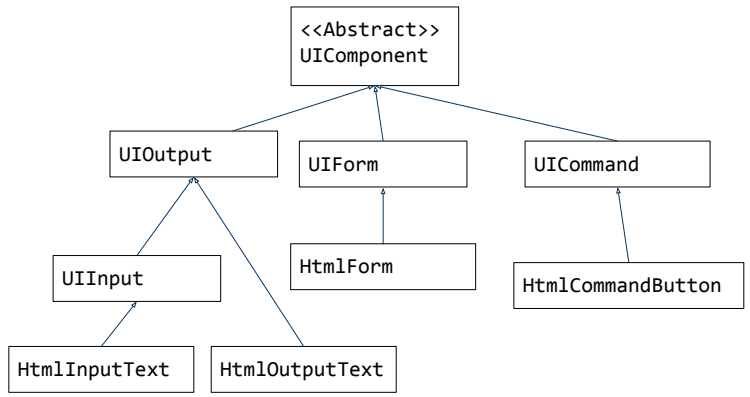
JSF and CDI

JSF Slides #2

Content

- Component Classes
- JSF Pages
- Faces Servlet and Faces Context
- Expression Language (again))
- POST and GET
- JavaBeans
- Context and Dependency Injection (CDI)
- View parameters
- MVC

JSF Component Classes

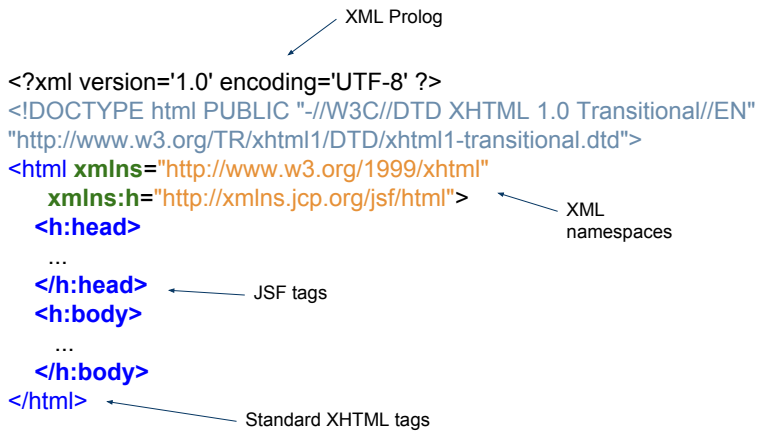


Many more classes ….

Server side component classes (tree of component objects),

- API
- Toplevel is UIComponent (similar to Swing)
- Incoming data from HTTP-request will be converted and stored in the components
- Outgoing data retrieved from components
- A renderkit to translate components (and data) to renderable format (spec. requires a HTML renderkit).
    - Handled transparently by JSF
    - Example: HtmlCommandButton object → <input type="submit" … > (String)

A JSF Page



Possible to create objects as usual (new HtmlCommandButton etc.)

- But tedious … build a component tree by hand (for each view)?!?!
- Also … must be programmer… NO!

Better use a declarative approach

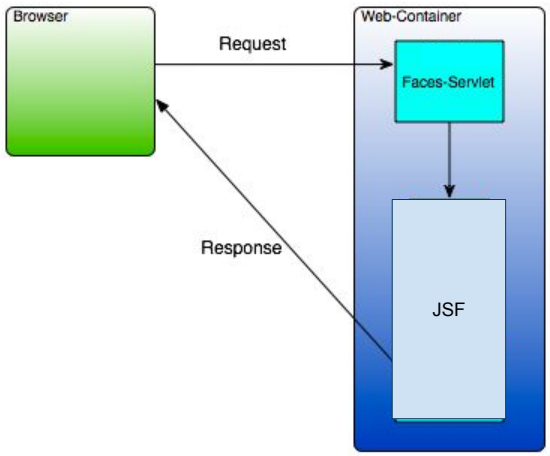
- Write a JSF "page" using element to represent the components in tree
  - One page for each view
- NOTE: The page is not sent to the client
  - It's used to build the tree (at request)…
  - … later transformed to HTML/CSS/JS and sent to client
- If needed: Possibly to manipulate tree components in other Java code using "binding attribute" for elements
- NOTE: The tree has state (the values in the components in tree)!

