# Procesos de la Ingeniería Software
# Tema 4

## *Soporte Java para construcción de aplicaciones empresariales*

## *4. Capa de presentación web en Jakarta EE: Jakarta Faces*

# Bibliografía

- Básica
  - Oracle and affiliates (2021): The Jakarta EE Tutorial (Release 9.1)
    - Parte III: The Web Tier (Capítulos 6 – 16)
- Complementaria
  - Antonio Goncalvez (2013): Beginning Java EE 7, Apress
    - Capítulo 10 - 11

# Aplicaciones web

- Una aplicación web es una extensión dinámica de la funcionalidad de un servidor web
  - Cada aplicación se asocia a un (o un conjunto de) URL
  - Carga estática o dinámica

- Dos tipos de aplicaciones web:
  - Orientadas a presentación o interacción con personas
    - Páginas con contenido dinámico basadas en lenguajes de marcado que se visualizan en un navegador
  - Orientadas a servicios o interacción con máquinas
    - Invocación programática (sin interfaz gráfica)

- En las orientadas a presentación, dos estilos de programación:
  - Estilo Script: Manejan invocaciones/respuestas HTTP a bajo nivel
    - Procesan los mensajes HTTP de invocación y respuesta como Strings (cabecera y cuerpo)
    - Ej: CGI scripts, Java Servlets
  - Estilo Server Pages: Centrados en la generación de la página de retorno
    - En lugar de generar a mano el String de respuesta, se genera la página HTML de respuesta
    - Ej: PHP, ASP .NET , JSP, JSF

# Componentes de la capa web en Jakarta EE

- Los componentes web de Jakarta EE, encargados de añadir funcionalidad a servidores web, son:
  - **Servlets**
  - Páginas JSP (**Jakarta Server Pages**)
  - Facelets (**Jakarta Faces**, antiguo **JSF**)
  - Endpoints de servicios web
    - Servicios WSDL/SOAP: a través de **Jakarta XML Web Services** (antigua **JAX-WS**)
    - Servicios REST: a través de **Jakarta REST** (antigua **JAX-RS**)

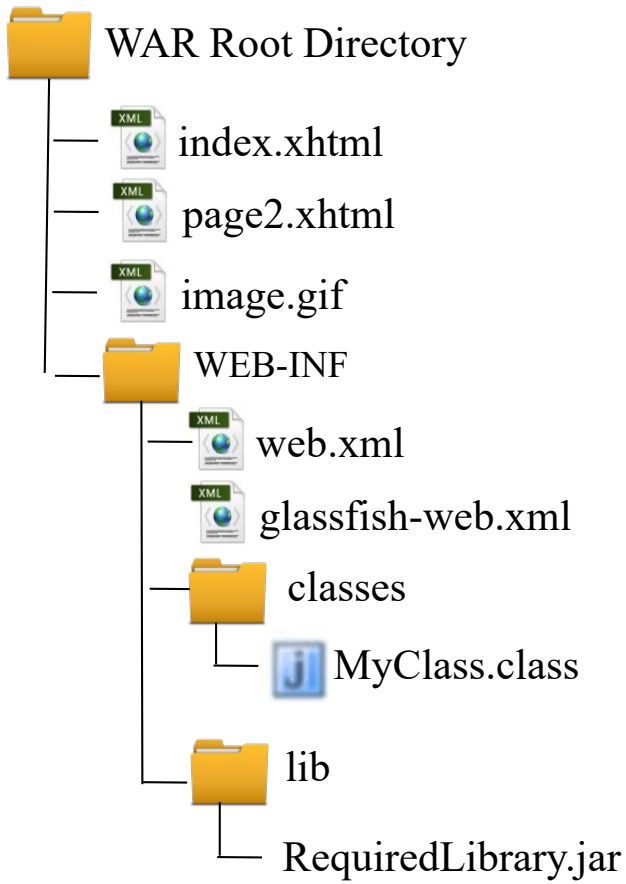
- Los componentes web de Jakarta EE son gestionados por el Web Container
  - El contenedor se encarga de los aspectos no funcionales
  - Los componentes web se encargan exclusivamente de generar las capas de presentación al usuario (basadas en HTML) o de implementar los servicios web
    - Usando para ello, en muchos casos, componentes de la capa de negocio (Enterprise beans)

- Los componentes web de Jakarta EE se distribuyen como módulos WAR
  - Desplegados independientemente
  - Desplegados dentro de una aplicación empresarial Jakarta EE (archivo EAR)

# Estructura de un componente web en Jakarta EE

- Un componente web Jakarta EE se compone de:
  - Recursos estáticos
    - Ficheros xhtml/html, imágenes, etc.
    - Almacenados en la raíz del archivo WAR
  - Clases de soporte
    - Responsables de:
      - generación de los contenidos dinámicos
      - enlace con la capa de negocio
      - navegación entre páginas
    - Almacenados en **WEB-INF/classes**
  - Librerías auxiliares requeridas
    - Almacenados en **WEB-INF/lib**
  - Descriptores de despliegue
    - **web.xml** (Obligatorio)
    - <appServer>-web.xml (específico de servidor y opcional)
    - Almacenados en la raíz del directorio **WEB-INF**

