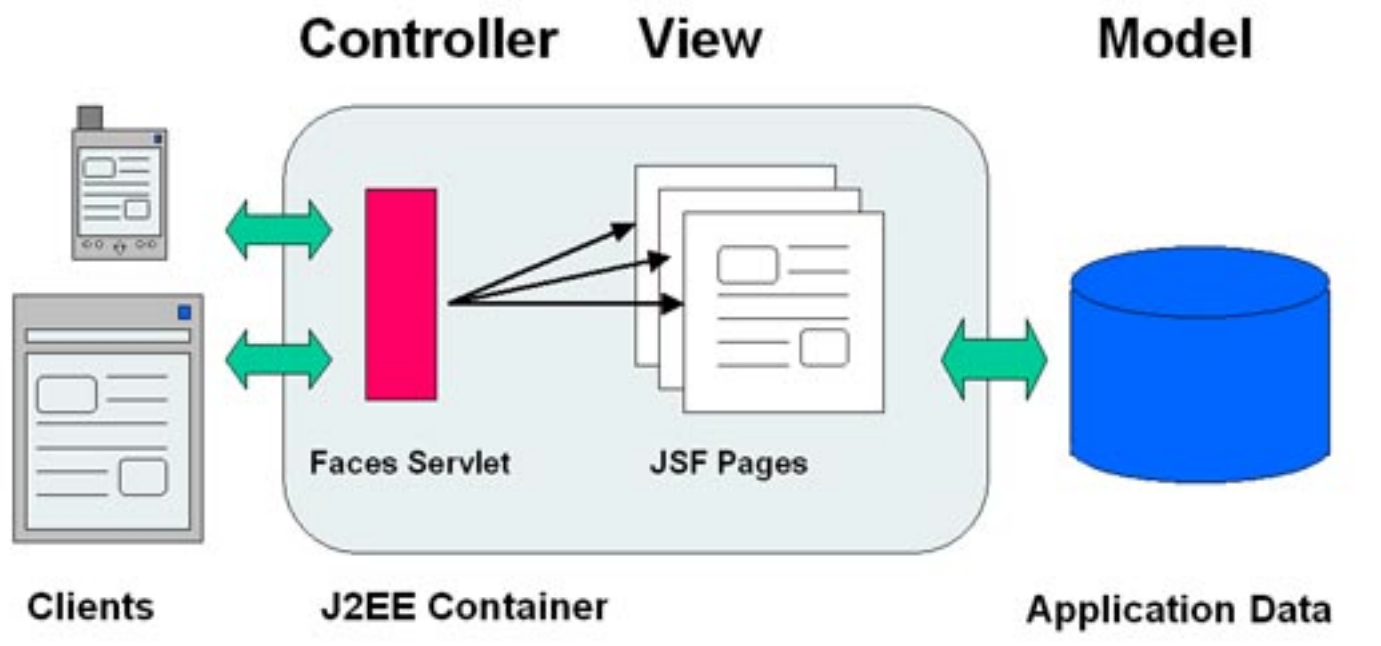
Enterprise Computing, Lecture 3 – Web Tier

Java Server Faces - JSF

Objective

- Context: what is JSF, and where does it fit in to the JEE architecture
- Architecture: Explain the components of the JSF architecture
- See it in action: Create a simple App using JSF and view the generated code

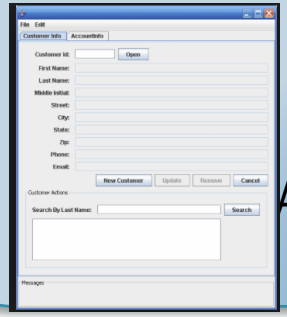


What is JSF?

- Java Server Faces is the recommended framework for generating the **User Interface component** of a JEE application.
- It's purpose was to bring graphical components to the Web, and was inspired by the SWING component model and other GUI frameworks
- Its goal is to make Web Development faster and easier by supporting user interface components, such as text boxes, list boxes, tabbed panes, and data grids, in a Rapid Application Development (RAD) environment.
  - RAD – iterative development through the construction of prototypes, with less time spent in planning stages

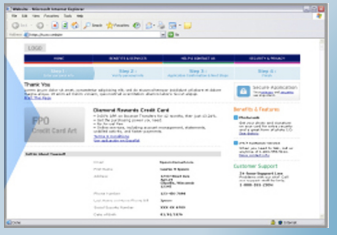
Java EE tiers

Java EE Application 1



Application Client

Java EE Application 2



Web Pages

Client Tier

Client Machine

Java Server Faces pages

Web Tier

Java EE server(s)