Tech talk

- JSF pages written in an XML-language (so must be well formed, i.e. closing tags)
  - If not exception!
- DOCTYPE generated by NetBeans, don't bother …
- Components as tags, a few tag families, most common is <h: …>, <f: …> and <ui: …>
  - Core JSF Html tag reference
  - Must be able to separate tag families, use XML-namespaces
  - To be able to use different tag families they must be set as namespace attribute (xmlns:) in <html> element

- There are implicit objects available to pages (avoid, low abstraction level)
- Possible to use JSTL in JSF but we avoid
    - Complex with tags executing at different times, strange behaviour …

Faces Servlet and FacesContext



JSF is a subsystem of the container

Container can handle many different types of request

- Could be an JSP or JAX-RS request (i.e. not a JSF request)
- To invoke the JSF subsystem, request must go through the FacesServlet (supplied by JSF)
  - Servlet "invisible" to developer
  - Configuration of Faces Servlet in web.xml
    - URL pattern to trigger Servlet (/faces/, *.xhtml or other, …), specify in web.xml
    - Possible with multiple patterns

FacesContext (object)

- Contains all of the per-request state information related to the processing of a single JavaServer Faces request, and the rendering of the corresponding response. It is passed to, and potentially modified by, each phase of the request processing lifecycle (see JSF life cycle)
- Globally accessible in code as: FacesContext.getCurrentInstance()
- Entry to
  - …the component tree, an event queue (storing events for later execution), messages (possibly failure messages to GUI)
  - Also possible to access low level data (HTTP request, session, …)

Session handling

- Same as before (handled by container)

- On logout or session time out we should invalidate the session
    - FacesContext.getCurrentInstance().getExternalContext().
      invalidateSession();

Expression Language in JSF

EL as read

```
<h:outputText value="#{person.name}"/>
```

NOTE

EL as write

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

Same idea as JSP. Used as in JSPs but

- … uses deferred syntax
- **#**{ … expression … }
  - Deferred is both read and write (depends on expression location)
  - Input vs output component
- JSP used immediate syntax ${…expression…}
  - Immediate syntax is read only

Tech talk

- outputText will call getName on some object
- inputText will call setName (input rendered as <input type="text" …>)

Post and Get

POST

```
<h:form>
   ...
   <h:commandButton ... />
   <h:commandLink action=.../>
</h:form>
```

GET

```
<h:link outcome=.../>
<h:button outcome=.../>
```

To send a POST request use

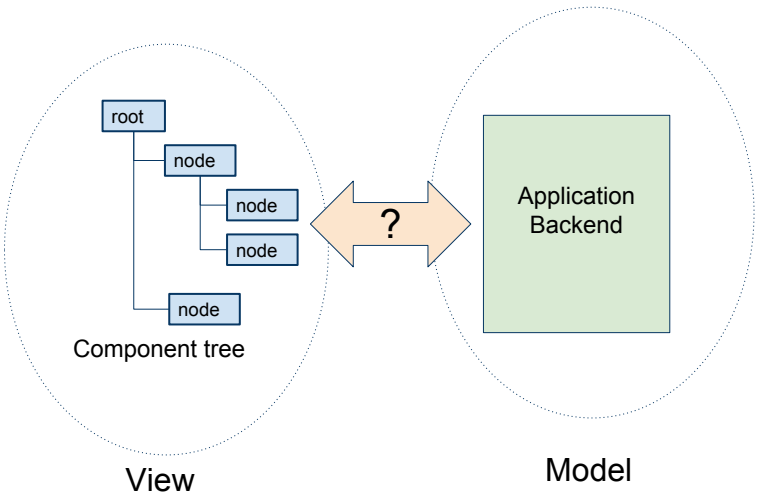
- commandButton or commandLink
- Must be inside <h:form…>
  - And so must the input controls too (<h:inputText…>, etc.)
- NOTE: <h:form> has no action attribute. Will "post back" to same page

To send a GET use

- link or button
- No need for <h:form …>

Important, POST and GET will control behaviour of JSF, see JSF Request Cycle, later …

View and Application Backend



We need to knit together the web tier (the server side component tree) and the transactional tier (rest of application, database)

We will not use Request or Session objects like in JSP

- We will use Beans … upcoming…

JavaBeans

Recurring term: Bean(s). Examples

- Session bean
- Managed bean
- Enterprise bean

A bean is a:

- Plain old Java class (POJO)
- Private attributes and read/write methods for (relevant) attributes (attribute +set + get is called a property)
  - Naming conventions for set/get-methods
  - private String data → public String getData() → public void setData(String str);
- Non-private default constructor
- Serializable

Sadly it's a bit of an antipattern... (immutable hard)

- Any way will show up ...