**Archivo WAR**



# Componentes web Jakarta EE

## Servlets

- Clase Java que procesa peticiones HTTP y genera respuestas HTTP (cabecera y cuerpo)
- La interfaz HTTPServlet proporciona una visión OO del lenguaje HTTP

```
public interface HTTPServlet {
     protected void doGet(HTTPServletRequest request, HTTPServletResponse response);
     protected void doPut(HTTPServletRequest request, HTTPServletResponse response);
     . . .
}
```

## Jakarta Server Pages (JSP)

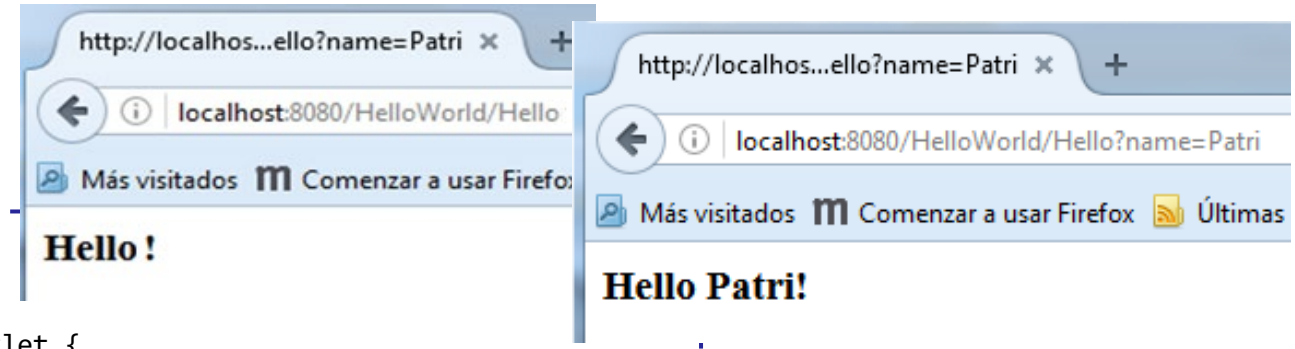
- Simplifican el desarrollo de páginas HTML dinámicas
- Una página JSP (.jsp) mezcla HTML puro con etiquetas JSP y código Java que permiten generar el contenido de forma dinámica
  - Sigue basándose en la interfaz HTTPServlet para gestionar las peticiones

## Jakarta Faces

- Evolución de JSP, para aplicaciones más interactivas y complejas
- Enfoque de modelo de componentes gráficos y eventos (similar a desarrollo de GUIs)
  - En lugar de basado en la petición HTTP
- Modo estándar de desarrollar aplicaciones web en Java desde Java EE 6

# Ejemplo de Servlet

- Aplicación web HelloWorld.war




```java
package myFirstServlet;

@WebServlet("/Hello")
public class HelloServlet extends HttpServlet {

    // Responde a una petición de tipo GET
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {

       // Extrae información de la petición
       String name= request.getParameter("name");

       // Cabecera ContentType de la respuesta
       response.setContentType("text/html");

      // Generación del cuerpo del mensaje HTTP de respuesta
       PrintWriter out = response.getWriter();
       out.print("<html><body>");
       out.print("<h3>Hello");
       if (name!=null)
           out.print(" "+name);
       out.print("!</h3>");
       out.print("</body></html>");
    }
}
```

El descriptor web.xml se puede usar en lugar de la anotación WebServlet

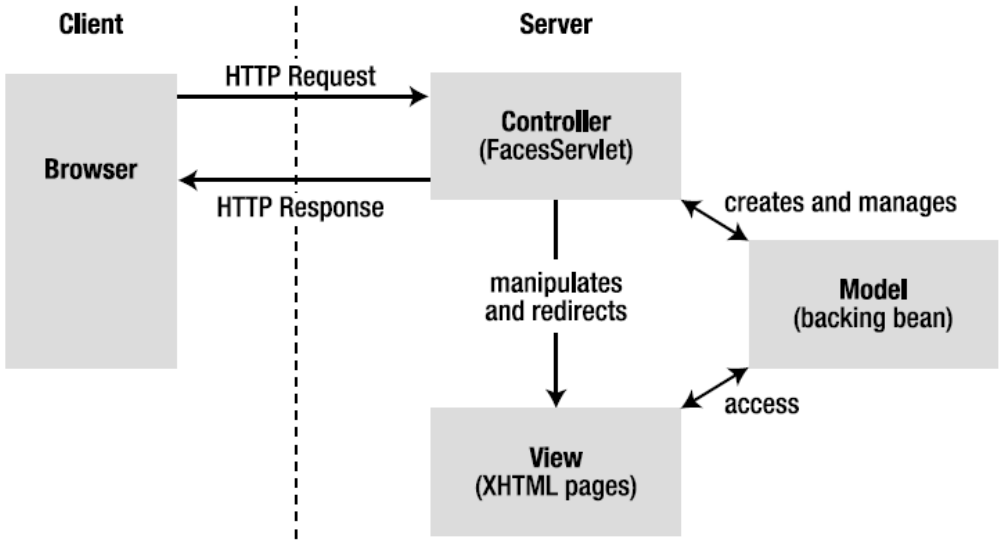
```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
         https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
         id="WebApp_ID" version="5.0">
  <servlet>
    <servlet-name>Hello</servlet-name>
    <servlet-class>myFirstServlet.HelloServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Hello</servlet-name>
    <url-pattern>/Hello.xhtml</url-pattern>
  </servlet-mapping>

</web-app>
```

