always run on this server).
  - Next ... Finish
- Browser: http://localhost:8080/demo01/hello

Java Servlets

Sunbeam Infotech

Servlets

- javax.servlet.Servlet interface
  - void init(ServletConfig config) throws ServletException;
  - void service(ServletRequest req, ServletResponse resp) throws IOException, ServletException;
  - void destroy();
- GenericServlet is abstract class that represents protocol-independent servlet.
- HttpServlet represent http based servlet class and user defined servlet classes are inherited from it.
  - Overrides service() method.
  - Provide doGet(), doPost(), doPut(), doDelete(), doHead(), doTrace(), doOptions()



Servlets

- Load on Startup

```
<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>pkg.Hello</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Hello</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

-ve number will load servlet upon first req.

+ve number defines seq of loading servlet in case of multiple startup servlets.

- Init params

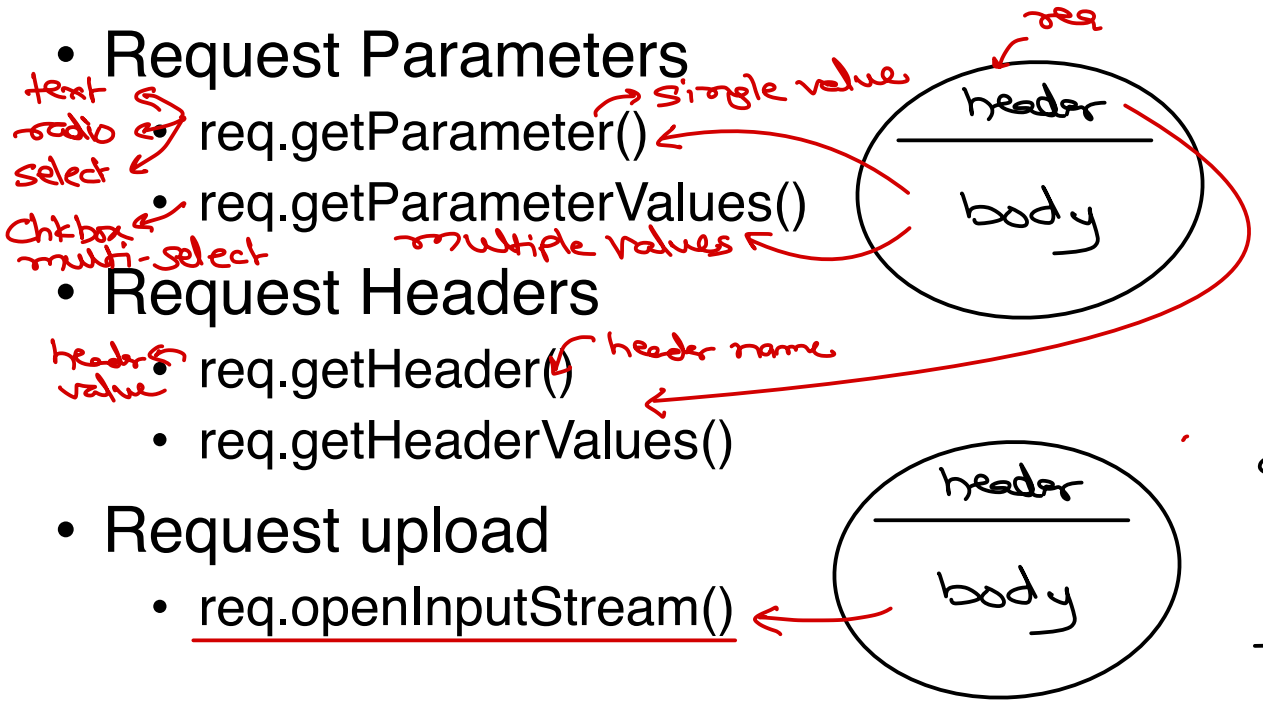
```
<servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>pkg.Hello</servlet-class>
  <init-param>
    <param-name>color</param-name>
    <param-value>red</param-value>
  </init-param>
</servlet>
<servlet-mapping>
  <servlet-name>Hello</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

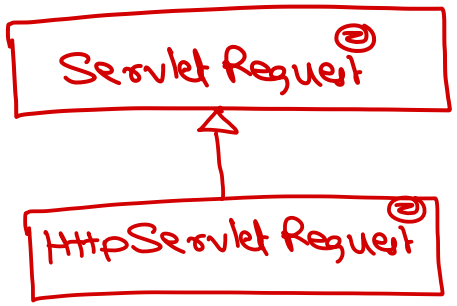
can be accessed via ServletConfig.

c = config.getInitParameter("color");

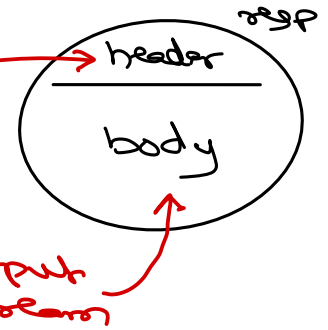
Servlets – HttpServletRequest & HttpServletResponse

- Request Parameters
  - req.getParameter()
  - req.getParameterValues()
- Request Headers
  - req.getHeader()
  - req.getHeaderValues()
- Request upload
  - req.openInputStream()

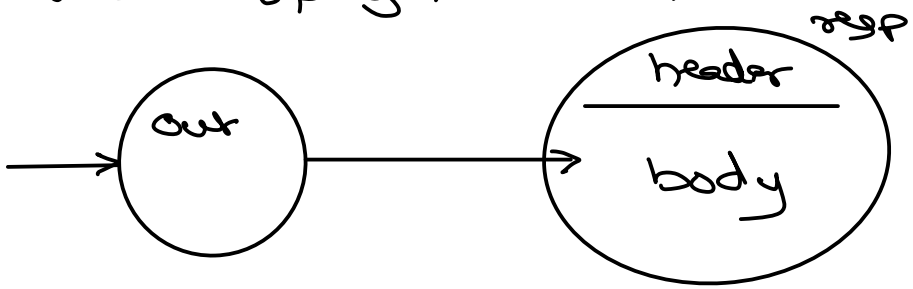


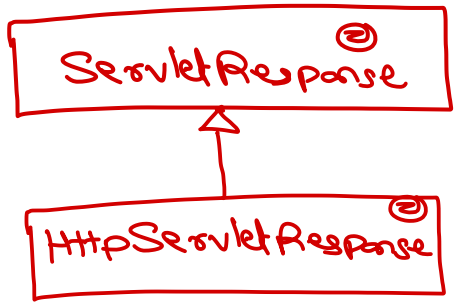


- Response content type
  - resp.setContentType()
- Response download
  - resp.openOutputStream()
- Response Body (text)

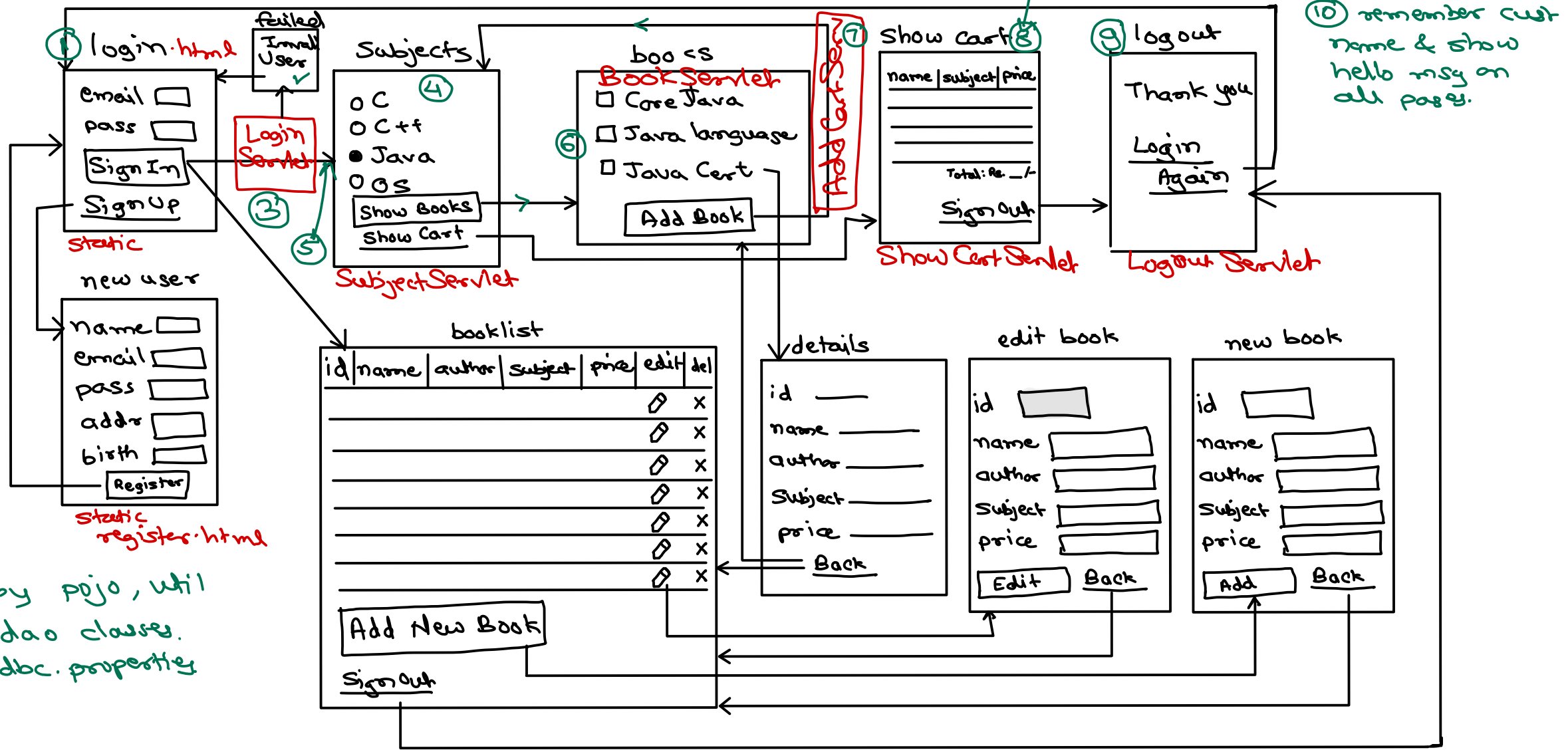



```
out = resp.getWriter()
```

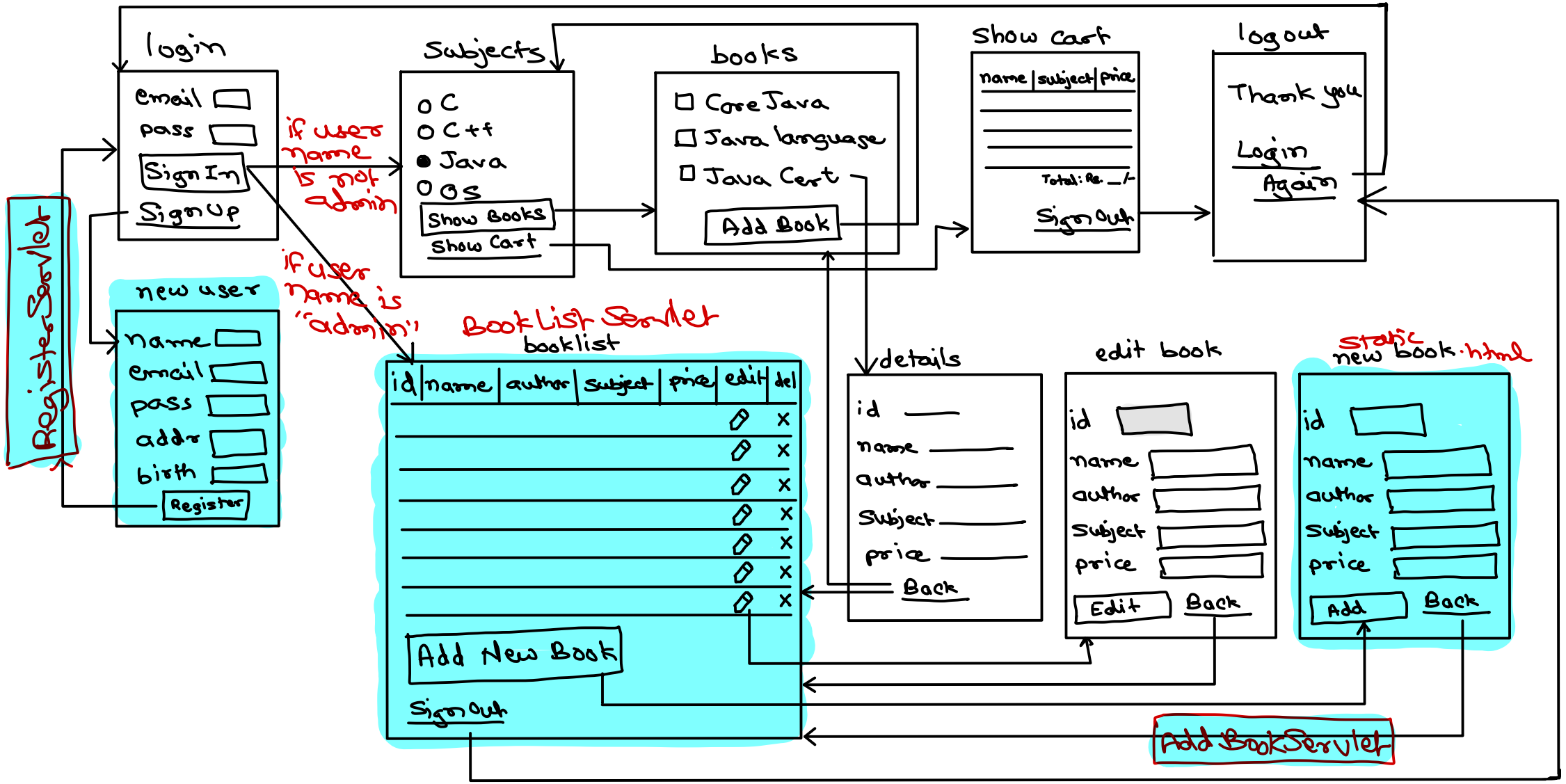




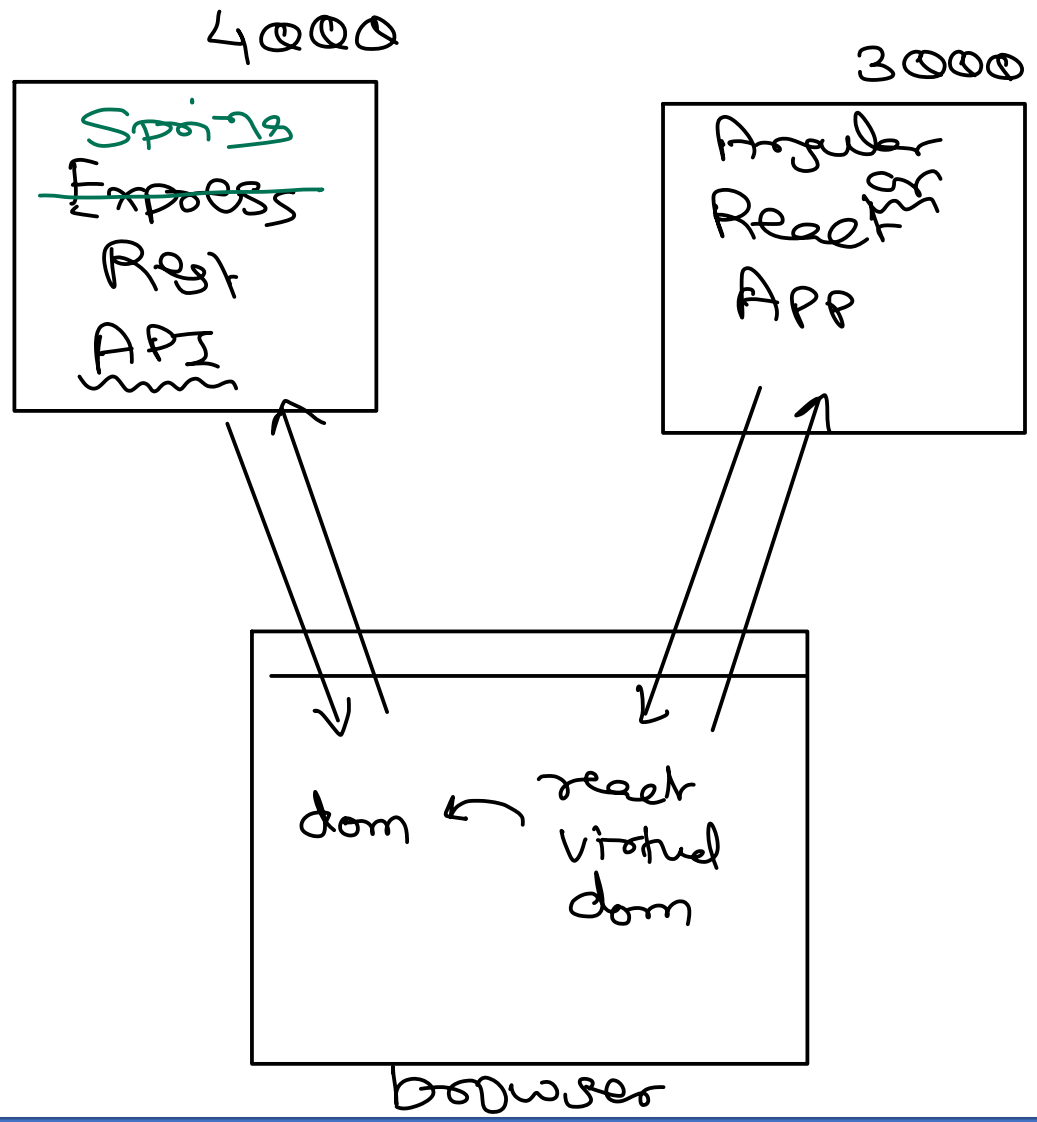
Project requirement - online book shop



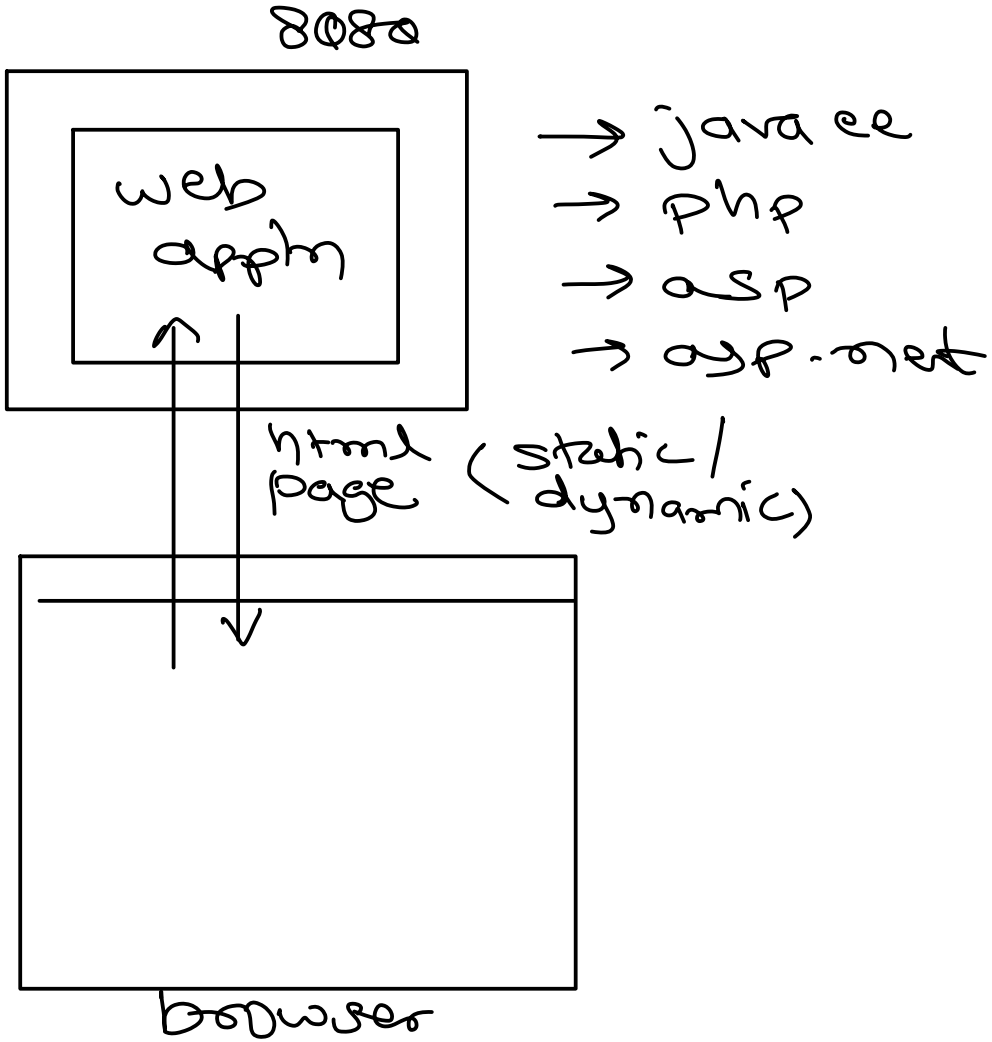
Assignment 2



React + Express/Rest app



Servlet/JSP web app

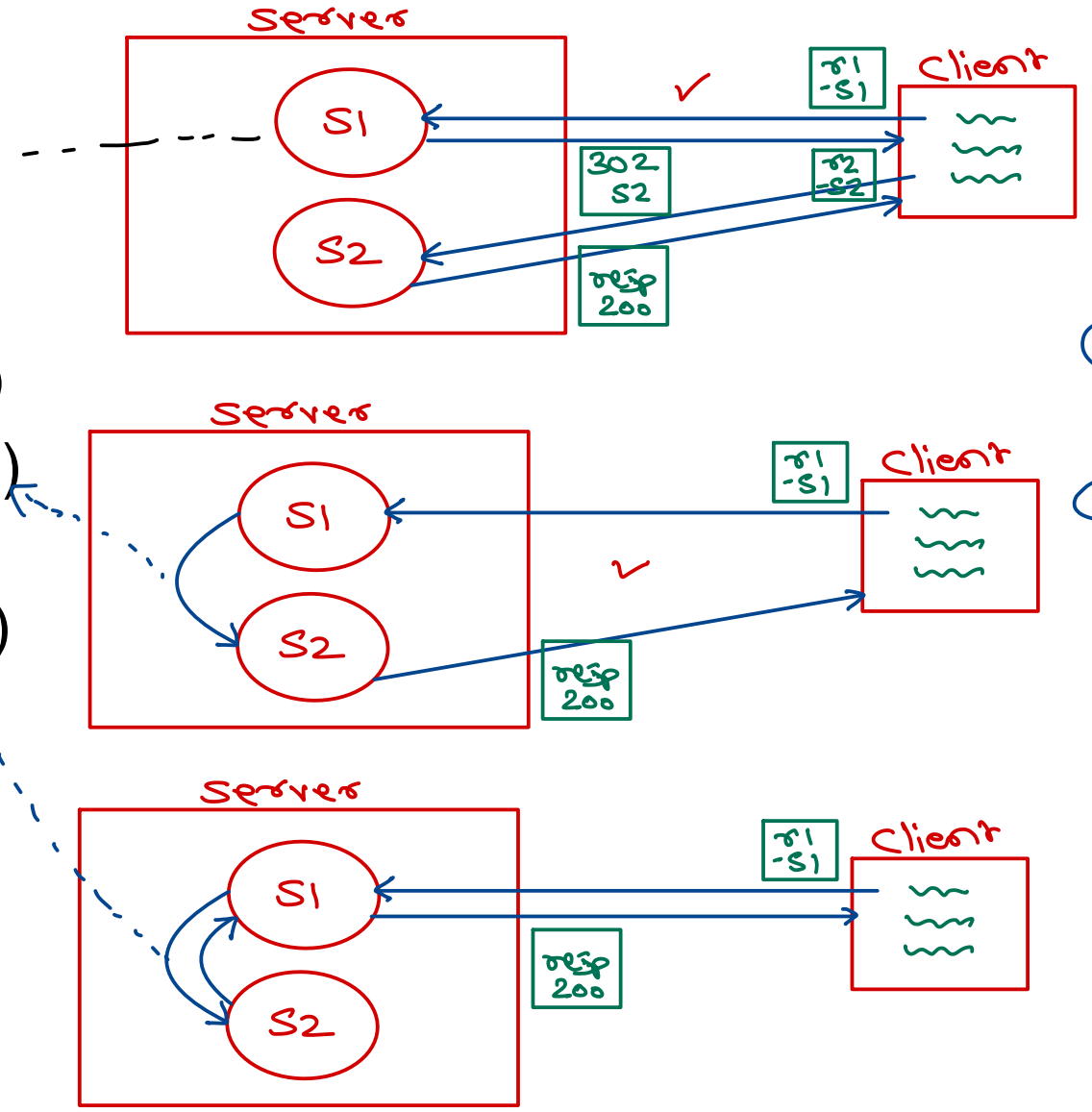


Servlet communication

- HTTP redirection
  - resp.sendRedirect("url")
- RequestDispatcher
  - req.getRequestDispatcher("url")
  - ctx.getRequestDispatcher("/url")
- RequestDispatcher – forward()
  - rd.forward(req, resp)
- RequestDispatcher – include()
  - rd.include(req, resp)

Using Request Dispatcher

another web Component in your applin (servlet, JSP, html, ...).



GET req

Using redirection

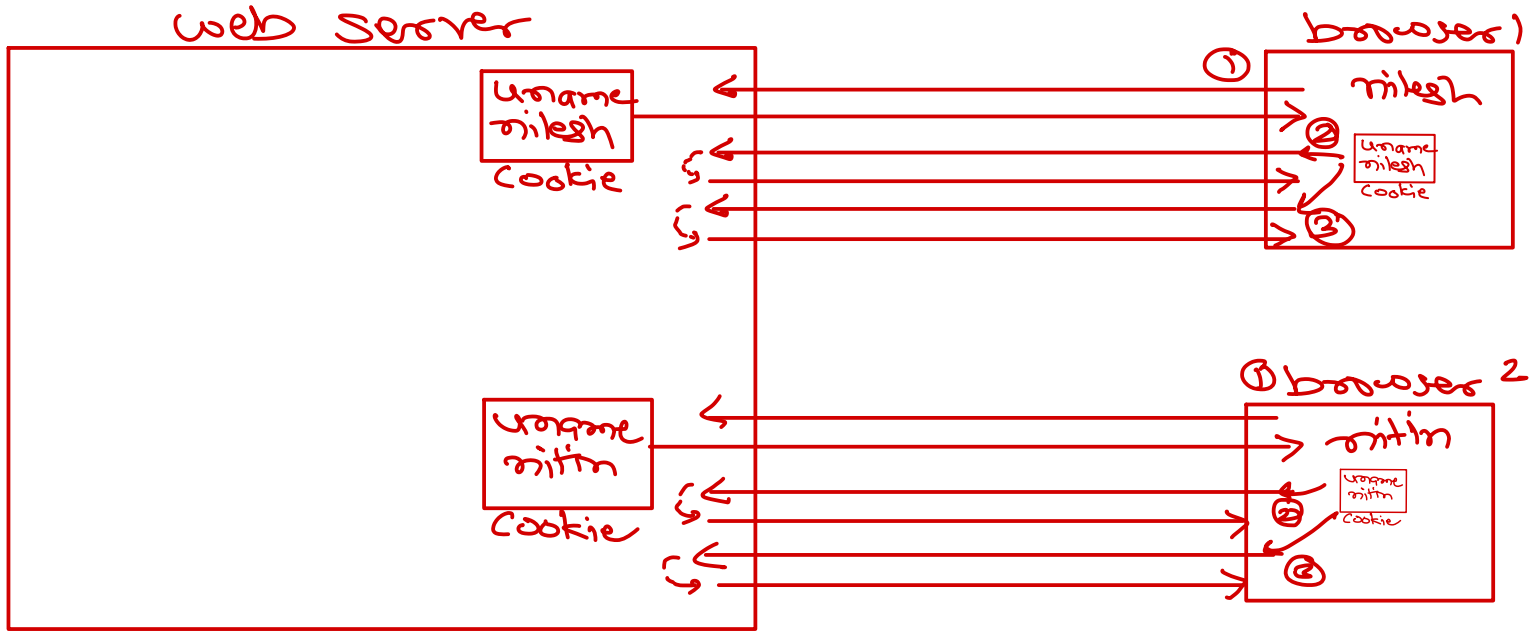
1. another web Component in your applin. (servlet, JSP, html, ...).
2. another web Component outside your applin. (e.g. www.google.com)

Controller (in MVC) internally use req dispatcher forward().

State Management

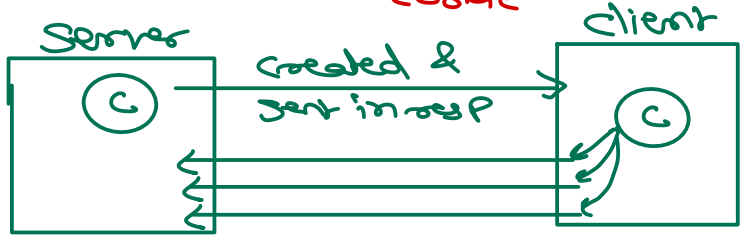
- HTTP is stateless protocol.
- State management is maintaining information of client.
- Client side state management
  - QueryString
  - Hidden form fields
  - Cookie
- Server side state management
  - Session
  - Request
  - ServletContext
- Session tracking
  - Cookie based
  - Url rewriting

State Management - Cookie



Cookie is text key-value data.

Max cookie size = 4KB.



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- Servlet init-param
- Servlet load-on-startup
- Inter-servlet communication/navigation
- State management

Annotation vs XML config

- XML based config
  - Need not to recompile the whole project.
  - If config has mistake, then deployment will fail. (No compilation in adavance).
- Annotation based config
  - Java code is compiled for syntax errors in advanced, so less chances of failure while deployment.
  - Need to recompile the project.
  - Nowadays preferred in industry.

Servlet Configuration

load-on-startup

- By default servlet class is loaded and instantiated, when first request is arrived for the servlet.
- Using `<load-on-startup>` under `<servlet>` tag, servlet can be loaded and instantiated as soon as application is deployed.
  - e.g. `<load-on-startup>1</load-on-startup>`
- A positive number given in body of tag, indicate sequence of servlet loading, if there are multiple servlets with load-on-startup tag.
- If this number is negative, servlet is loaded lazily (upon first request).
- Alternatively it can also be configured in `@WebServlet(initParams = { ... })`.

init-param

- Some configurable information can be associated with servlet using init-param under `<servlet>` tag in web.xml.
- Example:

```
<init-param>
    <param-name>DB_URL</param-name>
    <param-value>jdbc:mysql://localhost:3306/test</param-value>
</init-param>
```

- This value can be accessed in servlet class using ServletConfig object as:

```
// in init() method
String dbUrl = config.getInitParameter("DB_URL");
// ...
```

- Since GenericServlet class is inherited from ServletConfig, it can also be accessed using servlet object "this".

```
// in doGet(), doPost(), ... methods
String dbUrl = this.getInitParameter("DB_URL");
// ...
```

Inter-servlet communication/navigation

Redirection

- Can redirect from any page to any other page of the same application or different application.

```
resp.sendRedirect(url);
```

- When sendRedirect() is called, a temp response (status code 302 and destination url) is sent to the browser; due to which browser make a new request to the new link.
- In this case two requests are originated from the browser and hence browser is aware of the navigation.
- This is slower process.

RequestDispatcher

- Only one request is originated from the client, and single response is given back.

```
RequestDispatcher rd = req.getRequestDispatcher("url");
// ...
RequestDispatcher rd = ctx.getRequestDispatcher("/url");
```

```
rd.forward(req, resp);
// OR
rd.include(req, resp);
```

- This is faster than HTTP redirection.
- Browser is not aware of the navigation.
- Navigation can be done only to the pages within the same application.

Forwarding

- Request is forwarded to next servlet from which response will be given to the client.

Including:

- Request is given to next servlet, which performs some processing and return back to the calling servlet. The response generated by the second servlet will be included into first servlet's response.