Enterprise Beans

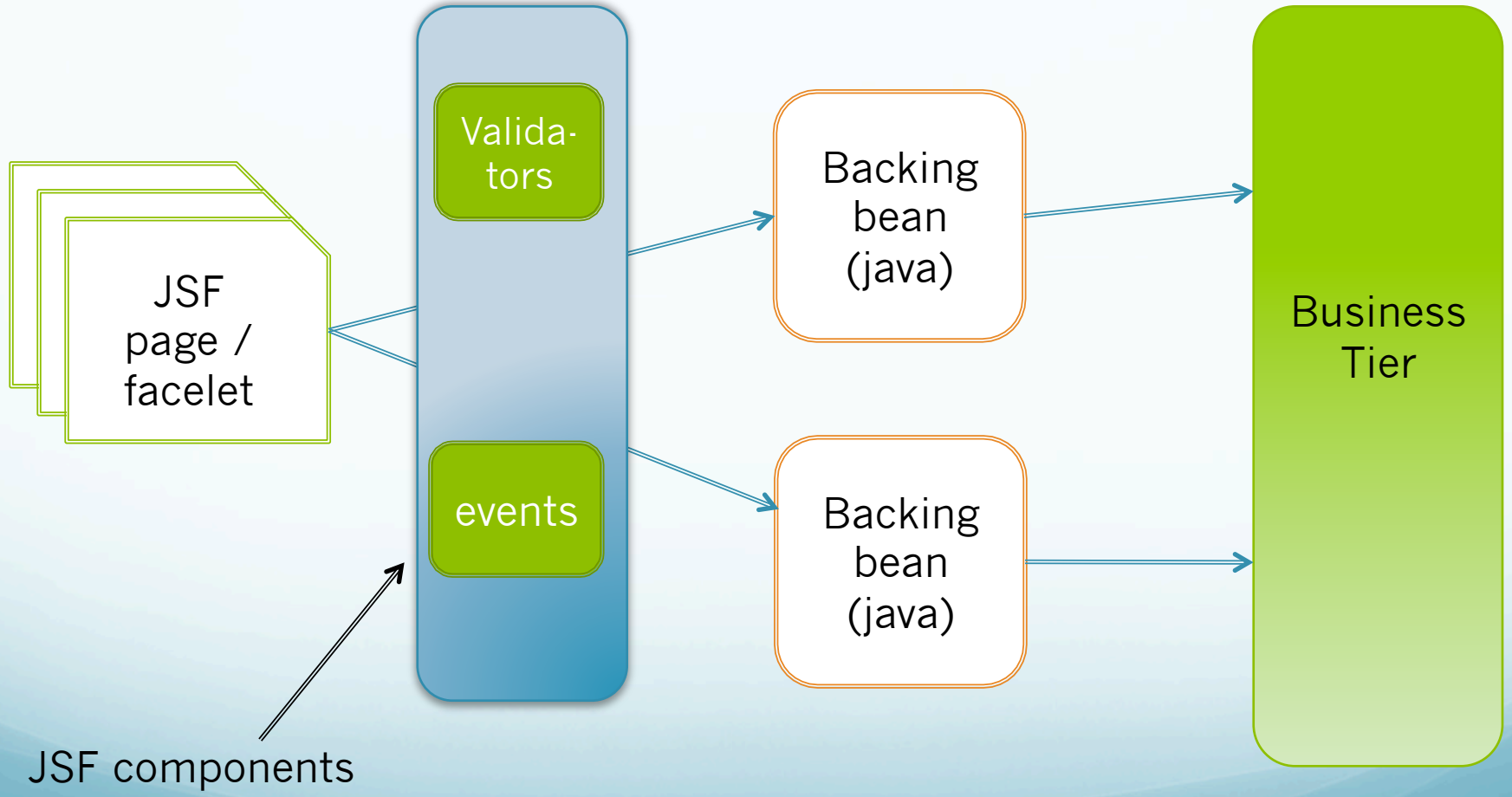
Enterprise Beans

Business Tier

Data Tier

Database server(s)

JSF overview



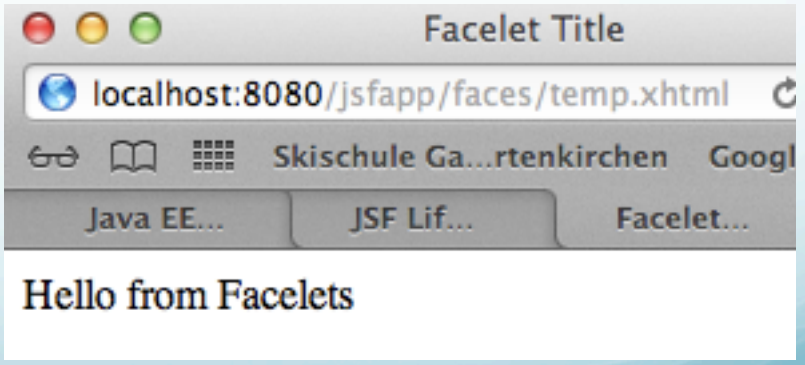
Some terminology

- JSF: Java Server Faces – the Java specification for component based user interfaces for the web. It is a standards specification, not an implementation of that standard.
- **Facelets**: a template of UI components that conform to the JSF standard. It's a view declaration **language** that extends XHTML.
  - Note: The are other languages for web applications that conform to JSF such as JSP and XUL (XML interface language).
- *Java EE currently recommends using facelets for JSF web apps, which is supported by Netbeans.*