# Jakarta Faces (antiguo JSF)

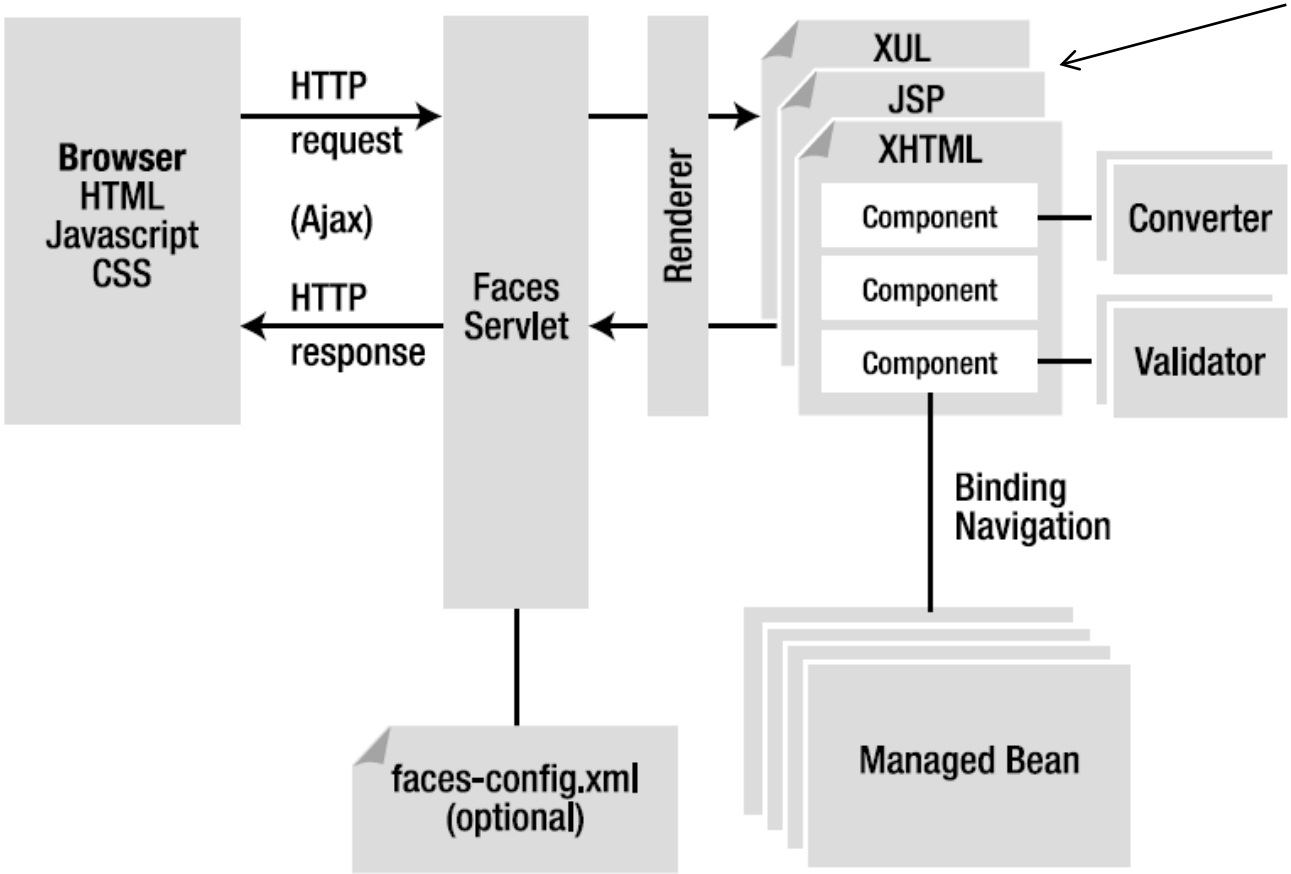
- Jakarta Faces es un framework de servidor para construir aplicaciones web Java
- Basado en el patrón **MVC**
  - Controlador: **FacesServlet**
    - Interno al contenedor web
    - Recoge todas las invocaciones HTTP y las redirige a la vista (facelet) adecuada
  - Vista: **Facelets** (Páginas .xhtml)
    - Intercambian información con el usuario
    - Conectan con el modelo
  - Modelo: **Backing Beans**
    - Implementan la lógica de negocio
      - Directamente (sin interacción con otros componentes Jakarta EE)
      - A través de invocaciones a Jakarta Enterprise Beans, servicios web, etc. (gateway)
    - Implementan la navegación entre las vistas




* Figura de Goncalves (2010)

- Implementación de referencia: Mojarra (incluida en Glassfish)

# JSF - Arquitectura




Soporta diferentes PDL (Page Definition Language) pero XHTML (Facelets) es el estándar