A Java EE component is a bean if the lifecycle of its instances may be managed by the container

- Container will create objects (and also passivate i.e. temporarily swap to disk, serializable important!)
  - Also all references must be serializable
  - Or: Use transient keyword
- Container will execute callbacks on bean

- AKA Managed beans

Context and Dependency Injection

```
import javax.enterprise.context.RequestScoped;

@Named("rbean")
@RequestScoped
public class RequestBean implements
                                Serializable {


  ...
}
```

Contexts and Dependency Injection for Java EE (CDI)

- Objects handled by container (i.e. beans)
- Act as "glue" layer between view and model. Will act as:
  - (temp) Holder for incoming data from view
    - JSF will automagically call setters to to transfer data from input gui components to beans
  - Supplier of data to output gui components (to display in client)
    - JSF will automagically call getters to to transfer data from beans to gui components in tree
  - Event listeners (like Swing listeners)
  - Control layer in MVC-architecture
- Possible to inject objects, more to come …
- There is much more to the picture
  - Reference implementation is Weld

Tech talk

- @Named annotation specify name to use in EL (must be unique in application)
- Scope annotation similar to other scopes we have seen
  - Must be annotated using annotations from javax.enterprise.context
    - @RequestScoped
    - @SessioScoped
    - @ApplicationScoped
    - @ConversationalScoped
    - @Singleton, a singleton (a pseudo scope), avoid

        - Dependant pseudo-scope, if no annotation, scope of owner
    - Must have file beans.xml in WEB-INF (NetBeans can generate)

- There are other annotations we can use, specified in JSF 2.2 (or elsewhere (?) … NOT inf CDI, confusing …)
    - @ViewScoped NOTE: javax.faces.view.ViewScoped
    - @FlowScoped NOTE: javax.faces.flow.FlowScoped
        - Used for page flows (collection of pages type wizards)
    - Flash scoped (no annotation) Flash Scope provides a way to pass objects between various JSF pages. The objects put in Flash scope are available only in the next view and then are cleared out (probably better use view parameters, upcoming …)

JEE is an evolving platform, things change…

- There are also the older "JSF managed beans", <u>we don't use</u> (replaced by CDI and the above)
    - Many (bad) code samples on the web, watch out!
    - <u>The annotation @ManagedBean should never occur in our code!!!</u>

NetBeans not so good at handle renaming of beans, possibly need to clean and build

- Exception: "… bean name ambiguous… "
- In pathological cases delete target (Files view)

Callbacks

```
import javax.annotation.PostConstruct;
import javax.annotation.PreDestroy;

@PostConstruct
public void post() {
  out.println("Application post contruct");
}

@PreDestroy
public void destroy() {
  out.println("Application pre destroy");
}
```

Methods automatically called by container (if specified)

- Used for initialization and cleanup
- Specified with annotations
  - Method names doesn't matter
- @PostConstruct, called after constructor and after injection done (upcoming)!
  - Good! Use to avoid NPE!
- @PreDestroy, called before garbage collection

CDI Beans as Listeners

Application Events

```
<h:commandButton ...
        actionListener="#{lbean.doActionListener}".../>

<h:commandButton ...
              action="#{lbean.anyMethod}" />

<h:inputText ... valueChangeListener="#{lbean.doValueChange}" />
```

System Events

```
<f:actionEvent action="#{lbean.preRender}" />
```

Possible to specify methods in CDI bean to be called as listeners (like Swing)

- If bean has listener method(s) possible to add as actionListener- , action-, or valueChangeListener-attribute on element or …
- … use as a system event listener (parameter type ComponentSystemEvent)

Tech talk

- Name of methods doesn't matter
- Application events
  - actionListener: Bean must have method with ActionEvent param
    - ActionEvent delivered to method
      - e.getSource() like Swing
      - Warning, common mistake
        - Importing java.awt.ActionEvent, BAD…should be
        - javax.faces.event.ActionEvent!
  - action attribute may specify any method (with params or not)
    - No event delivered but
    - If method returns a String, the string will be used in navigation, the name of the page (no .xhtml-suffix needed)
  - valueChangeListener: Bean must have method with ValueChangeEvent
    - Event delivered
    - Has e.getNewValue() and e.getOldValue()
  - Possible with many listeners on same element
- System events may be of many kinds, specify with attribute type on <f: actionEvent>

    - Not executed on post backs
- A data-model event occurs when a new row of a UIData component is selected (not shown)

CDI Injection

```
import javax.inject.Inject;

// In bean
private PersonBean pb;

@Inject
public void setPersonBean(PersonBean pb) {
        this.pb = pb;
}
```

