Advanced Java

Application architectures

Model-View architecture

- Also called as Model-1 architecture.
- Application is divided in two major parts. 1 * Model: Data handling and Business logic -- Java bean.
    - View: Presentation/appearence of the data -- JSP.
- Java beans are tightly coupled with JSP pages (jsp:useBean). Also a JSP page is tightly coupled with other JSP pages (e.g. href="...", action="..."). Any changes into bean or jsp will lead to changes in multiple other components.
- This architecture is suitable for small applications.

Model-View-Controller architecture

- Also called as Model-2 architecture.
- Application is divided in three major parts.
    - Model: Data handling and Business logic -- Java bean.
    - View: Presentation/appearence of the data -- JSP.
    - Controller: Handles communication/navigation between views and models.
- Models and views are loosely coupled with each other. Their navigation is centrally controlled by controller layer.
    - View1 --> Controller --> View2
    - View1 --> Controller --> Java Bean (Business Logic) --> Controller --> View2
- This architecture is suitable for bigger applications.
- Typically controller is implemented as a servlet that forwards the request to the next component.
- There are popular frameworks which implements MVC pattern (e.g. JSF, Spring MVC, Struts MVC). Spring MVC has predefined controller named as "DispatcherServlet".

Maven

- Maven is a build tool.
- Configuration file: pom.xml
- https://jenkov.com/tutorials/maven/maven-tutorial.html
- Video: https://youtu.be/lMXBrlVFYA0?si=vCP-_2egY1mQnKzV

JSP Custom Tags

- Not in Syllabus
- JSP has two types of tags:
    - Classic tags (javax.servlet.jsp.tagext.Tag)
    - Simple tags (javax.servlet.jsp.tagext.SimpleTag)
- Both are inherited from marker interface "javax.servlet.jsp.tagext.JspTag".
- To implement custom tags, "TagSupport" and "SimpleTagSupport" classes are preferred. These adapter classes, provides default implementation of the methods in respective interfaces.

Steps to implement custom (Simple) tag

- step 0: Secide the application, name, attributes and body of the tag.
  - Example: `<my:greet username="some_name"/>`
  - This tag should print greeting meesage for the given name.
- step 1: Write the tag handler class inherited from "SimpleTagSupport" e.g. GreetTagImpl.
  - Param-less ctor.
  - Fields corresponding to tag attributes.
  - Getter/setter for fields.
  - Override doTag() method of the "SimpleTagSupport" class. Write the business logic + presentation logic in it.
- step 2: Write tag library descriptor (tld) xml file inside WEB-INF. Important fields are:
  - taglib uri
  - tag (one for each tag)
    - name
    - class
    - body (none or jsp or scriptless)
    - attribute (one for each attribute)
      - name
      - type
      - required
      - rtexprvalue
- step3: use the tag into the JSP page
  - `<%@taglib prefix="my" uri="/WEB-INF/my-tags" %>`
  - `<my:greet username="${lb.username}"/>`

SimpleTag life cycle

1. When page containing custom tag is accessed first time, during translation stage:
   - the tld file is referred (via given prefix in @taglib)
   - from tld used tag is found and syntax is validated.
2. During request handling stage, for each occurrance of tag, the tag-class (in .tld file) is loaded and object is created. Paramless constructor will be called.
3. setJspContext() method will be called and current JSP's PageContext object will be passed to it. This object contains all info needed to process JSP page.
4. If tag is child of any other tag, then setParent() method will be called.
5. Then setter methods is called for all attributes used in the JSP file.
6. If tag has some body, then setJspBody() method will be called to set the tag body.
7. Then doTag() method is executed, which does intended processing and generate output if any.
8. After doTag() is completed, the tag's generated html will be added into page response.

Docs

- https://docs.oracle.com/javaee/7/api/javax/servlet/jsp/tagext/SimpleTag.html
- https://docs.oracle.com/javaee/7/api/javax/servlet/jsp/tagext/Tag.html

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();
- https://docs.oracle.com/javaee/7/api/javax/servlet/Filter.html
- Can be configured with @WebFilter or in web.xml (similar to servlets).
- Filter Life Cycle:
    - init() -- When application is deployed, filter object is created and init() is called. Programmer can do one-time initialization here.
    - destroy() -- When application is undeployed or server shutdown, filter object is destroyed. Programmer can do one-time de-initialization here.
    - doFilter() -- Executed for each request. Programmer implements pre-processing and post-processing code here.
- Refer slides.

Listeners

- Not in Syllabus
- Listeners are used to handle application level events.
- There are many listener interfaces. Refer docs.
    - ServletContextListener -- To handle application initialized and destroy events.
        - void contextInitialized(ServletContextEvent sce);
            - Called by web container when application is started/deployed i.e. servlet context is created.
            - Example: load and register JDBC driver, initialize a connection pool, ...
        - void contextDestroyed(ServletContextEvent sce);
            - Called by web container when application is stopped i.e. when web server shutdown.
            - Example: release a connection pool, ...
        - Implement this listener to perform one time initialization and destruction for the whole application.
    - HttpSessionListener -- To handle session initialized and destroy events.
        - sessionCreated() method is called when req.getSession() is called first time for any client. You may add any session attribute in it immediately after creating session.
        - sessionDestroyed() method is called when session is invalidated or time-out.
    - ServletRequestListener -- -- To handle request initialized and destroy events.
    - 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>
```