Sample JSF code

- A facelets/JSF page is like an XHTML page, with some differences in the tags used.

```
<?xml version='1.0' encoding='UTF-8' ?>
<DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://
www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html">
    <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        Hello from Facelets
    </h:body>
</html>
```



Complete steps 1 to 7 in lab sheet 2

. . .adding a form

- A pandelGrid can be used to layout a form, and is generally configured with three columns:

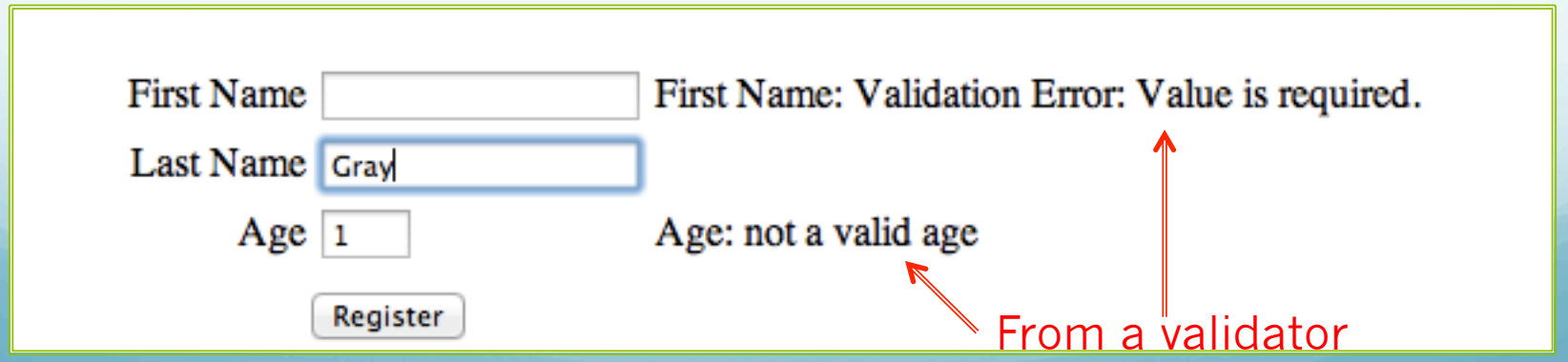
Defined in a css file

```
<h:form id=studentForm>

<h:panelGrid columns="3" columnClasses="rightalign,leftalign,leftalign">

 <h:outputLabel value="First Name" for="firstName"/>
 <h:inputText id="firstName" label="First Name" required="true" />
 <h:message for="firstName" />
```

Output Label	Input text	Error Message



. . .adding a form

- A command button can call another XHTML page directly (static), or retrieve the name of the next page from a backing bean (dynamic)

```
<h:commandButton id="register" value="Register"
action="confirmation"/>
```

Calls confirmation.xhtml

or

```
<h:commandButton id="register" value="Register"
action=#{registrationBean.nextPage()}/>
```

The *nextPage()* method in the *registrationBean* java class must return a string, which is the name of the html page to be called next. (The .xhtml extension will be added automatically.)

- Note: to indent the register button as per the last slide, the following adds a blank cell to the grid:
- <h:panelGroup/> <!--create an empty cell-->

Complete steps 8 & 9 in lab sheet 2

Creating a managed bean

- A bean is a java program that has a specific business purpose
- A managed bean manages some resource, in this case it manages the data entered by a user on a XHTML form.
- A managed bean for a XHTML form requesting a firstname, lastname and age would look like the following . . . .

```
@Named(value = "registrationBean")
@RequestScoped

public class RegistrationBean {

    public RegistrationBean() {
        }

       private String firstName;
        private String lastName;
        private Integer age;

    public String getFirstName() {
        return firstName;}
    public void setFirstName(String firstName) {
        this.firstName = firstName;  }
    public String getLastName() {
        return lastName; }
    public void setLastName(String lastName) {
        this.lastName = lastName;  }
    public Integer getAge() {
        return age; }
  public void setAge(Integer age) {
        this.age = age;  }
```

Declared as a managed bean referenced as 'registrationBean'

Scope defines who can see the variable's values, and how long they are stored

Declare a variable for each input

Add a get and set method for each input

Other methods can also be added.

