Procesos de la Ingeniería Software
Tema 4

Soporte Java para construcción de aplicaciones empresariales

4. Capa de presentación en Java EE: Java Server Faces

Bibliografía

- Lectura obligada
  - Antonio Goncalvez (2013): Beginning Java EE 7, Apress
    - Capítulo 10 - 11

- Lectura complementaria
  - Erik Jendrock et al. (2014): The Java EE 7 Tutorial
    - Capítulos 6 - 13

Aplicaciones web y servidores web

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

- Dos tipos de aplicaciones web:
  - Orientadas a presentación (interacción con personas): Páginas con contenido dinámico usando lenguajes de marcado
  - Orientadas a servicios (interacción con máquinas): Servicios web

- Dos estilos de programación:
  - Estilo Script: Programas que manejan invocaciones/respuestas HTTP a bajo nivel
    - Procesan los mensajes HTTP de invocación y respuesta como Strings
    - Ej: CGI scripts, Java Servlets
  - Estilo Server Pages: Programas centrados en la página de retorno
    - En lugar de escribir a mano el String, se genera la página HTML de respuesta
    - Ej: PHP, ASP .NET , JSP

Java EE Web Tier

- Los componentes Java EE de la capa web, encargados de añadir funcionalidad dinámica a los servidores web, son:
  - **Servlets**
  - Páginas JSP (**Java Server Pages**)
  - Páginas JSF (**Java Server Faces**)
  - Endpoints de Servicios web (clases desarrolladas con **JAX-WS** y **JAX-RS**)

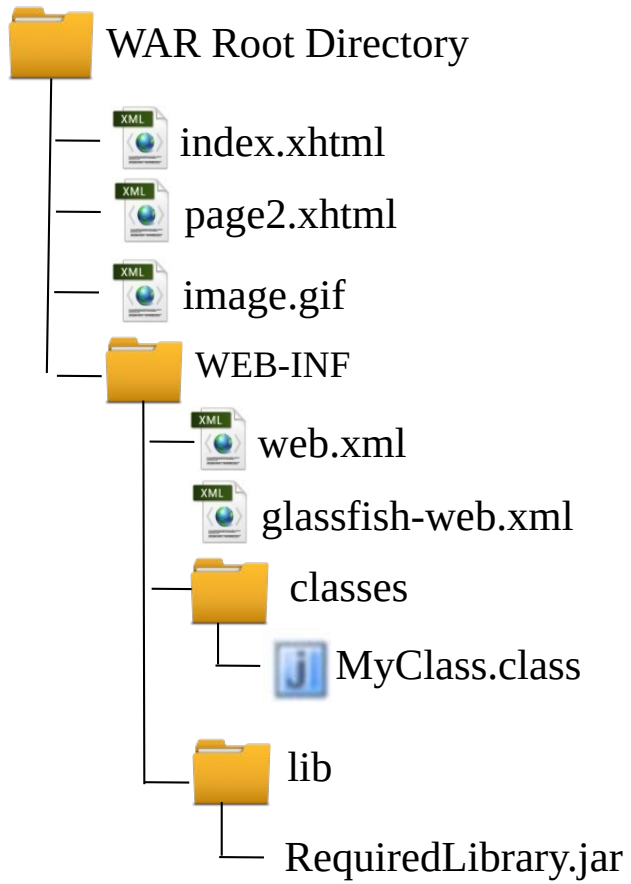
- Los componentes web de Java EE son gestionados por el contenedor web (Web Container):
  - El contenedor se encarga de los aspectos no funcionales
  - Los componentes web se encargan exclusivamente de generar las capas de presentación al usuario (basadas en HTML) o de implementar los servicios web
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

- Los componentes web de Java EE se distribuyen como:
  - Módulos Java EE desplegables independientemente (archivo WAR)
  - Módulos Java EE dentro de una aplicación empresarial Java EE (archivo EAR)

Estructura de una aplicación web en Java EE

- Una aplicación web Java EE se compone de:
  - Recursos estáticos
    - Ficheros xhtml/html, imágenes, etc.
    - Se almacenan en la raíz del archivo WAR
  - Clases de soporte
    - Responsables de:
      - la generación de los contenidos dinámicos
      - el enlace con la capa de negocio
    - Se almacenan en el paquete WEB-INF/classes
  - Librerías requeridas
    - Se almacenan en el paquete WEB-INF/lib
  - Descriptores de despliegue
    - web.xml (genérico)
    - <appServer>-web.xml (específico de servidor)
    - Se almacenan en la raíz del directorio WEB-INF

