Adding Business Logic Using Managed Beans and Expression Language

Jesper de Jong
SOFTWARE ARCHITECT

@jesperdj www.jesperdj.com

Overview

Implementing Business Logic

- Managed Beans
- Contexts and Dependency Injection
- Bean Scopes

Using Managed Beans in Facelets Pages

- Expression Language

Demo Application

Implementing Business Logic in Managed Beans

What Is a Managed Bean?

- Concepts: lifecycle management and dependency injection

- A regular Java object, managed by a container
- Container: Java EE application server

- The container manages the lifecycle of the object
- Client code lets the container provide the bean through dependency injection

Lifecycle Management and DI

- Example: ProductService

- Is a ProductService object stateful or stateless?
- How do I get an instance of the ProductService?

Using the ProductService

```
public interface ProductService {
    Product getProduct(long id);
}


public class Example {
    @Inject
    private ProductService productService;

    // ...
}
```

Contexts and Dependency Injection

- Standard Java EE API
- Annotations
- Scopes
- Dependency Injection
- Naming Beans

Contexts and Dependency Injection

- CDI uses annotations
- Beans have a scope which determines the lifecycle
- Beans can have a name

Contexts and Dependency Injection

- CDI Bean Requirements
  - No-arguments constructor, or constructor with `@Inject`
- Dependency Injection with `@Inject`
- Scope Annotations
  - CDI standard scopes
  - JSF specific scopes
- Name a bean with `@Named`

Applying Bean Scopes

CDI Scopes

JSF Scopes

Choosing a Scope for Your Beans

Making Beans Serializable

Standard CDI Scopes

- Package `javax.enterprise.context`
- Request scope: `@RequestScoped`
  - Single HTTP request-response cycle
- Session scope: `@SessionScoped`
  - HTTP user session
- Application scope: `@ApplicationScoped`
  - Application-wide, shared by all sessions
- Conversation scope: `@ConversationScoped`
  - For the duration of a programmer-defined conversation

Standard JSF Scopes

- View scope: `@ViewScoped`
  - All requests while staying in the same view
- Flow scope: `@FlowScoped`
  - Faces Flows

Choosing a Scope for Your Beans

- Request scope or view scope
  - Example: Holding query results
- Session scope
  - Examples: User information, shopping cart content
- Application scope
  - Example: Stateless objects that can be shared by all sessions
- Conversation scope
  - Example: The checkout process in a web shop

Making Beans Serializable

- Session, conversation, view scoped beans must be serializable
- Glassfish error message: "Bean declaring a passivating scope must be passivation capable"
- Serialization is contagious

Non-serializable Dependencies

```
@ApplicationScoped
public void ProductServiceImpl implements ProductService {
    // ...
}

@ViewScoped
public void ProductDetails implements Serializable {
    @Inject
    public ProductService productService;

    // ...
}
```

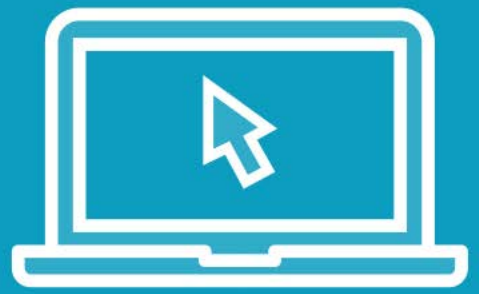
Non-serializable Dependencies

```
@ApplicationScoped
public void ProductServiceImpl implements ProductService {
    // ...
}

@ViewScoped
public void ProductDetails implements Serializable {
    @Inject
    public transient ProductService productService;

    // ...
}
```

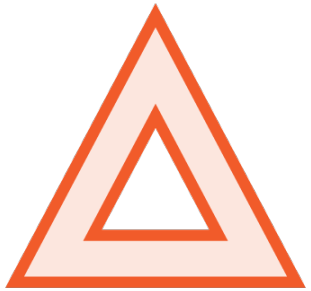
Demo



Implementing Managed Beans

- Get product information from a service

Beware of Old Versions

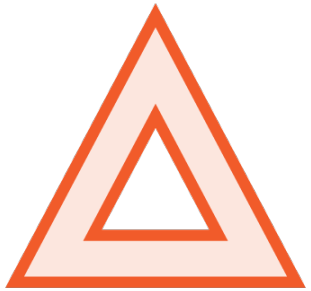


- JSF is older than CDI
- Beware of the old JSF managed bean annotations
  - Package `javax.faces.bean`
  - Easily confused with CDI annotations