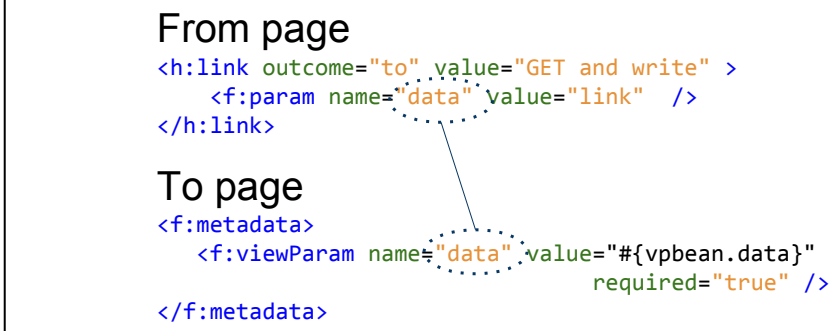
Injection = container creates and assigns instances of beans (to other beans)

Possible to inject using
- bare attribute, avoid
- constructor, ok
- or setter, ok
- The above are called injection points
    - If not possible to find bean matching injection point, error
    - NetBeans will show (injection point fail or similar)

This is the order
1. Constructor run (if constructor injection, do the inject)
2. Initializer methods and field injection
3. Life cycle callback executed (postConstruct)

Injection options much more complex than shown here.
- Variable may be interface type … many implementations.. which?
    - Very loose coupling if using interface types … nice!
- And more … see spec.

View Parameters



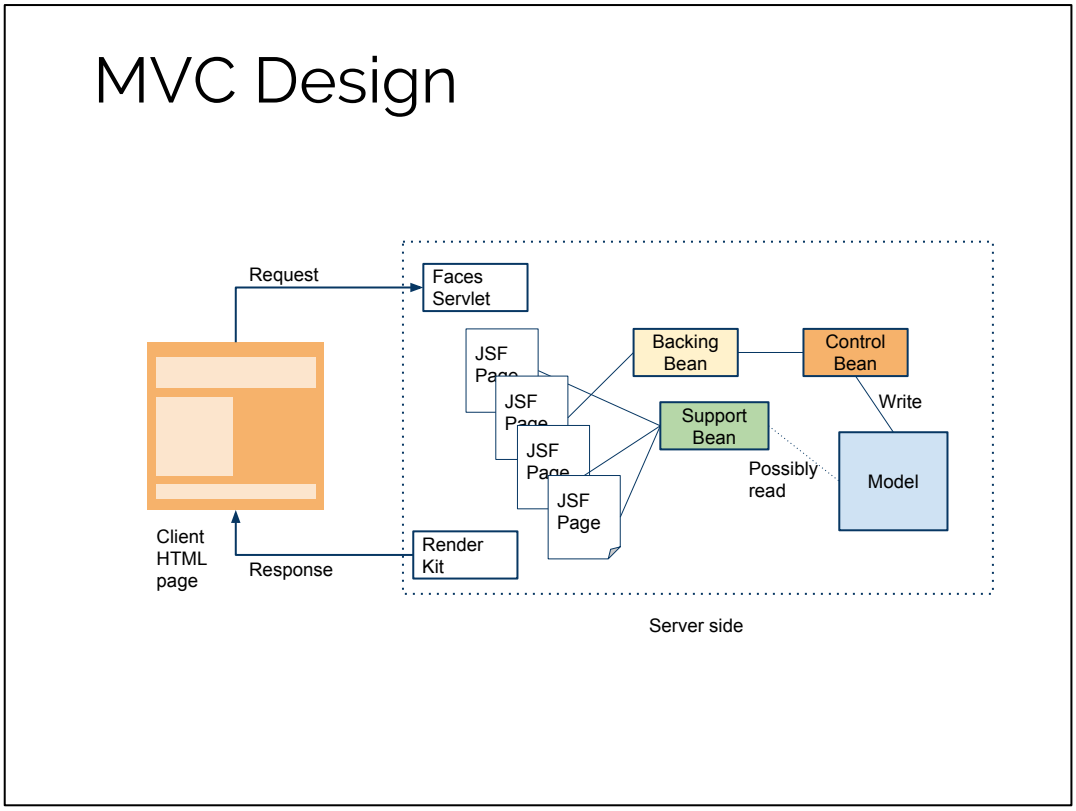
Normally GET is a read operation but sometimes we would like to write (to specify search criteria and alike).

- To be able to write to a bean with GET we use view parameters
- Bookmarkable GETs

Tech talk

- <h:link ..> will issue a GET request
- From page adds query string ("?data=link") to request using <f:param ..> tag
- "To-page" uses <f:metadata> and <f:viewParam> to retrieve query parameter data and call setData on some bean
- Final result: Query string data stored in bean

See also JSF Request Cycle later



[CDI bean roles]

- Backing bean: Holder for data. 1:1 with some page, request scoped
- Control bean: In control layer, transfer data from/to model. Act as listener (action attribute in element) , request scoped
- Support bean: Supply data common to more pages, session or application scoped
- Utility bean: General utility, application scoped (possibly reusable between applications)