State Management

State Management Classfication

Client Side State Management

- The state/info of client is stored on client machine
- Less memory is needed at the server side
- Less secure, client can access and/or modify
- Options: Cookie, QueryString, Hidden Form Fields

Server Side State Management

- The state/info of client is stored on server machine
- Large memory is needed at the server side
- More secure, client cannot directly access
- Options: Session, Application

Cookie

- To store info about the client, server send that info in textual key-value form to the client known as "cookie".
- Once cookie is received by the client, thereafter with each request cookie is sent back to the server.
- Cookie can store only text data upto 4KB.
- Cookies Types:
  - Temporary Cookie:
    - Cookie is stored in browser memory and will be destroyed when browser is closed.
  - Persistent Cookie:
    - Cookie is stored in client machine (disk) as text file and will be persisted even if browser is closed.
- To create and send cookie

```
Cookie c = new Cookie("key", "value");
// c.setMaxAge(secs); // to make cookie persistent
resp.addCookie(c);
```

- To receive cookie and get data

```
Cookie[] arr = req.getCookies();
for(Cookie c : arr) {
    if(c.getName().equals("key")) {
        String value = c.getValue();
        // ...
    }
}
```

- Cookie methods
  - Refer java docs

```
Cookie(String name, String value);
String getName();
String getValue();
```

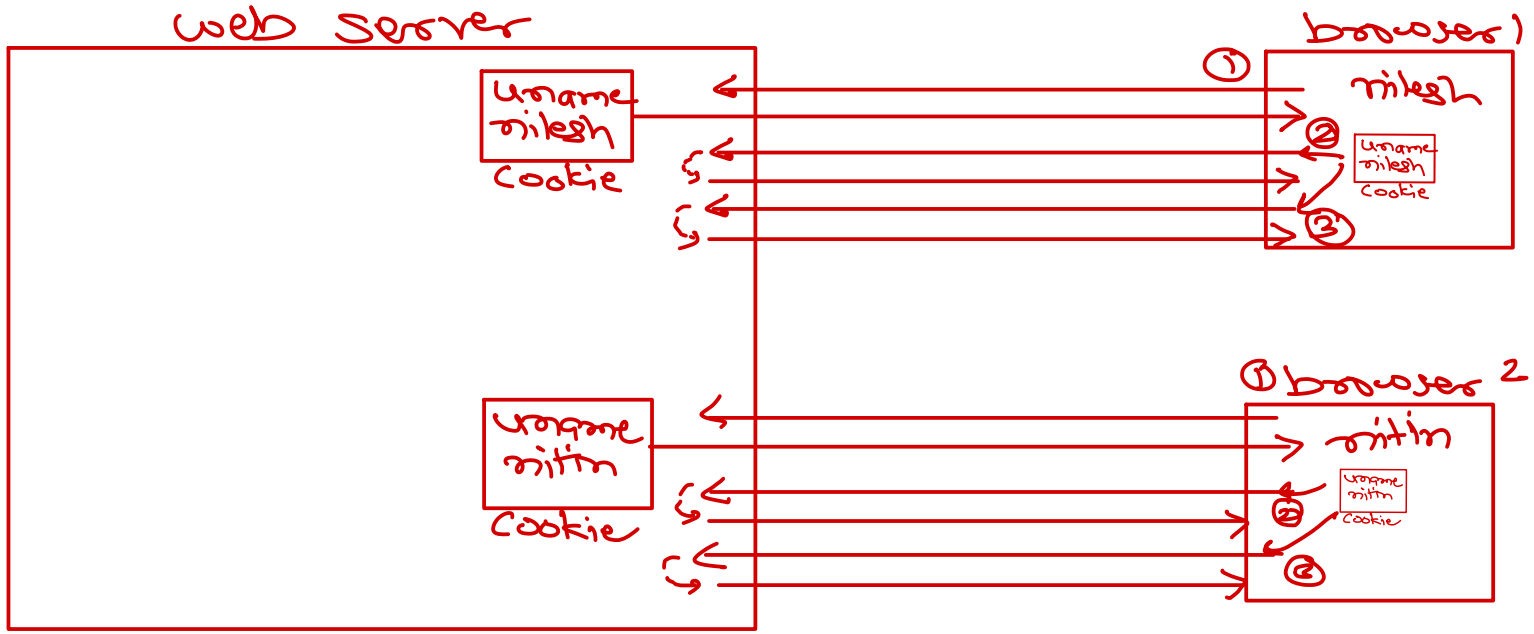
Java EE

Trainer: Nilesh Ghule

State Management

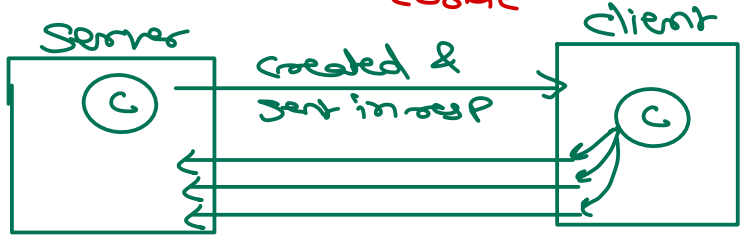
- HTTP is stateless protocol.
- State management is maintaining information of client.
- Client side state management
  - QueryString
  - Hidden form fields
  - Cookie
- Server side state management
  - Session
  - Request
  - ServletContext
- Session tracking
  - Cookie based
  - Url rewriting

State Management - Cookie

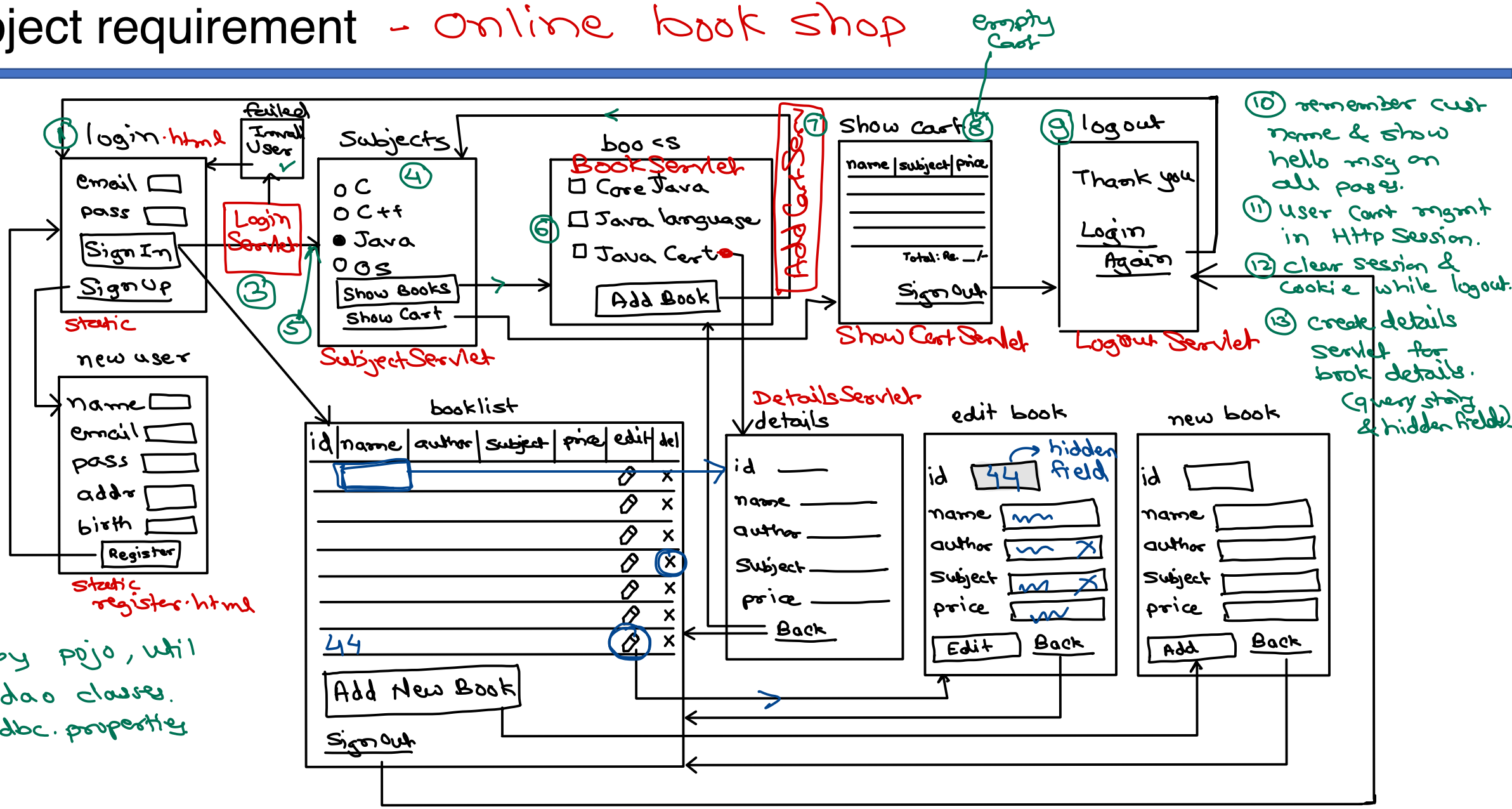


Cookie is text key-value data.

Max cookie size = 4KB.

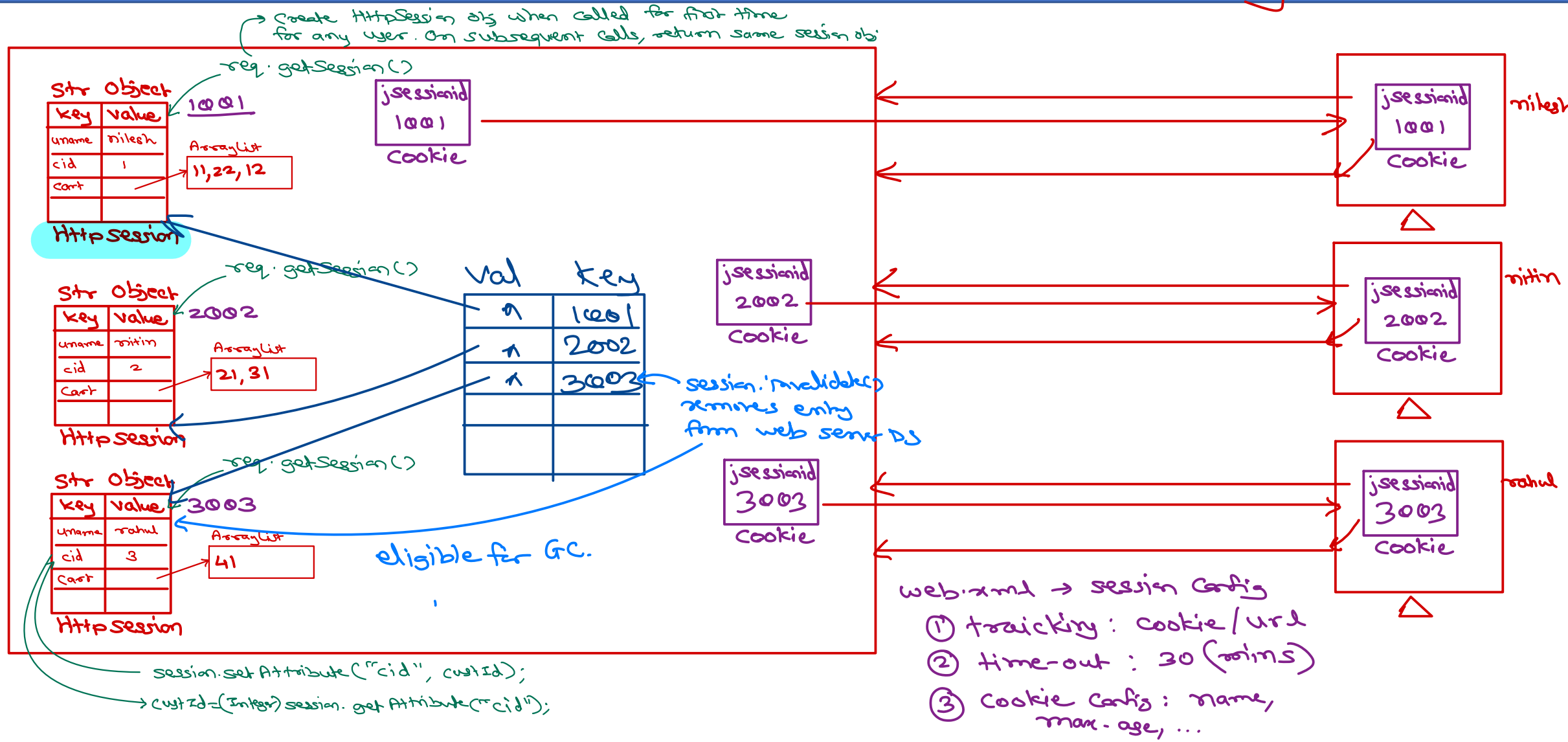


Project requirement - online book shop

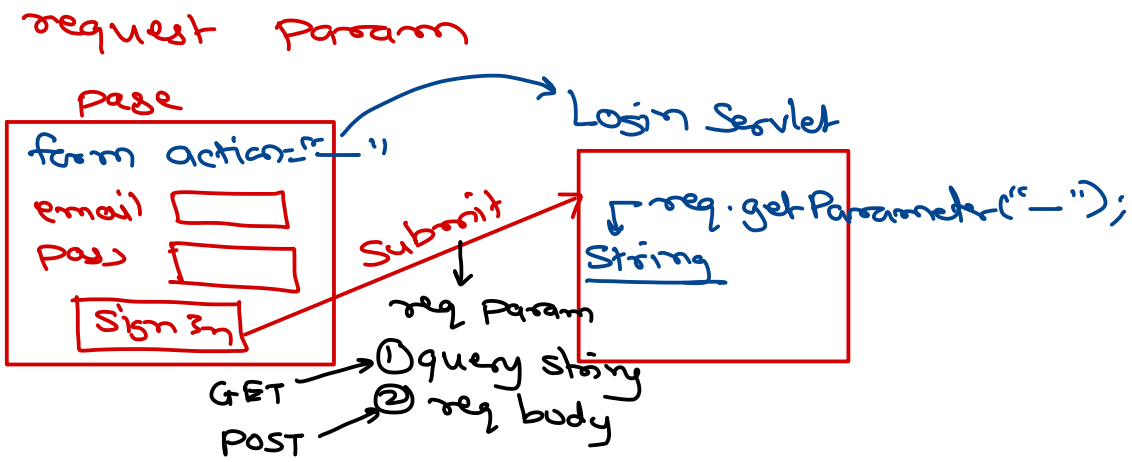


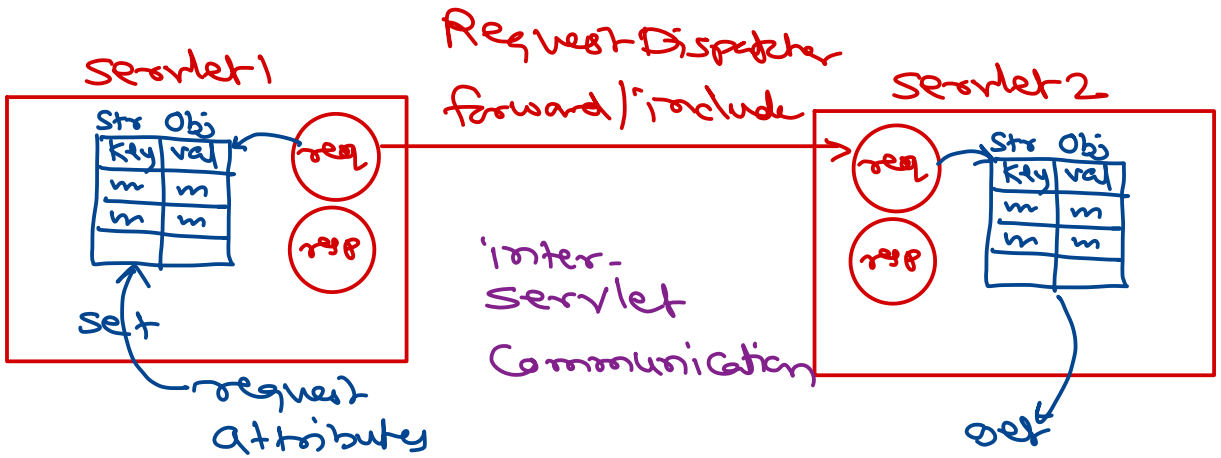
State Management - session tracking

- Cookie (default)
- Url-rewriting



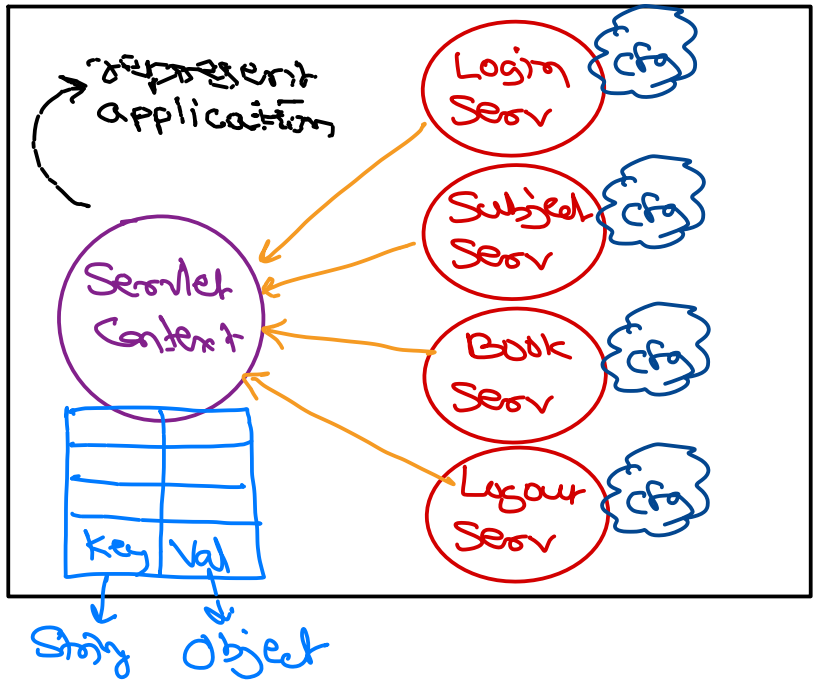
State Management





State Management

Servlet Configs




```
ctx.setAttribute("key", value);
value = ctx.getAttribute("key");
```

```
req.getServletContext()
session.getServletContext()
config.getServletContext()
this.getServletContext()
```

URL

http://localhost:8080/app/resource

X

URI

- used with ServletContext Req Dispatcher & url Pattern
- used with Request's Req Dispatcher.

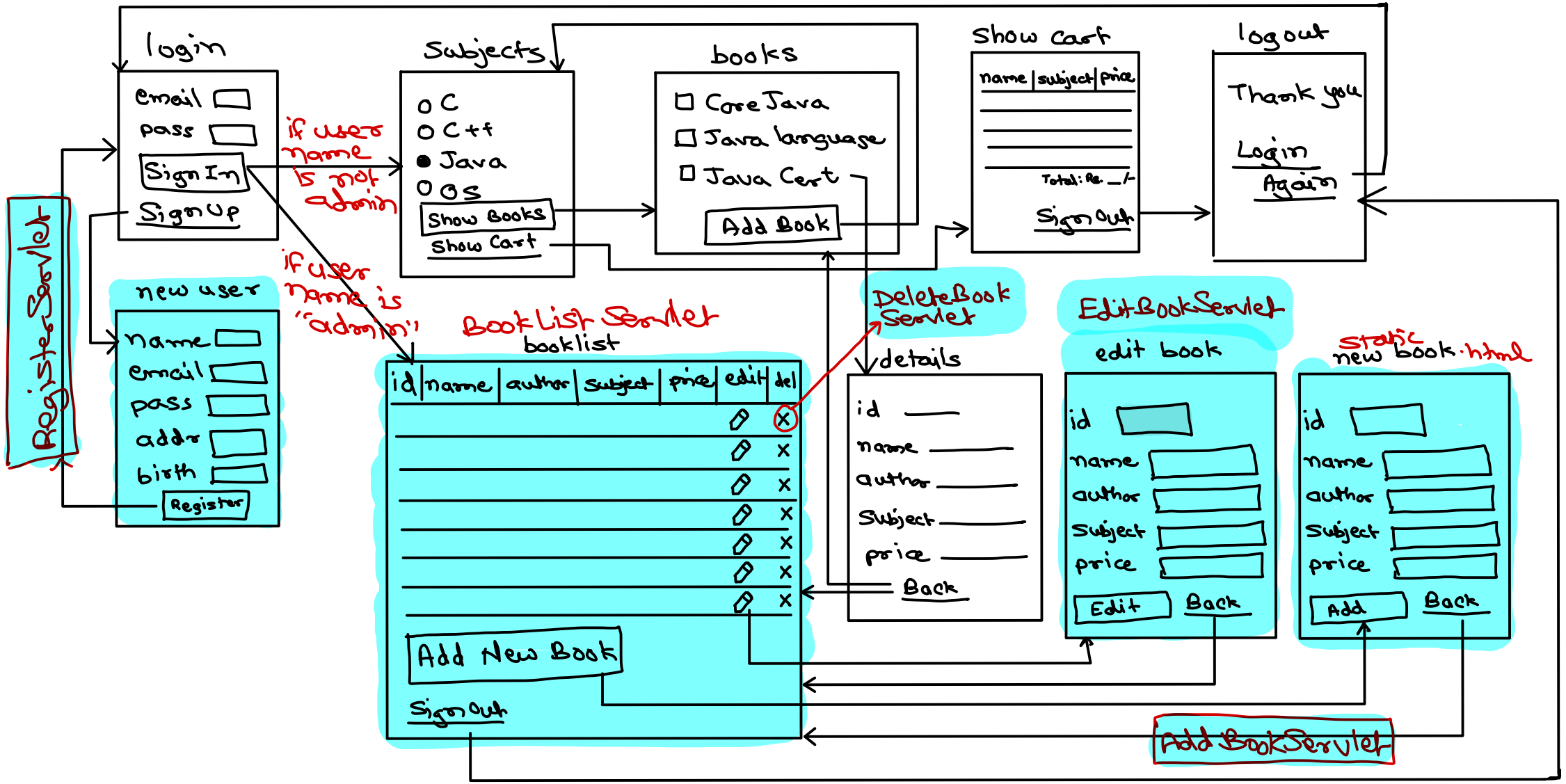
State Management

Servlet web appln

State mangement scopes

1. request : HttpServletRequest → state is accessible on all servlets to which current request is forwarded/ included.
2. session : HttpSession → state is accessible on all servlets for current user only. User specific data
3. application : ServletContext → state is accessible on all servlets for all users

Assignment 3



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- Error: tomcat v9.0 server at localhost failed to start
- State Management
  - Session
  - Session Tracking
  - Query String
  - Hidden form fields
  - Application (ServletContext)
- Filters

Error handling

- tomcat v9.0 server at localhost failed to start
- importing of Servlet packages

Cookie

```
Cookie c = new Cookie("uname", name);
c.setMaxAge(0);
resp.addCookie(c);
```

- Max age:
  - +ve value = number of seconds cookie should be persisted on client system.
  - -ve value = (-1) cookie is temporary (auto deleted when cookie when browser is closed).
  - 0 = cookie is deleted.

```
String uname = "";
Cookie[] arr = req.getCookies();
for(Cookie c: arr) {
    if(c.getName().equals("uname"))
        uname = c.getValue();
}
```

Session

- For each client one session object is created on the server side, in which client specific state/info can be saved.
- Session object is like a map, where data is stored in key/value pairs.
- The key must be a string, while value can be any object.
- HttpSession methods
  - void setAttribute(String key, Object value);

    - Object getAttribute(String key);
    - void invalidate();
    - boolean isNew();
    - String getId();
- To get session object
    - HttpSession session = req.getSession();
- to change session timeout: web.xml in `<web-app>`

```
<session-config>
    <session-timeout>10</session-timeout>
</session-config>
```

Session Tracking

- Each session is identified using a unique session id, which is associated with the client.
- There are two ways of this association (tracking):
    - cookie based
    - url rewriting

Cookie based

- By default, when new session is created (req.getSession() is called first time) a cookie is created and sessionid is sent to client via that cookie.
- For suqsequent calls to req.getSession() access the appropriate session object by getting sessionid from that cookie.

Url rewriting

- In case cookies disabled, sessionid can be maintained using url.
- resp.encodeURL() and resp.encodeRedirectURL() methods are used to embed sessionid into the url.
    - e.g. http://server:port/app/page;jsessionid=374334

ServletContext

- For each web application, web container creates a singleton object called as "ServletContext" during application deployment.
- This object is used
    - To store the data globally so that it can be accessed for all request to all pages (servlets) by all users.
    - To navigate from one page (servlet) to another using RequestDispatcher. (refer notes of RequestDispatcher)
    - To access application config params (context params).
- ServletContext object can be accessed using
    - req.getServletContext()

    - session.getServletContext()
    - config.getServletContext()
    - this.getServletContext() // In Servlet class

- To save and retrieve data from the ServletContext
    - void setAttribute(String key, Object value);
    - Object getAttribute(String key);

- Context Parameters
    - ServletConfig can be used to get init param for specific servlet (from web.xml).
    - In web.xml we can declare the values which can be accessible in entire application in form of context params:

```
<web-app>
    // ...
    <context-param>
        <param-name>color</param-name>
        <param-value>yellow</param-value>
    </context-param>
    // ...
</web-app>
```

- This param can be accessed into the web application via ServletContext object as:
    - String colorValue = context.getInitParameter("color");

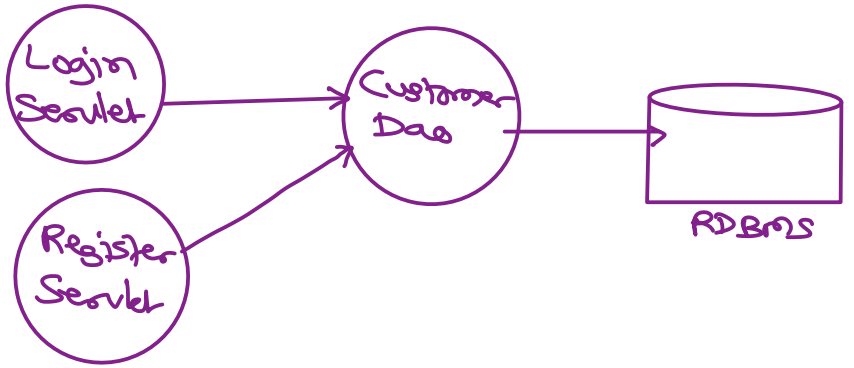
Java EE

Trainer: Nilesh Ghule

Filters

- Filters is way of implementing AOP in Java EE applications. Filters are used to perform pre-processing, post-processing or both for each request.
- Multiple filters can be executed in a chain/stack before/after handling request.
- javax.servlet.Filter interface is used to implement Filters.
  - void init(FilterConfig filterConfig);
  - void doFilter(ServletRequest req, ServletResponse resp, FilterChain chain);
  - void destroy();
- Can be configured with @WebFilter or in web.xml (similar to servlets).

AOP
Aspect Oriented Programming



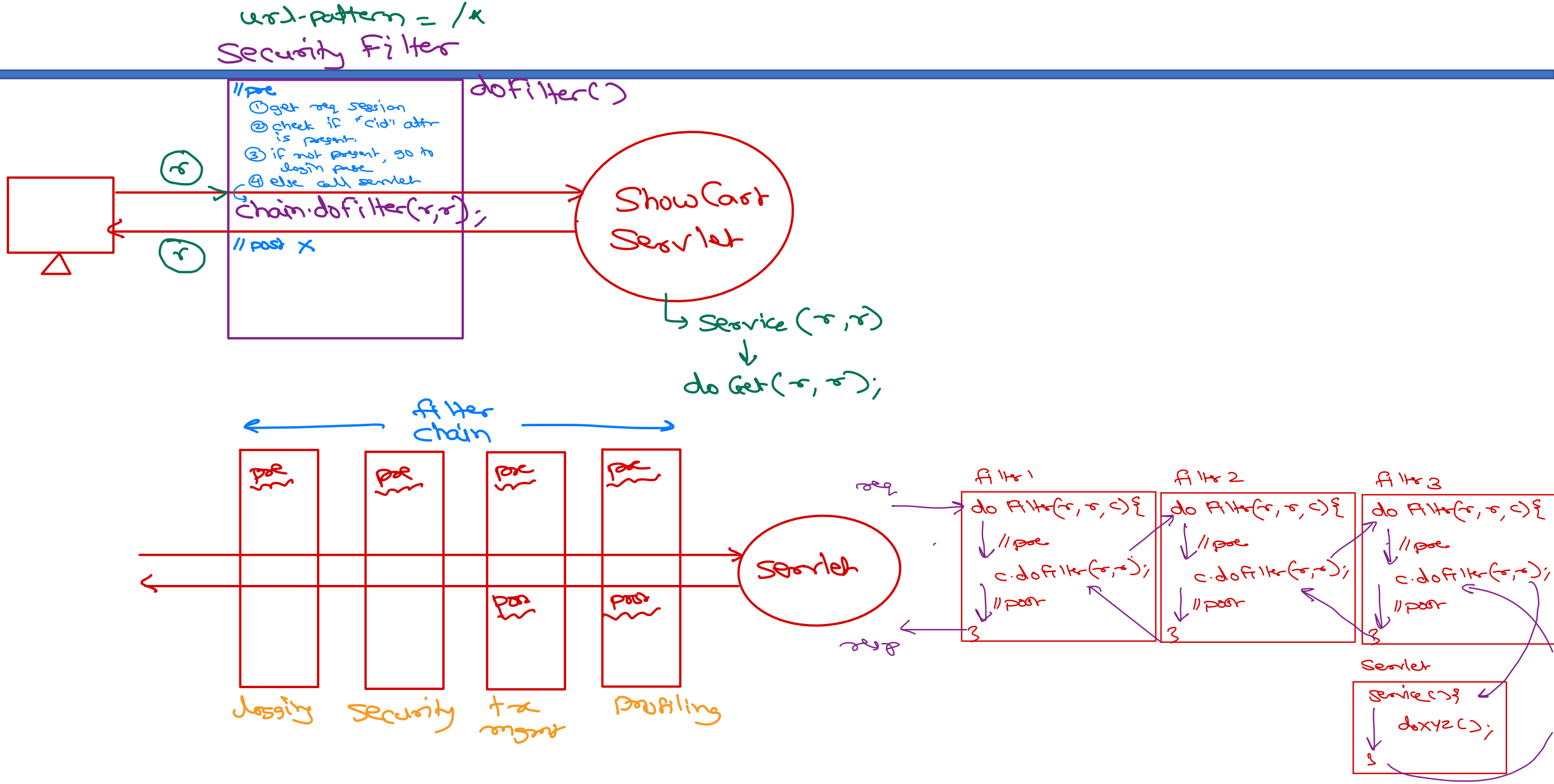
- AOP is implementation of cross cutting concerns without modifying business logic.
- Cross-cutting concerns = extra functionality
  - logging → pre or post
  - monitoring → post
  - profiling → pre and post
  - security → pre

success → commit()
exception → rollback()
pre and post ← tx management
autoCommit(false)

time - pstr = diff

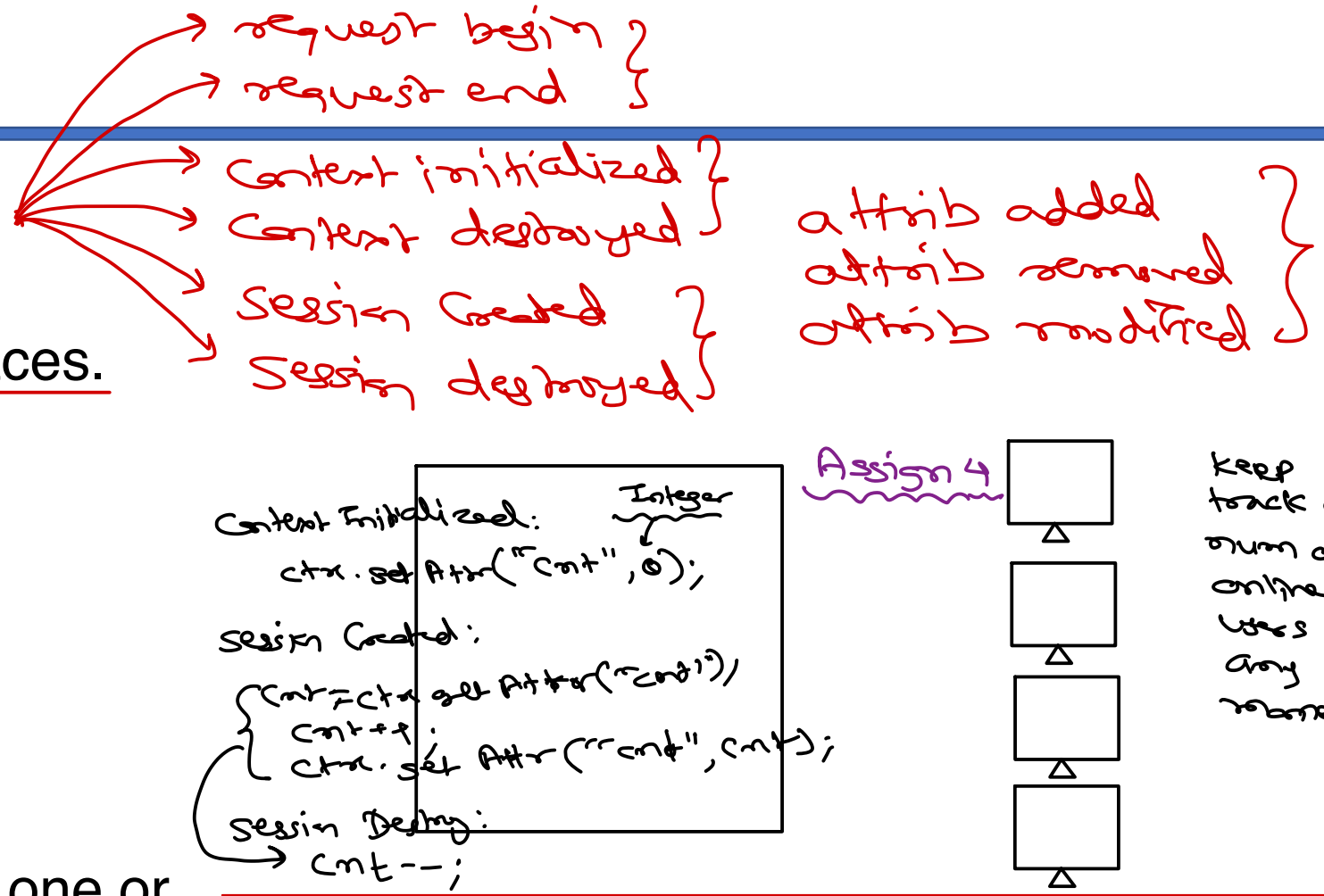
- CCC impl → Aspect → Advice
  - Aspect → class
  - Advice → method
    - pre-processing
    - post-processing
    - pre & post

(Java) AOP
- Java proxy
- AspectJ
- Spring AOP
- JavaEE filters


url-pattern = /*
Security Filter
doFilter()
//pre
1 get req session
2 check if "cid" attr is present.
3 if not present, go to login page
4 else call servlet
chain.doFilter(r,r);
//post
ShowCart Servlet
service(r,r)
doGet(r,r);
filter chain
pre
post
logging
security
tx mngmt
profiling
servlet
req
resp
filter 1
filter 2
filter 3
doFilter(r,r,c){
//pre
c.doFilter(r,r);
//post
Servlet
service(){
doXyz();

Listeners

- Listeners are used to handle application level events.
- There are many listener interfaces.
  - ServletContextListener
  - HttpSessionListener
  - ServletRequestListener
  - ServletContextAttributeListener
  - HttpSessionActivationListener
  - HttpSessionAttributeListener
  - ServletRequestAttributeListener
- Listener class must implement one or more listener interface.
- Can be configured with @WebListener or in web.xml.




```
<listener>
  <listener-class>pkg.MyListener</listener-class>
</listener>
```

Java Server Pages

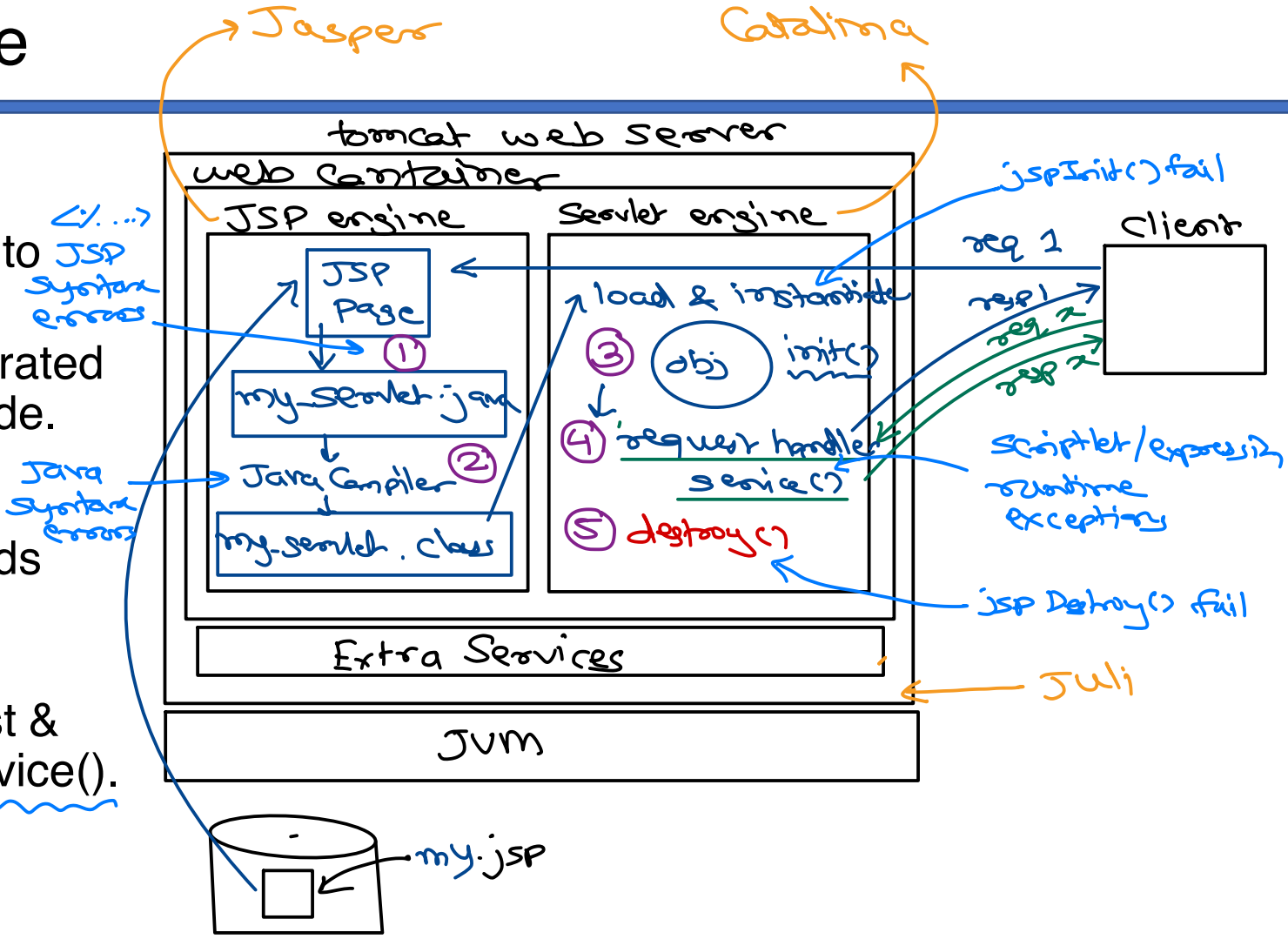
Sunbeam Infotech

Java Server Pages

- Servlet = Business logic + Presentation logic
- JSP = Presentation logic + Business logic
- JSP is converted into the servlet while execution.
- JSP syntax
  - Directive <%@ … %>
  - Declaration <%! … %>
  - Scriptlet <% … %>
  - Expression <%= … %>
  - Comment <%-- … --%>
- JSP Directives
  - <%@page ... %>
  - <%@include ... %>
  - <%@taglib ... %>

Java Server Pages – Lifecycle

- JSP Engine
  1. Translation stage: Converts JSP into servlet java class.
  2. Compilation stage: Compiles generated servlet java class into java byte code.
- Servlet Engine
  3. Loading & Instantiation stage: Loads servlet class into JVM & create its object. Invokes jspInit().
  4. Request handling: Handles request & produce response. Invokes jspService(). For each request.
  5. Destruction stage: De-initialize the object. Invokes jspDestroy().