# JSF - Facelets

- Implementan el rol de "Vista" en JSF, es decir, la interfaz de usuario
- Son contenedores de componentes (no necesariamente gráficos) que sirven para la recogida y/o muestra de información
  - Todos los componentes extienden a `jakarta.faces.component.UIComponent`
  - Componentes estilo Swing
- Internamente, la estructura del Facelet se almacena en forma de árbol:
  - Elemento raíz de tipo `jakarta.faces.component.UIViewRoot`
  - Elementos anidados: Comandos, Entradas, Salidas, Selección, Tablas, Gráficos, etc.
- El Facelet debe ser "renderizado" a **XHTML** para ser enviado/recibido
  - XHTML = HTML con sintaxis XML estricta

# JSF - Backing Beans

- Implementan el rol de "Modelo" en JSF
  - Sirven de enlace entre los Facelets y la lógica de negocio de la aplicación
    - La lógica pueden implementarla directamente o a través de llamadas a beans, servicios web…
  - Gestionan la navegación entre Facelets
  - Mantienen los datos (entradas/salidas) que se manejan en los Facelets
    - Los datos se almacenan como atributos del bean
    - Los mismos beans/datos pueden ser compartidos por varios Facelets

- Un **Backing Bean** es una clase POJO:
  - Anotada como `@Named` (`jakarta.inject.Named`)
  - Con un alcance (scope) asociado (ver siguiente transparencia)
  - Con, al menos, un constructor público sin parámetros
  - Con patrón get/set para todos aquellos atributos que se quieran vincular a los Facelets
  - Con uno o varios **métodos de acción** públicos
    - Invocados desde los Facelets y encargados de ejecutar la lógica de negocio y la navegación entre Facelets
    - El valor de retorno debe ser siempre String

# JSF - Backing Beans – Alcance (Scope)

- El ciclo de vida de los objetos (atributos) que se crean dentro de un Backing Bean se puede configurar a través de su alcance o **scope**

- Los tipos de alcance mas habituales son:
  - Application (`@ApplicationScoped`)
    - Disponibles para todos los clientes de la aplicación web durante todo su ciclo de vida
  - Session (`@SessionScoped`)
    - Disponibles mientras se mantenga la sesión con el cliente
    - Se debe indicar programáticamente el cierre de sesión

      ```
      FacesContext.getCurrentInstance().getExternalContext().invalidateSession();
      ```
  - Request (`@RequestScoped`)
    - Disponibles mientras se atiende la petición
    - Valor por defecto

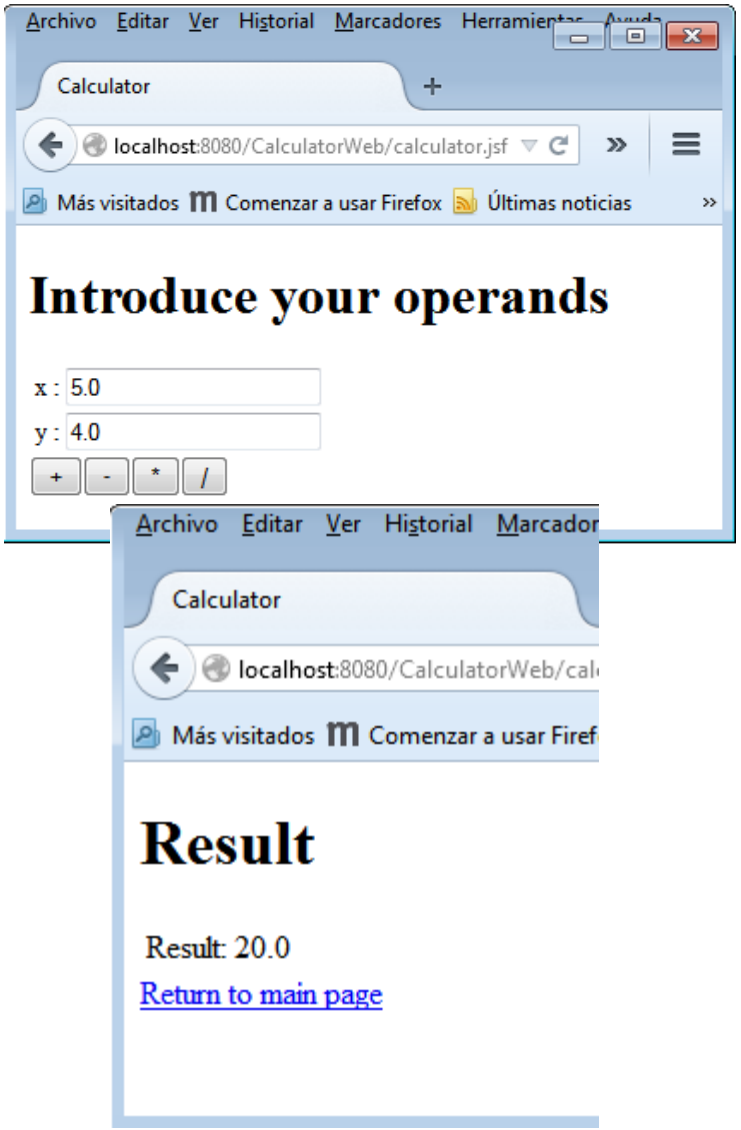
# JSF - Expression Language (EL)