Connecting a XHTML page to a managed bean

- Variables and methods can be called directly from within any XHTML page in the application as shown below:

```
<h:outputLabel value="First Name" for="firstName"/>

<h:inputText id="firstName" label="First Name"
required="true" value="#{registrationBean.firstName}"/>

<h:message for="firstName" />
```

The class name with the first letter changed to lower case

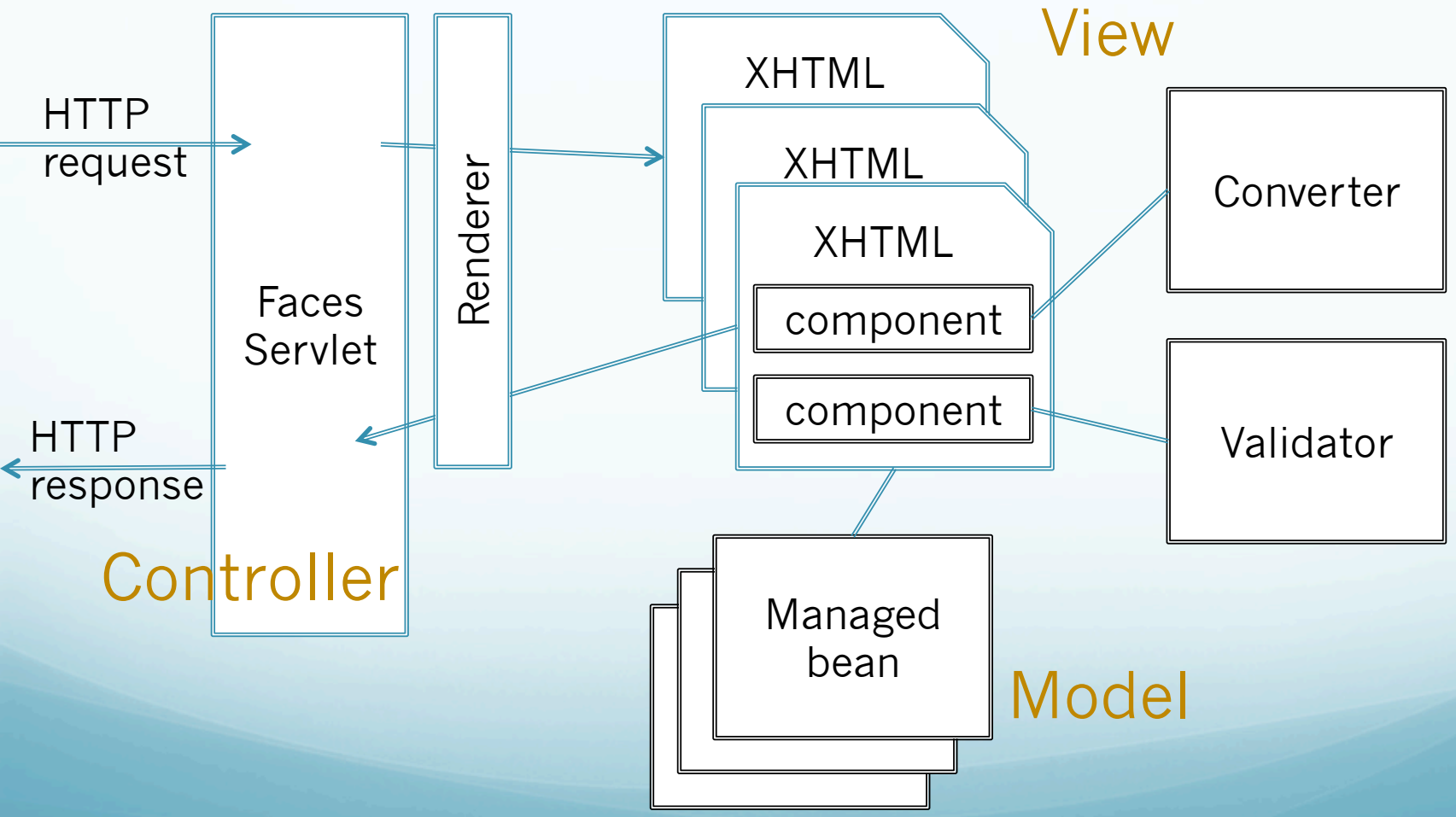
Variable or method name

An input text box will set the value of a variable in a managed bean, and output text box will retrieve the value

Complete steps 10 and 11 in lab sheet 2

JSF architecture

MVC pattern



JSF architecture

CONTROLLER – user interactions

- **Faces servlet & faces-config.xml**: The faces servlet is the controller in the MVC pattern, and is internal to JSF.