JSP – @page directive

- <%@page language = “java"%>
- <%@page import = "java.util.Date"%>
- <%@page contentType = "text/html" %>
- <%@page session = “true"%>
- <%@page isErrorPage = "false"%>
- <%@ page errorPage = "error.jsp" %>
- <%@page info = “This is hello JSP"%>
- <%@page buffer = “8"%>
- <%@page autoFlush = "false"%>
- <%@page extends = “javax.servlet.http.HttpServlet"%>
- <%@page isThreadSafe = “true"%>
- <%@page isELIgnored = “false"%>

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- Listeners
- Filters
- JSP
- JSP Life cycle
- ~~JSP directives~~
- ~~JSP implicit objects~~
- ~~JSP standard actions~~

Listeners

- These interfaces are used to handle events in the web application e.g. application start (context initialization), application stop (context uninit), session start, session end, new value added into session, any value removed from session, etc.
- interface ServletContextListener
    - void contextInitialized(ServletContextEvent e);
    - void contextDestroyed(ServletContextEvent e);
- interface HttpSessionListener
    - void sessionCreated(HttpSessionEvent e);
    - void sessionDestroyed(HttpSessionEvent e);
- interface HttpSessionAttributeListener:
    - void attributeAdded(HttpSessionBindingEvent e);
    - void attributeRemoved(HttpSessionBindingEvent e);
    - void attributeReplaced(HttpSessionBindingEvent e);
- interface HttpSessionActivationListener:
    - Refer docs.
- interface HttpSessionBindingListener:
    - Refer docs.
- interface ServletContextAttributeListener:
    - Refer docs.
- interface ServletRequestListener:
    - Refer docs.
- interface ServletRequestAttributeListener:

    - Refer docs.

- How to use listeners?
- step1: write a class implementing required listener
- step2: write the appropriate logic into appropriate method
- step3: inform web container about this listener in web.xml or using annotation @WebListener

Filters

- Filter is way of AOP in Java EE application.
- Can perform pre-processing and/or post-processing while accessing any servlet/jsp.
- Multiple filters can be executed in a chain.
- Filter life cycle (is similar to Servlet life cycle).
    - init(): Object is created when application is deployed and init() is called for one-time initialization.
    - destroy(): Executed when server shut-down for one-time de-initialization.
    - doFilter(): Pre-processing, invoke next component in chain and post-processing.
- Configured using @WebFilter annotation (refer docs) or in web.xml

```
<filter>
    <filter-name>SecurityFilter</filter-name>
    <filter-class>sunbeam.filters.SecurityFilter</filter-class>
</filter>
<filter-mapping>
    <filter-name>SecurityFilter</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

- Typically URL pattern is corresponds to some servlet. When servlet is requested, the filter doFilter() is invoked.
- URL may include wild-card /* indicating all the servlets.

JSP

- JSP is outdated.
- Simplest JSP page = HTML page.
- JSP is internally converted into servlets while execution.

JSP Life cycle

- step 1. Translation Stage:
    - When first request is made for the JSP page, it will be loaded by the web container inside JSP engine.
    - JSP engine translate the JSP page into a servlet's java code. This .java file can be found in tomcat's "work" folder.
    - If there is any error in JSP syntax (e.g. scriptlets), then this stage fails.
- step 2. Compilation Stage:
    - The translated servlet's .java file will be compiled into a .class file at runtime.
    - If there is any java syntax error, then this stage fails.

- step 3. Instantiation (Loading) Stage:
  - The .class file will be loaded and object of the translated servlet will be created.
  - Immediately after this init method of the JSP i.e. jspInit() will be executed.
  - If this method throws any exception, this stage fails.
  - This stage is also called as Loading or Initialization stage.
- step 4. Request Handling Stage:
  - All above stages are done only for the first request of the JSP file; However this stage is executed for each request.
  - For each request, _jspService() method is executed (which is made up of all the scriptlet and expressions in the JSP file).
- step 5. Destruction Stage:
  - When servlet object is no longer used or web container is going down, jspDestroy() method will be executed after which servlet object will be garbage collected.

JSP syntax

- directive `<%@ .... %>`
  - `<%@page language="java" import="java.util.*, java.io.*" ..%>`
    - mainly controls servlet translation
  - `<%@include file="file.ext" %>`
    - adding external html or jsp file statically.
  - `<%@taglib .... %>`
    - used for custom and third-party tags
- declaration `<%! .... %>`
  - used to declare fields and methods which will not be executed per request. e.g. jspInit(), jspDestroy(), other methods, all fields, etc.
  - can write one or more declaration blocks
- scriptlet `<% .... %>`
  - used to write java statement(s) to be executed per request.
  - all code written here will become part of _jspService() method during translation phase.
- expression `<%= .... %>`
  - the java expressions whose result (string) will be directly embedded into generated html.
  - Executed for each request.
  - will become part of _jspService() method during translation phase.
- comment `<%-- .... --%>`
  - Server side comment -- not visible to the client - HTML source
  - `<!-- ... -->` HTML client comment -- visible to the client - HTML source
- JSP Example

```
class HelloServlet extends HttpServlet {
    private int count;
    @Override
    public void init(ServletConfig config) throws ServletException {
        count = 0;
        System.out.println("HelloServlet.init() called");
    }
    @Override
    public void destroy() {
        System.out.println("HelloServlet.destroy() called");
    }
    @Override
    public void service(HttpServletRequest req, HttpServletResponse resp)
throws ServletException, IOException {
        count++;
        resp.setContentType("text/html");
        PrintWriter out = resp.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Even Odd</title>");
        out.println("</head>");
        out.println("<body>");
        if(count % 2 == 0)
            out.println("<h4>Even Count = " + count + "</h4>");
        else
            out.println("<h4>Odd Count = " + count + "</h4>");
        out.println(new Date().toString());
        out.println("</body>");
        out.println("</html>");
    }
}
```

```
<%@ page contentType="text/html" import="java.util.Date" %>
<html>
    <head>
        <title>Even Odd</title>
    </head>
    <body>
        <%! // data/method members of generated servlet code (declaration)
            private int count;
        %>

        <%! // data/method members of generated servlet code (declaration)
            public void jspInit() {
                count = 0;
                System.out.println("HelloServlet.init() called");
            }
            public void jspDestroy() {
                System.out.println("HelloServlet.destroy() called");
            }
        %>
```

```
        <%-- scriptlet syntax contains java statement(s) to be executed for
each request --%>
        <%
            count++;
            System.out.println("service() is called.");
        %>

        <%-- expression syntax contains java expression whose result will be
embedded in produced html --%>
        <% if(count % 2 == 0) { %>
            <h4>Even Count = <%= count %> </h4>
        <% } else { %>
            <h4>Odd Count = <%= count %> </h4>
        <% } %>

        <%
        out.println(new Date().toString());
        %>
        <br/>
        <%= new Date().toString() %>
    </body>
</html>
```

JSP good practices

- Never mix mark-up code and java code.
- JSP with scriptlets, declarations and expressions are considered to be BAD jsp.
- Ideal JSP is always zero-scriptlet JSP.
- JSP --> presentation logic
- Java Beans --> business logic

Assignments

- Maintain count of online users. Add a page for admin login to see number of online users.
- Complete the Bookshop assignment with SecurityFilter.
- Modify SecurityFilter to ensure that Admin pages are not accessible to the customers.
- Execute hello jsp using Eclipse. (Create it under webapp directory).

Java EE

Trainer: Nilesh Ghule

Java Server Pages

Sunbeam Infotech

JSP – @page directive

- <%@page language = “java"%>
- <%@page import = "java.util.Date"%>
- <%@page contentType = "text/html" %>
- <%@page session = “true"%>
- <%@page isErrorPage = "false"%>
- <%@ page errorPage = "error.jsp" %>
- <%@page info = “This is hello JSP"%>
- <%@page buffer = “8"%>
- <%@page autoFlush = "false"%>
- <%@page extends = “javax.servlet.http.HttpServlet"%>
- <%@page isThreadSafe = “true"%>
- <%@page isELIgnored = “false"%>

JSP – Implicit objects

- request: HttpServletRequest
- response: HttpServletResponse
- session: HttpSession
- out: ~~PrintWriter~~ JspWriter
- application: ServletContext
- config: ServletConfig
- pageContext: PageContext
- page: Object → represents current servlet object "this". Mainly used for synchronization
- exception: Throwable
  - ↳ available only in error page.
    isErrorPage = true

Same as Servlets

JSP Scopes
1. page → cur page cur request
2. request
3. session
4. application

Same as Servlet.

→ holds all essential objects to produce response e.g. request, response, out, session, context, buffer, config, ...

→ Can store page level attributes

pageContext.setAttribute()
pageContext.getAttribute()

synchronized (page) {
=
}

JSP – Standard actions

Part of JavaEE spec.
Standard prefix: jsp

- <jsp:forward page=… /> → RequestDispatcher.forward()
- <jsp:include page=… /> → RequestDispatcher.include() → dynamic inclusion @ request handling stage
- <jsp:param name=… value=… /> → child tag to be used with forward & include to send additional data.
  - name: String (req Param)
  - value: String
- <jsp:plugin type="applet" … />
- <jsp:fallback/>

Applet - Java class that downloaded at client side and execute into client browser JRE (plugin)

if plugin loading fails, display fallback.

- <jsp:element name = "xmlElement">
- <jsp:attribute name = "xmlEleAttr">
- <jsp:body>…</jsp:body>

produce xml output from JSP page.
contentType="text/xml"

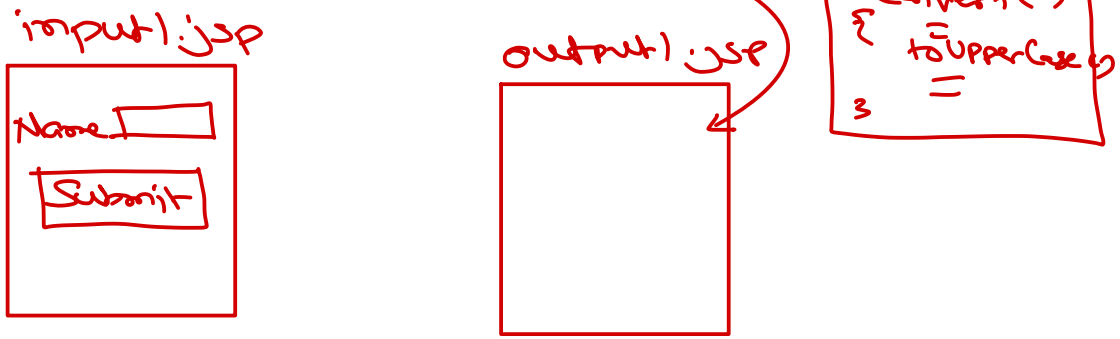
```
<element attr="val">
   body
</element>
```

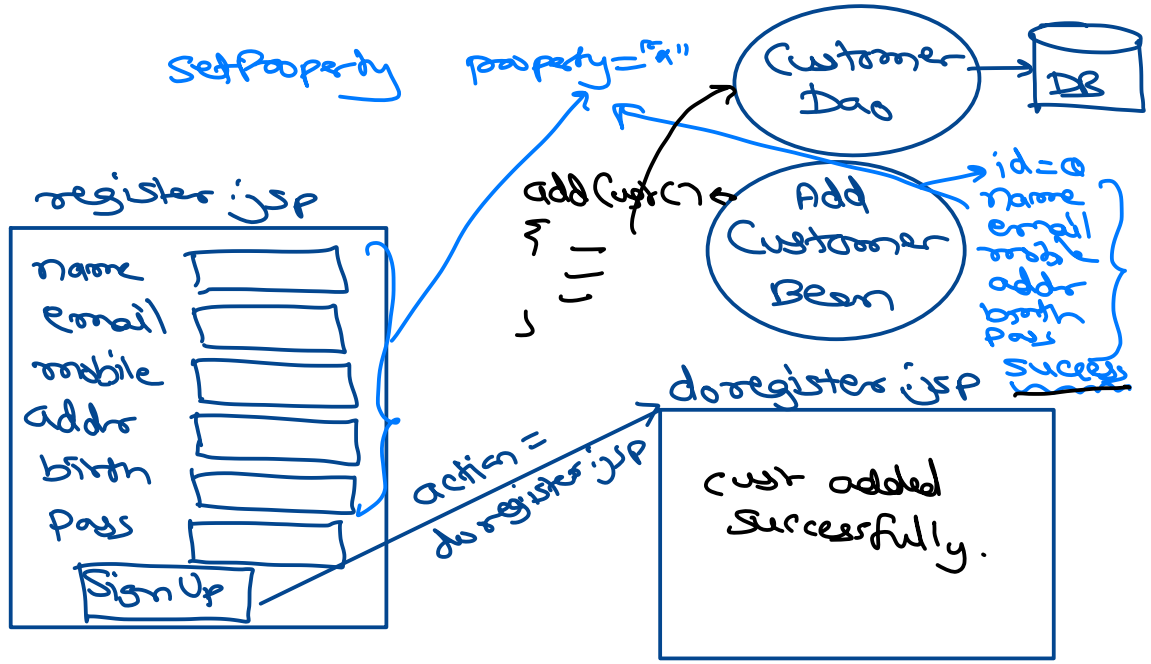
- <jsp:text>…</jsp:text>

JSP – Java beans

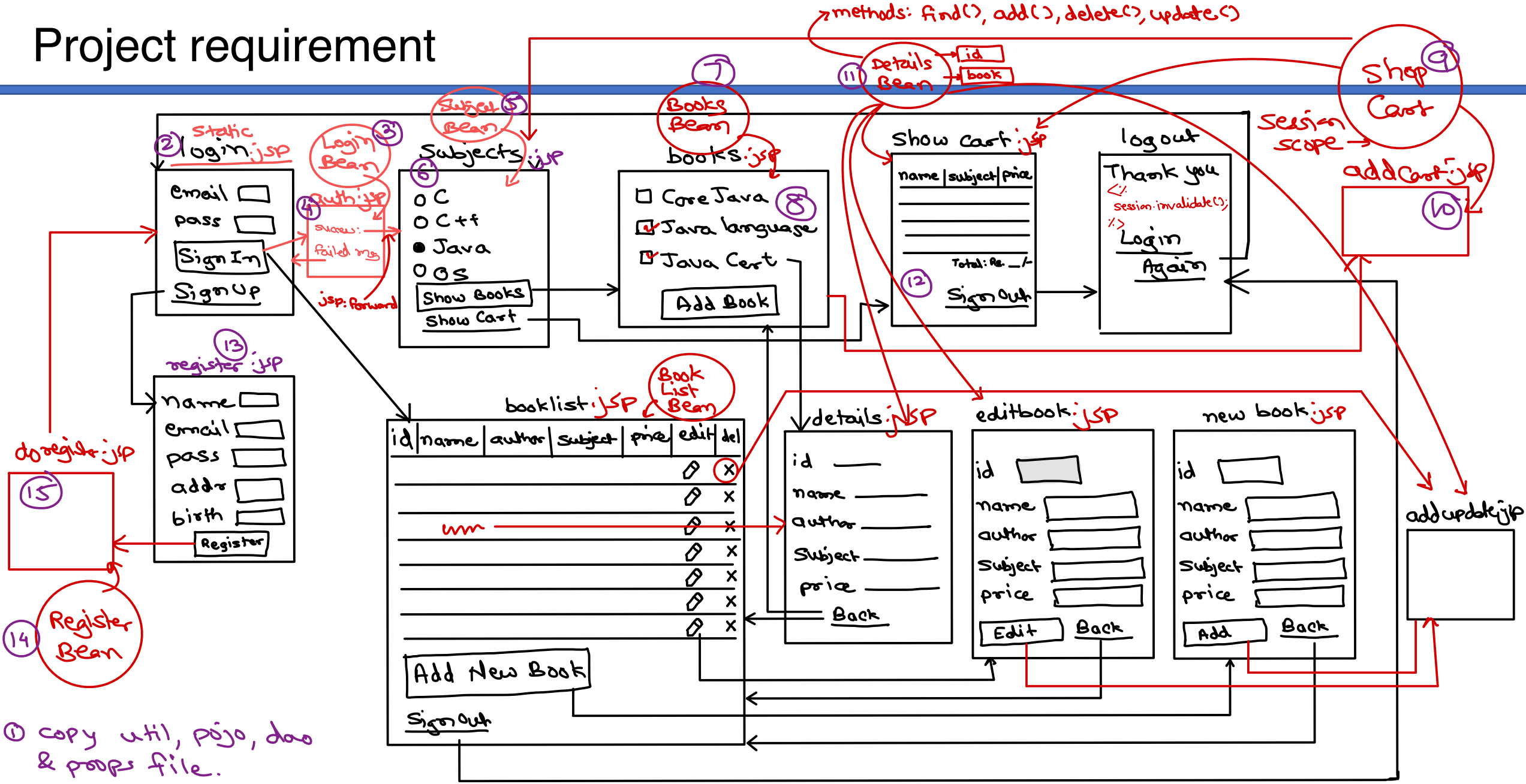
- Java beans are simple java classes which contain constructor, fields, getters/setters and one/more business logic methods.
- Java beans used in JSP pages using
  - <jsp:useBean id="var" class="pkg.BeanClass" scope="…"/>
  - <jsp:setProperty name="var" property="... " value="…"/>
  - <jsp:setProperty name="var" property="... " param="…"/>
  - <jsp:setProperty name="var" property="*"/>
  - <jsp:getProperty name="var" property="... "/>
- Java beans objects are created & accessed using reflection.

- Java bean scopes
  - page – PageContext attribute (default)
  - request – Request attribute
  - session – HttpSession attribute
  - application – ServletContext attribute
- jsp:useBean
  - Check if object with given name is present in given scope (using getAttribute()). If available, access it.
  - If not available create new bean object.
  - Set the object into given scope (using setAttribute()).
- jsp:setProperty, jsp:getProperty must be preceded by jsp:useBean.


input1.jsp
Name
Submit
UpperCaseBean1
userName
result
convert()
toUpperCase()
output1.jsp


setProperty
property="*"
CustomerDao
DB
register.jsp
name
email
mobile
addr
birth
pass
SignUp
addCust()
AddCustomerBean
id=0
name
email
mobile
addr
birth
pass
success
action=
doregister.jsp
doregister.jsp
cust added
successfully.

Project requirement



① copy util, pojo, dao & props file.

JSP – Expression Language

- Syntax: ${…}
- Used to
  - Access scoped objects
    - ${scopeName.objName}
    - pageScope, requestScope, sessionScope, applicationScope
    - Auto search from lowest to highest scope
  - Access fields (via getters)
    - ${objName.fieldName}
  - Access methods
    - ${objName.method()}
  - Perform arbitrary calculations
    - ${2 + 3 * 4 mod 5}
  - Use with EL implicit objects

- EL implicit objects
  - ${param.name}
  - ${paramValues.name}
  - ${header.name}
  - ${headerValues.name}
  - ${initParam.name}
  - ${cookie.name}
  - ${pageContext….}

JSP – JSTL

by Sun-microsystem / oracle.

- Standard JSP Tag Library
- Five components
  - Core → programming constructs if, loop, switch, ...
  - Formatting → formatting currency, date-time.
  - SQL → not recommended → can execute SQL queries on db. (no need of dao).
  - XML → xml generation / parsing.
  - Functions → utility fns like concat(), length(), ...
- <%@taglib prefix = "c" uri = "http://java.sun.com/jsp/jstl/core" %>
  - <c:if .../> if
  - <c:choose .../> if-else, switch-case
  - <c:forEach .../> loop
  - <c:set .../> variable in a scope.
  - <c:url .../> url encoding / re-writing
  - <c:redirect .../> http redirection

① add jstl.jar in project classpath.
WEB-INF/lib
OR tomcat/lib
② use <%@ taglib %> in jsp where you want to use jstl tags.
③ use appropriate jstl tags.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- JSP directives
- Custom Error Pages
- JSP Implicit objects
- JSP standard actions
- Java beans
- JSP Expression language
- JSTL
- JSP Custom tags
- MVC

JSP directives

- https://www.tutorialspoint.com/jsp/jsp_directives.htm
- `<%@ page ... %>`
  - language="java" --> server side code (scriptlet, declaration & expression language = java)
  - contentType="text/html"
  - session="true" --> automatically session is created for this page -- internally calls request.getSession() in generated servlet.
  - info="Description of the JSP page" --> kind of documentation
  - extends="javax.servlet.http.HttpServlet" --> generated servlet class must be inherited from this class.
  - isErrorPage="false" --> defines whether this page is designed to display error messages.
    - false --> simple JSP page (not an error page)
    - true --> an error page (can access "exception" object to display error).
  - errorPage="" --> name of error to be displayed when error is generated in current page.
  - isELIgnored="false"
    - false --> process JSP EL.
    - true --> ignore JSP EL syntax `${...}`
- `<%@ include file="..." %>`
  - Includes another file (jsp, html, ...) into current jsp.
  - As per standard, this is static inclusion (done at translation stage).
  - However web server implementors may follow dynamic inclusion (RequestDispatcher.include()).
  - Commonly used for consistent look & feel (common header & footer).
- `<%@ taglib ... %>`
  - Use custom tags/third party tags into JSP pages.
  - Tags = Business logic + Presentation logic

Custom Error Pages

- To display exceptions separate pages can be created with `<%@ page isErrorPage="true" %>`
- You can access "exception" object in such pages.
- These page can be linked to other JSP page with `<%@ page errorPage="error.jsp" %>`

- The error pages can be configured in web.xml, so that for specific error specific pages can be displayed.

```
<error-page>
    <error-code>404</error-code>
    <location>/notfound.jsp</location>
</error-page>
<error-page>
    <error-code>403</error-code>
    <location>/forbidden.jsp</location>
</error-page>
<error-page>
    <error-code>500</error-code>
    <location>/error.jsp</location>
</error-page>
```

- If error page is configured in web.xml as well as page directive, then page directive is get precedance.

JSP Implicit Objects

- Few objects can be directly accessible in JSP scriptlet and expressions (Request handling stage -- _jspService() method) without declaring them. They are JSP implicit objects.
- Internally these objects are local variables/arguments of _jspService().

1. request: HttpServletRequest
2. response: HttpServletResponse
3. config: ServletConfig
4. application: ServletContext
5. session: HttpSession
6. out: JspWriter
7. page: Object -> this pointer
8. pageContext: PageContext -> to save some data on page scope (accessible on that page for that request only)
9. exception: Throwable -> accessible only in error pages

JSP Standard Actions

- Refer slides.
- JSP built-in tags used for specific tasks. Reduces Java code in JSP pages.
- Standard actions is part of Java EE JSP specifications.

jsp:plugin & jsp:fallback

- These tags are used to embed applets (and other plugins like flash) into jsp pages.

```
<jsp:plugin type="applet" code="MyApplet.class" codebase="/application"
width="200"
height="300">
```

```
    <jsp:fallback>Applet Not Loaded</jsp:fallback>
</jsp:plugin>
```

- This code will be internally converted into html's or tag.

jsp:forward, jsp:include and jsp:param

- Inter-servlet communication

```
<jsp:forward page="pageurl"/>
<%-- OR --%>
<jsp:include page="pageurl"/>
```

- Will get converted into following java code

```
RequestDispatched rd = req.getRequestDispatcher("pageurl");
rd.forward(req, resp);
// OR
rd.include(req, resp);
```

- jsp:include is used to for dynamic inclusion of the file.
- Request param can be passed from page1.jsp to page2.jsp.

```
<jsp:forward page="page2.jsp">
    <jsp:param name="key1" value="value1"/>
</jsp:forward>
```

```
// in page2.jsp
String val = request.getParameter("key1");
```

JSP & Java Beans

Rules to write Java Beans

- Write a simple java class having following members:
    1. one or more fields
    2. param less constructor
    3. getter/setter methods (proper camel case)
    4. one or more business logic method (any name)

Standard actions for java beans

jsp:useBean

- jsp:useBean is used to access the bean object from the given scope if available; otherwise create a new object and add in the given scope.
  - This tag check the whether any object is present in the given scope with given name (id).
  - If object is present, it will be accessed by that name.
  - If object is not present, class will be loaded and object will be created at runtime.
  - Then object will added into given scope with given name.
- Example

```
<jsp:useBean id="ub" class="sunbeam.beans.UpperCaseBean1"
scope="page" />
```

- Internals

```
// pseudo code
UpperCaseBean1 ub = (UpperCaseBean1)
pageContext.getAttribute("ub");
if(ub == null) {
    ub = new UpperCaseBean1();  // object is created using
reflection
    pageContext.setAttribute("ub", ub);
}
```

- Scopes
  - page --> PageContext attribute --> smallest & default scope.
    - accessible in current request to current page.
  - request --> HttpServletRequest attribute
    - accessible in all pages to which current request is forwarded/included.
  - session --> HttpSession attribute
    - accessible in all requests to all pages for current user.
  - application --> ServletContext attribute --> highest scope.
    - accessible in all requests to all pages for all users.

jsp:setProperty

- Example

```
<jsp:setProperty name="obj" property="propName"
value="fix_value"/>
```

```
<jsp:setProperty name="obj" property="propName"
param="req_param"/>
```

```
<jsp:setProperty name="obj" property="*"/>
```

- Internally find corresponding setter method (i.e. setPropName()) and execute it on given object.
- "value" is used to give fix value, while "param" is used to give value from request parameter.
- If property="*", all properties having name same as request params will be automatically initialized.

jsp:getProperty

- Example