- El **Expression Language** (EL) vincula los atributos y métodos de acción de los Backing Beans con los componentes de los Facelets

- Desde un componente de un Facelet se puede:
  - Acceder (lectura y escritura) a un atributo de un Backing Bean:
    - `#{backingBeanName.anAttribute}`
    - `#{backingBeanName.anAttribute.nestedAttribute}`
  - Invocar un método de un Backing Bean:
    - `#{backingBeanName.aMethod}`
    - `#{backingBeanName.aMethod(parametro)}`
  - Ejemplo: `<h:outputLabel value="#{bookBean.book.title}"/>`

- Por defecto, `<backingBeanName>` es el nombre de la clase `@Named` en minúscula
  - Se puede sobrescribir con el atributo `name` de la anotación `@Named`

# Ejemplo de aplicación JSF - Calculadora




- Aplicación que usa CalculatorBean (tema 4.2), que se encuentra ya desplegado en el servidor de aplicaciones y que implementa la siguiente interfaz:

```
public interface ICalculatorRemote {
  public double add(double i, double j);
  public double subtract(double i, double j);
  public double multiply(double i, double j);
  public double divide(double i, double j);
}
```

- La aplicación va a estar formada por:
  - Dos Facelets:
    - Página inicial => `calculator.xhtml`
      - Recoge datos de entrada
    - Página de resultados => `calculatorResult.xhtml`
      - Muestra resultado y permite volver al inicio
  - Un Backing Bean
    - `CalculatorMBean`
    - Enlace entre los Facelets y CalculatorBean (bean de negocio)

# Ejemplo de aplicación JSF - Calculadora - Facelets

## calculator.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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>
        <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1"/>
        <title>Calculator</title>
    </h:head>

    <h:body>
        <h1>Introduce your operands</h1>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="x : "/>
                <h:inputText value="#{calculatorMBean.op1}"/>
                <h:outputLabel value="y :"/>
                <h:inputText value="#{calculatorMBean.op2}"/>
            </h:panelGrid>
            <h:commandButton value="+"action="#{calculatorMBean.add}"/>
            <h:commandButton value="-"action="#{calculatorMBean.subtract}"/>
            <h:commandButton value="*"action="#{calculatorMBean.multiply}"/>
            <h:commandButton value="/"action="#{calculatorMBean.divide}"/>
        </h:form>
    </h:body>
</html>
```

Atributos op1 y op2 de CalculatorMBean

Métodos de CalculatorMBean

# Ejemplo de aplicación JSF – Calculadora - Facelets (2)

## calculatorResult.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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>
         <meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1" />
         <title>Calculator</title>
    </h:head>

    <h:body>
        <h1>Result</h1>
        <h:form>
            <h:panelGrid columns="2">
                <h:outputLabel value="Result : "/>
                <h:outputText value="#{calculatorMBean.result}"/>
             </h:panelGrid>

             <h:link value="Return to main page" outcome="calculator.xhtml" />
        </h:form>
    </h:body>

</html>
```

Atributo result de CalculatorMBean

Navegación a vista inicial

# Ejemplo de aplicación JSF - Calculadora – Backing Bean

Anotaciones @Named y @RequestScoped

```
import jakarta.ejb.EJB;
import jakarta.inject.Named; import jakarta.enterprise.context.RequestScoped;
import es.unican.ps.calculator.ICalculatorRemote;

@Named
@RequestScoped
public class CalculatorMBean {
     @EJB
     private ICalculatorRemote myCalculator;

     // Valores vinculados a los Facelets
     private double op1;
     private double op2;
     private double result;

     //getters y setters de los atributos op1, op2 y result

    // Métodos de acción
     public String add()  {
            result = myCalculator.add(op1, op2);
            return "calculatorResult.xhtml";
     }

     public String subtract()  {
            result=  myCalculator.subtract(op1, op2);
            return "calculatorResult.xhtml";
     }   ...
}
```

Inyección de Beans de la capa de negocio

Métodos de acción (lógica de negocio y navegación entre páginas)

# Descriptor web.xml de aplicaciones JSF

Página principal de la aplicación (opcional)

Se declara el Faces Servlet y se mapea a la clase que lo implementa (interna) (Igual en todas las aplicaciones)

Indica que FacesServlet responde a las peticiones con extensión .xhtml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="https://jakarta.ee/xml/ns/jakartaee"
         xsi:schemaLocation="https://jakarta.ee/xml/ns/jakartaee
                             https://jakarta.ee/xml/ns/jakartaee/web-app_5_0.xsd"
         id="WebApp_ID" version="5.0">

  <display-name>CalculadoraJSF</display-name>
  <!- Opcional -->
  <welcome-file-list>
      <welcome-file>calculator.xhtml</welcome-file>
  </welcome-file-list>

  <!- Este bloque es igual en todas las aplicaciones-->
  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>jakarta.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

  <servlet-mapping>
    <servlet-name>Faces Servlet</servlet-name>
    <url-pattern>*.xhtml</url-pattern>
  </servlet-mapping>