MODEL – the data to be displayed on the page

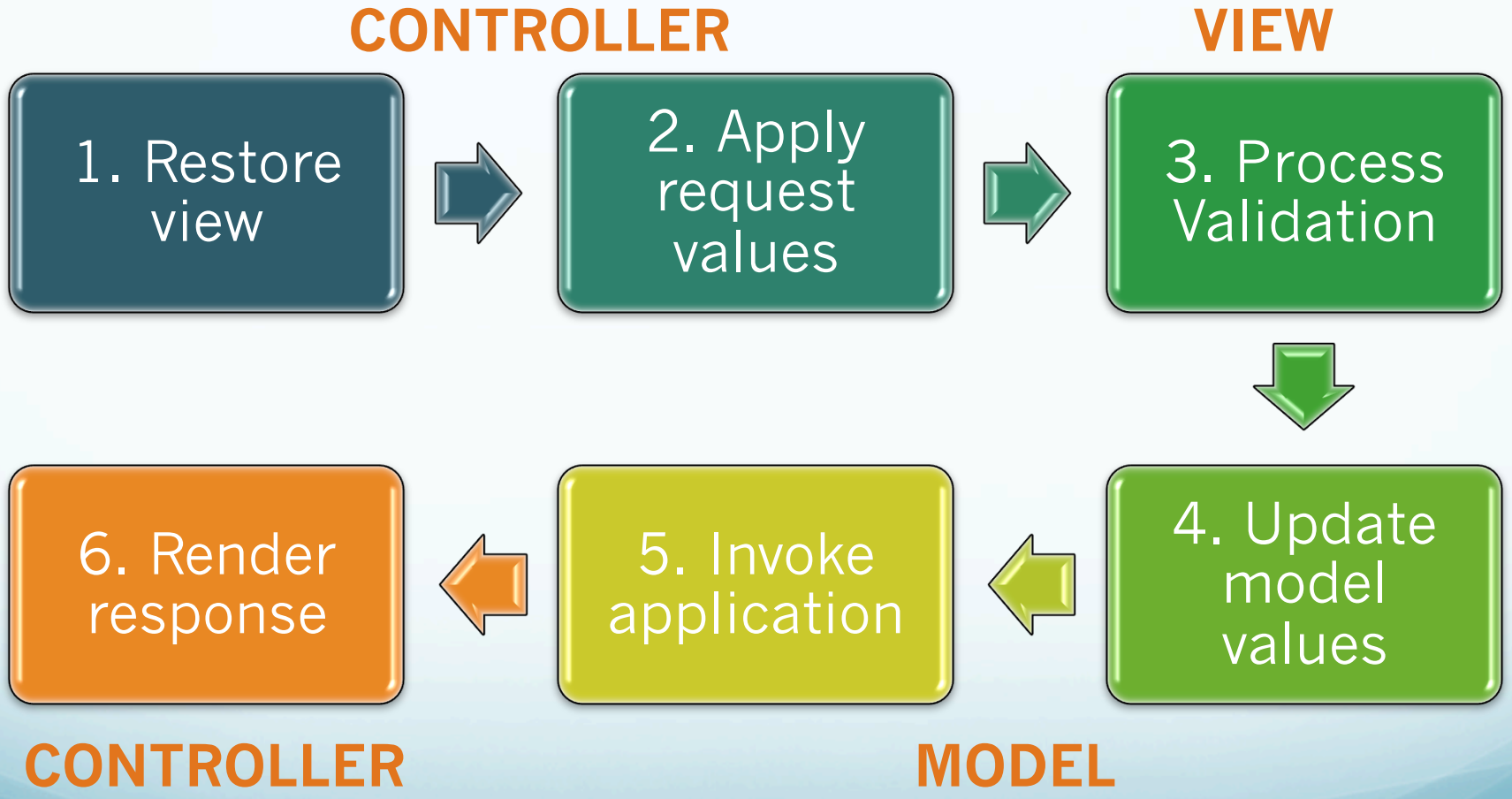
- **Managed Bean**: the backing beans supported each UI page. Variables and methods in the managed beans can be reference directly from a JSF page.

JSF architecture

VIEW- the web page:

- XHTML, JSP, XUL or facelets – i.e. **view technologies** for the user interface and its components.
- **Converters**: converters convert a components value (date, boolean etc.) to and from String values.
- **Validators**: the validation rules for the UI components, ensuring the value entered by the user is valid
- **Renderer**: This is an optional layer which controls how a UI component is displayed, allowing the same component to be rendered differently for different clients, e.g. a web browser or mobile phone.

