Web Applications Developments

ITWT413

LECTURE 5: JAKARTA EE CORE PROFILE

DR. HALA SHAARI

Course Structure

- Lecture (Monday& Tuesday)
- Time: 12:30-14:00
- Microsoft Teams Code (goi2d67)

(Lectures, labs, announcements, References):

- Grading :
  - 25% Midterm exam.
  - 25% Assignments
  - 25% Group Project (Groups of two).
  - 25% Final Exam.

Course References

Our Course Main References:

- Jakarta EE Tutorial
- Java EE to Jakarta EE 10 Recipes: A Problem-Solution Approach for Enterprise Java (2022)

key course objectives

1) Master Core Jakarta EE Concepts: Understand the fundamental architecture and components of Jakarta EE, including Servlets, JSP, JSF, and Enterprise JavaBeans (EJB).
2) Develop RESTful Web Services: Build and deploy scalable RESTful APIs using Jakarta RESTful Web Services (JAX-RS), with advanced features like exception handling, filters, and security.
3) Implement Dependency Injection and Persistence: Leverage Contexts and Dependency Injection (CDI) and Jakarta Persistence (JPA) to manage beans and database interactions in enterprise applications.
4) Ensure Application Security and Transaction Management: Apply Jakarta EE's security framework for authentication, authorization, and manage transactions using declarative and programmatic approaches.
5) Deploy Cloud-Native Applications: Utilize Jakarta EE to build, containerize, and deploy microservice-based applications in cloud environments, incorporating tools like Docker and Kubernetes.

Lecture Agenda

- Running the Basic Contexts and Dependency Injection Examples
  - Building and Running the CDI Samples
  - The simplegreeting CDI Example
  - The simplegreeting Source Files
  - The Facelets Template and Page
  - Running the simplegreeting Example
  - The guessnumber-cdi CDI Example
  - The guessnumber-cdi Source Files
  - The Facelets Page
  - Running the guessnumber-cdi Example

Required Software

The following software is required to run the examples

- Java Platform, Standard Edition
- Eclipse Glassfish Server
- Jakarta EE Tutorial Examples
- Apache NetBeans IDE
- Apache Maven
- Instructions from page 11 to page17 from Jakarta EE Tutorial tells you everything you need to know to install, build, and run the tutorial examples

Jakarta EE Tutorial- Structure

Basic Platform	Jakarta EE Core Profile	Jakarta EE Web Profile	Jakarta EE Platform
Resource Creation	Jakarta CDI Lite	Jakarta CDI Full	Jakarta Mail
Injection	Jakarta REST	Jakarta Validation	Jakarta Messaging
Packaging	Jakarta JSON	Jakarta Security	Jakarta Batch
		Jakarta Servlets	
		Jakarta Faces	
		Jakarta WebSocket	
		Jakarta Persistence	
		Jakarta Enterprise Beans Lite	

Building and Running the CDI Samples

- The examples are in the jakartaee-examples/tutorial/cdi/ directory.
- To build and run the examples, you will do the following:

  - Use NetBeans IDE or the Maven tool to compile and package the example.
  - Use NetBeans IDE or the Maven tool to deploy the example.
  - Run the example in a web browser.

The simplegreeting CDI Example

- The simplegreeting example illustrates some of the most basic features of CDI: scopes, qualifiers, bean injection, and accessing a managed bean in a Jakarta Faces application.
- When you run the example, you click a button that presents either a formal or an informal greeting, depending on how you edited one of the classes.
- The four source files for the simplegreeting example are:
  - The default Greeting class, shown in Beans as Injectable Objects
  - The @Informal qualifier interface definition and the InformalGreeting class that implements the interface, both shown in Using Qualifiers
  - The Printer managed bean class, which injects one of the two interfaces, shown in full in Adding Setter and Getter Methods