</web-app>
```

# Ciclo de vida de una página JSF

# Facelets - Librerías de tags JSF

| URI | Common Prefix | Description |
|---|---|---|
| http://xmlns.jcp.org/jsf/html | `h` | This tag library contains components and their HTML renderers (`h:commandButton`, `h:commandLink`, `h:inputText`, etc.). |
| http://xmlns.jcp.org/jsf/core | `f` | This library contains custom actions that are independent of any particular rendering (`f:selectItem`, `f:validateLength`, `f:convertNumber`, etc.). |
| http://xmlns.jcp.org/jsf/facelets | `ui` | Tags in this library add templating support. |
| http://xmlns.jcp.org/jsf/composite | `composite` | This tag library is used for declaring and defining composite components. |

# Facelets - Componentes básicos

- **Comandos**: Controles que pueden lanzar acciones (invocación de métodos en un Managed Bean o navegación entre páginas) enviando un mensaje HTTP POST
  - Botón: `<h:commandButton>`
    - `<h:commandButton value="A submit button" action="#{aManagedBean.aMethod}"/>`
    - `<h:commandButton image="anImage.gif" action="#{aManagedBean.aMethod}"/>`
  - Hiperenlace: `<h:commandLink>`
    - `<h:commandLink action="#{aManagedBean.aMethod}">Submit</h:commandLink>`

- **Targets:** Controles que permiten acceder a otras páginas enviando un HTTP GET
  - Botón: `<h:button>`
    - `<h:button value="Return to home" outcome="index.xhtml"/>`
  - Hiperenlace: `<h:link>`
    - `<h:link value="Back" outcome="index.xhtml"/>`

- **Imágenes**: Permiten añadir imágenes a la página
    - `<h:graphicImage id="mapImage" url="/template/world.jpg"/>`

# Facelets - Componentes básicos (2)

- **Entradas**: Permiten introducir información
  - Entrada de texto simple : `<h:inputText>`
    - ```
      <h:inputText value="Introduce tu edad" size="50" required="true"
                   requiredMessage="Debes introducir una edad"/>
      ```
  - Area de texto: `<h:inputTextArea>`
    - ```
      <h:inputTextArea value="Introduce un texto" rows="5" cols="20"/>
      ```
  - Entrada de password: `<h:inputSecret>`
    - ```
      <h:inputSecret value="Contraseña" maxlength="10"/>
      ```

- **Salidas**: Permiten mostrar información (no modificable)
  - Etiqueta: `<h:outputLabel>`
    - ```
      <h:outputLabel value="#{myManagedBean.attribute}"/>
      ```
  - Hiperenlace: `<h:outputLink>`
    - ```
      <h:outputLink value="www.example.org">An external link</h:outputLink>
      ```
  - Texto simple: `<h:outputText>`
    - ```
      <h:outputText value="#{myManagedBean.attribute}"/>
      ```
  - Texto parametrizado: `<h:outputFormat>`
    - ```
      <h:outputFormat value="Hello {0}!">
        <f:param value="#{userBean.Name}>
      </h:outputFormat>
      ```
      ```

# Facelets - Componentes básicos (3)

- **Selección**: Permiten seleccionar un elemento (o varios) dentro de un conjunto o lista
  - <h:selectBooleanCheckBox>, <h:selectOneRadio>, <h:selectOneMenu>, <h:selectOneListBox>
  - <h:selectManyCheckBox>, <h:selectManyMenu>, <h:selectManyListBox>
    - `<h:selectOneListBox value="#{asignaturasBean.asignatura}">`

```
<f:selectItem
   itemLabel="History"
   itemValue="asignaturasBean.History"/>
<f:selectItem
   itemLabel="Biography"
   itemValue="asignaturasBean.Biography"/>
</h:selectOneMenu>
```

| Tag | Rendering |
|---|---|
| h:selectBooleanCheckbox | ☐ |
| h:selectManyCheckbox | ☐ History ☐ Biography ☐ Literature ☐ Comics ☐ Child ☐ Scifi |
| h:selectManyListbox | History, Biography, Literature, Comics, Child, Scifi |
| h:selectManyMenu | History |
| h:selectOneListbox | History, Biography, Literature, Comics, Child, Scifi |
| h:selectOneMenu | History |
| h:selectOneRadio | ○ History ○ Biography ○ Literature ○ Comics ○ Child ○ Scifi |

# Facelets - Componentes básicos (4)

- **Tabla de datos**: `<h:dataTable>`
  - Muestra todos los elementos de una colección en forma tabular
  - Cada elemento de la colección en una fila, con tantas columnas como se configure

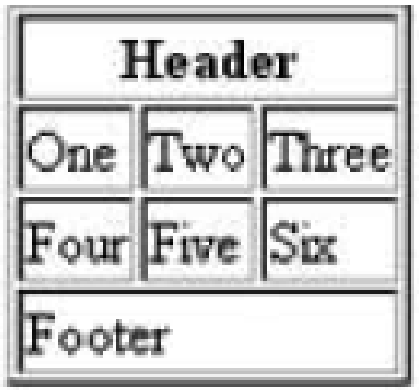
```
<h:dataTable value="#{curso.alumnos}" var="alumno" border="0" cellpadding="4"
             cellspacing="0" rules="all" style="border:solid 1px">
   <h:column>
        <f:facet name="header">
              <h:outputText value="Nombre"/>
        </f:facet>
        <h:outputText value="#{alumno.nombre}"/>
   </h:column>