JSF page life cycle



JSF page life cycle

- **Restore View**: Locate the requested page to either build it anew or restore it.
- **Apply Request Values**: values that have come from the request are applied to the various page components.
- **Process validations**: UI components now have their value set. JSF traverses the component tree and asks each component to validate the submitted value. This checks both conversion and validation rules.

JSF page life cycle

- **Update model values**: update the managed bean's elements associated with each component.
- **Invoke application**: perform the relevant business logic based on the actions triggered in the managed bean following the user input.
  - This could include running SQL commands and/or page navigation.
- **Render response**: Send the response back to the user, and save the state of the view (for restoration at the start of the next cycle).

Five scopes of a managed bean

- Objects that are created from a managed bean class have a certain life span, based on the scope of the managed bean, which can be specified as an annotation at the top of the program. There are five possible scopes:
  1. **@ApplicationScope**: This is the longest life span. Objects are available to ALL clients using the web application for as long as the application is active. The objects can be called concurrently, and need to the thread safe.
  2. **@sessionScope**: The object is available to all request/response cycles that belong to a client session until the session is invalidated.

Five scopes of a managed bean

3. **@ViewScoped**: Objects are available within a given view until the view is changed
   - A view is a JSF page
4. **@RequestScoped**: Objects are available from the beginning of a request until a response is send to the client. This is the default scope if no scope is specified.
5. **@NoneScope**: managed beans with this scope are not visible in any JSF page, and define objects used by other managed beans in the application.

JSF: Navigation

Static & dynamic navigation

- As was mentioned earlier, the action parameter on a command button can contain the name of the next XHTML page to navigate to. This statically defines page navigation.
- Page navigation can be done dynamically by calling a method in a managed bean. This method MUST return a string, which will be the name of the next page, giving more flexibility to navigation paths through your application.

Example of dynamic navigation . .

In the XHTML page:

- <h:commandButton id="register" value="Register" action=#{navigateBean.nextPage()}/>

In the NavigateBean class:

```
public String nextPage() {
        if (age>=18)
          return “over18”;
       else if (age<18)
          return “under18”;
        else
          return “errorPage”;
   }
```

JSF will navigate to either over18.xhtml or under18.xhtml depending on the age entered on the form. A catchall of ‘error.xhtml’ is also added to the code.

Complete steps 12 to 14 in lab sheet 2

JSF validation

Validation

- There are a number of ways to implement validation in JSF. We will look at three of these:
  - Built-in validation components – defined in the XHTML file.
  - Custom validation components (defined as a method in a java program which implements the Validator interface)
  - Application level validation

Built in validators

- In JSF, the data type of an input field is defined by the corresponding data type in the managed bean.
  - For example, age is an integer in the managed bean, so only integer values will be accepted on the form.
- Additional, common validations are generally available through JSF tags. These tags must be nested within <h:inputText> <\h:inputText> tags. Below is a list of some validators from Netbeans:

Built in validators

- f:validateDoubleRange : Component's value must be numeric and must be in range specified by minimum and/or maximum parameter values.
- f:validateLongRange : Component's value must be numeric and convertible to long; must be in range specified by minimum and/or maximum parameter values.
- f:validateLength: Type must be string; length must be in range specified by minimum and/or maximum parameter values.
- f:validateRegex: validates against a specified regular expression.

Built in validators -Examples

```
<h:outputLabel value="Age" for="age"/>
<h:inputText id="age" label="Age" size="2"
value="#{registrationBean.age}">
    <f:validateLongRange minimum="16"
   maximum=’’120"/>
</h:inputText>
<h:message for="age" />
```

Ensures age will be between 16 and 120

```
<h:outputLabel value=”First Name" for=”firstname"/>
<h:inputText id="firstName"
   value="#{registrationBean.firstName}">
    <f:validateLength minimum="2"  maximum="25" />
</h:inputText>
<h:message for=”firstName" />
```

Limit first name to between 2 and 25 characters

Built in validators - examples

```
<h:outputLabel value=”Password" for=”password"/>
<h:inputText id="password" label="Password”
value="#{registrationBean.password}">
    <f:validateRegex pattern="((?=.*\d)(?=.*[a-z])(?
   =.*[A-Z])(?=.*[@#$%]).{6,20})" />
</h:inputText>
<h:message for="age" />
```

The above regex pattern requires a 6 to 20 character string with at least one digit, one upper case letter, one lower case letter and one special symbol (“@#$%”). More details are given in **Java Regular Expression Notation.docx** on Moodle.

