Managed Beans, Context and Dependency Injection (CDI) and Bean Validation

JSF Slides #3

Managed Beans

Managed Beans used as
- Glue between server side GUI (component tree) and model
- Receiver of incoming data (get from components in GUI-tree, set in bean), possible handle over to model or subsystem
- Supplier of outgoing data (get from bean, set in components in GUI-tree), possible read from model or subsystem
- Possible (often) as event listener
- Possible (often) define navigation (depending on processing outcome)
- Must run in some container (that can manage)

Design: Beans have various "roles", to be continued...

Managed Beans Confusion

JEE is an evolving platform, things change...

We will only (for now) use Context and Dependency Injection (CDI) managed beans .

There are also JSF managed beans, we don't use (replaced by CDI)
- Many code samples on the web, watch out!
- The annotation @ManagedBean is **not** used by us!!!

```
@ManagedBean
public class HelloBean ... {
                // BAD, BAD, BAD, BAD...
```

CDI Managed Beans Packages

Packages used for CDI managed beans
- javax.annotation.*
- javax.inject.*
- javax.enterprise.context.*

Never
- javax.faces.*
- EXCEPT javax.faces.event.*;

Warning, common mistake
- Importing java.**awt**.ActionEvent, BAD...should be
- javax.faces.**event**.ActionEvent!

CDI Built-in Scopes

As before (again: This is from javax.enterprise.*)
-@RequestScoped
-@SessionScoped
-@ApplicationScoped
And a new one
-@ConversationalScoped, more to come (see navigation)

Dependant pseudo-scope, if **no annotation**. Same scope as other object (user of bean) ... and more...

Warning also javax.faces.* scopes

A CDI Managed Bean

```
@Named("mybean") // CDI for managed beans, name used in pages
@SessionScoped   // Created at session, destroyed when finished
public class SomeBean implements Serializable {

   private int data;

   ... set/get for data
}
```

For this to work must have (almost empty) file **WEB-INF/beans.xml**, NetBeans will normally create

Managed Bean as Event Listeners

Managed beans may have listener methods, can act as event listener

```
@Named("mybean") @RequestScoped
public class MyBean ... {
   //Callback for valueChangeListener attribute in <h: ...>
   public void valueChange(ValueChangeEvent evt){
    ...// name of method arbitrary
   }
   // Callback for actionListener attribute in <h: ...>
   public void buttonClicked(ActionEvent evt) {
    ...
   }
   // Callback for action attribute in <h: ...>
   public String navigate() {
    ...
   }
}
```

Component Binding

Possible to access UIComponents in GUI tree in managed bean, **component binding**

In page

```
<!-- Tag will create a component in server GUI -->
<...binding="#{bean.txtData}".../>
```

In bean

```
// txtData reference to component in tree
private UIInput txtData; // Also need set and get
public void buttonClicked(ActionEvent evt) {
     // Will show up in GUI
     txtData.setValue("Hello");
}
```

CDI Dependency Injection

Injection: Container create object and set reference to

Possible to inject interface types (loose coupling)

Possible to inject "any managed into any other managed" (often CDI beans into CDI bean), but must have an **injection point**

Injection points:
- Constructor injection, prefer...(can only have one)
- Initializer method injection, ok ...
- Field (attribute) injection, ... bad (makes it hard to test)

CDI Injection Points

```
// Constructor. THIS DOESN'T SEEM TO WORK WITHOUT A DEFAULT CTOR?!?
public class Checkout {
private final ShoppingCart cart; // Other CDI bean
@Inject
public Checkout(ShoppingCart cart) {
this.cart = cart;
}}

// Initializer method
public class Checkout {
private ShoppingCart cart;
@Inject
void setShoppingCart(ShoppingCart cart) {
this.cart = cart;
}}

// Field
public class Checkout {
@Inject
private ShoppingCart cart;
}}
```

What Get Injected

Very many options to specify
- Simple case just one type possible

Complicated
- Many implementations of interface and more...
- See CDI ref.

Injection Timing

This is the order

1. Constructor run (if constructor injection, do inject)
2. Initializer methods and field injection
3. Life cycle callback executed, upcoming...

Life Cycle Callbacks

@PostContruct
- Called after constructor
- After injection done (i.e. no null pointer exception)
- Only one method can have

```
@PostConstruct
public void postContruct() { // Any name
    // Do something...
}
```

@PreDestroy
- Called before destruction (release resources)

Validation

Principle: All layers should validate input data, validate client-side and server side

Client side : HTML5/CSS3/JavaScript client side validation

Server side: Two different ways to validate

JSF Validation, part of JSF specification

Beans Validation JSR 000303, a different specification
- JSF validation targeting just JSF, Bean Validation targets "any" application (also Java SE), prefer ...

JSF Validation

Tags...

```
<f:validateLength maximum="8" minimum="1"  />
<f:validateRegex ..... />

    ...

<!-- In a page -->
<h:inputText id="txtData" value="#{bean.data}" required=
                    "true" validatorMessage = "Must be 1-8 chars" >
   <f:validateLength minimum="1" maximum="8"   />
</h:inputText>
<h:message for="txtData" showSummary="true" showDetail="false"
                                    style="color: red" />
```

..and many more options (possible create custom Validator)

Bean Validation

Works seamless with JSF, use annotations (validation errors will end up in <h:message(s)../>)

```
@NotNull
@Size(min = 1, max = 8, message = "Must use 1-8 chars")
private String data;

@Pattern(regexp = "[A_Z][a-z]*")
private String pattern;

@Min(value = 1, message = "To small")
@Max(value = 10, message = "To big")
private int value;
```

JSF and Bean validation not in same phase, JSF first then Beans (if not JSF failed, then skipped)

Validation Messages

Possible to store localized messages in ValidationMessages.properties file (exact that name, possible suffix for language _en, _sv!)

```
// ValidationMessages.properties
user.password =  Password name must be 3-20 characters
…
```

Location of file in NetBeans:
Other Sources > src/main/resources/default package
Location in war: WEB-INF/classes

Usage in code

```
@Size(min = 3, max = 20, message = "{user.password}")
private String password;
```

Bean Validation Advanced

Bean validation is extremely customizable! See links

Possible to define advanced validations
- Creating own annotations for attributes, example @OrderNumber
- Creating validation annotations for complete class (object)
- Validation groups
- Different stages, i.e. validation rules vary over time

Exceptions

Exceptions in control/model layer (if not using Bean Validation)

```
// In managed bean
try {
     cr.add(new Customer(login, passwd, "...", 10));
     return "login";
 } catch (Exception e) {
     // Will end up in <h:message(s) ../>
     FacesContext.getCurrentInstance().addMessage(
                  new FacesMessage("Bad Login name"));
 }
```

JSF and CDI Testing

Beans must run in container, how test?

If using constructor or method injection possible to supply needed objects, so just run JUNit test like POJO's

Some thought's

If using JSF/CDI as a thin "technical" layer between user and model, there should not be much of testing need, no application logic in pages, beans

CDI has a LOT more...

Producer methods, Interceptors, Decorators, Events, Stereotypes, create own scopes, ... not used by us, see CDI ref.