```
<jsp:getProperty name="obj" property="propName"/>
```

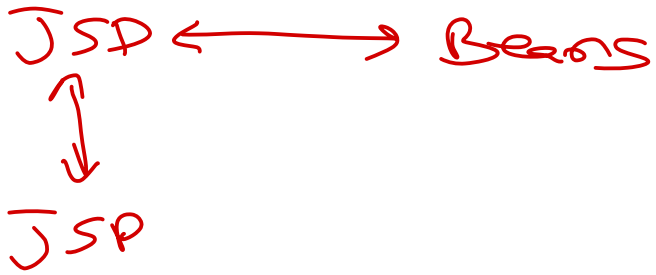
- Internally find corresponding getter method (i.e. getPropName()) and execute it. The return value will be added into html response.
- Before calling jsp:setProperty or jsp:getProperty the jsp page must have jsp:useBean tag.

Sunbeam Infotech

Nilesh Ghule <nilesh@sunbeaminfo.com>

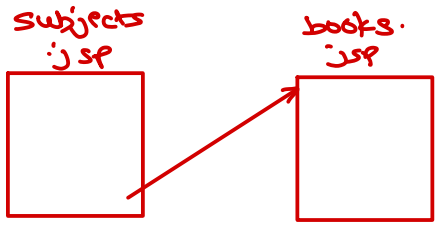
Model-View architecture

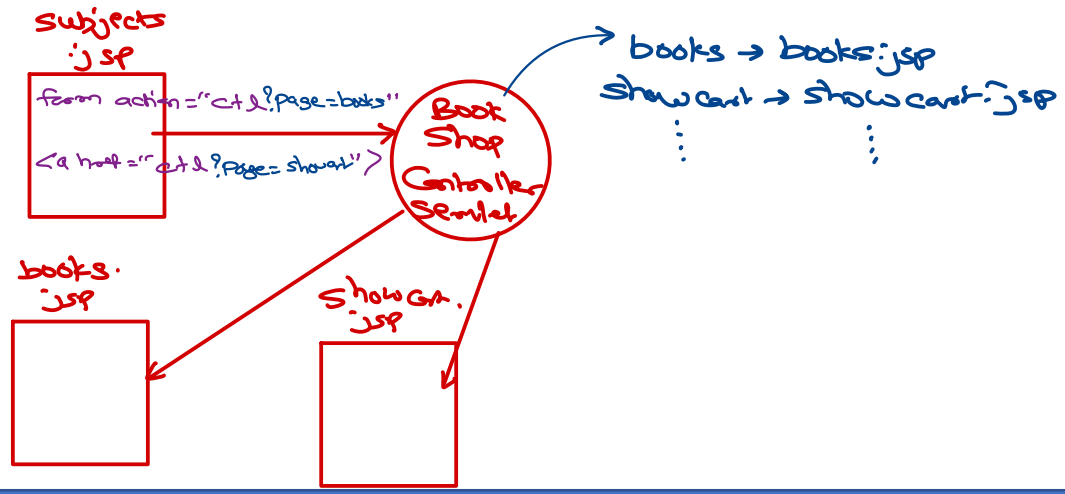
- Typical JSP based web applications use Java beans to implement business logic while JSP pages implement presentation logic.
- Java beans collect input from JSP/HTML forms and hold data to displayed on JSP.
- Such application architecture is referred as Model-View architecture.
  - Model: Java bean
  - View: JSP page
- This architecture is easy to implement, however not suitable for huge applications.



Model-View-Controller architecture

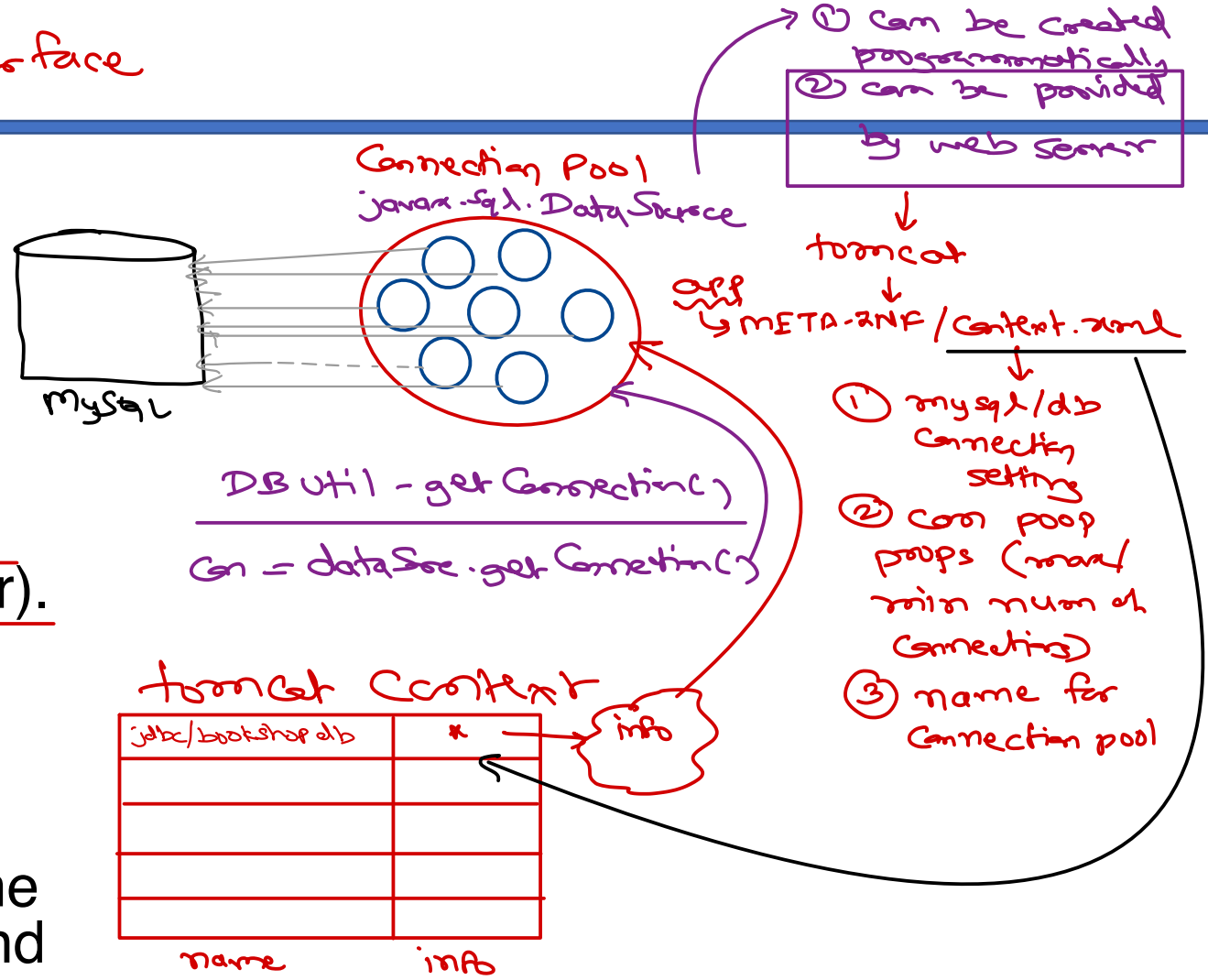
- MVC design pattern makes Java EE web applications extendable & maintainable.
- Typical MVC implementation
  - Model: Java bean/POJO
  - View: JSP page
  - Controller: Servlet (forwarding request)
- Popular MVC frameworks: Struts, JSF, Spring.





JNDI - Java Naming & Directory Interface

- JNDI is Java API used to discover and lookup resources by name in form of Java object.
- JNDI is independent of underlying implementation.
- JNDI is commonly used for RMI, JDBC (Web server), EJB and JMS (App server).
- JNDI names are in hierarchical fashion.
- Name is bound and looked up in some context.
- Typical example is web server bind some name with the JDBC connection pool and it can be looked up from the web applications.



Maven

tool to build java se/ee applns
e.g. ant, maven, gradle.

package.json → npm → download node modules.

pom.xml → maven → download jar

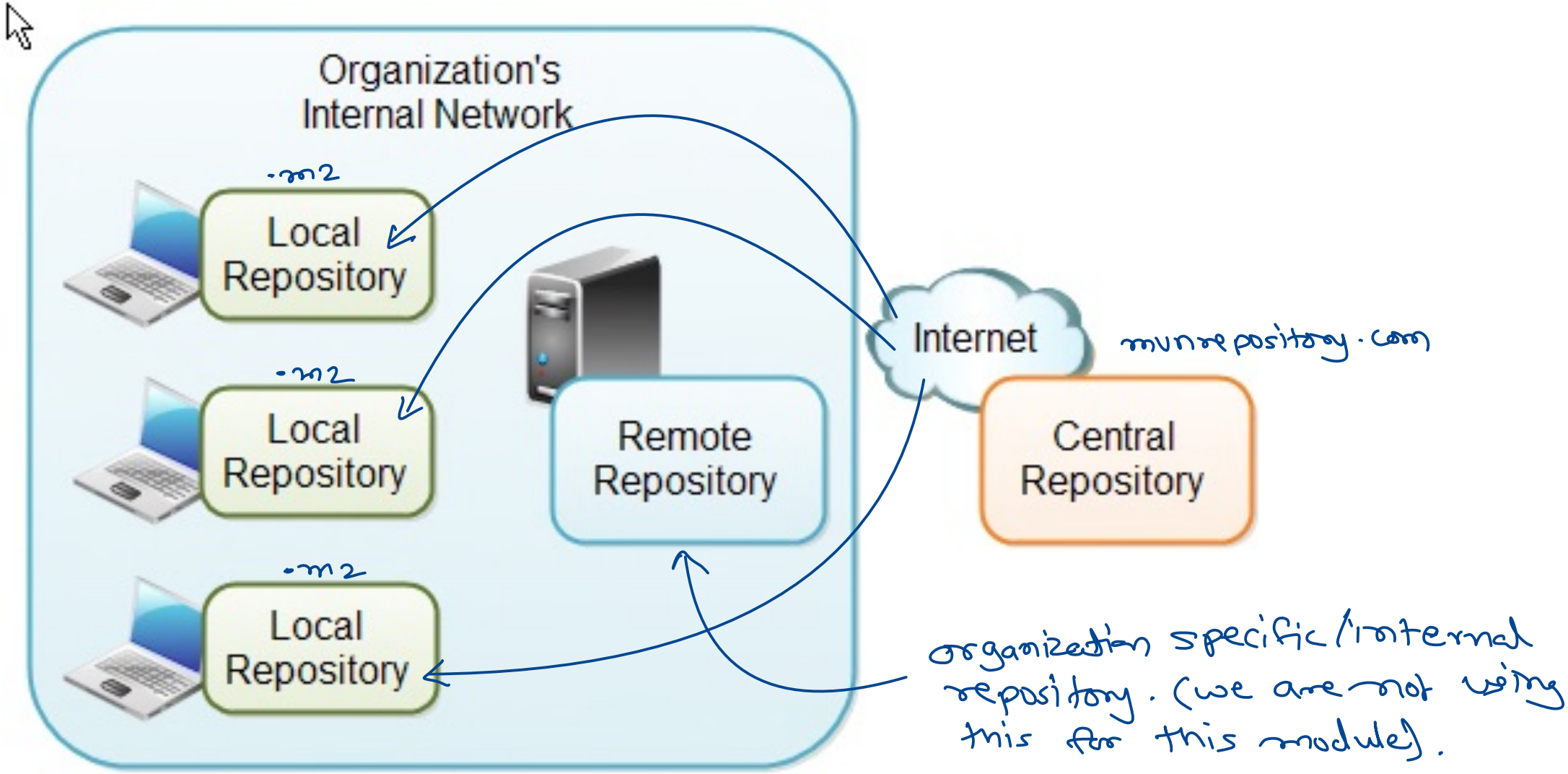
- Maven is Java Build Tool.
  - tool to build appln.
  - e.g. make/cmake, react npx,... c/c++
- Maven does following:
  - Download dependencies from central/remote repository into local repository.
  - Compile source code. → calls 'javac' (as per config in pom.xml)
  - Package compiled code into JAR/WAR files.
  - Install the packaged code.
- POM.XML – Heart of Maven
  - Configurations, Dependencies, Build plugins.
  - third-party jars
  - how to build (customization) & packaging.
  - dev pom ...
- Maven build life cycles: default or clean
  - build appln.
  - delete all generated jars/classes/output files.
- Maven build phases: validate, compile, test, package, install, deploy
  - validate: ① check pom ② download all dependencies
  - compile: ① set classpath ② call javac
  - test: run all unit test case (JUnit)
  - package: create jar/war
  - install: copy jar/war into local repo.
  - deploy: deploy on server (as per config).
- Details: http://tutorials.jenkov.com/maven/maven-tutorial.html

Maven is cmd line tool by Apache

maven.apache.org

All modern Java IDE have builtin support for maven.

Maven repositories



Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- MVC pattern
- JNDI
- Maven
- ~~Spring Introduction~~

MVC Pattern

Model I architecture

- Also called as "Model-View" Architecture.
- View -> Presentation Logic -> JSP pages
- Model -> Business Logic / Data Transfer Objects -> Java Beans
- Used for very small web applications

Model II architecture

- Also called as "Model-View-Controller" Architecture.
- Used for huge web applications
- e.g. Third party frameworks like struts, spring, etc.

JNDI

- https://docs.oracle.com/javase/tutorial/jndi/overview/index.html

Maven

- Maven is a powerful build tool for Java software projects.

Build tool

- A build tool is a tool that automates everything related to building the software project. Building a software project typically includes one or more of these activities:
    - Generating source code (if auto-generated code is used in the project).
    - Generating documentation from the source code.
    - Compiling source code.
    - Packaging compiled code into JAR files or ZIP files.
    - Installing the packaged code on a server, in a repository or somewhere else.
- The advantage of automating the build process is that you minimize the risk of humans making errors while building the software manually.
- Additionally, an automated build tool is typically faster than a human performing the same steps manually.

Maven Core Concepts

- Maven is centered around the concept of POM files (Project Object Model).
- A POM file is an XML representation of project resources like source code, test code, dependencies (external JARs used) etc.
- The POM file should be located in the root directory of the project it belongs to.
- Maven working
    1. Reads pom.xml
    2. Download dependencies into local repository
    3. Executes life cycles, build phases and goals.
    4. Execute plugins.

POM file

- The groupId element is a unique ID for an organization, or a project (an open source project, for instance). Most often you will use a group ID which is similar to the root Java package name of the project.
- The artifactId element contains the name of the project you are building.
- The versionId element contains the version number of the project. SNAPSHOP versions indicate that project is in active development (not production ready).

Dependencies

- Maven check the dependencies needed for your project.
- Dependencies are external JAR files (Java libraries) that your project uses.
- If the dependencies are not found in the local Maven repository, Maven downloads them from a central Maven repository and puts them in your local repository.
- Maven add all dependencies into current project classpath.
- Dependency scopes
    - compile: This is the default scope (when no other scope is given).
    - provided: Dependencies that are provided at runtime by JDK or a container. These jars are not deployed in final project archive. Example: java-ee api
    - runtime: The dependencies with this scope are required at runtime, but they're not needed for compilation of the project code. Example: jdbc driver jars
    - test: Only present for test and execution classpaths. Example: JUnit.

Build life cycle, Phases & Goals

- The build process in Maven is split up into build life cycles, phases and goals.
- A build life cycle consists of a sequence of build phases, and each build phase consists of a sequence of goals.
- When you run Maven you pass a command to Maven.
- This command is the name of a build life cycle, phase or goal.
- If a life cycle is requested executed, all build phases in that life cycle are executed.
- If a build phase is requested executed, all build phases before it in the pre-defined sequence of build phases are executed too.

Build Life Cycles

- Maven has built-in build life cycles. These are:
    - default: related to compiling and packaging your project.
    - clean: related to removing temporary files from the output directory.
- Life cycle is further divided into phases.
- When any phase is given, all phases before it are executed.

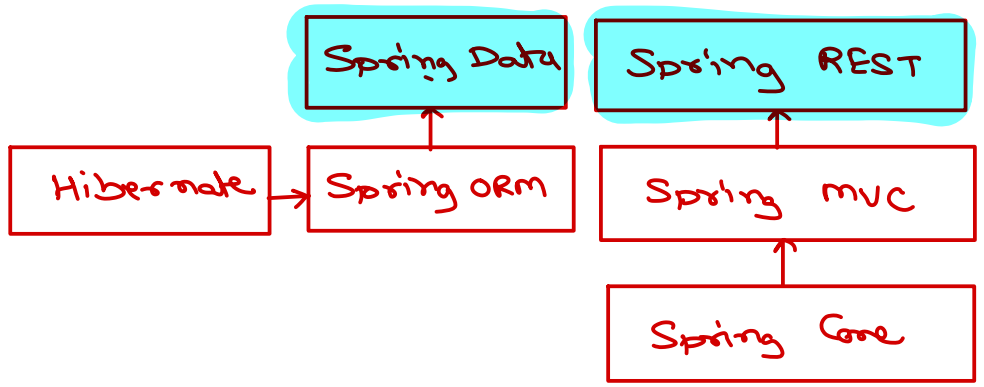
Default Build Life Cycle Phases

- validate: Validates that the project is correct and all necessary information is available. This also makes sure the dependencies are downloaded.
- compile: Compiles the source code of the project.
- test: Runs the tests against the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed.
- package: Packs the compiled code in its distributable format, such as a JAR.
- install: Install the package into the local repository, for use as a dependency in other projects locally.
- deploy: Copies the final package to the remote repository for sharing with other developers and projects.

Spring and Hibernate

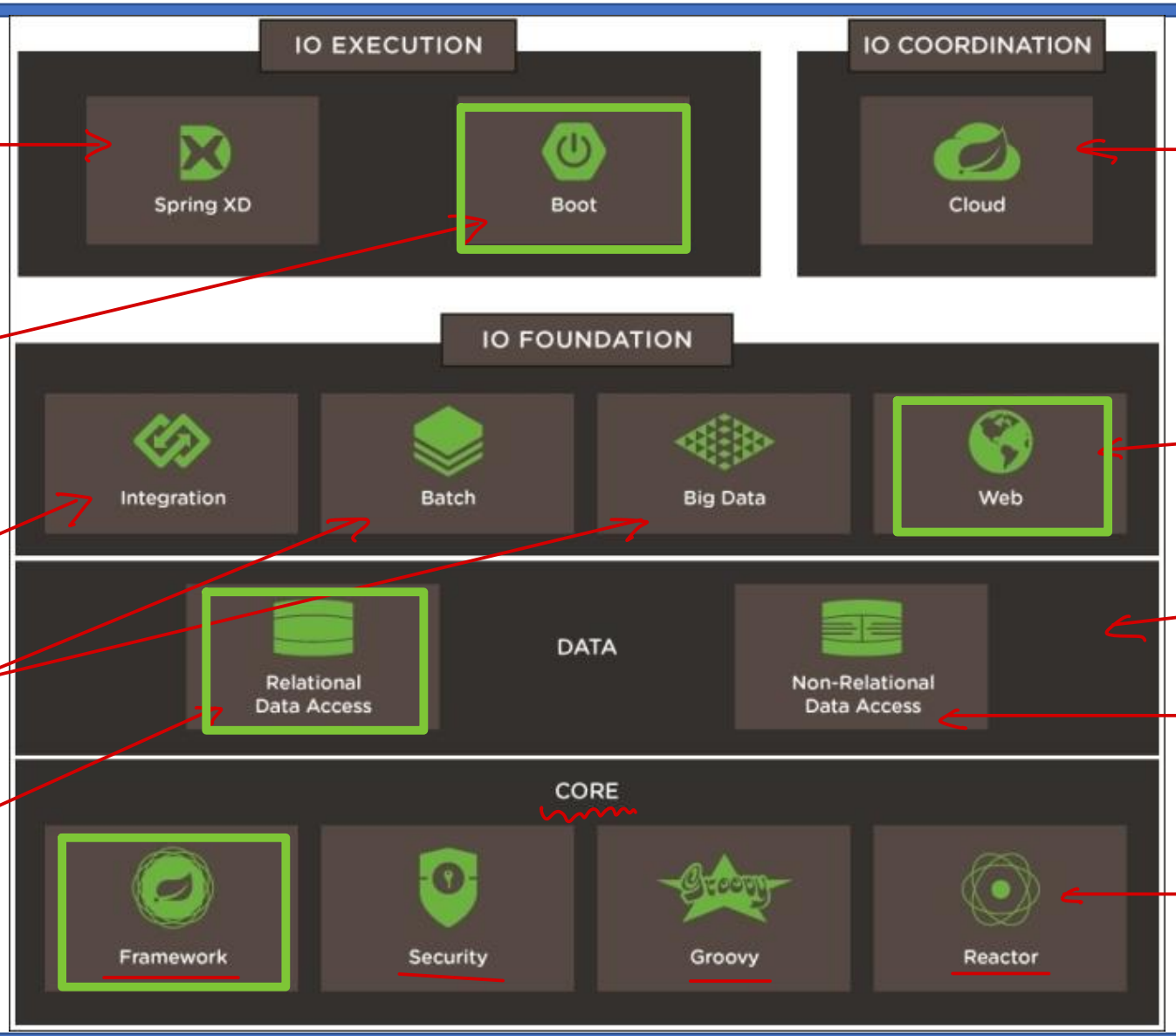
Nilesh Ghule <nilesh@sunbeaminfo.com>

Spring Introduction



Dependency Injection

Spring IO platform



data ingestion / export
RT analytics

micro-services development

simplified access to existing spring frameworks

Spring MVC
Spring REST

messaging among spring apps

Spring data

processing huge data

NoSQL

RDBMS → ORM
→ JDBC

Spring 5

Spring Framework

Java syntax is simple. Java development is not → huge apps, testing, deployment, maintenence

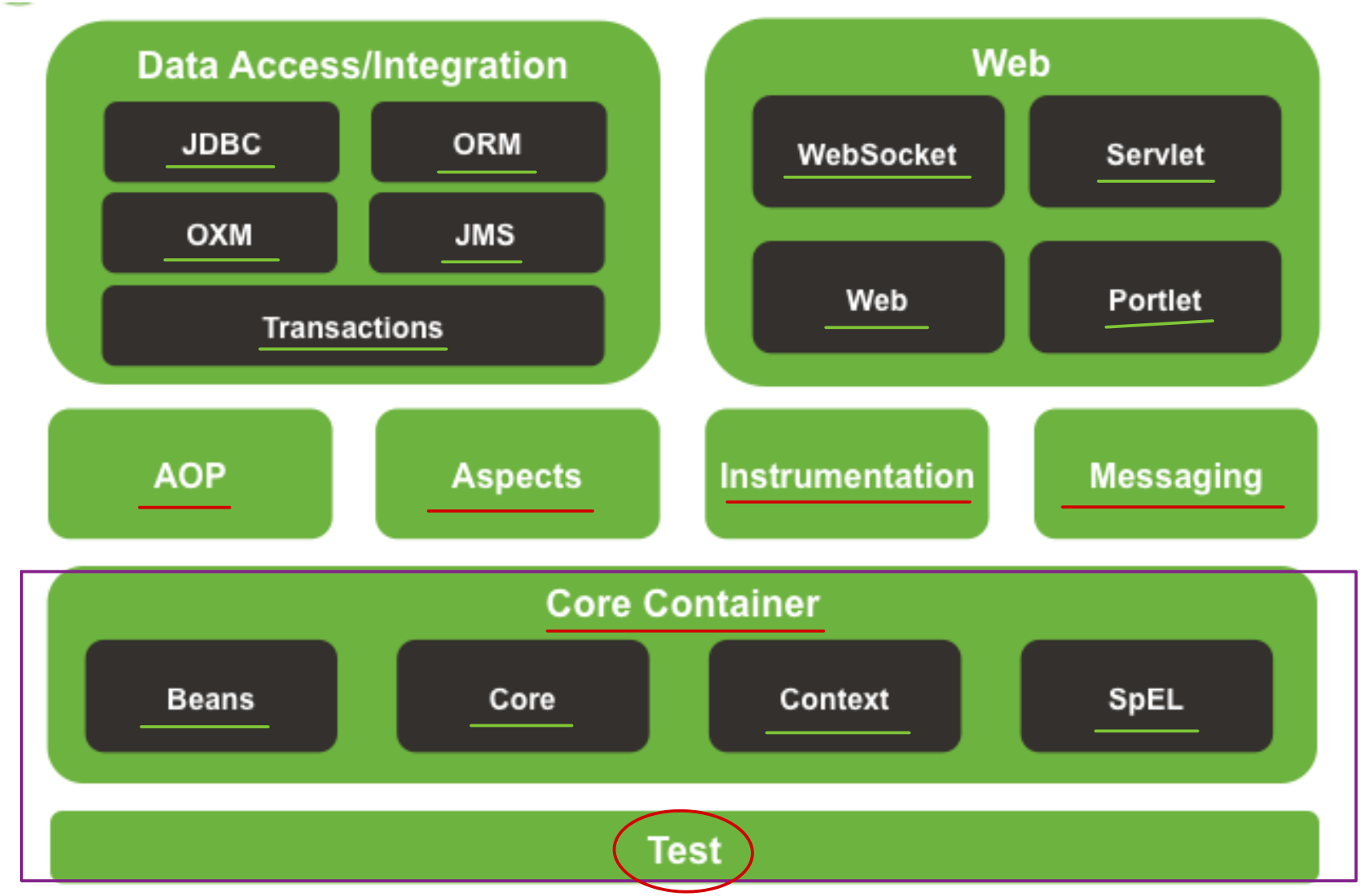
spring-core (2MB) → covering all aspects

- Spring is **light-weight** **comprehensive** framework to **simplify** Java development.
- Developed by Rod Johnson. Spring 3 added annotations while Spring 5 added reactive.
  2003
- Spring pros
  - Inversion of control (Dependency injection)
  - Test driven development
  - Extensive but Flexible and modular
  - Smooth integration with existing technologies
  - Eliminate boilerplate code
  - Extendable
  - Maintainable

Struts : all-or-nothing
Spring : flexible

- Spring cons
  - Heavy configuration (XML, Java, Both)
  - Module versioning & compatibility
  - Application deployment

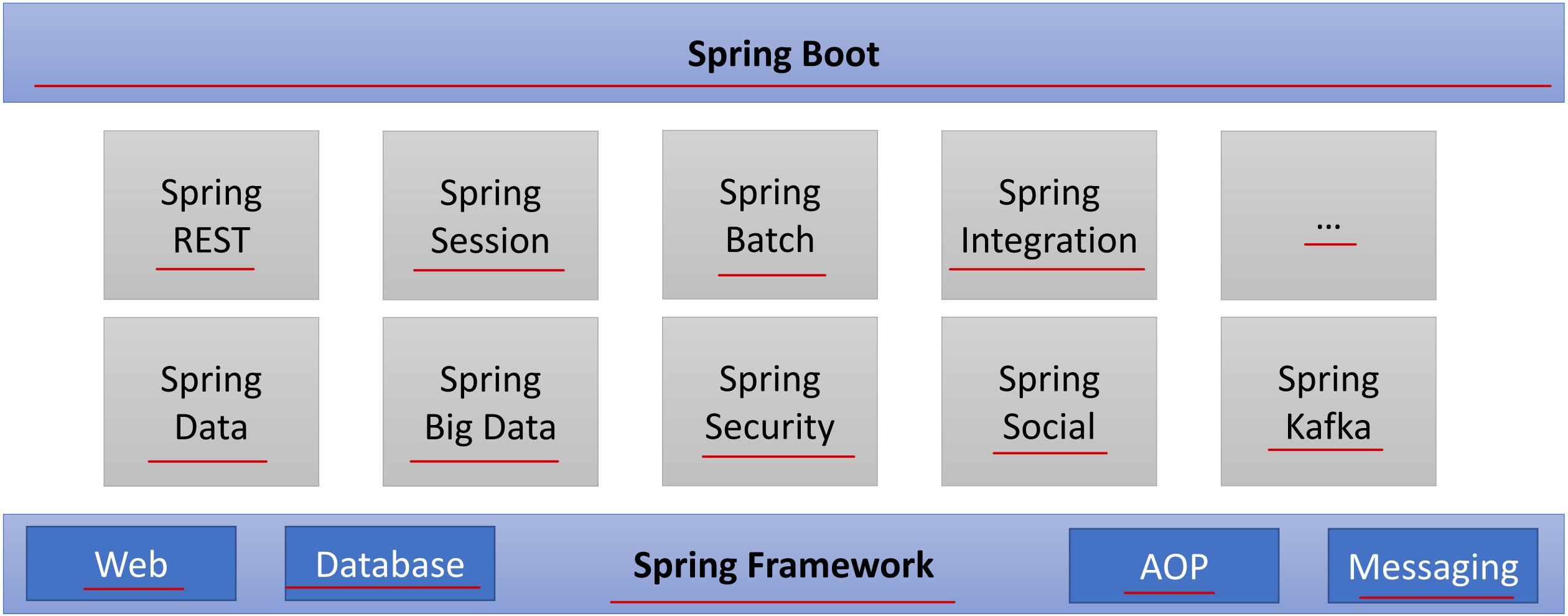
Spring architecture (3.x)



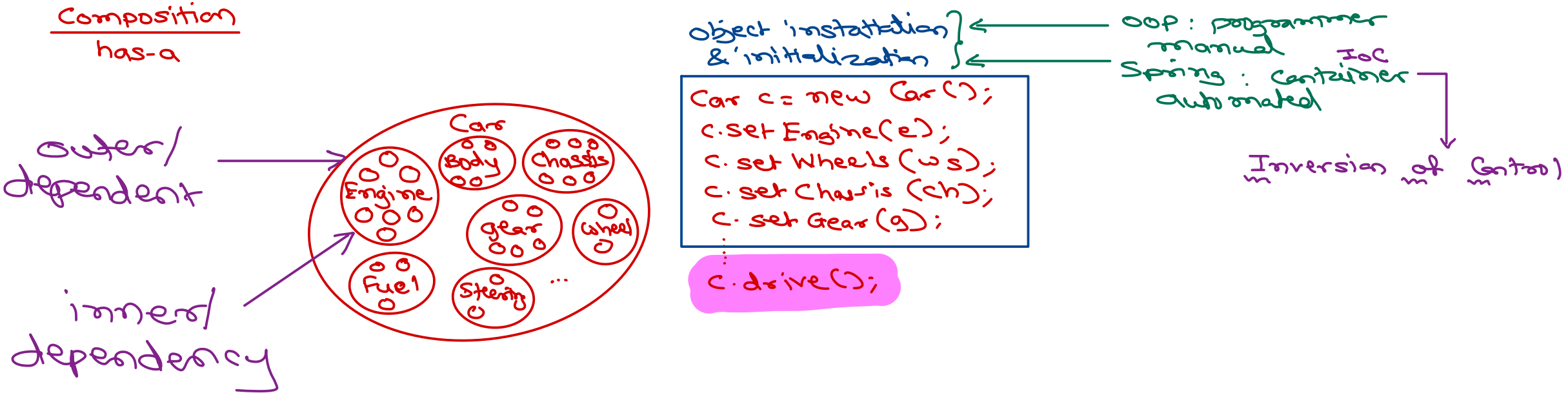
Spring Boot

- Spring Boot is NOT a new standalone framework. It is combination of various frameworks on spring platform i.e. spring-core, spring-webmvc, spring-data, …
- Spring Boot makes it easy to create stand-alone, production-grade Spring based Applications that you can “just run”.
- Spring Boot take an "opinionated view" of the Spring platform and third-party libraries. So most of applications need minimum configuration.
- Primary Goals/Features
  - Provide a radically faster and widely accessible "Quick Start".
  - Opinionated config, yet quickly modifiable for different requirements.
  - Managing versions of dependencies with starter projects.
  - Provide lot of non-functional common features (e.g. security, servers, health checks, ...).
  - No extra code generation (provide lot of boilerplate code) and XML config.
  - Easy deployment and containerisation with embedded Web Server.
- Recommended to use for new apps and not to port legacy Spring apps.

Spring Boot



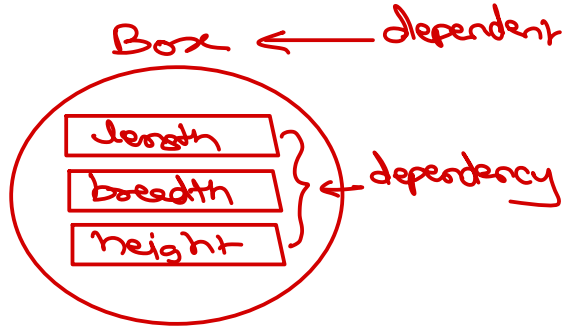
Spring Boot = Spring framework + Embedded web-server + Auto-configuration - XML config - Jar conflicts



Spring Dependency Injection

Dependency Injection

- Spring container injects dependency beans into dependent beans.
- Spring container is also called as IoC container.
- Dependency Injection
  - Setter based DI <property ...>
  - Constructor based DI <constructor-arg ...>
  - Field based DI




```
Box b=new Box();
b.setLength(5);
b.setBreadth(4);
b.setHeight(3);
sop(b.calcVolume());
```

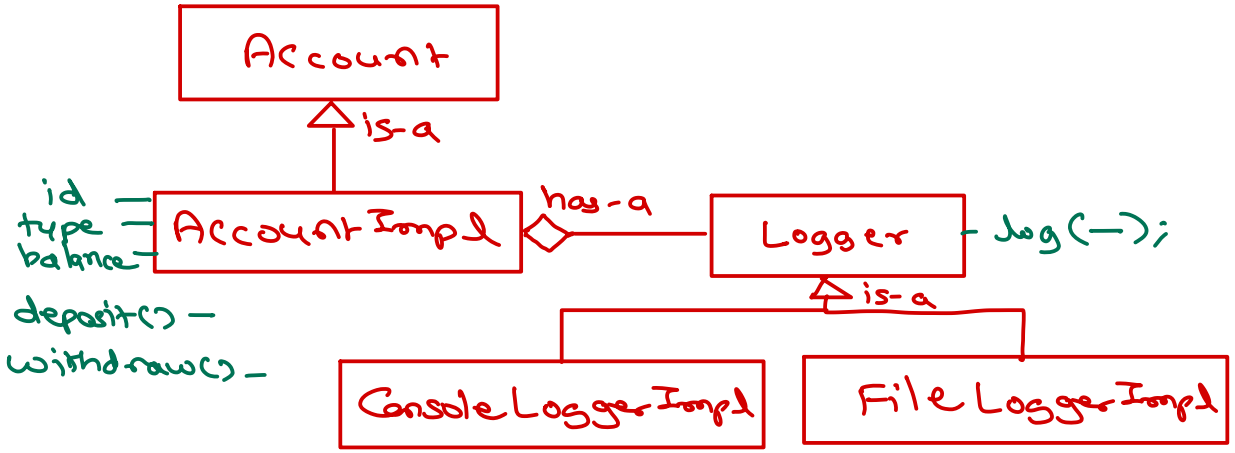
```
Box b=ctx.getBean(...);
sop(b.calcVolume());
```

XML based DI

- Spring beans are declared into *Spring bean configuration file*.
- Bean factory or application context reads the file. It instantiate and initialize bean objects at runtime.
- XML configuration
  - Initializing properties
  - Initializing dependency beans
  - Initializing collections

Dependency Injection

- One interface may have multiple implementations.
- This makes application more extendable and maintainable.
- Desired implementation can be plugged in using DI.



XML Bean Factory

- Deprecated (no more used)
- Minimal Dependency injection features
  - Beans loading
  - Auto-wiring
- Create a singleton bean when it is accessed first time in the application.
- No bean life cycle management x

Application Context

- Features
  - Dependency injection - Beans loading and Auto-wiring
  - Automatic BeanPostProcessor registration i.e. Bean life cycle management
  - Convenient MessageSource access (Internationalization)
  - ApplicationEvent publication
- Create all singleton beans when application context is loaded.

- ApplicationContext (base interface (refer spring docs))
  - ClassPathXmlApplicationContext
  - FileSystemXmlApplicationContext
  - AnnotationConfigApplicationContext
  - WebApplicationContext
    - XmlWebApplicationContext
    - AnnotationConfigWebApplicationContext

impl classes →

Annotation config

- Does spring configuration without using any XML file.
- The bean creation is encapsulated in @Configuration class and beans are represented as @Bean methods.
- Annotation configuration is processed using AnnotationConfigApplicationContext.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Spring

Spring techologies overview

- Spring IO platform
  - Spring XD
  - Spring cloud (https://spring.io/projects/spring-cloud-netflix)
  - Spring boot (https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/)
- Spring framework (https://docs.spring.io/spring-framework/docs/4.3.29.RELEASE/spring-framework-reference/htmlsingle/)
- Spring Documentation (https://docs.spring.io/spring-framework/docs/5.3.8/javadoc-api/)

Spring vs Spring Boot

- Spring is lightweight comprehensive framework that simplifies Java development.

Why Spring

- Spring is lightweight. The basic version of Spring framework is around 2MB.
- Good programming practices based on interfaces and POJOs.
- Inversion of Control - Dependency injection.
  - OOP -- Object Oriented Programming -- Composition
  - Outer object --> Inner object(s)
    - Car <>-- Engine
    - Car <>-- Wheels

```
// ...
Car c = new Car();
c.setEngine(e);
c.setWheels(w);
c.setChasis(ch);
// ...
c.drive();
```

  - Spring automates object creation as well as initialization.