Custom validation components

If validation requires more specific, custom written code, this can be implemented in a validator, which is a Java class that implements the Validator interface, and has one method called validate(). It typically starts as follows:

Used to reference this validator from a JSF page

```
@FacesValidator("emailValidator")
public class EmailValidator implements Validator {

    @Override
    public void validate(FacesContext facesContext,
            UIComponent uIComponent, Object value) throws
            ValidatorException {
        Pattern pattern = Pattern.compile("\\w+@\\w+\\.\\w+");
        Matcher matcher = pattern.matcher(
                (CharSequence) value);
```

Java REGEX syntax is defined here: http://docs.oracle.com/javase/tutorial/essential/regex/

Code is explained in the following slides . . . .

Customer validation components

- Each validator class will implement ONE validation only.
- The class is called by referencing the string defined in @FacesValidator:
  - @FacesValidator("emailValidator”)
- By implementing Validator, three parameters are expected as input, which are sent automatically by JSF:
  - FacesContext facesContext: recording state information on the JSF page
  - **UIComponent uIComponent**: the JSF component to be validated
  - **Object value**: the value entered by the user.

Customer validation components

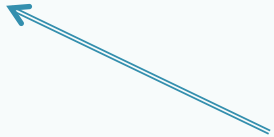
- Pattern represents the compiled form of a regular expression.
- It's method matcher creates a Matcher object that can compare a given input with the regular expression.
  - Calling matches() will do the comparison.

```
if (!matcher.matches()) {
        FacesMessage facesMessage =
                new FacesMessage(label
                + ": not a valid email address");

        throw new ValidatorException(facesMessage);
    }
```

Calling a validator from an XHTML page:

```
<h:outputLabel value="Email" for="email"/>
            <h:inputText id="email" label="Email" required="true"
value="#{registrationBean.email}">
                <f:validator validatorId="emailValidator"/>
            </h:inputText>
            <h:message for="email" />
```



Matches the text in @FacesValidator

- See labsheet 2 for more details and a worked example.

Application level validation

- Like the last example, application level validation is validation implemented in a bean method, but called from an action on the form (rather than an f:validator tag)
- It can be used to validate business logic, such as
  - Library: 'is the member over their borrowing limit'
  - Banking: 'Is the customer over their credit limit'

There is an example on the next slide . . .

Application level validation, implementing business rules

```
Public String register() {

FacesContext = context.getCurrentInstance();
if(StringUtils.isEmpty(studentForm.firstName)&&StringUtils.isEmpty
(studentForm.lastName) {
        FacesMessage message = new FacesMessage();
        message.setSeverity(FacesMessage.SEVERITY_ERROR);
        message.setSummary("blanks names")
        message.setDetail("Both firstname and last name are blank");
        context.addMessage("studentForm:firstName",message)
        return "error"
        }
return "success";
}
```

Get current state of the form

Check there is data entered for either firstname or lastname

Construct the error message

Link the error message to a component on the form

JSF xhtml tags

A full list of JSF tags are available at:
http://docs.oracle.com/cd/E17802_01/j2ee/javaee/javaserverfaces/1.2_MR1/docs/tlddocs/

And are explained with examples at:
http://www.exadel.com/tutorial/jsf/jsftags-guide.html

Tags are grouped as follows:
h: tags – xhtml components
f: tags – actions
ui: tags – defining templates
c: tags – logic such as if statements and loops
fn: tags – additional functions that can be applied to an input string (like the string functions in Excel)
composite: creating composite objects