   <h:column>
         <f:facet name="header">
              <h:outputText value="DNI"/>
         </f:facet>
        <h:outputText value="#{alumno.dni}"/>
   </h:column>
</h:dataTable>
```

# Facelets - Componentes básicos (5)

- **Grid:** `<h:panelGrid>`
  - Organiza los componentes en filas y columnas
  - Se pueden agrupar componentes internamente con `<h:panelGroup>`

```
<h:panelGrid columns="3" border="1">
   <f:facet name="header">
     <h:outputText value="Header"/>
   </f:facet>
   <h:outputLabel value="One"/>
   <h:outputLabel value="Two"/>
   <h:outputLabel value="Three"/>
   <h:outputLabel value="Four"/>
   <h:outputLabel value="Five"/>
   <h:outputLabel value="Six"/>
   <f:facet name="footer">
     <h:outputText value="Footer"/>
   </f:facet>
</h:panelGrid>
```



## Mensajes de error

- Mensaje asociado a un único componente `<h:message>`

```
<h:form id="usuarioForm">
   <h:outputLabel value="Edad">
   <h:inputText id="edad" required="true" requiredMessage="Debes introducir una edad"/>
   <h:message for="edad" style="color:red"/>
   <h:outputLabel value="Nombre">
   <h:inputText id="edad" required="true"/>
   <h:message for="nombre" style="color:red"/>
</h:form>
```

Edad [ ] Edad: Validation Error: Debes introducir una edad

Nombre [ ] Nombre: Validation Error: Nombre required

- Mensaje global `<h:messages>`

```
<h:form id="usuarioForm">
   <h:outputLabel value="Edad">
   <h:inputText id="edad" required="true" requiredMessage="Debes introducir una edad"/>
   <h:outputLabel value="Nombre">
   <h:inputText id="edad" required="true"/>
   <h:messages style="color:red"/>
</h:form>
```

Edad [ ]

Nombre [ ]

Edad: Validation Error: Debes introducir una edad

Nombre: Validation Error: Nombre required

# Gestión de mensajes de error desde Backing Beans

- Para gestionar la generación de mensajes de error en los Facelets desde los Backing Beans, se puede usar el método **addMessage** de la clase FacesContext
  - `void addMessage(String clientId, FacesMessage message)`
  - `FacesMessage(Severity severity, String summary, String detail)`

- Mismo ejemplo que en la transparencia anterior

```
<h:form id="usuarioForm">
   <h:outputLabel value="Edad">
   <h:inputText id="edad"
                value="#{userController.edad}"/>
   <h:message for="edad" style="color:red"/>
   <h:outputLabel value="Nombre" >
   <h:inputText id="nombre"
                value="#{userController.nombre}"/>
   <h:message for="nombre" style="color:red"/>
</h:form>
```

Edad [ ] Debes introducir una edad

Nombre [ ] Debes introducir un nombre

```
public class UserController {
   public String doCreaUsuario() {
      FacesContext ctx = FacesContext.getCurrentInstance();

      if (nombre== null || "".equals(nombre)) {
         ctx.addMessage("usuarioForm:nombre",
             new FacesMessage(FacesMessage.SEVERITY_WARN,
                 "Error", "Debes introducir un nombre"));
      }
      if (edad == null || "".equals(edad)) {
         ctx.addMessage("usuarioForm:edad",
            new FacesMessage(FacesMessage.SEVERITY_WARN,
                "Error", "Debes introducir una edad"));
      }
   ...
}
```

# Backing Beans - Inyección de propiedades

- A través de la anotación **@Inject** se pueden inicializar los atributos de un Backing Bean con valores procedentes de otros Backing Beans
  - Haciendo uso también del Expression Language

```
@Named
public class AlumnosBean {
     @Inject
     private CursosBean cursoBean;

     private String nombre;
     private String dni;
     private String nombreCurso;

     public AlumnosBean() {

     }

     @PostConstruct() {
     public void inicializaCurso {
         nombreCurso = cursoBean.nombre;
     }
}
```

# Conversión en aplicaciones JSF

- La conversión se aplica para convertir el String que el usuario introduce en la página al tipo/clase que corresponda (y viceversa)

- Mecanismos de conversión
  - Automática para tipos primitivos y enumerados
  - Conversión de números (a números, moneda o porcentajes)

```
<h:inputText value="#{aBean.factorConversion}">
        <f:convertNumber minFractionDigits="2"/>
</h:inputText>
```

  - Conversión de fechas

```
<h:inputText value="#{userBean.fechaNacimiento}">
        <f:convertDateTime pattern="DD/MM/YYYY"/>
</h:inputText>
```

  - Convertidores de usuario, para otro tipo de conversiones más complejas:
    - Definir una clase anotada como `@FacesConverter` que implemente la interfaz `Converter`:

```
public Object getAsObject(FacesContext ctx, UIComponent component, String value)
public String getAsString(FacesContext ctx, UIComponent component, Object value)
```

    - En el componente que queramos usar el convertidor, usar el atributo converter o el tag `<f:converter>`

# Conversiones de usuario - Ejemplo

## EuroConverter.java