The Facelets Template and Page

- To use the managed bean in a simple Facelets application:
  - Use a very simple template file and index.xhtml page.
  - The template page, /WEB-INF/templates/template.xhtml, looks like slide 11.
- To create the Facelets page, redefine the title and head, then add a small form to the content, as shown in slide 12.

```
<!DOCTYPE html>
<html lang="en"
      xmlns:h="jakarta.faces.html"
      xmlns:ui="jakarta.faces.facelets">
    <h:head>
        <title><ui:insert name="title">Default Title</ui:insert></title>
        <h:outputStylesheet name="css/default.css" />
    </h:head>
    <h:body>
        <main>
            <header>
                <h1>
                    <ui:insert name="head">
                        <ui:insert name="title">Default Header</ui:insert>
                    </ui:insert>
                </h1>
            </header>

            <article>
                <ui:insert name="content" />
            </article>
        </main>
    </h:body>
</html>
```

```
<ui:composition template="/WEB-INF/templates/template.xhtml"
                xmlns:ui="jakarta.faces.facelets"
                xmlns:h="jakarta.faces.html">
    <ui:define name="title">Simple Greeting</ui:define>
    <ui:define name="content">
        <h:form id="simpleGreetingForm">
            <div class="input">
                <h:outputLabel for="name" value="Enter your name" />
                <h:inputText id="name" value="#{printer.name}"/>
            </div>
            <div class="actions">
                <h:commandButton id="createSalutation"
                                 value="Say Hello"
                                 action="#{printer.createSalutation}">
                    <f:ajax execute="@form" render="salutation" />
                </h:commandButton>
            </div>
            <div class="output">
                <p>
                    <h:outputText id="salutation"
                                  value="#{printer.salutation}" />
                </p>
            </div>
        </h:form>
    </ui:define>
</ui:composition>
```

Running the simplegreeting Example (1)

To Build, Package, and Run the simplegreeting Example Using NetBeans IDE

- Make sure that GlassFish Server has been started (see Starting and Stopping GlassFish Server).
- From the File menu, choose Open Project.
- In the Open Project dialog box, navigate to: jakartaee-examples/tutorial/cdi
- Select the simplegreeting folder.
- Click Open Project.
- To modify the Printer.java file, perform these steps:
  - Expand the Source Packages node.
  - Expand the greetings node.
  - Double-click the Printer.java file.
  - In the editor, comment out the @Informal annotation:

```
@Inject
//@Informal
Greeting greeting;
```

  - Save the file.
- In the Projects tab, right-click the simplegreeting project and select Build.

Running the simplegreeting Example (2)

- To Build, Package, and Deploy the simplegreeting Example Using Maven
- Make sure that GlassFish Server has been started (see Starting and Stopping GlassFish Server).
- In a terminal window, go to:

  jakartaee-examples/tutorial/cdi/simplegreeting/

- Enter the following command to deploy the application:

  mvn install

- This command builds and packages the application into a WAR file, simplegreeting.war, located in the target directory, and then deploys it to GlassFish Server.

Running the simplegreeting Example (3)

- To Run the simplegreeting Example
- In a web browser, enter the following URL:
  http://localhost:8080/simplegreeting
- The Simple Greeting page opens.
- Enter a name in the field.
  For example, suppose that you enter Duke.
- Click Say Hello.
- If you did not modify the Printer.java file, then the following text string appears below the button:
  Hi, Duke!
- If you commented out the @Informal annotation in the Printer.java file, then the following text string appears below the button:
  Hello, Duke.

Using Jakarta EE Interceptors

- Interceptors are used in conjunction with Jakarta EE managed classes to allow developers to invoke interceptor methods on an associated target class, in conjunction with method invocations or lifecycle events. Common uses of interceptors are logging, auditing, and profiling.
- You can use interceptors with session beans, message-driven beans, and CDI managed beans. In all of these cases, the interceptor target class is the bean class.
- An interceptor can be defined within a target class as an interceptor method, or in an associated class called an interceptor class. Interceptor classes contain methods that are invoked in conjunction with the methods or lifecycle events of the target class.
- Interceptor classes and methods are defined using metadata annotations, or in the deployment descriptor of the application that contains the interceptors and target classes.

Interceptor Metadata Annotations

- Interceptor methods within the target class or in an interceptor class are annotated with one of the metadata annotations defined in Interceptor Metadata Annotations.