Complete steps 15 to 18 in lab sheet 2

Facelets Templates

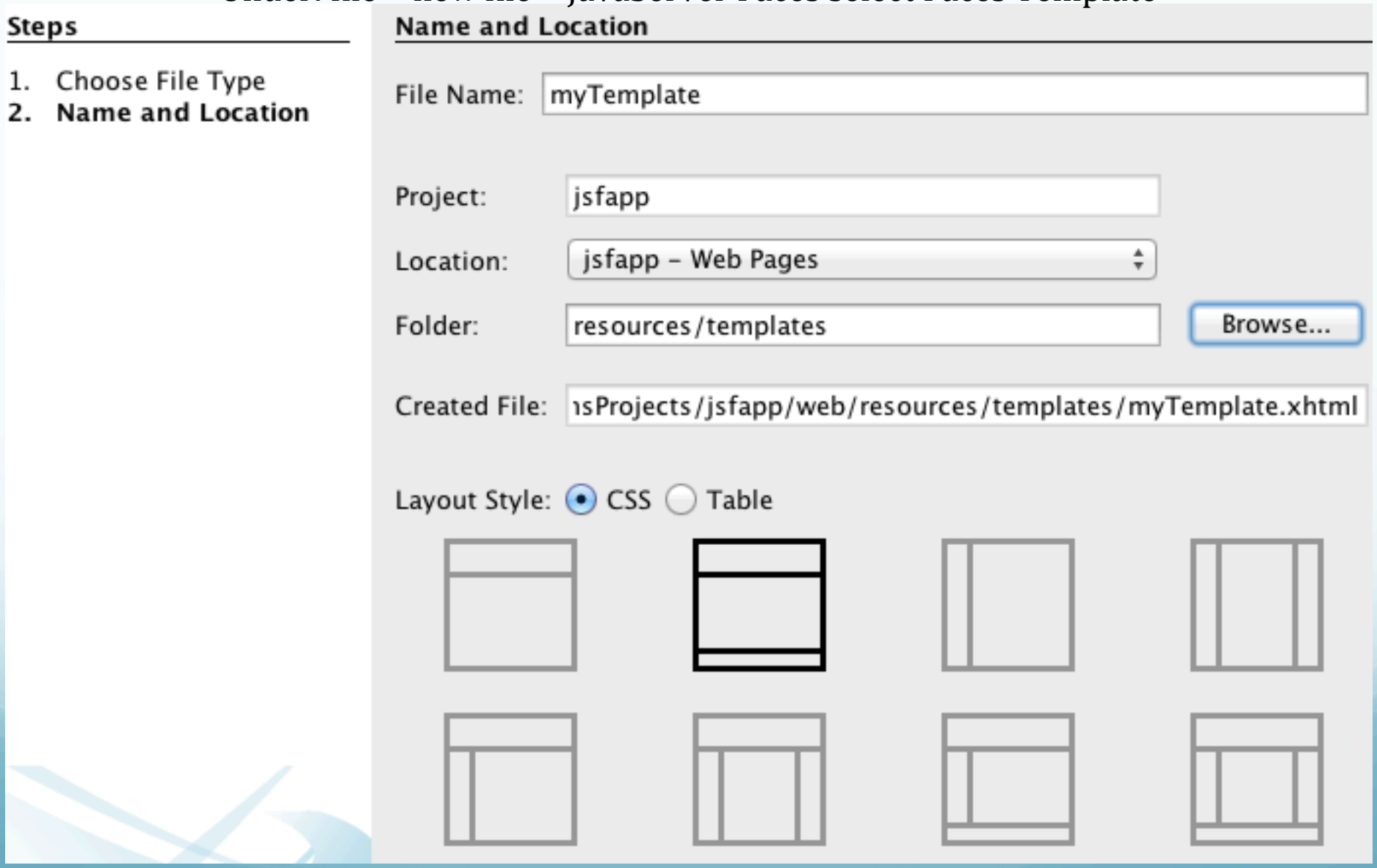
JSF pages can be created from a template to make consistency of design across pages easier.

Facelets Templates

- Facelet Templates (i.e. templates for JSF facelet pages) allow you to define a page layout for pages in a web app.
  - An application can have more than one template
- Content to be replicated on every page is coded ONCE in the template.
- Individual pages inherit code from the template, but change some or all of the inherited content.
- The are a number of predefined page layouts in Netbeans to use, from which you can base a your own template . . .

Creating a Facelets Templates in netbeans:

Under: file > new file > JavaServer Faces select Faces Template



Template example

```
?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets"
      xmlns:h="http://xmlns.jcp.org/jsf/html">

    <h:head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <link href="./resources/css/default.css" rel="stylesheet" type="text/css" />
        <link href="./resources/css/cssLayout.css" rel="stylesheet" type="text/css" />
        <title>My JSF application</title>
    </h:head>

    <h:body>
        <div id="top">
            <ui:insert name="top"><H1> Registration Application</H1></ui:insert>
        </div>

        <div id="content" class="center_content">
            <ui:insert name="content">put page content here</ui:insert>
        </div>

        <div id="bottom">
            <ui:insert name="bottom">Developed by G. Gray</ui:insert>
        </div>
    </h:body>
</html>
```

Using a template

- To use a template, rather than creating a JSF page, you need to create Facelets Template Client pages.
  - Under file > new file > JavaServer Faces select Faces Template Client.

- This creates a file with each section from the template.
- To inherit a section, **DELETE** it from the client page.
- To override a section, **LEAVE** it in the client page.

Using a template

- For example, the following template client will use the top and bottom sections from the template but replace the middle content section:

```
<body>
        <ui:composition template="./resources/templates/
                                lab2Template.xhtml">

            <ui:define name="content">
        add new content here for mid section of the webpage
            </ui:define>
        </ui:composition>
 </body>
```

Summary

- JSF architecture
- The life cycle of a JSF page
- JSF navigation
  - Static – hard coded in XHTML
  - Dynamic – controlled in a Java program
- JSF validation
  - Built in
  - custom
- Look online at the definition for all JSF XHTML tags
- Facelets Templates: Design page layout ONCE

Complete lab sheet 2