```
@FacesConverter(value = "euroConverter")
public class EuroConverter implements Converter {

   @Override
   public Object getAsObject(FacesContext ctx, UIComponent comp, String value) {
      return value;
   }

   @Override
   public String getAsString(FacesContext ctx, UIComponent comp, Object value) {
      double amountInEuros =  Double.parseDouble(value.toString()) * 0.8;
      DecimalFormat df = new DecimalFormat("###,##0.##");
      return df.format(amountInEuros);
   }
}
```

## SomePage.xhtml

```
// in dollars
<h:outputText value="#{book.price}"/>

// in euros
<h:outputText value="#{book.price}">
      <f:converter converterId="euroConverter"/>
</h:outputText>
```

# Validación en páginas JSF

- La validación comprueba que los datos proporcionados por el usuario son válidos (antes de asignárselos al Backing Bean)
  - Se usan en conjunto con mensajes de error

- Mecanismos de validación
  - Validación de atributos no nulos => Atributos **required** y **requiredMessage** en XHTML

```
<h:inputText value="#{UserNumberBean.userNumber}" required="true"
             requiredMessage="El número de usuario no puede ser nulo">
<h:messages/>
```

  - Validadores estándar (uso conjunto con h:message)

```
<h:inputText id="userNo" value="#{UserNumberBean.userNumber}">
         <f:validateLongRange minimum="0" maximum="10"/>
</h:inputText>
<h:message style="color:red" for="userNo"/>
```

  - Validadores de usuario
    - Definir una clase anotada como @FacesValidator que implemente la interfaz Validator

```
public void validate(FacesContext context, UIComponent component, Object value)
```

    - En el componente que queramos usar el validador, usar el atributo validator o el tag <f:validator>

# Validador de usuario - Ejemplo

**ISBNValidator.java**

```java
@FacesValidator(value = "isbnValidator")
public class IsbnValidator implements Validator {
    private Pattern pattern;
    private Matcher matcher;

    @Override
    public void validate(FacesContext arg0, UIComponent arg1, Object value) throws ValidatorException {
       String componentValue = value.toString();
       pattern = Pattern.compile("(?=[-0-9xX]{13}$)");
       matcher = pattern.matcher(componentValue);
       if (!matcher.find()) {
           String message = MessageFormat.format("{0} is not a valid isbn format", componentValue);
           FacesMessage facesMessage = new FacesMessage(message, message);
           throw new ValidatorException(facesMessage);
       }
    }
}
```

**SomePage.xhtml**

```xml
<h:inputText value="#{book.isbn}" validator="isbnValidator"/>

<h:inputText value="#{book.isbn}">
          <f:validator validatorId="isbnValidator" />
</h:inputText>
```

# Plantillas en páginas JSF

- JSF permite definir una estructura (aspecto) común a todas las páginas de una aplicación mediante el uso de plantillas (**templates**)

- Mecanismo de definición de plantillas:
  1. Se define la plantilla como un fichero .xthml
     - A través de elementos **<ui:insert>** se definen áreas que deberán ser reemplazas por contenido específico en cada página

```
<ui:insert name="title">Default title</ui:insert>
```

  2. Se definen páginas .xhtml acordes a la plantilla
     - El cuerpo de la página se define como un elemento **<ui:composition>** acorde a una determinada plantilla

```
<ui:composition template="myTemplate.xhtml">
```

     - A través de elementos **<ui:define>** se rellenan los huecos <ui:insert> de la plantilla
       - `<ui:define name="title">Introduce your operands</ui:define>`

# Ejemplo Calculadora con plantillas

## layout.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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">

<head>
   <meta http-equiv="Content-Type"
         content="text/html; charset=ISO-8859-1" />
   <style type="text/css">
      body {font-size:14pt}
   </style>
   <title>Calculator</title>
</head>

<body>
   <h1 style="color:blue">
      <ui:insert name="title">Default title</ui:insert>
   </h1>

   <ui:insert name="content">Default content</ui:insert>

   <hr/>
   <h3 style="color:green">Procesos de la Ingeniería
     Software - Curso 2014/2015</h3>

</body>
</html>
```

## calculator.xhtml

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!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"
      xmlns:ui="http://xmlns.jcp.org/jsf/facelets">

   <ui:composition template="WEB-INF/templates/layout.xhtml">

      <ui:define name="title">Introduce your operands</ui:define>

      <ui:define name="content">
         <h:form>
            <h:panelGrid columns="2">
               <h:outputLabel value="x : " />
               <h:inputText value="#{calculatorMBean.op1}" />

               <h:outputLabel value="y :" />
               <h:inputText value="#{calculatorMBean.op2}" />
             </h:panelGrid>
             <h:commandButton value="+"
                              action="#{calculatorMBean.add}" />
             <h:commandButton value="-"
                              action="#{calculatorMBean.subtract}" />
             <h:commandButton value="*"
                              action="#{calculatorMBean.multiply}" />
             <h:commandButton value="/"
                              action="#{calculatorMBean.divide}"
         </h:form>
      </ui:define>
   </ui:composition>
</html>
```