```
Car c = ctx.getBean(Car.class);
c.drive();
```

  - Inversion-of-Control

        - OOP (Bottom-up) vs POP (Top-down)
        - Manual Object Initialization vs DI Object Initialization (Inverted)
- Test driven approach: Easy testing support.
    - Unit testing/Mocking support in out-of-box.
    - Beans can be tested independently.
- Extensive but modular and flexible.
    - Extensive: Huge -- so many modules, sub-projects, wide spectrum
    - Modular: Separate sub-projects/modules are available
    - Flexible: Can use only modules you need on top of Spring core
- Smooth integration with existing technologies like JDBC, JNDI, ORM, Mailing, etc.
- Eliminate boiler-plate code - ORM, JDBC, Security, etc.
    - JDBC
        - Load & register class (managed by spring)
        - Create connection (managed by spring)
        - Create statement (managed by spring)
        - Execute the query (managed by spring)
        - Process the result (ResultSet) -- supply SQL statement, parameters and process result.
        - Close all (managed by spring)
    - Hibernate
        - Hibernate configuration (.cfg.xml or coding) (managed by spring)
        - Create SessionFactory (managed by spring)
        - Create session (managed by spring)
        - transaction management (managed by spring)
        - CRUD operations or Query execution -- user-defined
        - Cleanup (managed by spring)
- Unified transaction management (local & distributed/global) -- @Transactional
    - Local transactions: Within same database
    - Global transactions - JTA: Across the databases
- Easier J2EE development through Spring Web MVC (Pull) and Web sockets (Push).
- Consistent and "readable" unchecked exceptions - wraps technology-specific exceptions.

Core feature

- Dependency injection
- Spring IoC container - Bean life-cycle management
    - Container -- Life-cycle
        - JRE container in browser: Applet life-cycle
        - Servlet/JSP container in web-server: Servlet/JSP life-cycle
        - EJB container: EJB life-cycle

Spring hurdles

- Heavy configuration
    - XML configuration
    - Java (annotation) configuration
    - Mixed configuration
- Module versioning/compatibility

    - Spring & Spring-Security version
    - Hibernate & Cache version
    - ...
- Application deployment
    - Web-server deployment
    - Containerisation - Docker
    - Complex Microservice development

Spring Boot

- Spring Boot is a Framework from "The Spring Team" to ease the bootstrapping and development of new Spring Applications.
    - Framework
    - Ease bootstrapping - Quick Start (Rapid Application Development)
    - New applications - Not good choice for legacy applications
- Abstraction/integration/simplification over existing spring framework/modules.
    - Spring core
    - Spring Web MVC
    - Spring Security
    - Spring ORM
    - ...
- NOT replacement or performance improvement of Spring framework.
    - Not replacement, but it is abstraction.
    - Underlying same Spring framework is working -- with same speed.
- Spring Boot = Spring framework + Embedded web-server + Auto-configuration - XML configuration - Jar conflicts

Why Spring Boot

- Provide a radically faster and widely accessible "Quick Start".
    - Very easy to develop production grade applications.
    - Reduce development time drastically.
    - Increase productivity.
    - Spring Boot CLI (https://docs.spring.io/spring-boot/docs/current/reference/html/spring-boot-cli.html)
    - Spring Initializr (https://start.spring.io/)
    - IDE support - Eclipse STS
    - Build tools - Maven/Gradle support
- Managing versions of dependencies with starter projects.
    - Set of convenient dependency descriptors
    - Hierarchially arranged so that minimal dependecies to be added in application
    - e.g. spring-boot-starter-web
        - spring-web
        - spring-webmvc
        - spring-boot-starter
            - spring-core
            - spring-context

        - spring-boot-autoconfigure
        - spring-boot-starter-logging
            - log4j-to-slf4j
            - jul-to-slf4j
    - spring-boot-starter-json
        - jackson-databind
    - spring-boot-starter-tomcat
        - tomcat-embed-core
        - tomcat-embed-websocket
        - jakarta.el
    - hibernate-validator
- No extra code generation (boiler-plate is pre-implemented) and XML config.
    - Minimal configuration required.
    - Ready to use in-memory/embedded databases, ORM/JPA, MVC/REST, ...
- Opinionated configuration, yet quickly modifiable for different requirements.
    - Easy/quick integration with standard technologies/frameworks.
    - Examples:
        - Web applications -- Tomcat
        - ORM -- Hibernate
        - JSON -- Jackson
        - Test -- JUnit
    - You can change the configuration/technologies by adding alternates on pom.xml (classpath).
- Provide lot of non-functional common features (e.g. security, servers, health checks, ...).
- Easy deployment and Containerisation.
    - Embedded Web Server for web applications.

Spring XML configuration BoxImpl demo

- step 1: Create new Maven project. Fill project details like groupId, artifactId, packaging=jar, etc.
- step 2: Create bean class BoxImpl with appropriate fields and methods (as discussed in the class).
- step 3: In resources folder, create Spring Bean Configuration File (beans.xml) and configure beans b1 & b2 (as discussed in class).
- step 4: Create a Main class and create ClassPathXmlApplicationContext, create beans and invoke business logic (as discussed in class).

Spring Annotation configuration BoxImpl demo

- step 1: Create new Maven project. Fill project details like groupId, artifactId, packaging=jar, etc.
- step 2: Create bean class BoxImpl with appropriate fields and methods (as discussed in the class).
- step 3: Create AppConfig class annotated with @Configuration and create beans b1 & b2 in it (as discussed in class).
- step 4: Create a Main class and create AnnotationConfigurationXmlApplicationContext, create beans and invoke business logic (as discussed in class).

Spring Boot BoxImpl demo

- step 1: New Spring Boot Starter project. Fill project groupId, artifactId, name, etc details.

- step 2: Inherit main class from CommandLineRunner interface and implement run() method. Implement code in run() method.
- step 3: Create bean class BoxImpl with appropriate fields and methods (as discussed in the class).
- step 4: Create AppConfig class annotated with @Configuration and create beans b1 & b2 in it (as discussed in class).
- step 5: @Autowired ApplicationContext ctx; into main class.
- step 6: In run() method, create beans and invoke business logic (as discussed in class).

Understanding Simple Spring Boot Console Application

- step 1: New Spring Boot Starter project. Fill project groupId, artifactId, name, etc details.
- step 2: Inherit main class from CommandLineRunner interface and implement run() method. Implement code in run() method.

@SpringBootApplication annotation

- Combination of three annotations
    - @ComponentScan
    - @Configuration
    - @EnableAutoConfiguration
- @EnableAutoConfiguration annotation
    - Intelligent and automatic configuration
    - Spring @Configuration
    - @Conditional beans
        - @ConditionalOnClass
        - @ConditionalOnMissingBean

SpringApplication class

- Entrypoint of Spring boot application
- SpringApplication.run() does following
    - Create an ApplicationContext instance
    - Prepare spring container for spring bean life-cycle management
    - Load all singleton beans
    - Execute CommandLineRunner if available.
- By default take bean config from @Configuration classes. Can also take from
    - explicit AnnotatedBeanDefinitionReader
    - XML resource by XmlBeanDefinitionReader
    - groovy script by GroovyBeanDefinitionReader
    - beans by ClassPathBeanDefinitionScanner

Spring and Hibernate

Nilesh Ghule <nilesh@sunbeaminfo.com>

Spring beans

Beans created & managed by Spring containers

Bean = object of a java class

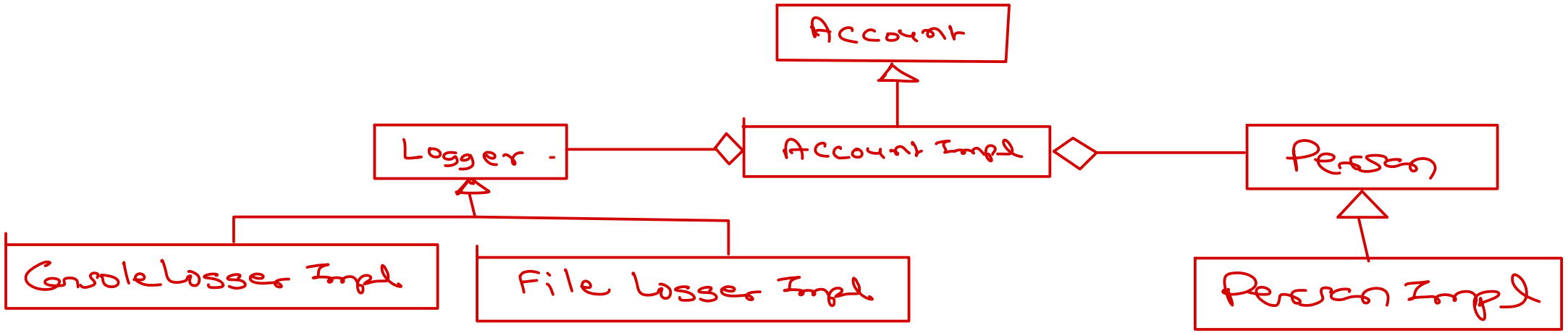
Java class = fields + constructors + getter/setters + business logic

Spring bean life cycle

- Spring container creates (singleton) spring bean objects when container is started.
- Spring container controls creation and destruction of bean objects.
- There are four options to control bean life cycle.
  - InitializingBean and DisposableBean callback interfaces
  - Aware interfaces for specific behaviours
    - BeanNameAware, BeanFactoryAware, ApplicationContextAware , etc.
  - BeanPostProcessor callback interface
  - Custom init() / @PostConstruct and destroy() / @PreDestroy methods

Spring bean life cycle

- Bean creation
  1. Instantiation (Constructor)
  2. Set properties (Field/setter based DI)
  3. BeanNameAware.setBeanName()
  4. ApplicationContextAware.setApplicationContext()
  5. Custom BeanPostProcessor.postProcessBeforeInitialization()
  6. Custom init() (@PostConstructor)
  7. InitializingBean.afterPropertiesSet()
  8. Custom BeanPostProcessor.postProcessAfterInitialization()
- Bean destruction
  1. Custom destroy() (@PreDestroy)
  2. DisposableBean.destroy()
  3. Object.finalize()



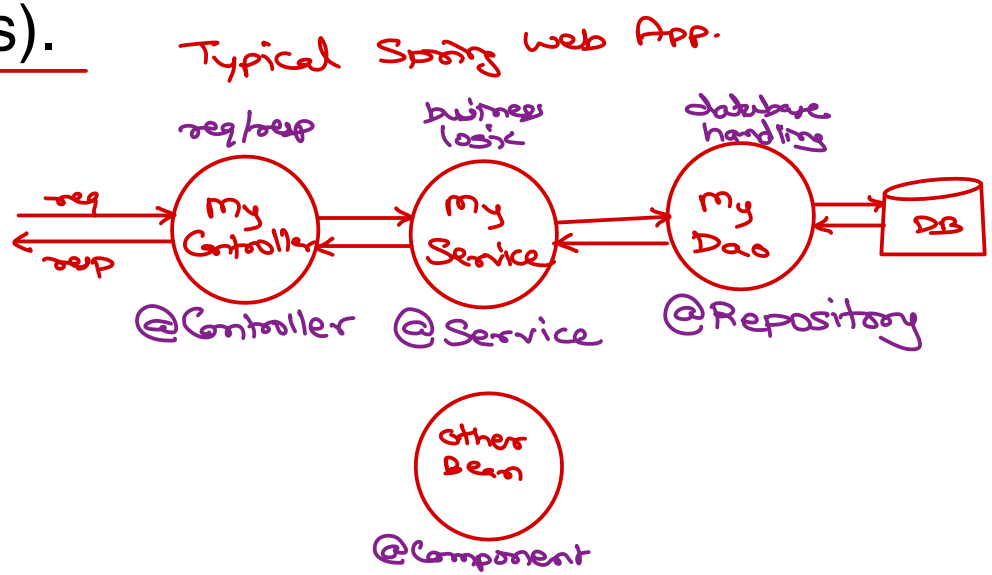
Stereo-type annoatations

Stereo type annotations

- Auto-detecting beans (avoid manual config of beans).
  - @Component
  - @Service
  - @Repository
  - @Controller and @RestController
- In XML config file
  - <context:component-scan basePackages="___"/>
- Annotation based config
  - @ComponentScan(basePackages = "pkg")
  - includeFilters and excludeFilters can be used to control bean detection.

@Controller for REST service Spring 4.0 +

@SpringBootApplication
↳ by default scan current pkg & sub pkg.



Auto-wiring

Auto-wiring

- Automatically injecting appropriate dependency beans into the dependent beans.

Auto-wiring (XML config)

- <bean id="..." class="..." autowire="defaultInolbyTypelbyNamelconstructor" .../>
  - default / no: Auto-wiring is disabled.
  - byType: Dependency bean of property type will be assigned (via setter).
    - If multiple beans of required type are available, then exception is thrown.
    - If no bean of required type is available, auto-wiring is not done.
  - byName: Dependency bean of property name will be assigned (via setter).
    - If no bean of required name is available, auto-wiring is not done.
  - constructor: Dependency bean of property type will be assigned via single argument constructor (of bean type).
- <bean id="..." class="..." autowire-candidate="truelfalse" .../>
  - false -- do not consider this dependency bean for auto-wiring.

Auto-wiring (Annotation config)

- @Autowired
  - Setter level: setter based DI
  - Constructor level: constructor based DI
  - Field level: field based DI
- @Autowired: Find bean of corresponding field type and assign it.
  - If no bean is found of given type, it throws exception.
    - @Autowired(required=false): no exception is thrown, auto-wiring skipped.
  - If multiple beans are found of given type, it try to attach bean of same name. and if such bean is not found, then throw exception.
  - If multiple beans are found of given type, programmer can use @Qualifier to choose expected bean. @Qualifier can only be used to resolve conflict in case of @Autowired.
- @Resource (JSR 250)
  - @Resource: DI byName, byType, byQualifier
- @Inject (JSR 330) – same as @Autowired
  - @Inject: DI byType, byQualifier (@Named), byName

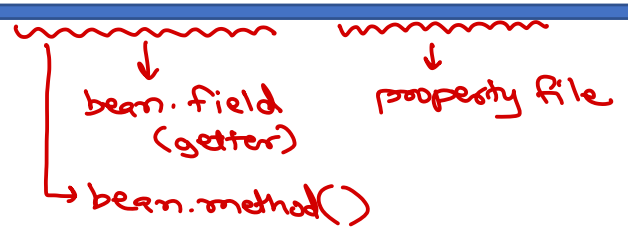
Mixed configuration

- XML config + Annotations
  - XML config file is loaded using ClassPathXmlApplicationContext.
  - <context:annotation-config/> activate annotation processing.
    - @PostConstruct
    - @PreDestroy
    - @Autowired
    - @Qualifier

Spring EL

Spring Expression Language

JSP EL ${ — }
SpEL #{ — } ${ — }



- SPEL can be used with XML config or @Value annotation.
- Using XML config
  - value="#{bean.field}"
  - value="${property-key}"
    - In beans.xml: <context:property-placeholder location="classpath:app.properties" />

if using xml config file.

- Using Annotation config
  - Auto-wire dependency beans: #{bean}
  - Values using SPEL expressions: #{bean.field} or #{bean.method()}
  - Values using SPEL expressions from properties files: ${property-key}
    - In @Configuration class: @PropertySource("classpath:app.properties")

if using annotation config / spring boot

Spring Expression Language

- SpEL is a powerful expression language that supports querying and manipulating an object graph at runtime. Syntactically it is similar to EL(JSP)
- SpEL can be used in all spring framework components/products.
- SpEL supports Literal expressions, Regular expressions., Class expressions, Accessing properties, Collections, Method invocation, Relational operators, Assignment, Bean references, Inline lists/maps, Ternary operator, etc.
- SpEL expressions are internally evaluated using SpELExpressionParser.
  - ExpressionParser parser = new SpELExpressionParser();
  - value = parser.parseExpression("'Hello World'.concat('!')");
  - value = parser.parseExpression("new String('Hello World').toUpperCase()");
  - value = parser.parseExpression("bean.list[0]");
- SpEL expressions are slower in execution due parsing. Spring 4.1 added SpEL compiler to speed-up execution by creating a class for expression behaviour at runtime.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- Spring Beans
- Spring Auto-wiring
- Spring SpEL
- ~~Hibernate Introduction~~

Spring Boot Hello World

@SpringBootApplication annotation

- Combination of three annotations
  - @ComponentScan
    - Auto detection of stereo-type annotated beans e.g. @Component, @Service, @Repository, ...
    - By default basePackage is to search into "current" package (and its sub-packages).
    - @ComponentScan("other.package") can be added "explicitly" to restrict the search of beans into given package.
  - @Configuration
    - Spring annotation configuration to create beans.
    - Contains @Bean methods which create and return beans.
    - @Configuration classes are also auto-detected by @ComponentScan
  - @EnableAutoConfiguration
- @EnableAutoConfiguration annotation
  - Spring @Configuration -- create the beans
  - Intelligent and automatic configuration
  - @Conditional beans
    - @ConditionalOnClass
    - @ConditionalOnMissingBean

SpringApplication class

- Entrypoint of Spring boot application
  - main() method is called by JVM.
- SpringApplication.run() does following
  - Create an ApplicationContext instance
  - Prepare spring container for spring bean life-cycle management
  - Create all singleton beans
  - Execute CommandLineRunner if available.
    - Console application.
- By default take bean config from @Configuration classes.

Spring Beans

Spring Bean Life Cycle

Demo 04 (Bean life cycle)

- interface Box
- class BoxImpl implements Box, InitializingBean, DisposableBean, BeanNameAware, ApplicationContextAware
- Add @PostConstruct and @PreDestroy methods
- create @Configuration AppConfig class
    - create BoxImpl bean in a @Bean method
- class BoxBeanPostProcessorImpl implements BeanPostProcessor
    - Create this as @Bean in AppConfig.
    - Examples of pre-defined BeanPostProcessor: BeanValidationPostProcessor, InitDestroyAnnotationBeanPostProcessor, etc.
- Execute the application and understand bean life cycle.

Dependency Injection

Demo 05 (DI Revision)

- interface Logger
- class ConsoleLoggerImpl implements Logger
- class FileLoggerImpl implements Logger
- interface Person
- class PersonImpl implements Person
- interface Account
- class AccountImpl implements Account
    - Logger dependency
    - Person dependency (accHolder)

Stereo-Type Annotations & SpEL

Demo 06 (Strereo-type annotations and @Value)

- Copy Demo 05 project.
- Declare ConsoleLoggerImpl and FileLoggerImpl as @Component.
- Use @Value in FileLoggerImpl on "logFilePath" field.
- resources/demo.properties
    - log.filePath=demo06.log
- On AppConfig class use @PropertySource("classpath:demo.properties")
- In main test Console Logger and File Logger.

ConfigurationProperties

Demo 06 (@ConfigurationProperties and @ConfigurationPropertiesBinding)

- Demo 06 Continued.
- Declare Person p1 and p2 properties into demo.properties.

- In AppConfig class create p1 & p2 @Bean with appropriate @ConfigurationProperties(prefix="??").
- Declare Account a1 and a2 properties into demo.properties.
- In AppConfig class create a1 & a2 @Bean with appropriate @ConfigurationProperties(prefix="??").
- In demo.properties
    - a1.accHolder = p1
    - a2.accHolder = p2
- For these ConfigurationProperties binding special converter should be implemented to convert "p1" String to p1 @Bean.
- So implement PersonConverterImpl class inherited from Converter<String,Person> interface and implement its convert() method.
    - Mark this class as @ConfigurationPropertiesBinding and @Component (spring bean).
    - To have access to ApplicationContext in this bean class use ApplicationContextAware interface.
- In main() test p1, p2, a1 and a2.

SpEL

Demo 06 (@Value)

- Demo 06 Continued.
- Create @Component Employee and initialize it in constructor.
- Create @Component EmployeeContact and initialize initialize its members using SpEL @Value.
- In main() test EmployeeContact.

Auto-wiring

Demo 07 (Auto-wiring)

- Copy Demo 06 project.
- Remove Employee and EmployeeContact class.

Test 1

- Comment @Component from ConsoleLoggerImpl and FileLoggerImpl.
- Use @Autowired for AccountImpl class's "logger" field.
- In main() test Account a1 deposit().
- Expected output: Error - no bean found for auto-wiring "logger".

Test 2

- Comment @Component from ConsoleLoggerImpl and FileLoggerImpl.
- Use @Autowired(required=false) for AccountImpl class's "logger" field.
- In main() test Account a1 deposit().
- Expected output: No Error but logger is not auto-wired, so no deposit() log produced.

Test 3

- Mark @Component from ConsoleLoggerImpl and FileLoggerImpl.
- Use @Autowired for AccountImpl class's "logger" field.

- In main() test Account a1 deposit().
- Expected output: Error - Expected single bean for "logger" found 2 beans. Either use @Primary or @Qualifier.

Test 4

- Mark @Component from ConsoleLoggerImpl and FileLoggerImpl.
- Use @Autowired and @Qualifier("fileLogger") for AccountImpl class's "logger" field.
- In main() test Account a1 deposit().
- Expected output: No error. File logger is autowired and deposit() log added into log file.

Test 5

- Mark @Component from ConsoleLoggerImpl and FileLoggerImpl.
- Mark @Primary to ConsoleLoggerImpl.
- Use @Autowired for AccountImpl class's "logger" field.
- In main() test Account a1 deposit().
- Expected output: No error. Console logger is autowired and deposit() log displayed on console.

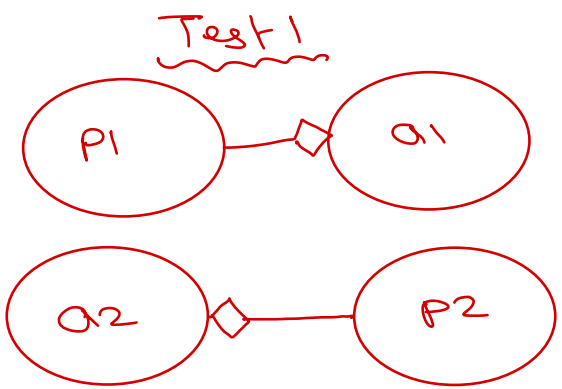
Java EE

Trainer: Nilesh Ghule

Auto-wiring

Auto-wiring

- Automatically injecting appropriate dependency beans into the dependent beans.



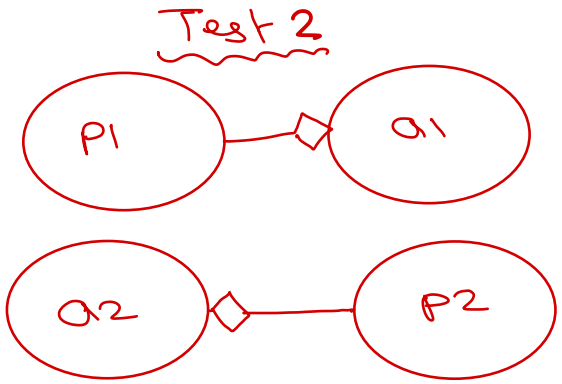
Account Impl

@Autowired

Logger logger;

↓

error: no bean found.

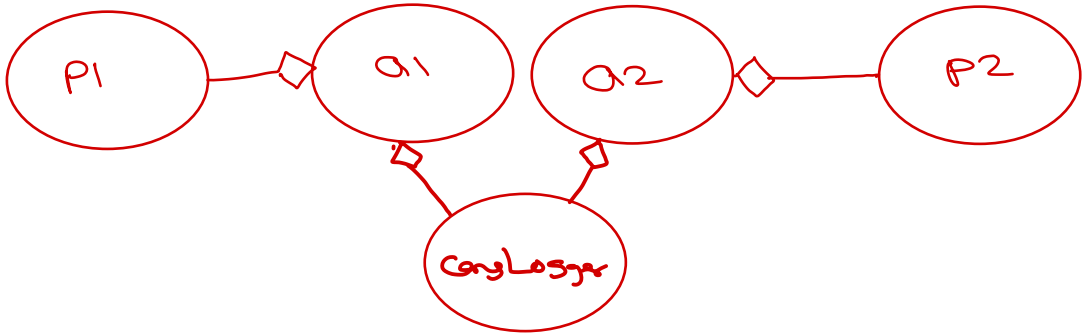


Account Impl

@Autowired(required=false)

Logger logger;

no bean found so logger is kept null. no error raised.



when single bean of logger is available, it is auto wired.

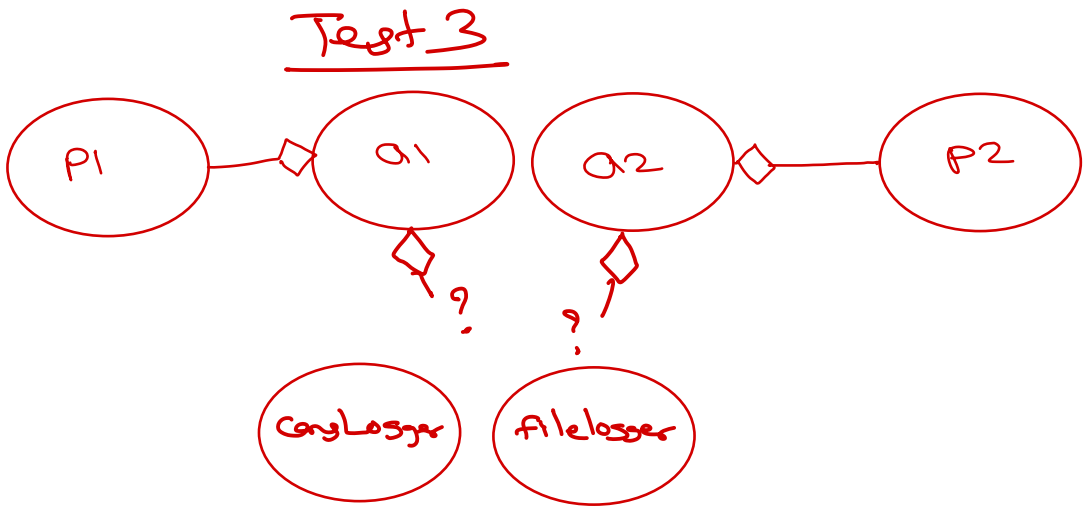
Account Impl

@Autowired

Logger logger;

Auto-wiring

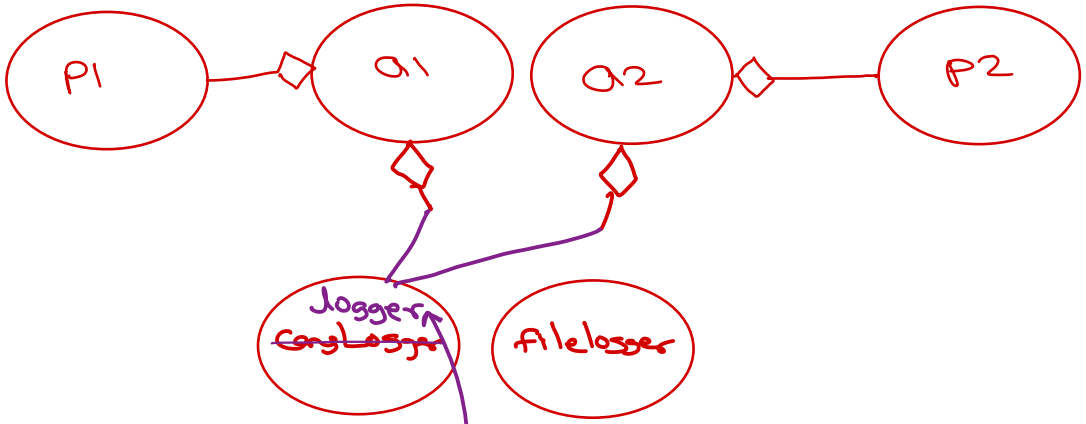
- Automatically injecting appropriate dependency beans into the dependent beans.



Account Impl

@Autowired

Logger logger;



Account Impl

@Autowired

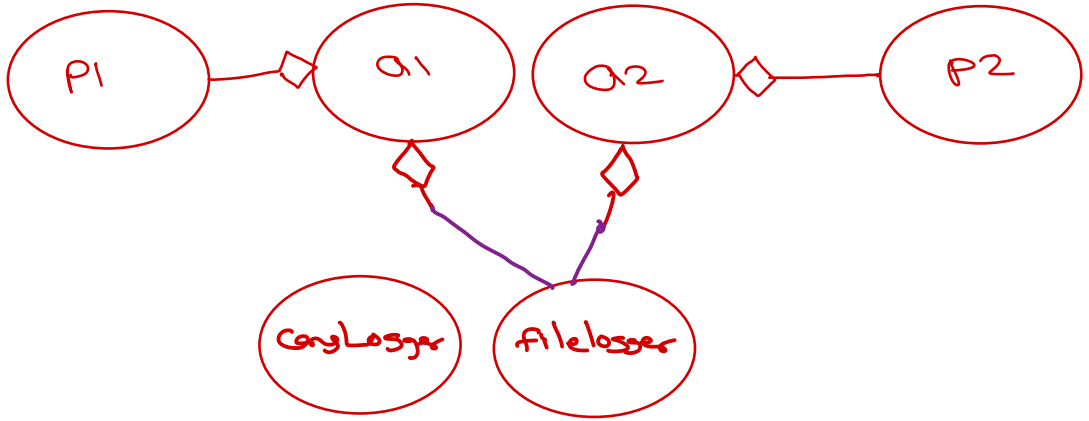
Logger logger;

If multiple beans with same type, it will find bean with name same as fieldname. If found, it will auto wire.

Auto-wiring

- Automatically injecting appropriate dependency beans into the dependent beans.

Test 4

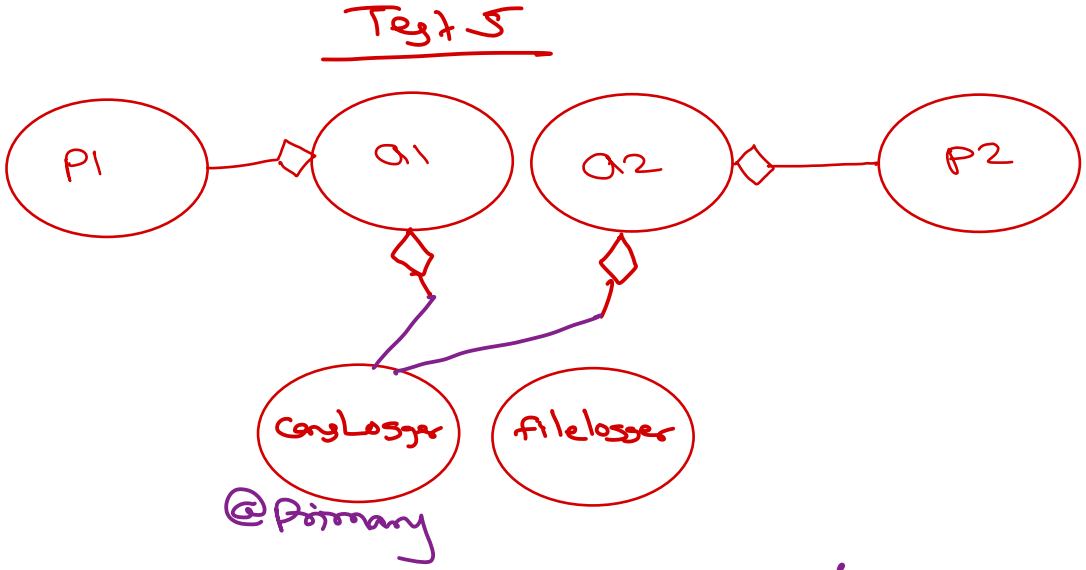


Account Impl

```
@Qualifier("fileLogger")
@Autowired
Logger logger;
```

The ambiguity can be resolved by specifying bean to be autowired using @Qualifier.

Test 5



In case of multiple beans of same type, raise importance of one of the bean using @Primary. So if conflict, the primary bean will be selected by spring container (instead of error).

Auto-wiring (XML config)

- <bean id="..." class="..." autowire="default|no|byType|byName|constructor" .../>
  - default / no: Auto-wiring is disabled.
  - byType: Dependency bean of property type will be assigned (via setter).
    - If multiple beans of required type are available, then exception is thrown.
    - If no bean of required type is available, auto-wiring is not done.
  - byName: Dependency bean of property name will be assigned (via setter).
    - If no bean of required name is available, auto-wiring is not done.
  - constructor: Dependency bean of property type will be assigned via single argument constructor (of bean type).
- <bean id="..." class="..." autowire-candidate="true|false" .../>
  - false -- do not consider this dependency bean for auto-wiring.

Auto-wiring (Annotation config)

- @Autowired
  - Setter level: setter based DI
  - Constructor level: constructor based DI
  - Field level: field based DI
- @Autowired: Find bean of corresponding field type and assign it.
  - If no bean is found of given type, it throws exception.
    - @Autowired(required=false): no exception is thrown, auto-wiring skipped.
  - If multiple beans are found of given type, it try to attach bean of same name. and if such bean is not found, then throw exception.
  - If multiple beans are found of given type, programmer can use @Qualifier to choose expected bean. @Qualifier can only be used to resolve conflict in case of @Autowired.
- @Resource (JSR 250)
  - @Resource: DI byName, byType, byQualifier
- @Inject (JSR 330) – same as @Autowired
  - @Inject: DI byType, byQualifier (@Named), byName

Mixed configuration

- XML config + Annotations
  - XML config file is loaded using ClassPathXmlApplicationContext.
  - <context:annotation-config/> activate annotation processing.
    - @PostConstruct
    - @PreDestroy
    - @Autowired
    - @Qualifier

singleton class

↓

only one object of singleton class is created.

if tried to create next obj, get ref of already created object.

spring bean

↓

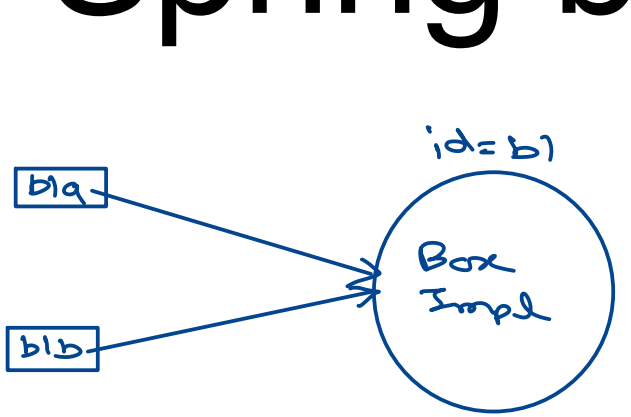
object of a java class created by spring container.

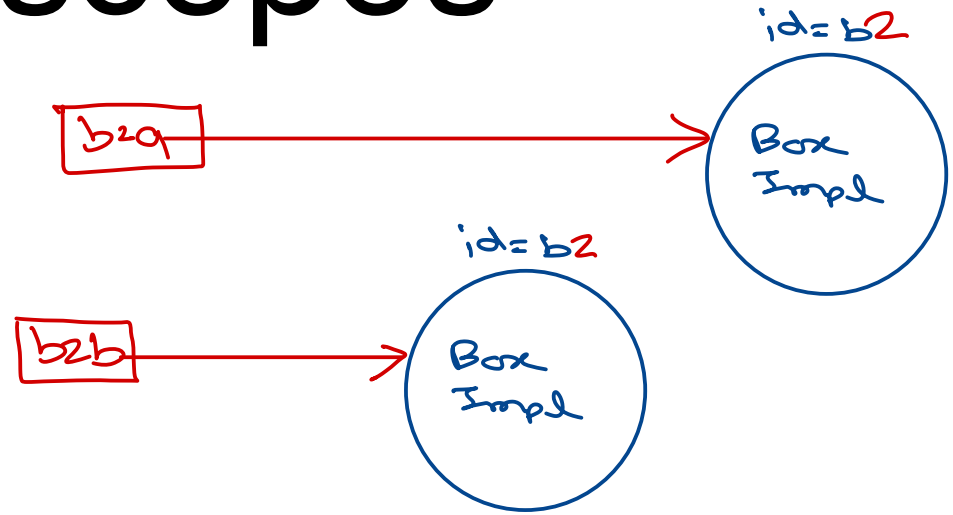
singleton bean

↓

- only one bean object is created (of given id).
- typically when appctx is created.
- each access to bean return same obj.
- managed/stored in spring container.
- destroyed when spring container shutdown.

Spring bean scopes

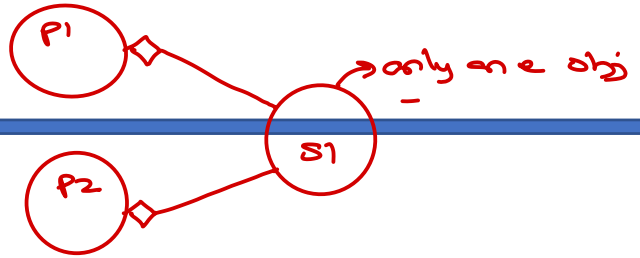




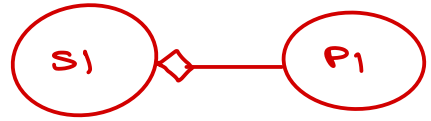
Bean scopes

- Bean scope can be set in XML or annotation.
  - <bean id="___" class="___" scope="singleton|prototype|request|session" />
  - @Scope("singleton|prototype|request|session")
- singleton
  - Single bean object is created and accessed throughout the application.
  - XMLBeanFactory creates object when getBean() is called for first time for that bean.
  - ApplicationContext creates object when ApplicationContext is created.
  - For each sub-sequent call to getBean() returns same object reference.
  - Reference of all singleton beans is managed by spring container.
  - During shutdown, all singleton beans are destroyed (@PreDestroy will be called).
- prototype
  - No bean is created during startup.
  - Reference of bean is not maintained by ApplicationContext.
  - Beans are not destroyed automatically during shutdown.
  - Bean object is created each time getBean() is called.
- request and session: scope limited to current request and session.

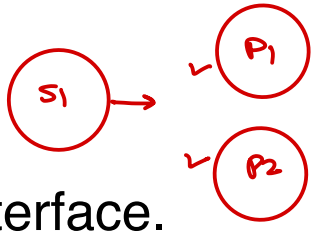
Bean scopes



- Singleton bean inside prototype bean
  - Single singleton bean object is created.
  - Each call to getBean() create new prototype bean. But same singleton bean is autowired with them.
- Prototype bean inside singleton bean



  - Single singleton bean object is created.
  - While auto-wiring singleton bean, prototype bean is created and is injected in singleton bean.
  - Since there is single singleton bean, there is a single prototype bean.
- Need multiple prototype beans from singleton bean (solution 1)

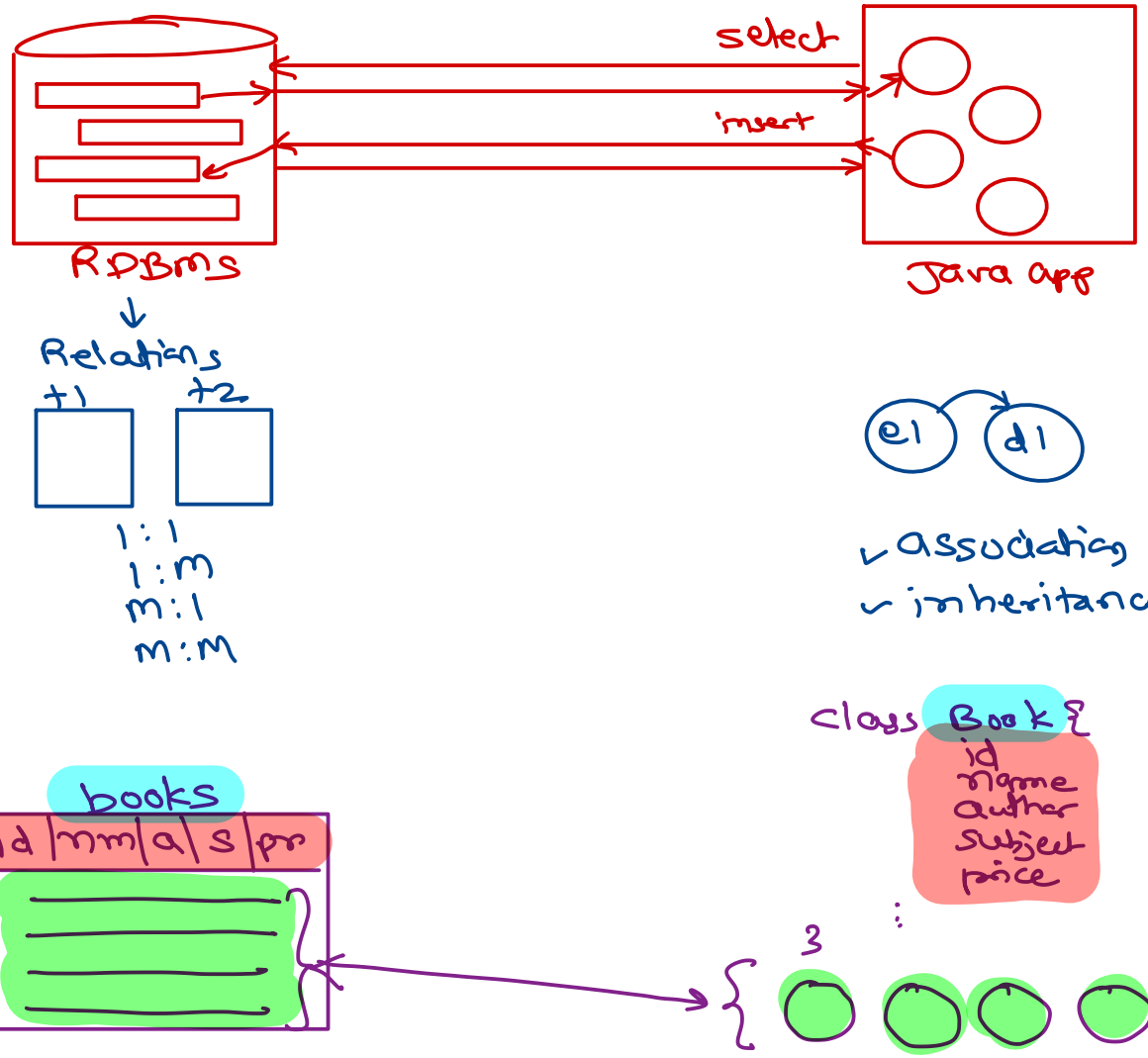


  - Using ApplicationContextAware
  - The singleton bean class can be inherited from ApplicationContextAware interface.
  - When its object is created, container call its setApplicationContext() method and give current ApplicationContext object. This object can be used to create new prototype bean each time (as per requirement).
- Need multiple prototype beans from singleton bean (solution 2)
  - using @Lookup method
  - The singleton bean class contains method returning prototype bean.
  - If method is annotated with @Lookup, each call to the method will internally call ctx.getBean(). Hence for prototype beans, it returns new bean each time.

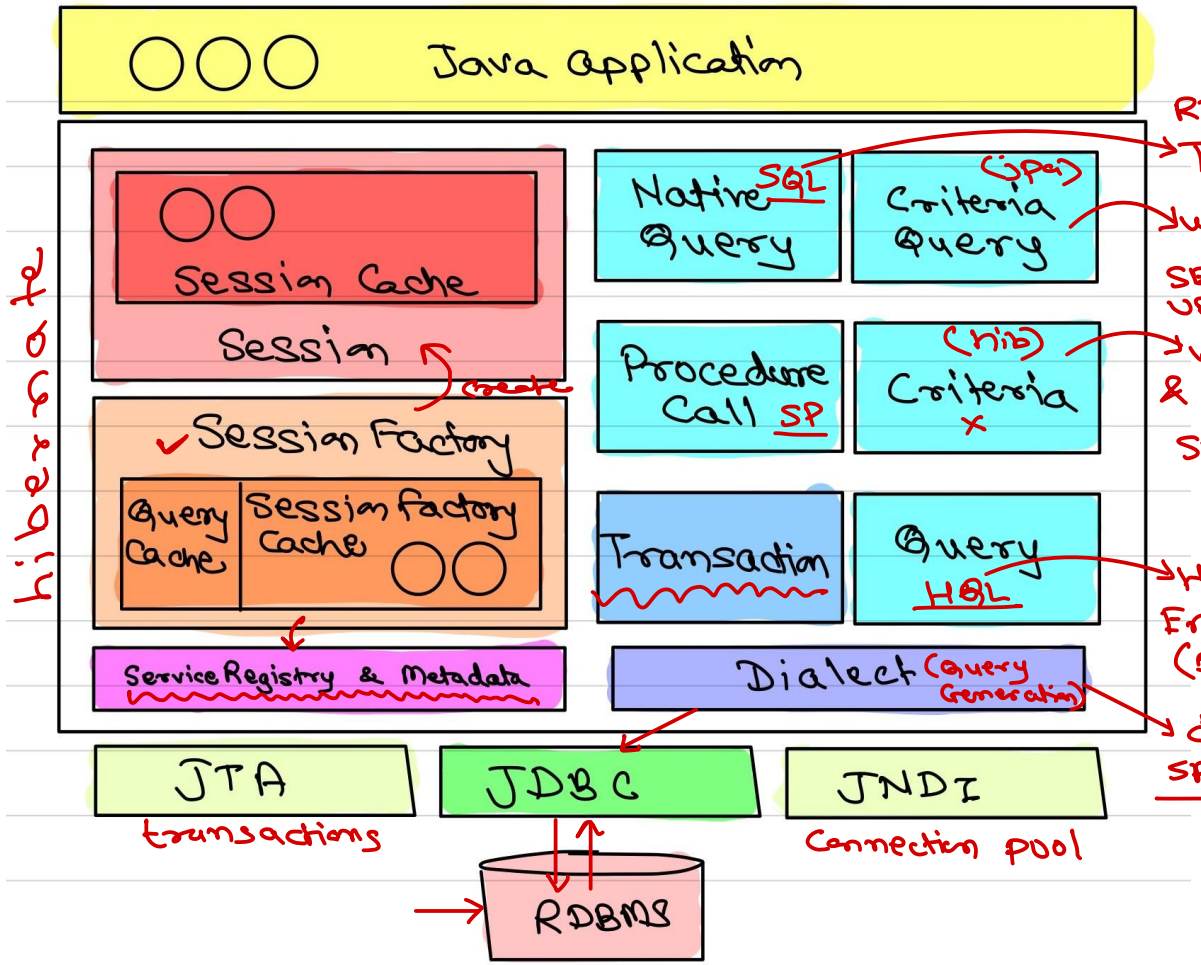
Hibernate

Object Relational Mapping

- Converting Java objects into RDBMS rows and vice-versa is done manually in JDBC code.
- This can be automated using Object Relational Mapping.
- Class → Table and Field → Column
- It also map table relations into entities associations/inheritance and auto-generates SQL queries.
- Hibernate is most popular ORM tool.
- Other popular ORM are EclipseLink, iBatis, Torque, ...
- **JPA is specification for ORM.**
  - ↳ Java Persistence API → Sun/Oracle.

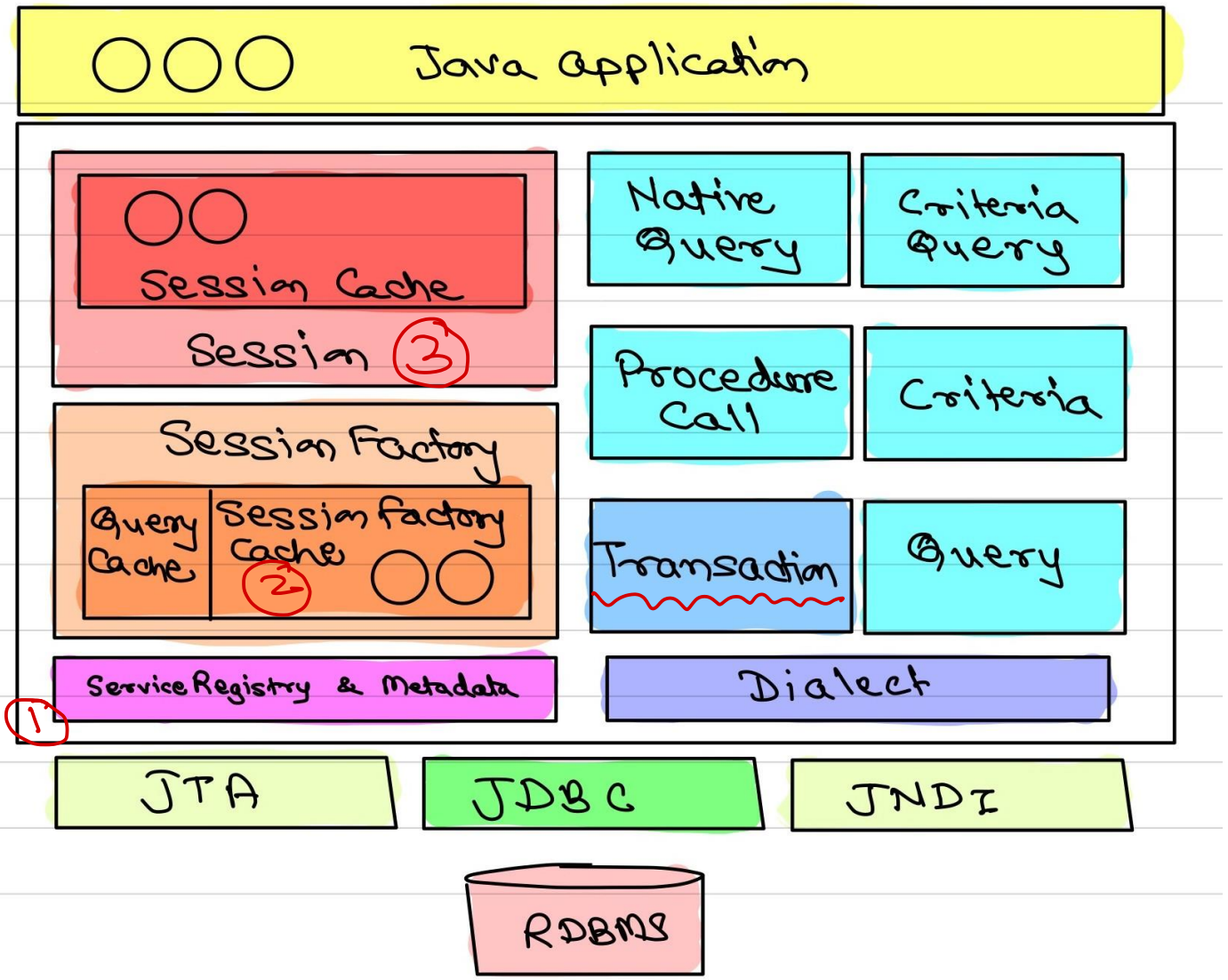


Hibernate



- SessionFactory
  - One SessionFactory per application (per db).
  - Heavy-weight object. Not recommended to create multiple instances.
  - Thread-safe. Can be accessed from multiple threads (synchronization is built-in).
  - Typical practice is to create singleton utility class for that.
- Session
  - Created by SessionFactory & it encapsulate JDBC connection.
  - All hibernate operations are done on hibernate sessions.
  - Is not thread-safe. Should not access same session from multiple threads.
  - Light-weight. Can be created and destroyed as per need.

Hibernate



- Transaction
  - In hibernate, autocommit is false by default.
  - DML operations should be performed using transaction.
  - session.beginTransaction()
    - to start new transaction.
  - tx.commit()
    - to commit transaction.
  - tx.rollback()
    - to rollback transaction.

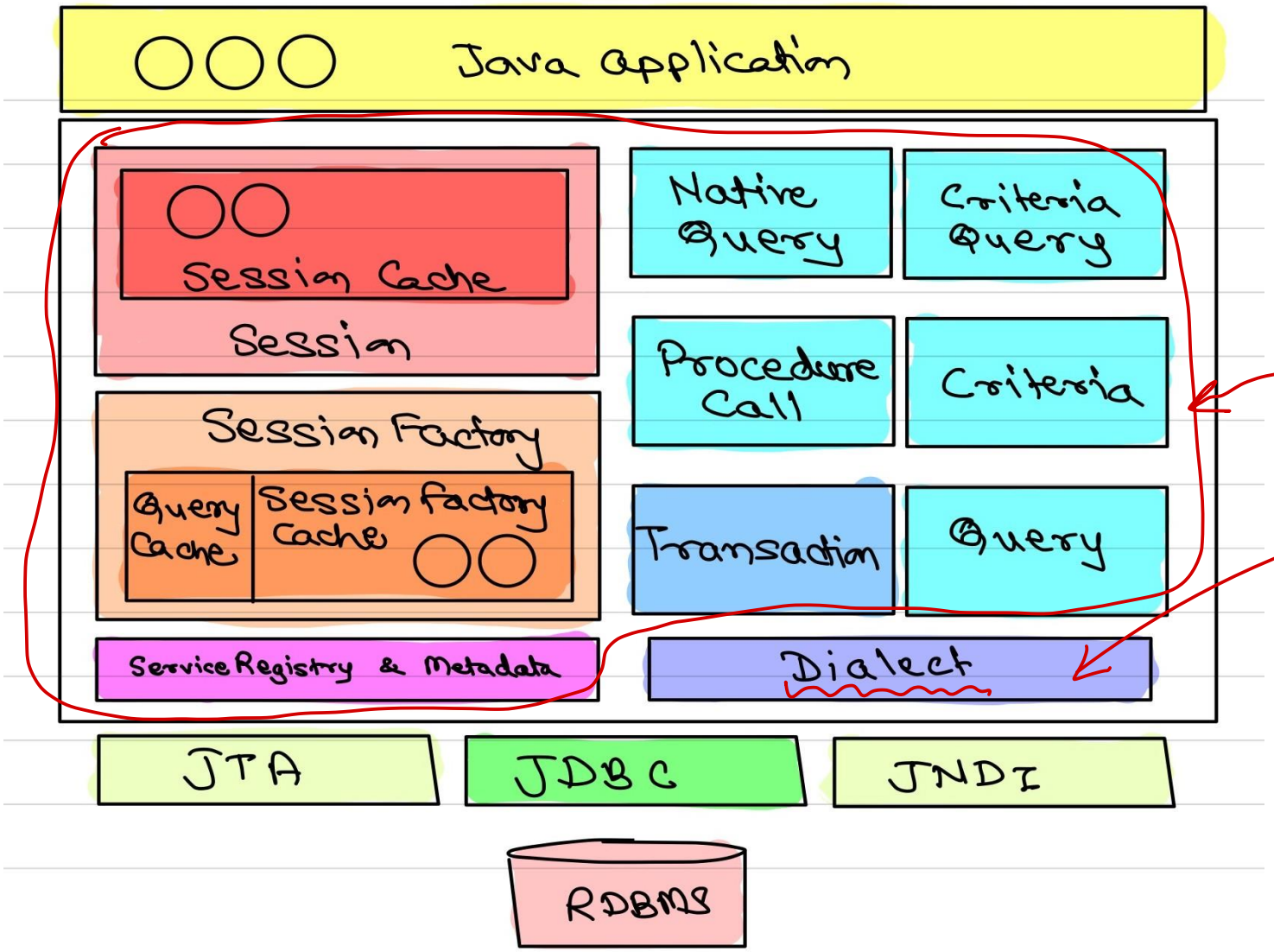
```
try {
  // begin tx
  tx = session.beginTx();
  ...
  // commit
  tx.commit();
} catch(...) {
  // roll back
  tx.rollback();
}
```

- Hibernate CRUD methods
  - get() → select
  - save() → insert
  - update() → update
  - delete() → delete

session

transaction

Hibernate



- Dialect
  - RDBMS have specific features like data types, stored procedures, primary key generation, etc.
  - Hibernate support all RDBMS.
  - Most of code base of Hibernate is common.
  - Database level changes are to be handled specifically and appropriate queries should be generated. This is handled by Dialect.
  - Hibernate have dialects for all RDBMS. Programmer should configure appropriate dialect to utilize full features of RDBMS.

Hibernate 3 Bootstrapping