Interceptor Metadata Annotation	Description
`jakarta.interceptor.AroundConstruct`	Designates the method as an interceptor method that receives a callback after the target class is constructed
`jakarta.interceptor.AroundInvoke`	Designates the method as an interceptor method
`jakarta.interceptor.AroundTimeout`	Designates the method as a timeout interceptor for interposing on timeout methods for enterprise bean timers
`jakarta.annotation.PostConstruct`	Designates the method as an interceptor method for post-construct lifecycle events
`jakarta.annotation.PreDestroy`	Designates the method as an interceptor method for pre-destroy lifecycle events

Interceptor Classes

- Interceptor classes may be designated with the optional jakarta.interceptor.Interceptor annotation, but interceptor classes are not required to be so annotated. An interceptor class must have a public, no-argument constructor.
- The target class can have any number of interceptor classes associated with it. The order in which the interceptor classes are invoked is determined by the order in which the interceptor classes are defined in the jakarta.interceptor.Interceptors annotation. However, this order can be overridden in the deployment descriptor.
- Interceptor classes may be targets of dependency injection. Dependency injection occurs when the interceptor class instance is created, using the naming context of the associated target class, and before any @PostConstruct callbacks are invoked.

Interceptor Lifecycle

- Interceptor classes have the same lifecycle as their associated target class.
- When a target class instance is created, an interceptor class instance is also created for each declared interceptor class in the target class.
- That is, if the target class declares multiple interceptor classes, an instance of each class is created when the target class instance is created.
- The target class instance and all interceptor class instances are fully instantiated before any @PostConstruct callbacks are invoked, and any @PreDestroy callbacks are invoked before the target class and interceptor class instances are destroyed

Using Interceptors

- To define an interceptor, use one of the interceptor metadata annotations listed in Interceptor Metadata Annotations within the target class, or in a separate interceptor class. The following code declares an @AroundTimeout interceptor method within a target class.

```
@Stateless
public class TimerBean {
    ...
    @Schedule(minute="*/1", hour="*")
    public void automaticTimerMethod() { ... }

    @AroundTimeout
    public void timeoutInterceptorMethod(InvocationContext ctx) { ... }
    ...
}
```

Intercepting Method Invocations

- Use the @AroundInvoke annotation to designate interceptor methods for managed object methods. Only one around-invoke interceptor method per class is allowed. Around-invoke interceptor methods have the following form:

```
@AroundInvoke
visibility Object method-name(InvocationContext) throws Exception { ... }
```

For example:

```
@AroundInvoke
public void interceptOrder(InvocationContext ctx) { ... }
```

Intercepting Lifecycle Callback Events

- Interceptors for lifecycle callback events (around-construct, post-construct, and pre-destroy) may be defined in the target class or in interceptor classes.
- The jakarta.interceptor.AroundConstruct annotation designates the method as an interceptor method that interposes on the invocation of the target class's constructor.
- The jakarta.annotation.PostConstruct annotation is used to designate a method as a post-construct lifecycle event interceptor.
- The jakarta.annotation.PreDestroy annotation is used to designate a method as a pre-destroy lifecycle event interceptor.

```
void method-name() { ... }
```

For example:

```
@PostConstruct
void initialize() { ... }
```

Lifecycle event interceptors defined in an interceptor class have the following form:

```
void method-name(InvocationContext) { ... }
```

For example:

```
@PreDestroy
void cleanup(InvocationContext ctx) { ... }
```

Binding Interceptors to Components

- Interceptor binding types are annotations that may be applied to components to associate them with a particular interceptor.
- Interceptor binding types are typically custom runtime annotation types that specify the interceptor target.
- Use the jakarta.interceptor.InterceptorBinding annotation on the custom annotation definition and specify the target by using @Target, setting one or more of TYPE (class-level interceptors), METHOD (method-level interceptors), CONSTRUCTOR (around-construct interceptors), or any other valid target:

```
@InterceptorBinding
@Target({TYPE, METHOD})
@Retention(RUNTIME)
@Inherited
pubic @interface Logged { ... }
```

Questions?