`javax.faces.bean.SessionScoped`

`javax.enterprise.context.SessionScoped`

Beware of Old Versions



`javax.faces.bean.ViewScoped` 

`javax.faces.view.ViewScoped` 

Old Versus New Annotations

Old Annotation	New Annotation
javax.faces.bean.RequestScoped	javax.enterprise.context.RequestScoped
javax.faces.bean.SessionScoped	javax.enterprise.context.SessionScoped
javax.faces.bean.ApplicationScoped	javax.enterprise.context.ApplicationScoped
javax.faces.bean.ViewScoped	javax.faces.view.ViewScoped
javax.faces.bean.ManagedBean	javax.annotation.ManagedBean javax.inject.Named
javax.faces.bean.ManagedProperty	javax.inject.Inject

Accessing Managed Beans Using Expression Language

Referencing Managed Bean Properties

- **Binding a component value to a bean property**

```
<h:inputText value="#{user.name}"/>
```

Referencing Managed Bean Methods

- **Calling a method to perform an action**

```
<h:commandButton action="#{loginPage.submit}"/>
```

```
<h:commandLink action="#{shoppingCart.addProduct(product)}">
                    Add to cart
                  </h:commandLink>
```

Accessing Collections

- Accessing elements of an array or list

```
#{user.addressLines[0]}
```

- Accessing values in a map

```
#{user.phoneNumbers["mobile"]}
```

Dot and Square Bracket Syntax

- Accessing bean properties

```
#{user.name}
#{user['name']}
```

- Accessing map values

```
#{user.phoneNumbers["mobile"]}
#{user.phoneNumbers.mobile}
```

Expression Language Overview

- **Referencing Bean Properties**

```
<h:inputText value="#{user.name}"/>
```

- **Referencing Bean Methods**

```
<h:commandButton action="#{loginPage.submit}"/>
```

- **Accessing Collections**

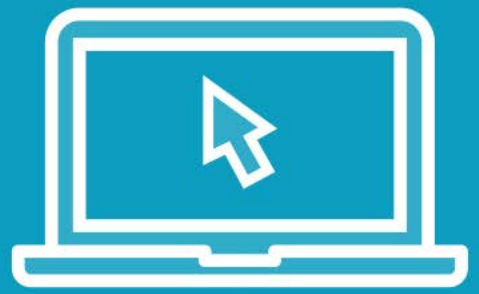
```
#{user.addressLines[0]}
#{user.phoneNumbers["mobile"]}
```

- **Dot and Square Bracket Syntax**

Immediate and Deferred Evaluation

```
${user.name}
#{user.name}
```

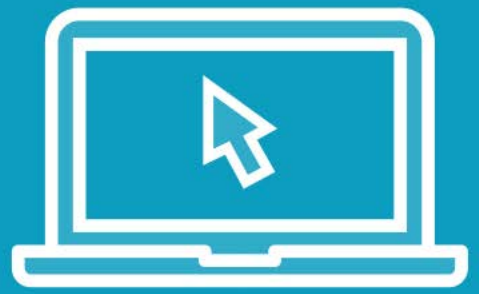
Demo



Using Expression Language

- Access the managed beans in the Facelets pages

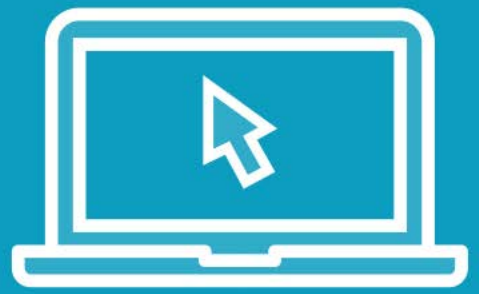
Demo



Making the Shopping Cart Work

- Bean to store the content of the shopping cart
- Make the “Add to cart” links work
- Make the top panel work
- Implement the shopping cart page

Demo



Implementing the Theme Selector

- Dynamically selecting the theme

Summary

Managed Beans

- Container manages the lifecycle and provides dependency injection
- Contexts and Dependency Injection
- Scopes

Expression Language

- Accessing managed beans in Facelets pages

Coming Up

- Handling user input
- The Facelets lifecycle
- A deeper look at JSF components