```
public class HbUtil {
  private static SessionFactory factory;

  static {
    try {
      Configuration cfg = new Configuration();
      cfg.configure();
      factory = cfg.buildSessionFactory();
    } catch (Exception e) {
      e.printStackTrace();
    }
  }

  public static SessionFactory getSessionFactory() {
    return factory;
  }

  public static void shutdown() {
    factory.close();
  }
}
```

- Hibernate Configuration (hibernate.cfg.xml)
  - hibernate.connection.driver_class
  - hibernate.connection.url
  - hibernate.connection.username
  - hibernate.connection.password
  - hibernate.dialect
  - hibernate.show_sql
- Hibernate3 Bootstrapping
  1. Create Configuration object.
  2. Read hibernate.cfg.xml file using its configure() method.
  3. Create SessionFactory using its buildSessionFactory() method.

Hibernate 5 Bootstrapping

```
public class HbUtil {
  private static final SessionFactory factory
      = createSessionFactory();
  private static ServiceRegistry serviceRegistry;

  private static SessionFactory createSessionFactory() {
    serviceRegistry = new StandardServiceRegistryBuilder()
        .configure() // read from hibernate.cfg.xml
        .build();

    Metadata metadata = new MetadataSources(serviceRegistry)
        .getMetadataBuilder()
        .build();

    return metadata.getSessionFactoryBuilder().build();
  }
  public static void shutdown() {
    factory.close();
  }
  public static SessionFactory getSessionFactory() {
    return factory;
  }
}
```

Hibernate 5 Bootstrapping

- Create ServiceRegistry.
- Create Metadata.
- Create SessionFactory.

ServiceRegistry

- ServiceRegistry is interface.
- Some implementations are StandardServiceRegistry, BootstrapServiceRegistry, EventListenerRegistry, ...
- Add, manage hibernate services.

Metadata

- Represents application's domain model & its database mapping.

https://docs.jboss.org/hibernate/orm/4.3/topical/html/registries/ServiceRegistries.html

Hibernate

- Hibernate3 added annotations for ORM.

- ORM using annotations
  - @Entity
  - @Table
  - @Column
  - @Id
  - @Temporal
  - @Transient

to convert date & time classes from java.sql to java.util.Date and/or java.util.Calendar.

Can be used to map

Temporal.DATE → mysql DATE

Temporal.TIME → mysql TIME

Temporal.TIMESTAMP → mysql DATETIME TIMPSTAMP

- @Column can be used on field level or on getter methods.

Hibernate Transaction

- In hibernate, autocommit is false by default.
- DML operations should be performed using transaction.
  - session.beginTransaction(): to start new tx.
  - tx.commit() & tx.rollback(): to commit/rollback tx.
- session.flush()
  - Forcibly synchronize in-memory state of hibernate session with database.
  - Each tx.commit() automatically flush the state of session.
  - Manually calling flush() will result in executing appropriate SQL queries into database.
  - Note that flush() will not commit the data into the RDBMS tables.
  - The flush mode can be set using session.setHibernateFlushMode(mode).
    - ALWAYS, AUTO, COMMIT, MANUAL
- If hibernate.connection.autocommit is set to true, we can use flush to force executing DML queries.

Thank you!

Nilesh Ghule <nilesh@sunbeaminfo.com>

Java EE

Agenda

- Auto-wiring
- Bean scopes
- Hibernate
- ~~Spring Hibernate Integration~~

Auto-wiring

- Demo 07
- Refer yesterday's PDF for steps
- Refer slides for better understanding

Bean scopes

- Demo 08

Demo 08 (Singleton & Prototype)

- Implement Box interface and BoxImpl class. In BoxImpl, also implement BeanNameAware interface.
- In AppConfig create @Bean b1 as Singleton and @Bean b2 as Prototype.
- In main() try creating objects of the beans.
- Note that:
    - "b1" bean is created at the start of application. The getBean() returns same reference.
    - "b2" bean is created each time getBean() is called. Obviously returns different reference for each call.

Demo 08 (Prototype bean inside Singleton bean)

- Demo 08 continued.
- Implement Inner1 as Prototype bean and Outer1 as Singleton bean. Autowire Inner1 bean inside Outer1.
- Create Outer1 bean in main() and try calling getInner() multiple times.
- Note that:
    - Both Outer1 & Inner1 beans are created at the start of appplication. Inner1 is created to Autowire in Outer1.
    - Multiple calls to getInner() returns same Inner1 bean reference.

Demo 08 (Prototype bean inside Singleton bean)

- Demo 08 continued.
- Implement Inner2 as Prototype bean and Outer2 as Singleton bean.
- Inherit Outer2 from ApplicationContext and return Inner2 bean from getInner() method (using ctx.getBean()).
- Create Outer2 bean in main() and try calling getInner() multiple times.

- Note that:
  - Outer2 bean is created at the start of application.
  - Multiple calls to getInner() returns different (new) Inner2 bean reference.

Demo 08 (Prototype bean inside Singleton bean)

- Demo 08 continued.
- Implement Inner3 as Prototype bean and Outer3 as Singleton bean.
- Inherit Outer3 from ApplicationContext and return null from getInner() method and mark that method as @Lookup.
- Create Outer3 bean in main() and try calling getInner() multiple times.
- Note that:
  - Outer3 bean is created at the start of application.
  - Multiple calls to getInner() returns different (new) Inner3 bean reference.

Hibernate

Demo 09 (Hibernate CRUD operations)

- Create new Maven project. In pom.xml set Java version 11 and add dependencies hibernate-core and mysql-connector-java.
- Create resources/hibernate.cfg.xml to specify database params.
- Write HbUtil class to create singleton SessionFactory.
- Create Book class and do ORM mapping using annotations.
- Add Book class mapping into hibernate.cfg.xml `<mapping/>`.
- Create BookDao class and provide methods for getBook(), addBook(), updateBook(), deleteBook(), getAll(), getSubjectBooks().
- Write a main class and test all methods into main() one by one.

Assignment

- Implement Customer entity class.

```
@Entity
@Table(name="customers")
class Customer {
    // ...
    @Column
    @Temporal(TemporalType.DATE)
    private Date birth; // java.util.Date
    // ...
}
```

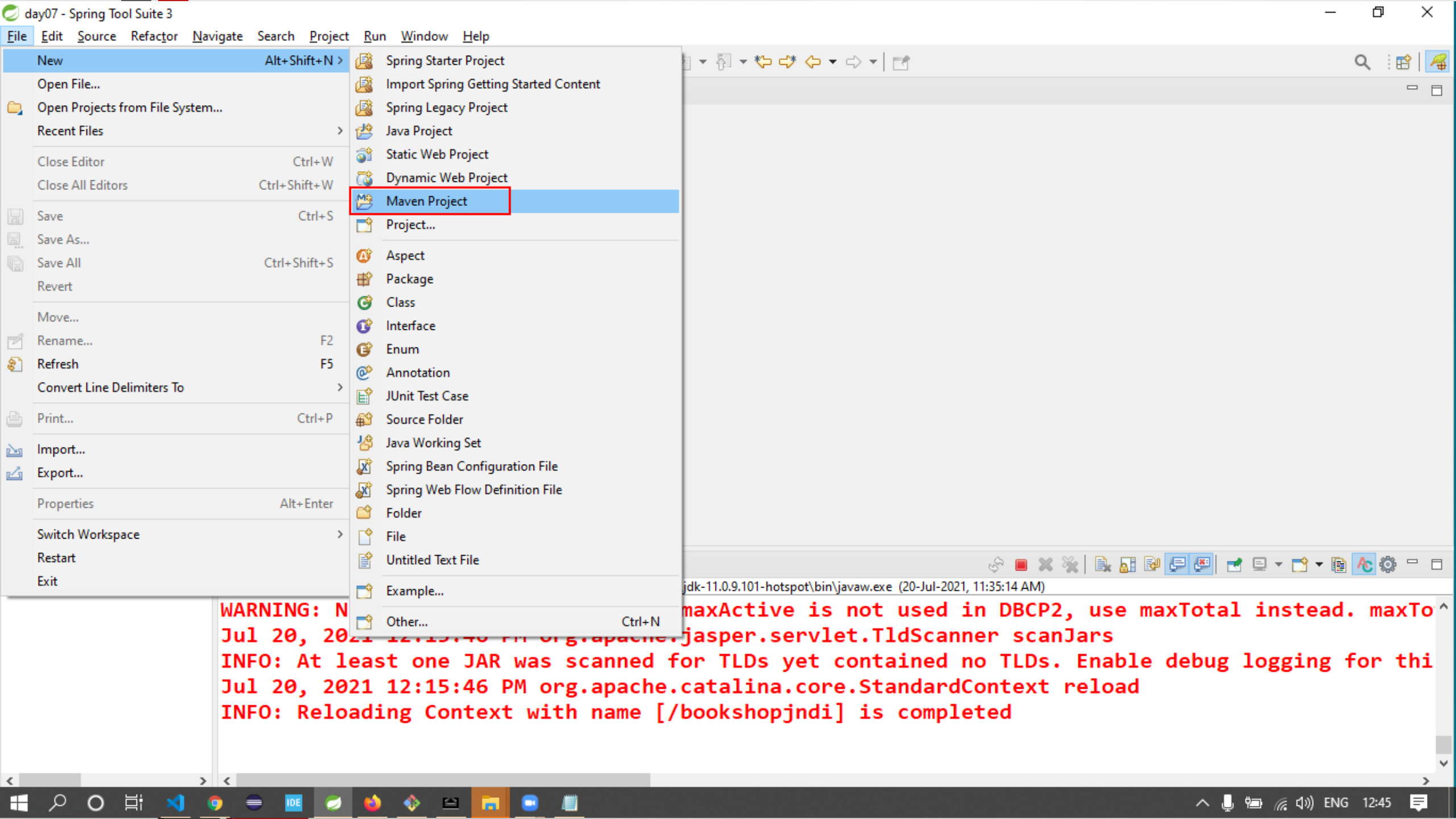
- Implement CustomerDao with folling functinalities.
  - Customer findById(int id);
  - Customer findByEmail(String email);
  - List findAll();
  - void update(Customer cust);

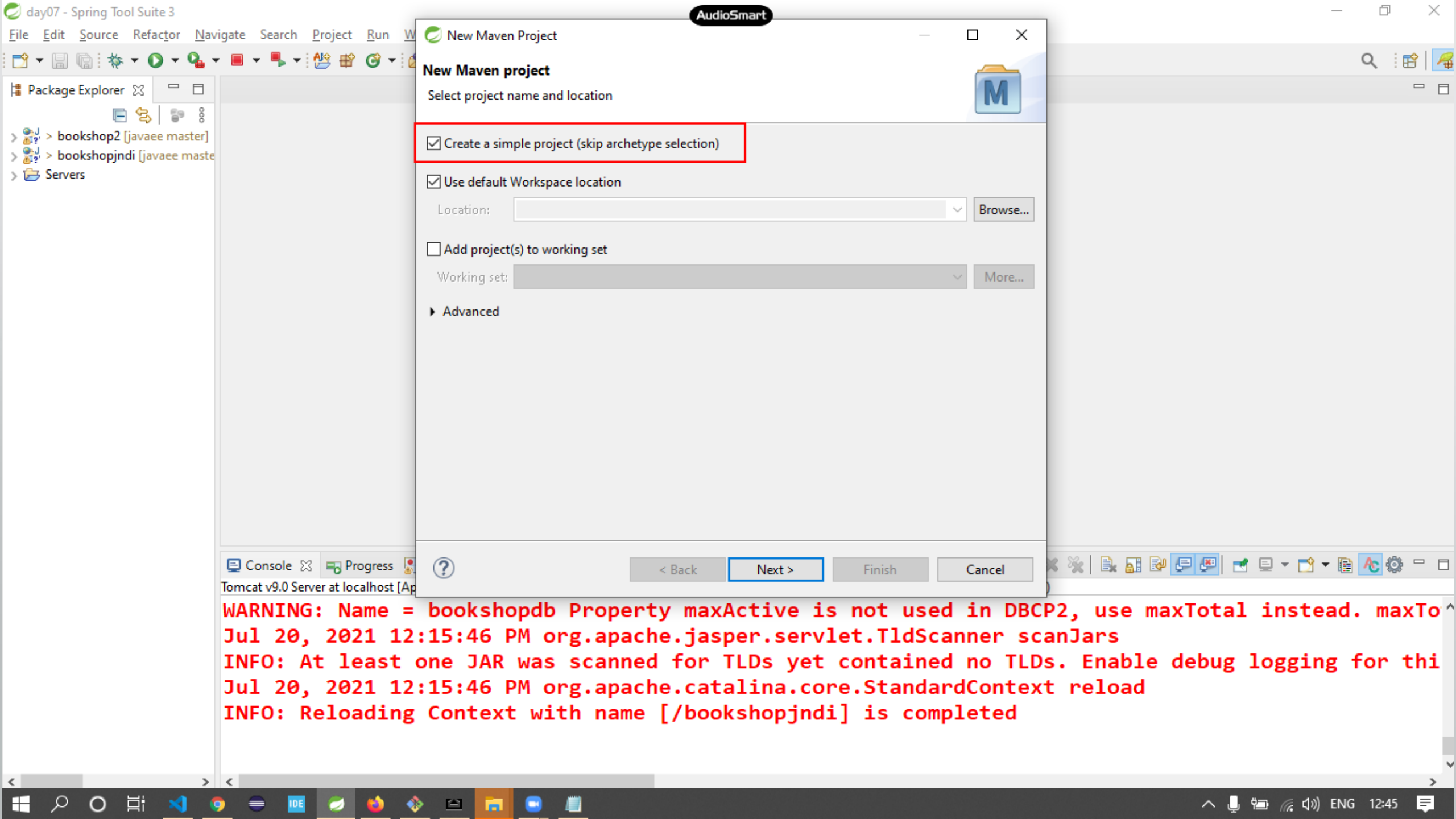
    - void deleteById(int id);
    - void save(Customer cust);
- Write a main class and implement all steps one by one.

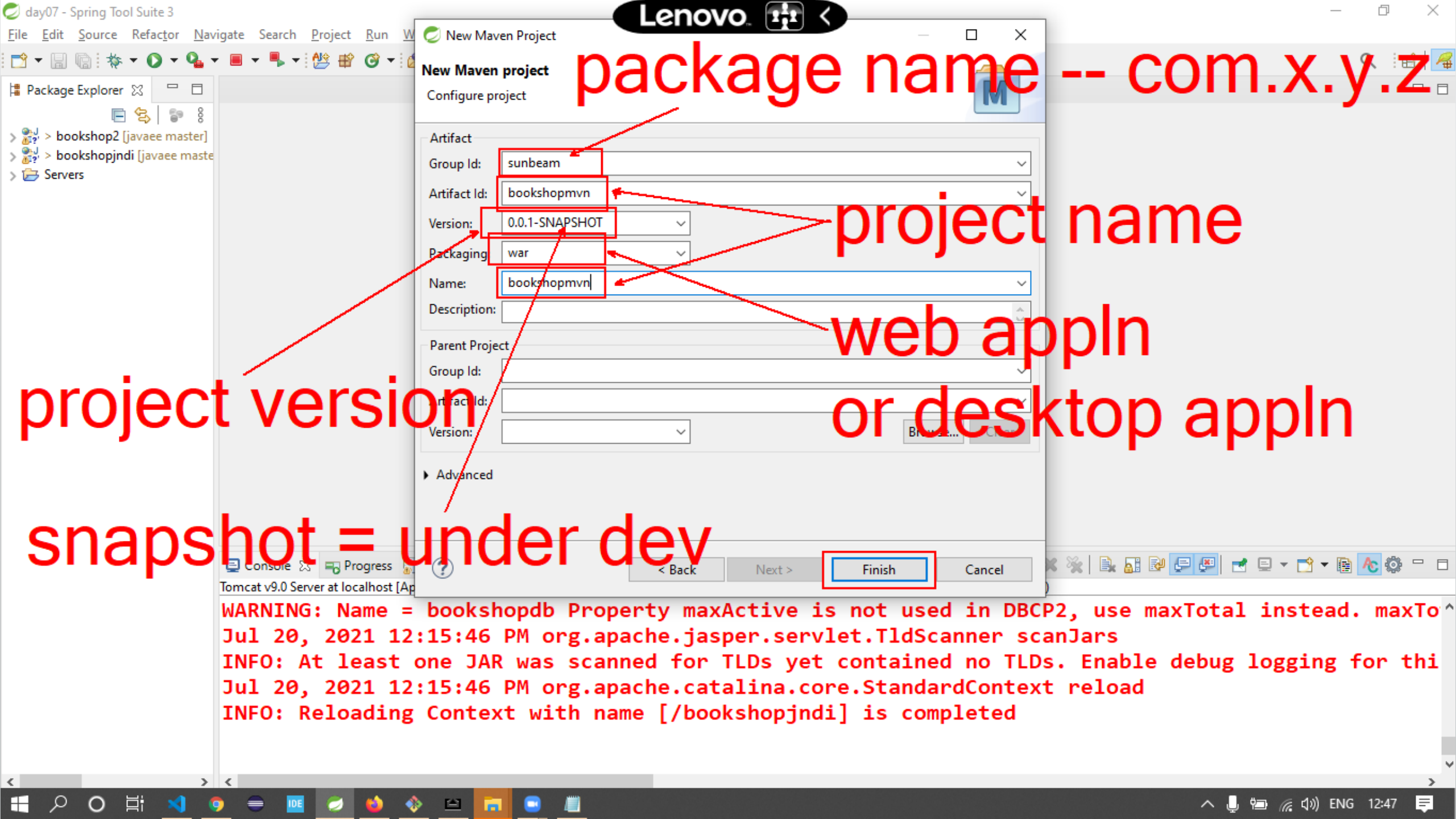
Java Web Application using Maven

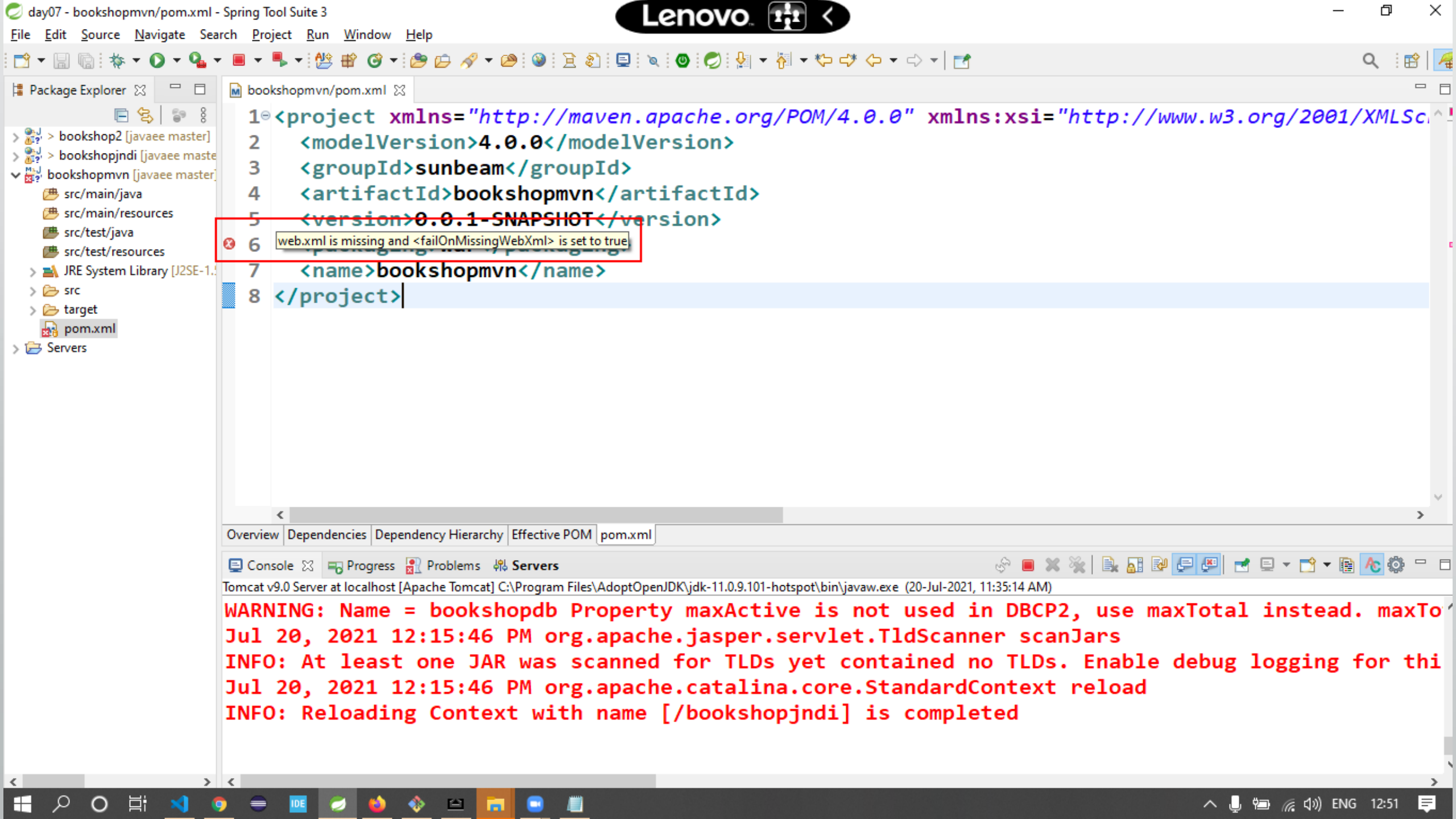
Nilesh Ghule @ Sunbeam Infotech

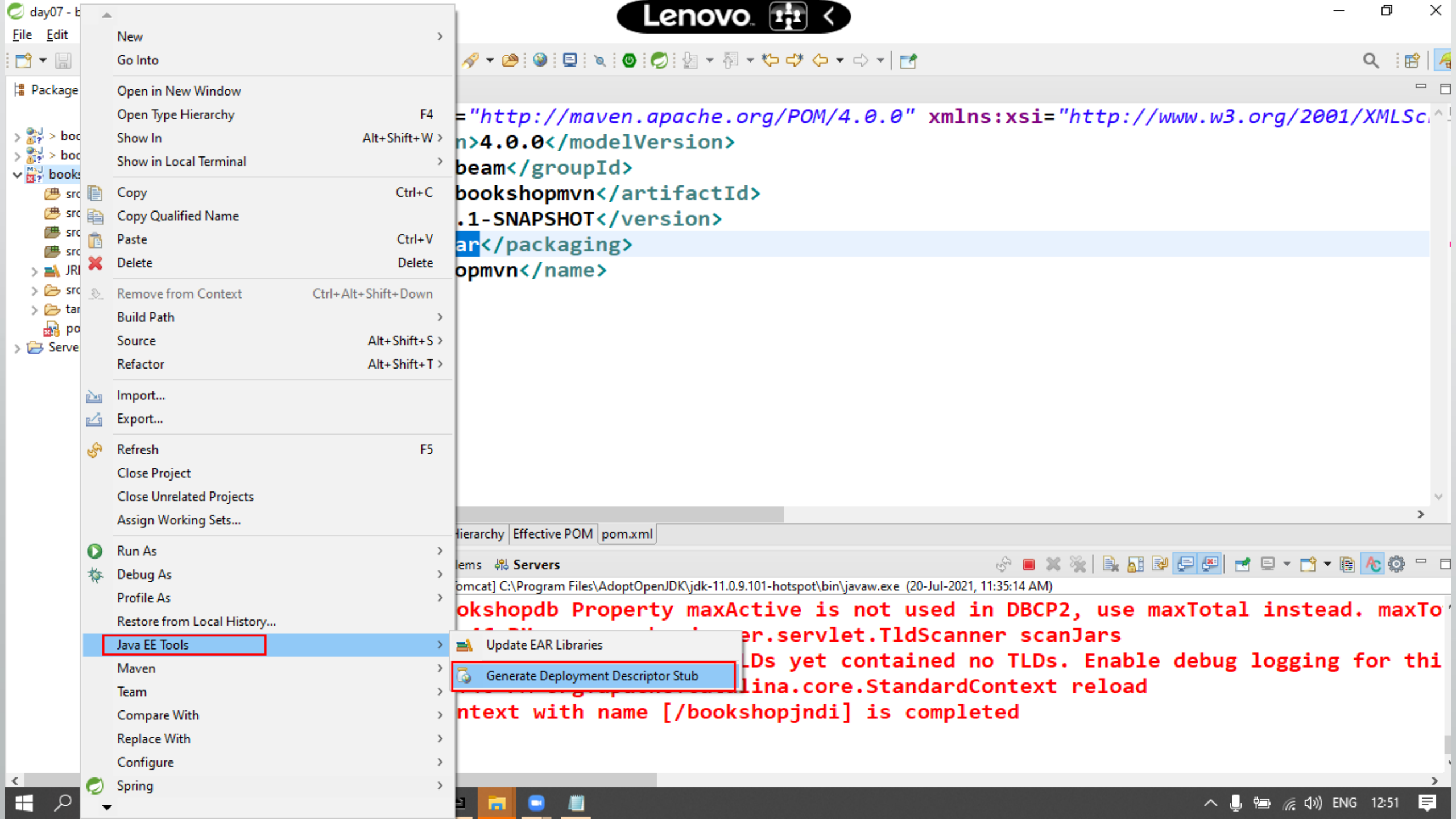
Develop Maven web application in Eclipse

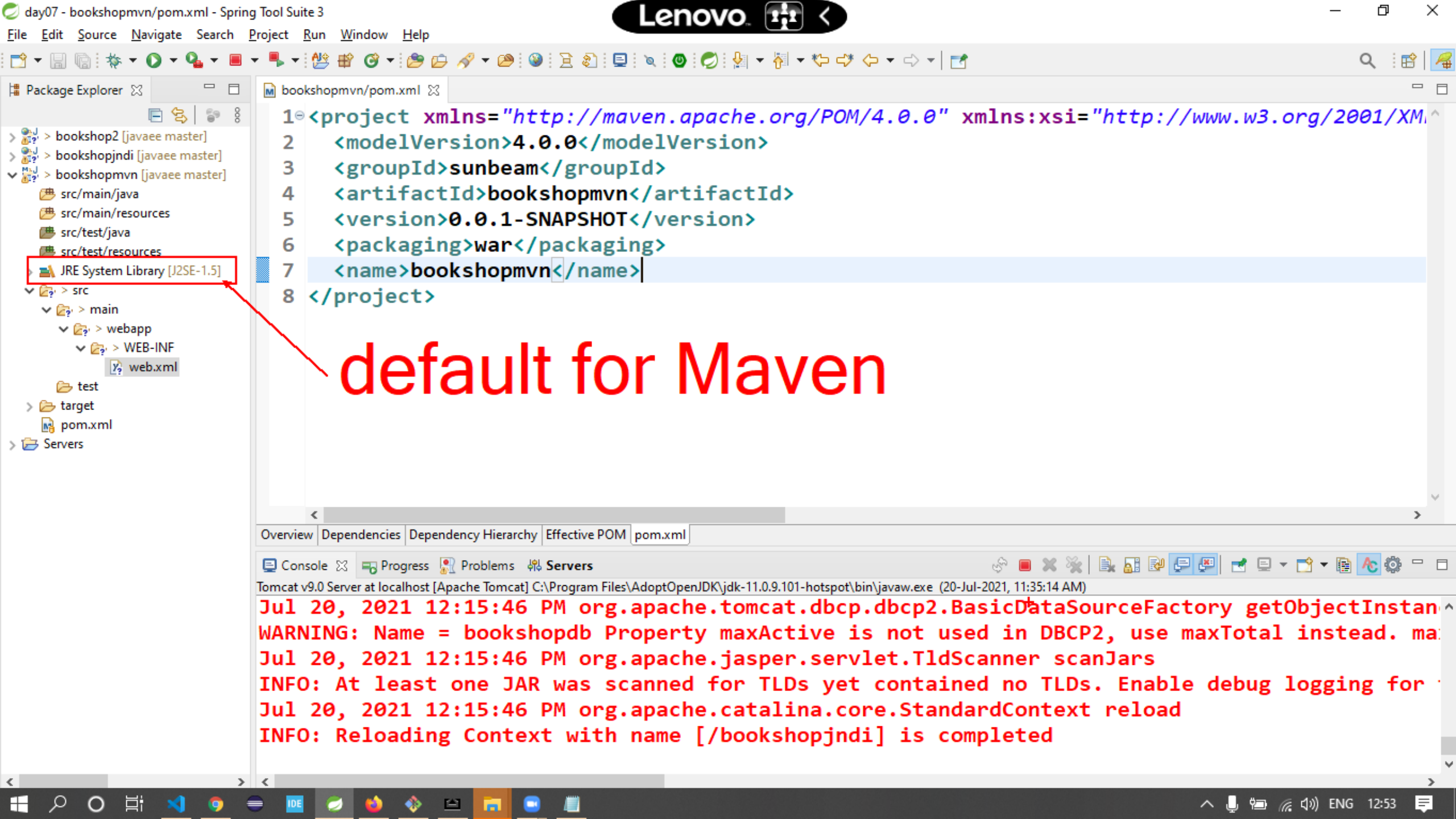

day07 - Spring Tool Suite 3
File Edit Source Refactor Navigate Search Project Run Window Help
New Alt+Shift+N
Open File...
Open Projects from File System...
Recent Files
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Revert
Move...
Rename... F2
Refresh F5
Convert Line Delimiters To
Print... Ctrl+P
Import...
Export...
Properties Alt+Enter
Switch Workspace
Restart
Exit
Spring Starter Project
Import Spring Getting Started Content
Spring Legacy Project
Java Project
Static Web Project
Dynamic Web Project
Maven Project
Project...
Aspect
Package
Class
Interface
Enum
Annotation
JUnit Test Case
Source Folder
Java Working Set
Spring Bean Configuration File
Spring Web Flow Definition File
Folder
File
Untitled Text File
Example...
Other... Ctrl+N
jdk-11.0.9.101-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
WARNING: N
maxActive is not used in DBCP2, use maxTotal instead. maxTo
Jul 20, 202
jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for thi
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
ENG 12:45

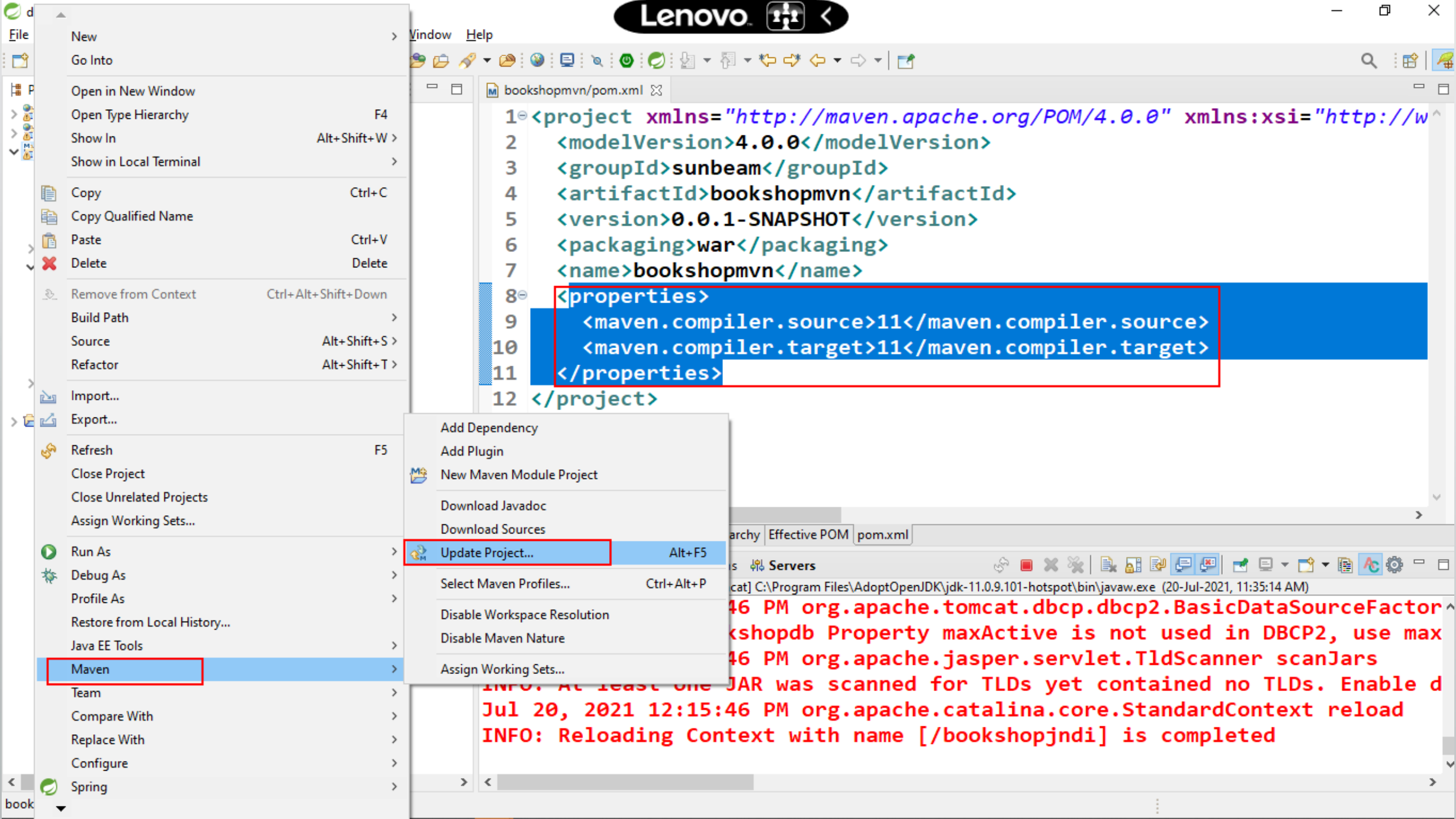

day07 - Spring Tool Suite 3
AudioSmart
File Edit Source Refactor Navigate Search Project Run
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee maste
Servers
New Maven Project
New Maven project
Select project name and location
Create a simple project (skip archetype selection)
Use default Workspace location
Location:
Browse...
Add project(s) to working set
Working set:
More...
Advanced
< Back
Next >
Finish
Cancel
Console
Progress
Tomcat v9.0 Server at localhost [Ap
WARNING: Name = bookshopdb Property maxActive is not used in DBCP2, use maxTotal instead. maxTo
Jul 20, 2021 12:15:46 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for thi
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
ENG
12:45

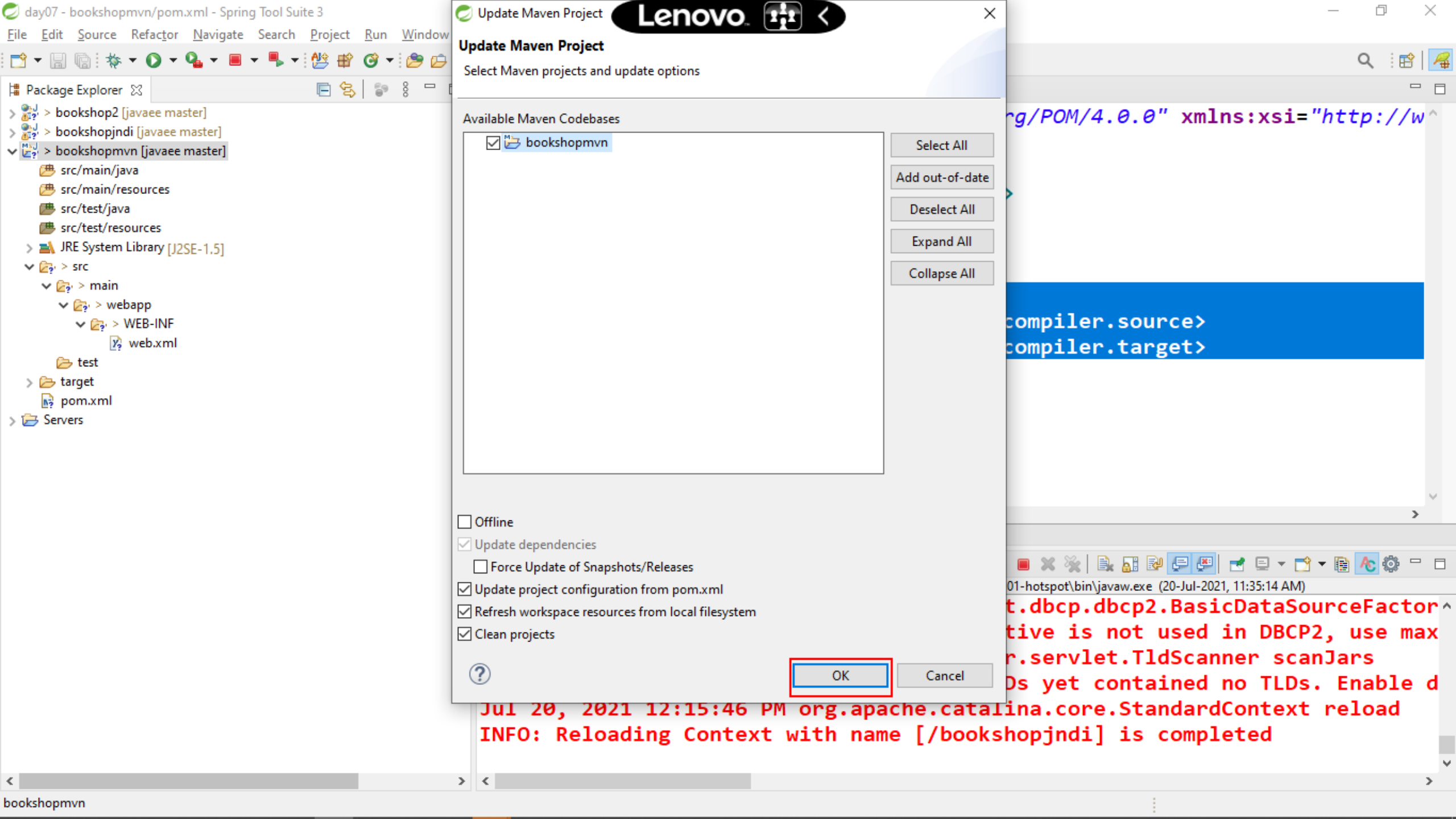

day07 - Spring Tool Suite 3
File Edit Source Refactor Navigate Search Project Run
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee maste
Servers
New Maven Project
New Maven project
Configure project
package name -- com.x.y.z
Artifact
Group Id: sunbeam
Artifact Id: bookshopmvn
Version: 0.0.1-SNAPSHOT
Packaging: war
Name: bookshopmvn
Description:
project name
web appln
or desktop appln
Parent Project
Group Id:
Version:
Advanced
project version
snapshot = under dev
< Back
Next >
Finish
Cancel
Console
Progress
Tomcat v9.0 Server at localhost [Ap
WARNING: Name = bookshopdb Property maxActive is not used in DBCP2, use maxTotal instead. maxTo
Jul 20, 2021 12:15:46 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for thi
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
ENG 12:47

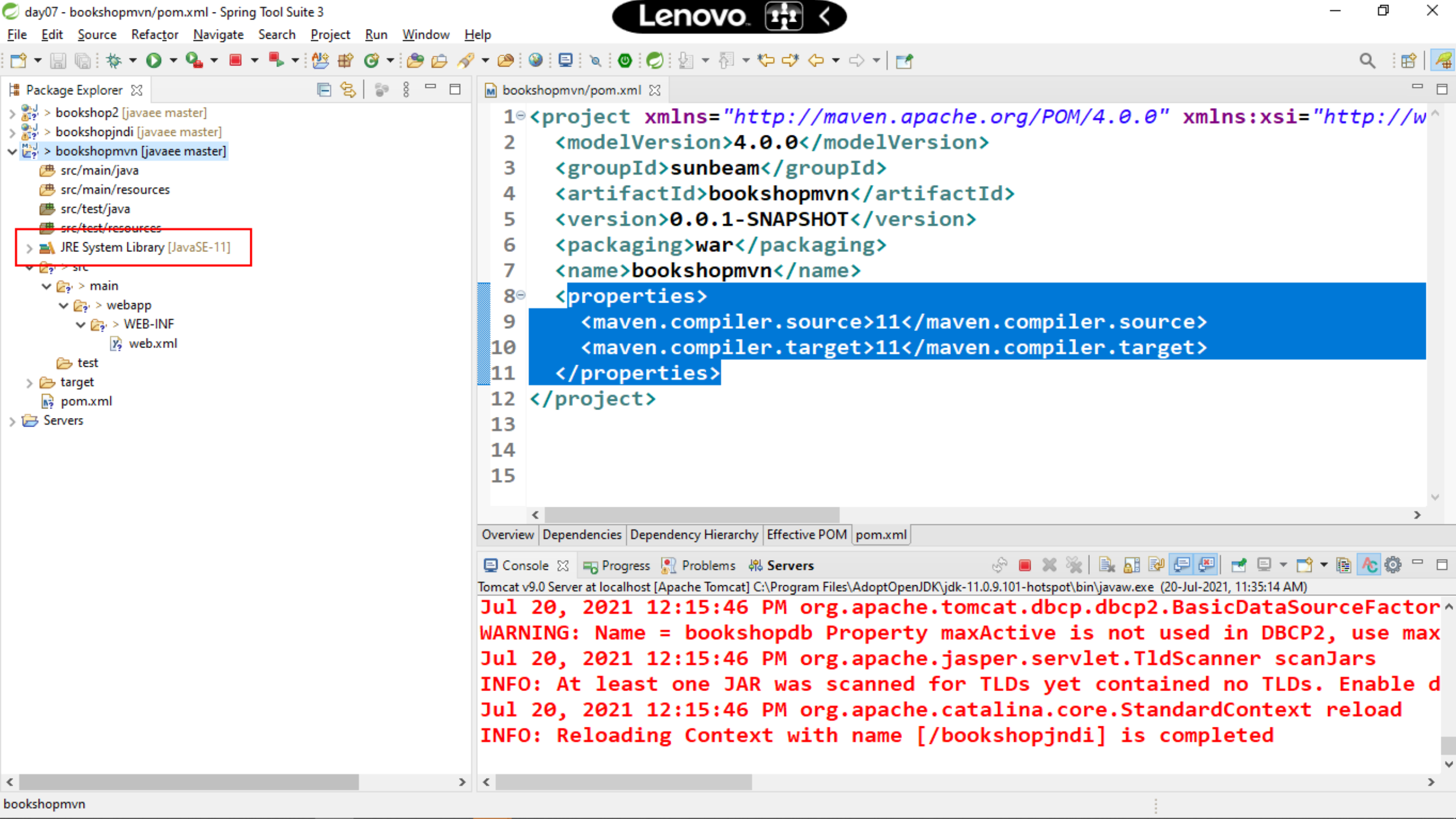

day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Navigate Search Project Run Window Help
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee maste
bookshopmvn [javaee master]
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.
src
target
pom.xml
Servers
bookshopmvn/pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSc
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>sunbeam</groupId>
4 <artifactId>bookshopmvn</artifactId>
5 <version>0.0.1-SNAPSHOT</version>
6 web.xml is missing and <failOnMissingWebXml> is set to true
7 <name>bookshopmvn</name>
8 </project>
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Console Progress Problems Servers
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
WARNING: Name = bookshopdb Property maxActive is not used in DBCP2, use maxTotal instead. maxTo
Jul 20, 2021 12:15:46 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for thi
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
ENG 12:51


File Edit
Package
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
Compare With
Replace With
Configure
Spring
Update EAR Libraries
Generate Deployment Descriptor Stub
="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSc
n>4.0.0</modelVersion>
beam</groupId>
bookshopmvn</artifactId>
.1-SNAPSHOT</version>
ar</packaging>
opmvn</name>
Hierarchy Effective POM pom.xml
Servers
omcat] C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
okshopdb Property maxActive is not used in DBCP2, use maxTotal instead. maxTo
er.servlet.TldScanner scanJars
LDs yet contained no TLDs. Enable debug logging for thi
lina.core.StandardContext reload
ntext with name [/bookshopjndi] is completed
ENG 12:51


day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Navigate Search Project Run Window Help
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.5]
src
main
webapp
WEB-INF
web.xml
test
target
pom.xml
Servers
bookshopmvn/pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XM
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>sunbeam</groupId>
4 <artifactId>bookshopmvn</artifactId>
5 <version>0.0.1-SNAPSHOT</version>
6 <packaging>war</packaging>
7 <name>bookshopmvn</name>
8 </project>
default for Maven
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Console Progress Problems Servers
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
Jul 20, 2021 12:15:46 PM org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactory getObjectInstan
WARNING: Name = bookshopdb Property maxActive is not used in DBCP2, use maxTotal instead. ma
Jul 20, 2021 12:15:46 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
ENG 12:53


File
Window
Help
bookshopmvn/pom.xml
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://w
<modelVersion>4.0.0</modelVersion>
<groupId>sunbeam</groupId>
<artifactId>bookshopmvn</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>war</packaging>
<name>bookshopmvn</name>
<properties>
<maven.compiler.source>11</maven.compiler.source>
<maven.compiler.target>11</maven.compiler.target>
</properties>
</project>
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
Compare With
Replace With
Configure
Spring
Add Dependency
Add Plugin
New Maven Module Project
Download Javadoc
Download Sources
Update Project... Alt+F5
Select Maven Profiles... Ctrl+Alt+P
Disable Workspace Resolution
Disable Maven Nature
Assign Working Sets...
Effective POM
pom.xml
Servers
cat] C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
46 PM org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactor
kshopdb Property maxActive is not used in DBCP2, use max
46 PM org.apache.jasper.servlet.TldScanner scanJars
JAR was scanned for TLDs yet contained no TLDs. Enable d
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed


day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Refactor Navigate Search Project Run Window
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [J2SE-1.5]
src
main
webapp
WEB-INF
web.xml
test
target
pom.xml
Servers
Update Maven Project
Lenovo
Update Maven Project
Select Maven projects and update options
Available Maven Codebases
bookshopmvn
Select All
Add out-of-date
Deselect All
Expand All
Collapse All
Offline
Update dependencies
Force Update of Snapshots/Releases
Update project configuration from pom.xml
Refresh workspace resources from local filesystem
Clean projects
OK
Cancel
g/POM/4.0.0" xmlns:xsi="http://w
compiler.source>
compiler.target>
01-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
t.dbcp.dbcp2.BasicDataSourceFactor
tive is not used in DBCP2, use max
r.servlet.TldScanner scanJars
Ds yet contained no TLDs. Enable d
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
bookshopmvn


day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
src/main/java
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
src
main
webapp
WEB-INF
web.xml
test
target
pom.xml
Servers
bookshopmvn/pom.xml
1 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://w
2 <modelVersion>4.0.0</modelVersion>
3 <groupId>sunbeam</groupId>
4 <artifactId>bookshopmvn</artifactId>
5 <version>0.0.1-SNAPSHOT</version>
6 <packaging>war</packaging>
7 <name>bookshopmvn</name>
8 <properties>
9 <maven.compiler.source>11</maven.compiler.source>
10 <maven.compiler.target>11</maven.compiler.target>
11 </properties>
12 </project>
13
14
15
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Console Progress Problems Servers
Tomcat v9.0 Server at localhost [Apache Tomcat] C:\Program Files\AdoptOpenJDK\jdk-11.0.9.101-hotspot\bin\javaw.exe (20-Jul-2021, 11:35:14 AM)
Jul 20, 2021 12:15:46 PM org.apache.tomcat.dbcp.dbcp2.BasicDataSourceFactor
WARNING: Name = bookshopdb Property maxActive is not used in DBCP2, use max
Jul 20, 2021 12:15:46 PM org.apache.jasper.servlet.TldScanner scanJars
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable d
Jul 20, 2021 12:15:46 PM org.apache.catalina.core.StandardContext reload
INFO: Reloading Context with name [/bookshopjndi] is completed
bookshopmvn

day07 - bookshopmvn/src/main/webapp/WEB-INF/web.xml - Spring Tool Suite 3

File Edit Source Refactor Navigate Search Project Run Design Window Help

Package Explorer

- bookshop2 [javaee master]
- bookshopjndi [javaee master]
- bookshopmvn [javaee master]
  - src/main/java
    - sunbeam.beans
    - sunbeam.controllers
    - sunbeam.dao
    - sunbeam.pojo
    - sunbeam.util
    - jdbc.properties
  - src/main/resources
  - src/test/java
  - src/test/resources
  - JRE System Library [JavaSE-11]
  - Maven Dependencies
  - src
    - main
      - webapp
        - WEB-INF
          - web.xml
        - addcart.jsp
        - auth.jsp
        - books.jsp
        - doregister.jsp
        - login.jsp
        - logout.jsp
        - register.jsp
        - showcart.jsp
        - subjects2.jsp
      - test
  - target
    - m2e-wtp
  - pom.xml
- Servers

bookshopmvn/pom.xml | web.xml

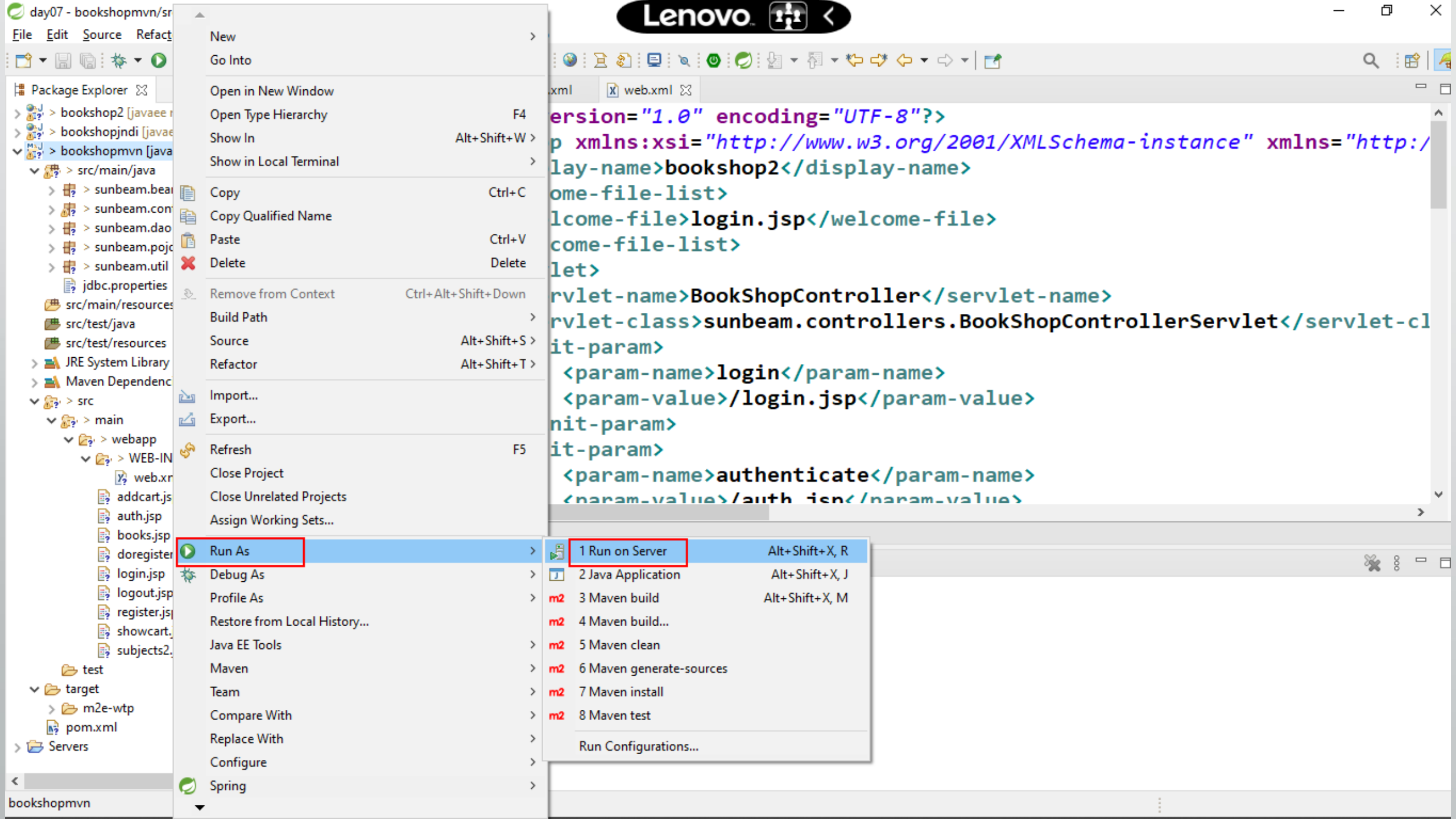
```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:/
3    <display-name>bookshop2</display-name>
4    <welcome-file-list>
5      <welcome-file>login.jsp</welcome-file>
6    </welcome-file-list>
7    <servlet>
8      <servlet-name>BookShopController</servlet-name>
9      <servlet-class>sunbeam.controllers.BookShopControllerServlet</servlet-cl
10     <init-param>
11         <param-name>login</param-name>
12         <param-value>/login.jsp</param-value>
13     </init-param>
14     <init-param>
15         <param-name>authenticate</param-name>
16         <param-value>/auth.jsp</param-value>
```

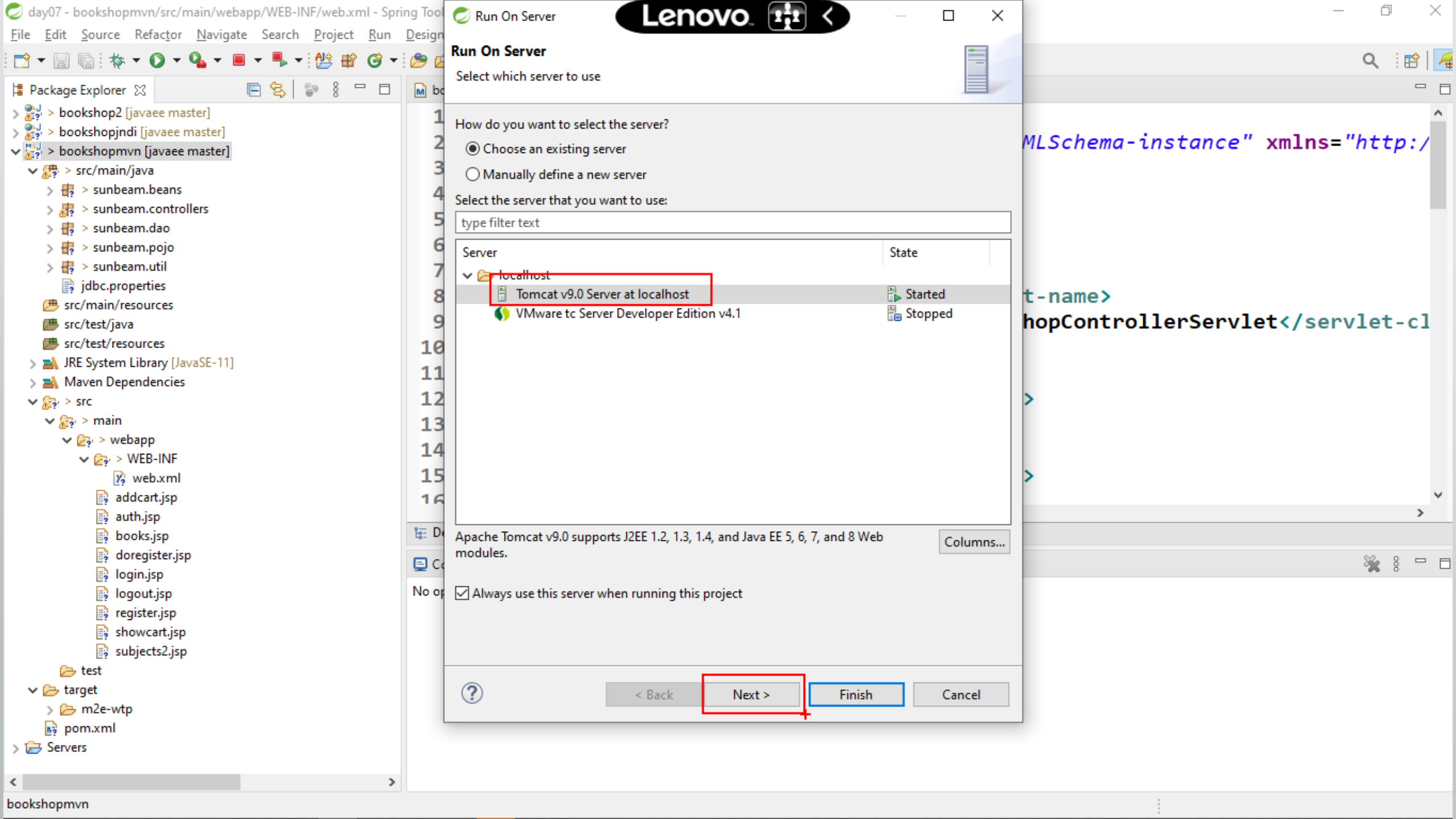
Design | Source

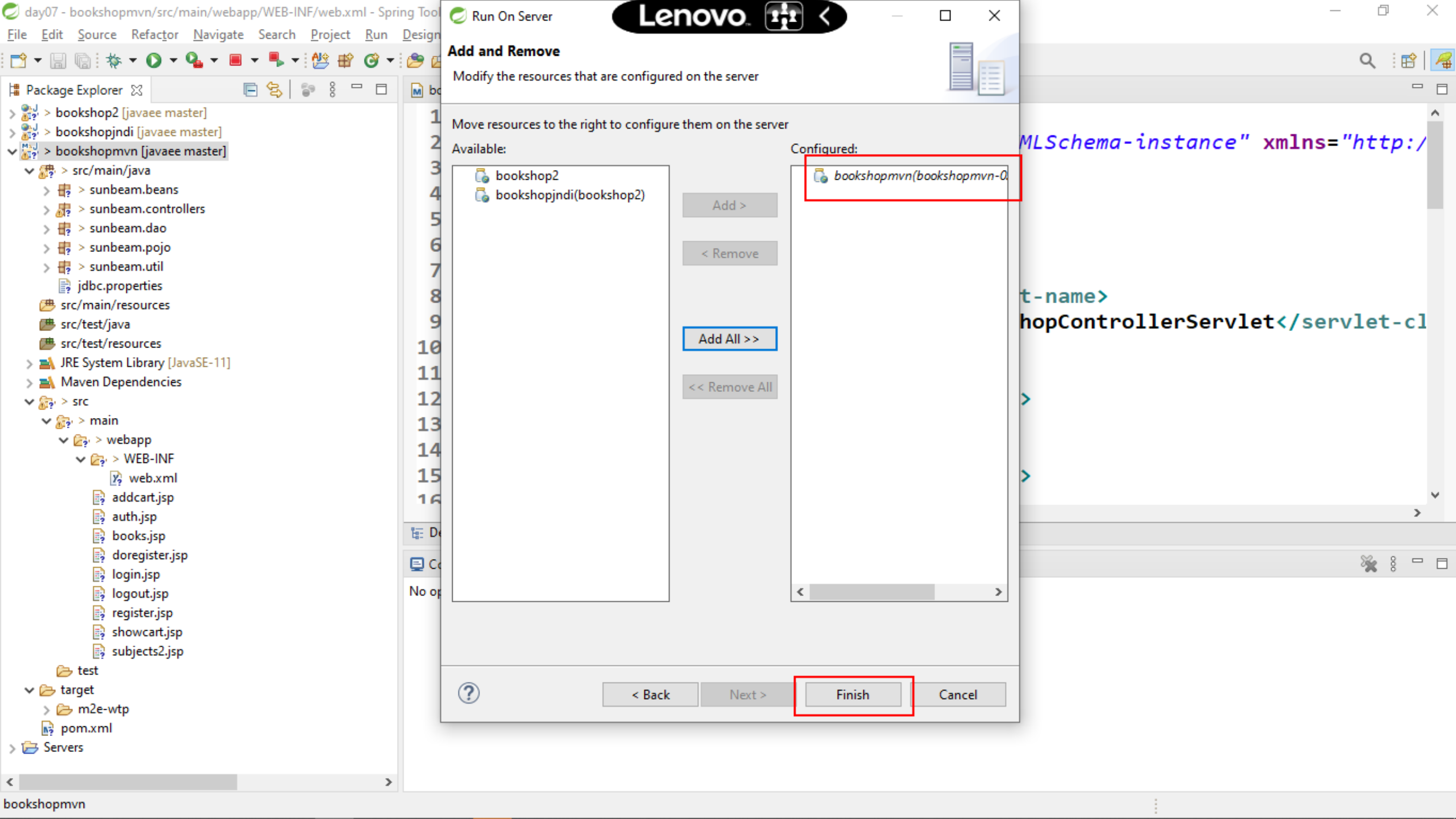
Console | Progress | Problems | Servers

No operations to display at this time.

Run Maven web application from Eclipse


day07 - bookshopmvn/sr
File Edit Source Refact
Package Explorer
bookshop2 [javaee
bookshopjndi [javae
bookshopmvn [java
src/main/java
sunbeam.bea
sunbeam.con
sunbeam.dao
sunbeam.pojo
sunbeam.util
jdbc.properties
src/main/resources
src/test/java
src/test/resources
JRE System Library
Maven Dependenc
src
main
webapp
WEB-IN
web.xr
addcart.js
auth.jsp
books.jsp
doregister
login.jsp
logout.jsp
register.js
showcart.
subjects2.
test
target
m2e-wtp
pom.xml
Servers
bookshopmvn
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
Compare With
Replace With
Configure
Spring
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean
6 Maven generate-sources
7 Maven install
8 Maven test
Run Configurations...
.xml
web.xml
ersion="1.0" encoding="UTF-8"?>
p xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:/
lay-name>bookshop2</display-name>
ome-file-list>
lcome-file>login.jsp</welcome-file>
come-file-list>
let>
rvlet-name>BookShopController</servlet-name>
rvlet-class>sunbeam.controllers.BookShopControllerServlet</servlet-cl
it-param>
<param-name>login</param-name>
<param-value>/login.jsp</param-value>
nit-param>
it-param>
<param-name>authenticate</param-name>

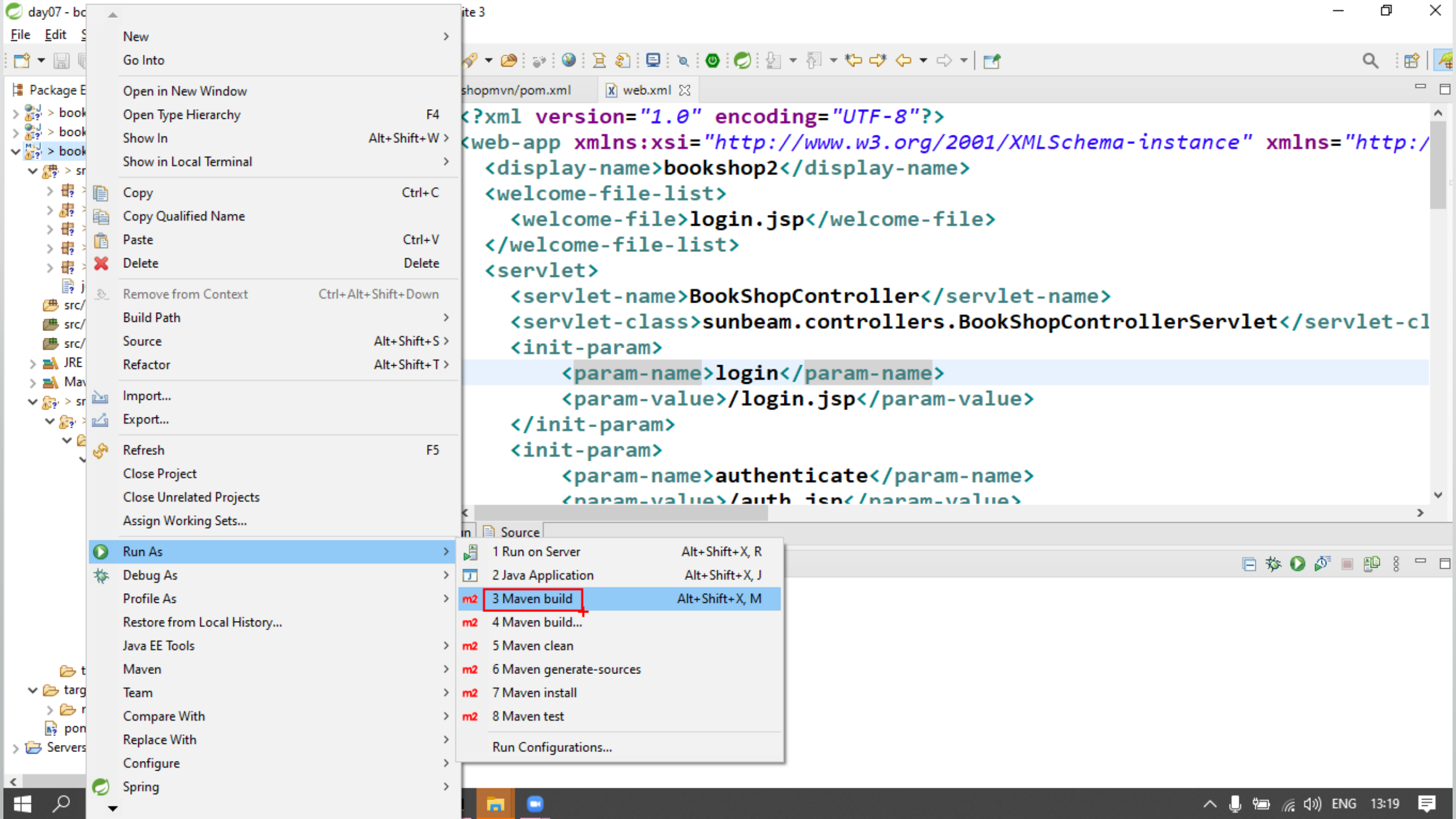

day07 - bookshopmvn/src/main/webapp/WEB-INF/web.xml - Spring Tool
File Edit Source Refactor Navigate Search Project Run Design
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
src/main/java
sunbeam.beans
sunbeam.controllers
sunbeam.dao
sunbeam.pojo
sunbeam.util
jdbc.properties
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
Maven Dependencies
src
main
webapp
WEB-INF
web.xml
addcart.jsp
auth.jsp
books.jsp
doregister.jsp
login.jsp
logout.jsp
register.jsp
showcart.jsp
subjects2.jsp
test
target
m2e-wtp
pom.xml
Servers
bookshopmvn
Run On Server
Lenovo
Run On Server
Select which server to use
How do you want to select the server?
Choose an existing server
Manually define a new server
Select the server that you want to use:
type filter text
Server
State
localhost
Tomcat v9.0 Server at localhost
Started
VMware tc Server Developer Edition v4.1
Stopped
Apache Tomcat v9.0 supports J2EE 1.2, 1.3, 1.4, and Java EE 5, 6, 7, and 8 Web modules.
Columns...
Always use this server when running this project
< Back
Next >
Finish
Cancel
MLSchema-instance" xmlns="http:/
t-name>
hopControllerServlet</servlet-cl
No op

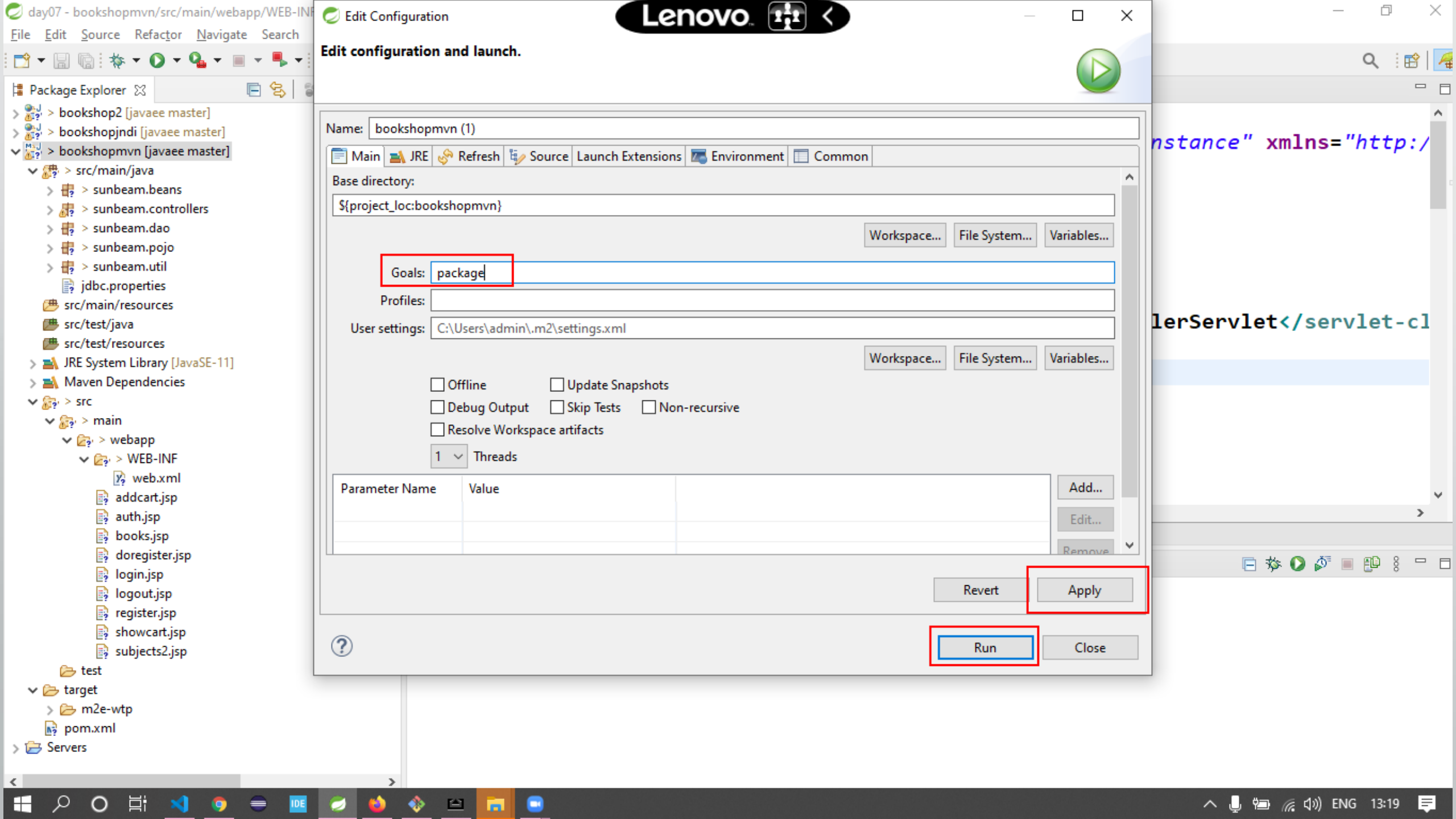

day07 - bookshopmvn/src/main/webapp/WEB-INF/web.xml - Spring Tool
File Edit Source Refactor Navigate Search Project Run Design
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
src/main/java
sunbeam.beans
sunbeam.controllers
sunbeam.dao
sunbeam.pojo
sunbeam.util
jdbc.properties
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
Maven Dependencies
src
main
webapp
WEB-INF
web.xml
addcart.jsp
auth.jsp
books.jsp
doregister.jsp
login.jsp
logout.jsp
register.jsp
showcart.jsp
subjects2.jsp
test
target
m2e-wtp
pom.xml
Servers
bookshopmvn
Run On Server
Lenovo
Add and Remove
Modify the resources that are configured on the server
Move resources to the right to configure them on the server
Available:
bookshop2
bookshopjndi(bookshop2)
Add >
< Remove
Add All >>
<< Remove All
Configured:
bookshopmvn(bookshopmvn-0
MLSchema-instance" xmlns="http:/
t-name>
hopControllerServlet</servlet-cl
< Back
Next >
Finish
Cancel

Run application
(Same as Dynamic Web Project execution)

- Open the browser
- Enter application login page URL
- Login and test all features.

Deploy Maven web application in Tomcat


day07 - bo
File Edit
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Remove from Context Ctrl+Alt+Shift+Down
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Close Unrelated Projects
Assign Working Sets...
Run As
Debug As
Profile As
Restore from Local History...
Java EE Tools
Maven
Team
Compare With
Replace With
Configure
Spring
1 Run on Server Alt+Shift+X, R
2 Java Application Alt+Shift+X, J
3 Maven build Alt+Shift+X, M
4 Maven build...
5 Maven clean
6 Maven generate-sources
7 Maven install
8 Maven test
Run Configurations...
shopmvn/pom.xml
web.xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http:/
<display-name>bookshop2</display-name>
<welcome-file-list>
<welcome-file>login.jsp</welcome-file>
</welcome-file-list>
<servlet>
<servlet-name>BookShopController</servlet-name>
<servlet-class>sunbeam.controllers.BookShopControllerServlet</servlet-cl
<init-param>
<param-name>login</param-name>
<param-value>/login.jsp</param-value>
</init-param>
<init-param>
<param-name>authenticate</param-name>
Source
ENG 13:19


day07 - bookshopmvn/src/main/webapp/WEB-INF
File Edit Source Refactor Navigate Search
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
src/main/java
sunbeam.beans
sunbeam.controllers
sunbeam.dao
sunbeam.pojo
sunbeam.util
jdbc.properties
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
Maven Dependencies
src
main
webapp
WEB-INF
web.xml
addcart.jsp
auth.jsp
books.jsp
doregister.jsp
login.jsp
logout.jsp
register.jsp
showcart.jsp
subjects2.jsp
test
target
m2e-wtp
pom.xml
Servers
Edit Configuration
Edit configuration and launch.
Name: bookshopmvn (1)
Main JRE Refresh Source Launch Extensions Environment Common
Base directory:
${project_loc:bookshopmvn}
Workspace... File System... Variables...
Goals: package
Profiles:
User settings: C:\Users\admin\.m2\settings.xml
Workspace... File System... Variables...
Offline
Update Snapshots
Debug Output
Skip Tests
Non-recursive
Resolve Workspace artifacts
1 Threads
Parameter Name
Value
Add...
Edit...
Remove
Revert
Apply
Run
Close
nstance" xmlns="http:/
lerServlet</servlet-cl
ENG
13:19

Run application in Tomcat

- Stop tomcat from Eclipse (if already running).
- Run tomcat. <tomcat>/bin/startup.bat.
- Check if tomcat running well. Browser http://localhost:8080
- Copy application war file (created while "package") into <tomcat>/webapps
- Check if appln running well. Browser http://localhost:8080/appln_dir_name

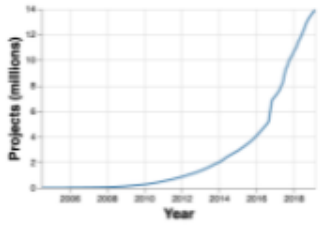
Understanding Maven Central & Local Repository

Thanks | Subjects | Thanks | Repository: mysql » mysq

mvnrepository.com/artifact/mysql/mysql-connector-java/8.0.24

Maven Central

Indexed Artifacts (21.4M)



Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers
- HTTP Clients
- I/O Utilities
- JDBC Extensions
- JDBC Pools

Home » mysql » mysql-connector-java » 8.0.24

MySQL Connector/J » 8.0.24

JDBC Type 4 driver for MySQL

License	GPL 2.0
Categories	MySQL Drivers
Organization	Oracle Corporation
HomePage	http://dev.mysql.com/doc/connector-j/en/
Date	(Apr 19, 2021)
Files	jar (2.3 MB) View All
Repositories	Central
Used By	5,417 artifacts

Note: There is a new version for this artifact

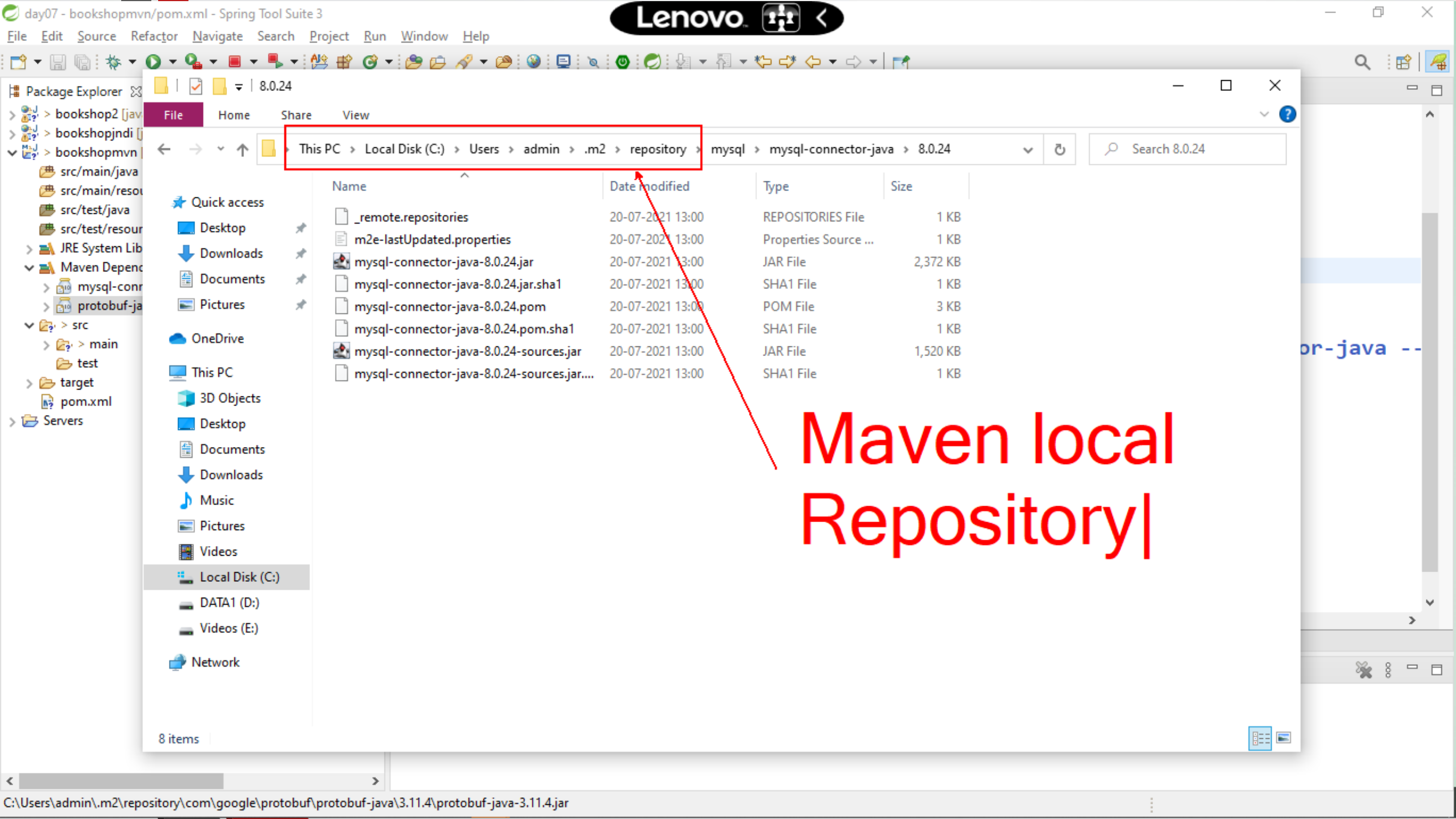
New Version	8.0.25

Maven | Gradle | Gradle (Short) | Gradle (Kotlin) | SBT | Ivy | Grape | Leiningen | Buildr

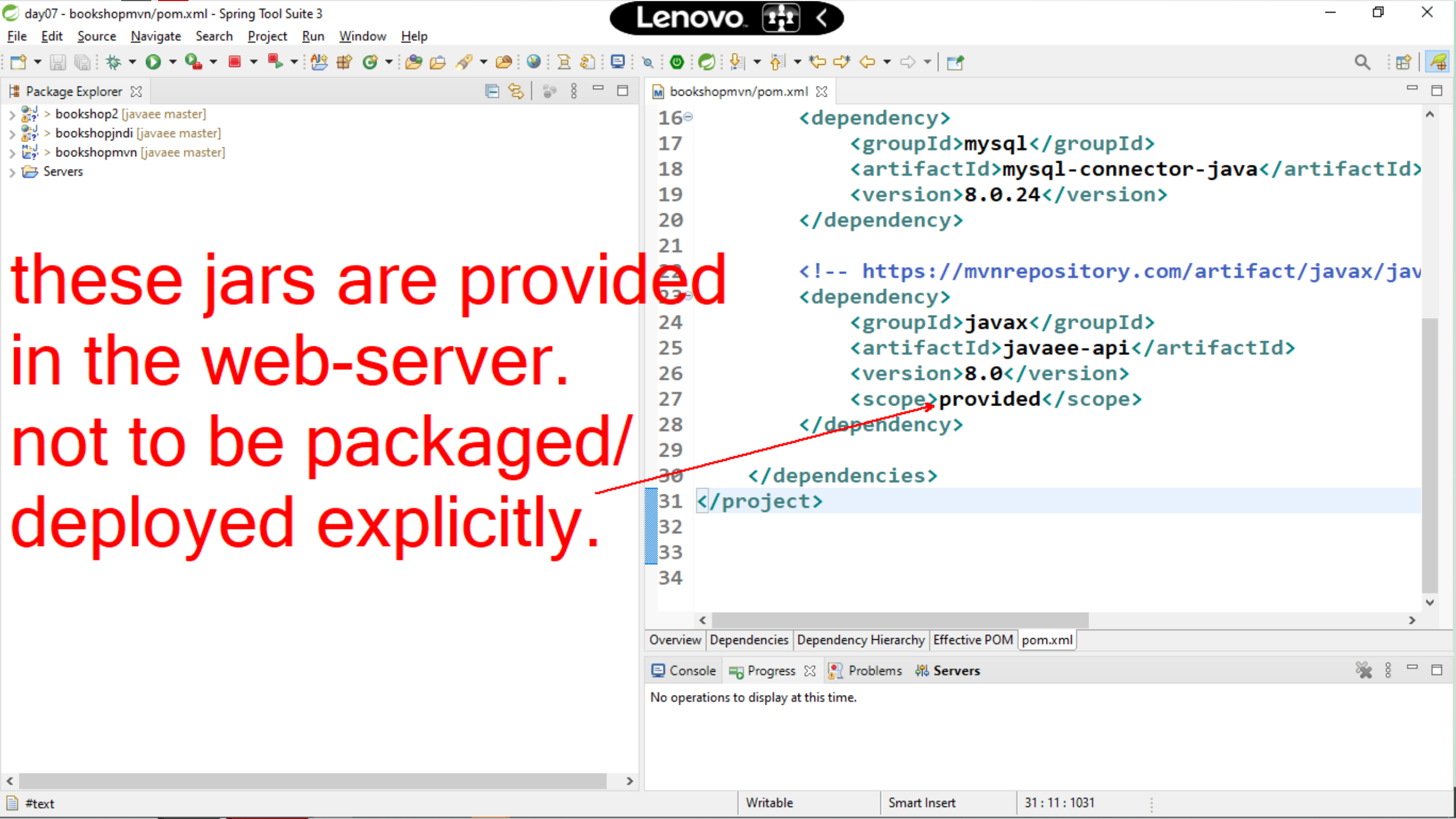
```
<!-- https://mvnrepository.com/artifact/mysql/mysql-connector-java -->
<dependency>
    <groupId>mysql</groupId>
    <artifactId>mysql-connector-java</artifactId>
    <version>8.0.24</version>
</dependency>
```

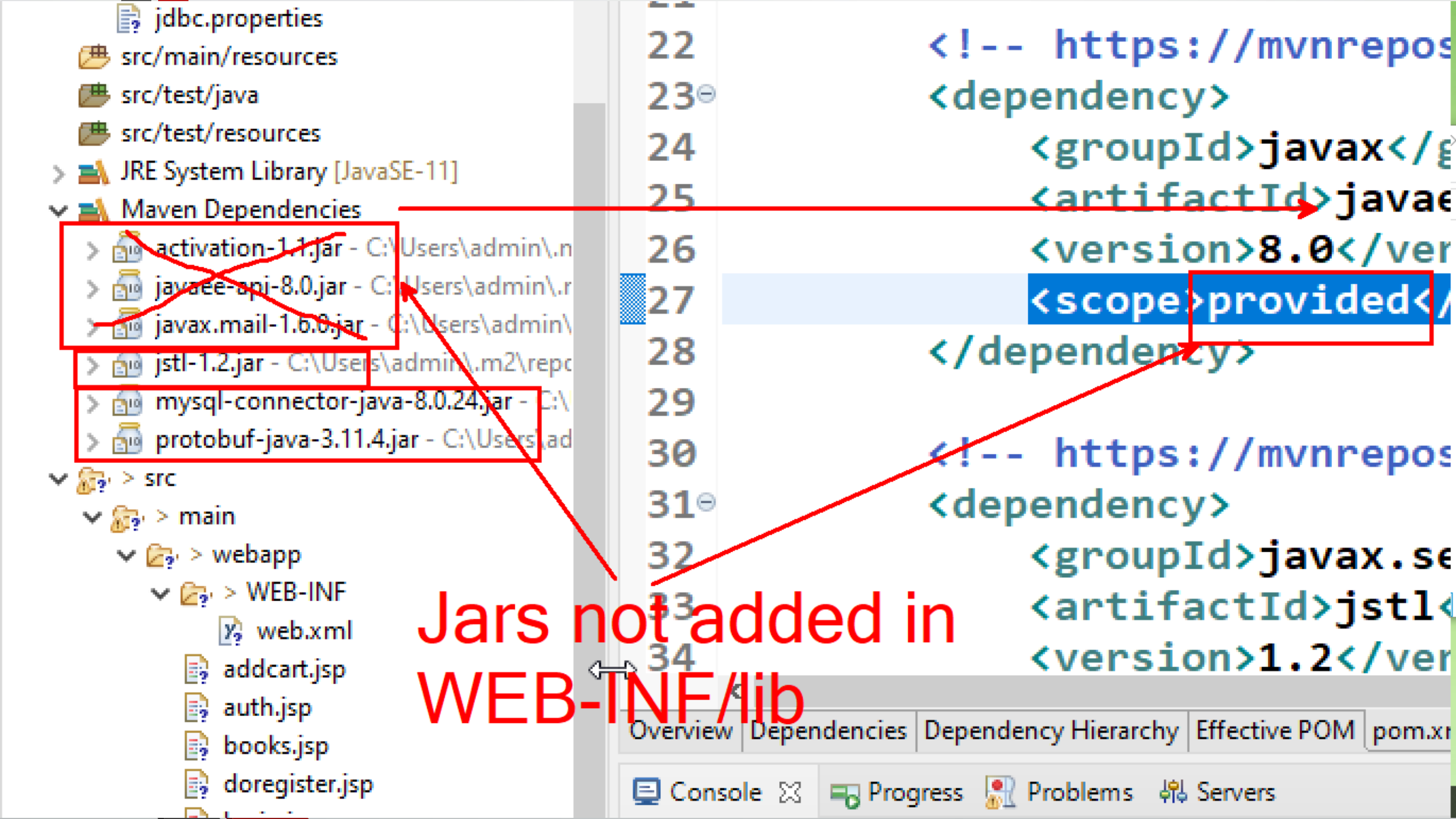
Include comment with link to declaration

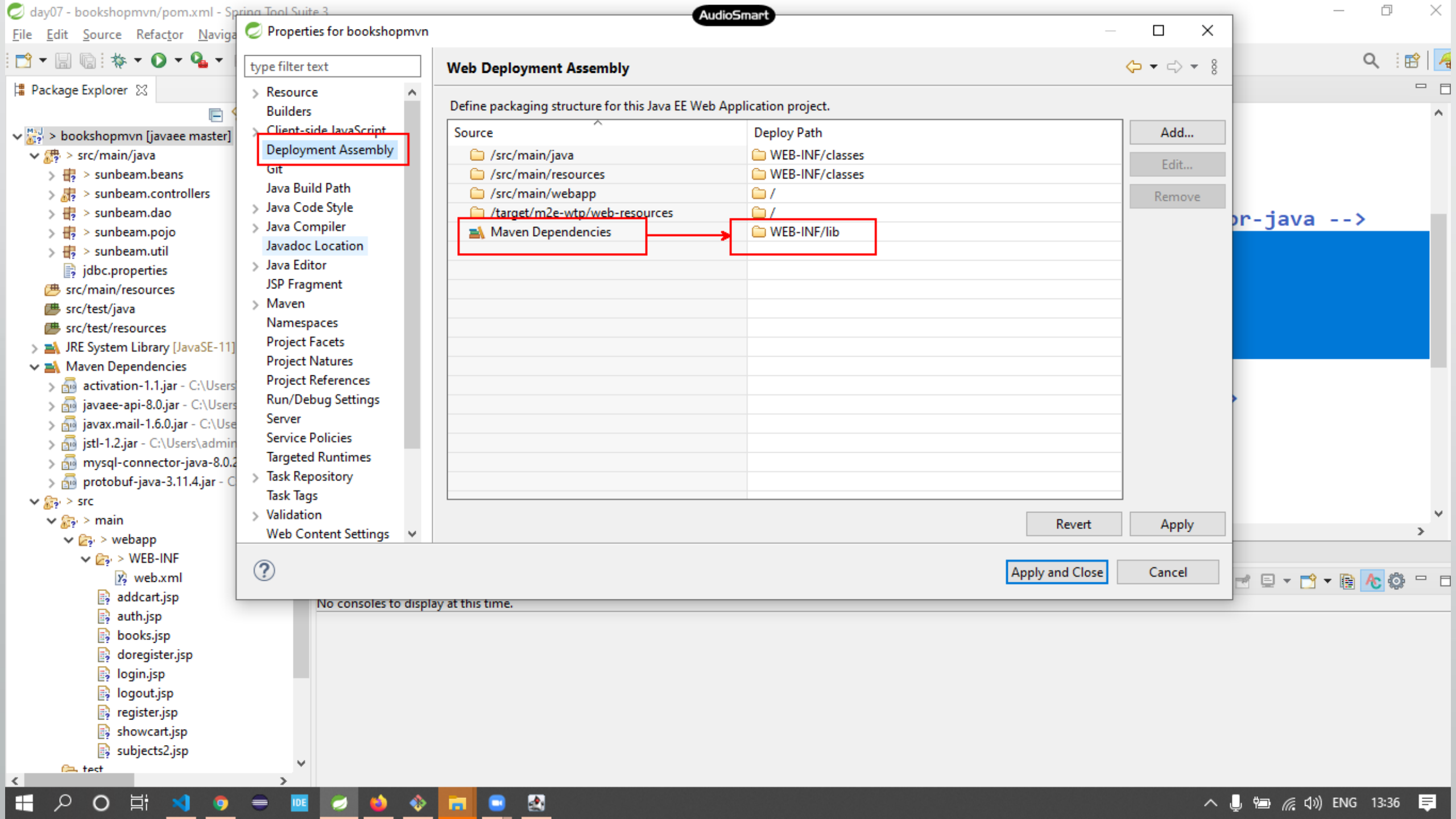
Copied to clipboard!


day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
bookshop2 [jav
bookshopjndi [j
bookshopmvn
src/main/java
src/main/resou
src/test/java
src/test/resour
JRE System Lib
Maven Depend
mysql-conr
protobuf-ja
src
main
test
target
pom.xml
Servers
8.0.24
File Home Share View
This PC › Local Disk (C:) › Users › admin › .m2 › repository › mysql › mysql-connector-java › 8.0.24
Search 8.0.24
Quick access
Desktop
Downloads
Documents
Pictures
OneDrive
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
DATA1 (D:)
Videos (E:)
Network
Name
Date modified
Type
Size
_remote.repositories 20-07-2021 13:00 REPOSITORIES File 1 KB
m2e-lastUpdated.properties 20-07-2021 13:00 Properties Source ... 1 KB
mysql-connector-java-8.0.24.jar 20-07-2021 13:00 JAR File 2,372 KB
mysql-connector-java-8.0.24.jar.sha1 20-07-2021 13:00 SHA1 File 1 KB
mysql-connector-java-8.0.24.pom 20-07-2021 13:00 POM File 3 KB
mysql-connector-java-8.0.24.pom.sha1 20-07-2021 13:00 SHA1 File 1 KB
mysql-connector-java-8.0.24-sources.jar 20-07-2021 13:00 JAR File 1,520 KB
mysql-connector-java-8.0.24-sources.jar.... 20-07-2021 13:00 SHA1 File 1 KB
8 items
Maven local
Repository
C:\Users\admin\.m2\repository\com\google\protobuf\protobuf-java\3.11.4\protobuf-java-3.11.4.jar

Understanding Maven dependency scope = provided


day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Navigate Search Project Run Window Help
Package Explorer
bookshop2 [javaee master]
bookshopjndi [javaee master]
bookshopmvn [javaee master]
Servers
these jars are provided
in the web-server.
not to be packaged/
deployed explicitly.
bookshopmvn/pom.xml
16 <dependency>
17 <groupId>mysql</groupId>
18 <artifactId>mysql-connector-java</artifactId>
19 <version>8.0.24</version>
20 </dependency>
21
22 <!-- https://mvnrepository.com/artifact/javax/jav
23 <dependency>
24 <groupId>javax</groupId>
25 <artifactId>javaee-api</artifactId>
26 <version>8.0</version>
27 <scope>provided</scope>
28 </dependency>
29
30 </dependencies>
31 </project>
32
33
34
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Console Progress Problems Servers
No operations to display at this time.
#text
Writable
Smart Insert
31 : 11 : 1031


jdbc.properties
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
Maven Dependencies
activation-1.1.jar - C:\Users\admin\.n
javaee-api-8.0.jar - C:\Users\admin\.r
javax.mail-1.6.0.jar - C:\Users\admin\
jstl-1.2.jar - C:\Users\admin\.m2\repo
mysql-connector-java-8.0.24.jar - C:\
protobuf-java-3.11.4.jar - C:\Users\ad
src
main
webapp
WEB-INF
web.xml
addcart.jsp
auth.jsp
books.jsp
doregister.jsp
22 <!-- https://mvnrepos
23 <dependency>
24 <groupId>javax</g
25 <artifactId>javae
26 <version>8.0</ver
27 <scope>provided</
28 </dependency>
29
30 <!-- https://mvnrepos
31 <dependency>
32 <groupId>javax.se
33 <artifactId>jstl<
34 <version>1.2</ver
Jars not added in
WEB-INF/lib
Overview
Dependencies
Dependency Hierarchy
Effective POM
pom.x
Console
Progress
Problems
Servers


AudioSmart
day07 - bookshopmvn/pom.xml - Spring Tool Suite 3
File Edit Source Refactor Naviga
Package Explorer
> bookshopmvn [javaee master]
> src/main/java
> sunbeam.beans
> sunbeam.controllers
> sunbeam.dao
> sunbeam.pojo
> sunbeam.util
jdbc.properties
src/main/resources
src/test/java
src/test/resources
JRE System Library [JavaSE-11]
Maven Dependencies
activation-1.1.jar - C:\Users
javaee-api-8.0.jar - C:\Users
javax.mail-1.6.0.jar - C:\Use
jstl-1.2.jar - C:\Users\admin
mysql-connector-java-8.0.2
protobuf-java-3.11.4.jar - C
> src
> main
> webapp
> WEB-INF
web.xml
addcart.jsp
auth.jsp
books.jsp
doregister.jsp
login.jsp
logout.jsp
register.jsp
showcart.jsp
subjects2.jsp
test
Properties for bookshopmvn
type filter text
Resource
Builders
Client-side JavaScript
Deployment Assembly
Git
Java Build Path
Java Code Style
Java Compiler
Javadoc Location
Java Editor
JSP Fragment
Maven
Namespaces
Project Facets
Project Natures
Project References
Run/Debug Settings
Server
Service Policies
Targeted Runtimes
Task Repository
Task Tags
Validation
Web Content Settings
Web Deployment Assembly
Define packaging structure for this Java EE Web Application project.
Source
Deploy Path
/src/main/java
WEB-INF/classes
/src/main/resources
WEB-INF/classes
/src/main/webapp
/
/target/m2e-wtp/web-resources
/
Maven Dependencies
WEB-INF/lib
Add...
Edit...
Remove
Revert
Apply
Apply and Close
Cancel
No consoles to display at this time.
r-java -->
ENG
13:36

Thank you

Nilesh Ghule <nilesh@sunbeaminfo.com>

https://www.youtube.com/@codewitharrays

https://www.instagram.com/codewitharrays/

https://t.me/codewitharrays Group Link: https://t.me/cceesept2023

+91 8007592194 +91 9284926333

codewitharrays@gmail.com

https://codewitharrays.in/project