Archivo WAR



Componentes web Java EE

Servlets

- Clase Java que procesa peticiones y genera respuestas HTTP
- La interfaz Servlet proporciona una visión OO del lenguaje HTTP

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

Páginas JSP

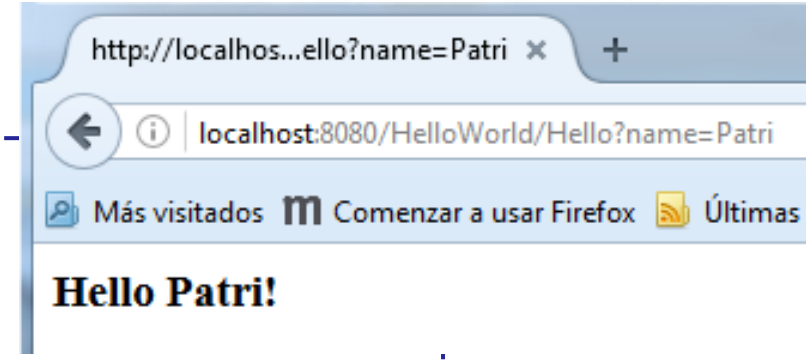
- Simplifican el desarrollo de páginas HTML dinámicas
- Basadas en HTML y generación dinámica de Servlets

Framework JSF

- Mejora de JSP
- Visión centrada en un modelo de componentes gráficos y eventos (similar a entornos de desarrollo de GUIs)
- Modo estándar de desarrollar aplicaciones web en Java EE (desde Java EE 6)

Ejemplo de Servlet

- Aplicación web HelloWorld



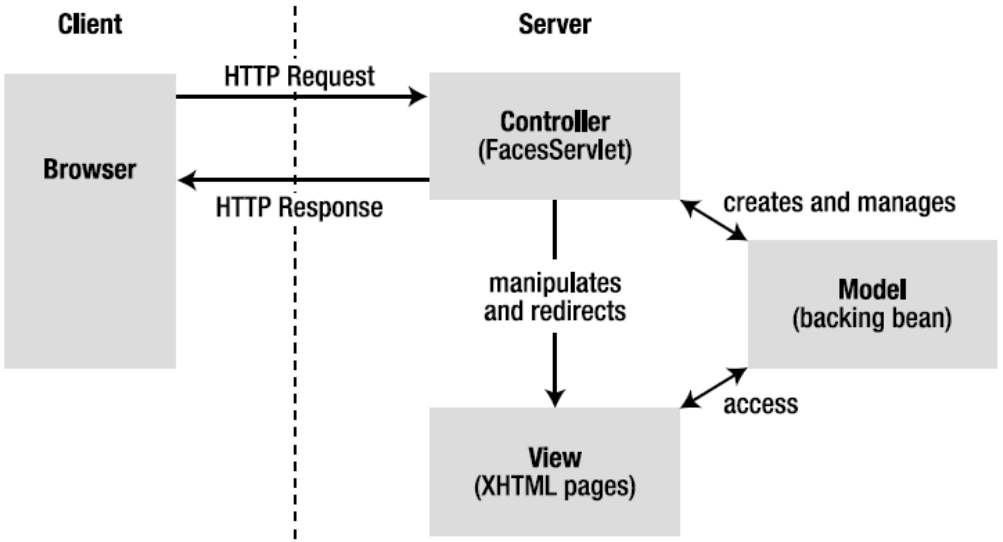

```java
package myFirstServlet;

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

    private static final long serialVersionUID = 1L;

    protected void doGet(HttpServletRequest request, HttpServletResponse response)
              throws ServletException, IOException {

       response.setContentType("text/html");
       PrintWriter out = response.getWriter();

       String name= request.getParameter("name");

       out.print("<html><body>");
       out.print("<h3>Hello");
       if (name!=null)
           out.print(" "+name);
       out.print("!</h3>");
       out.print("</body></html>");
    }
}
```

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
         http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         id="WebApp_ID" version="3.0">

  <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</url-pattern>
  </servlet-mapping>

</web-app>
```

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

JSF: Java Server Faces

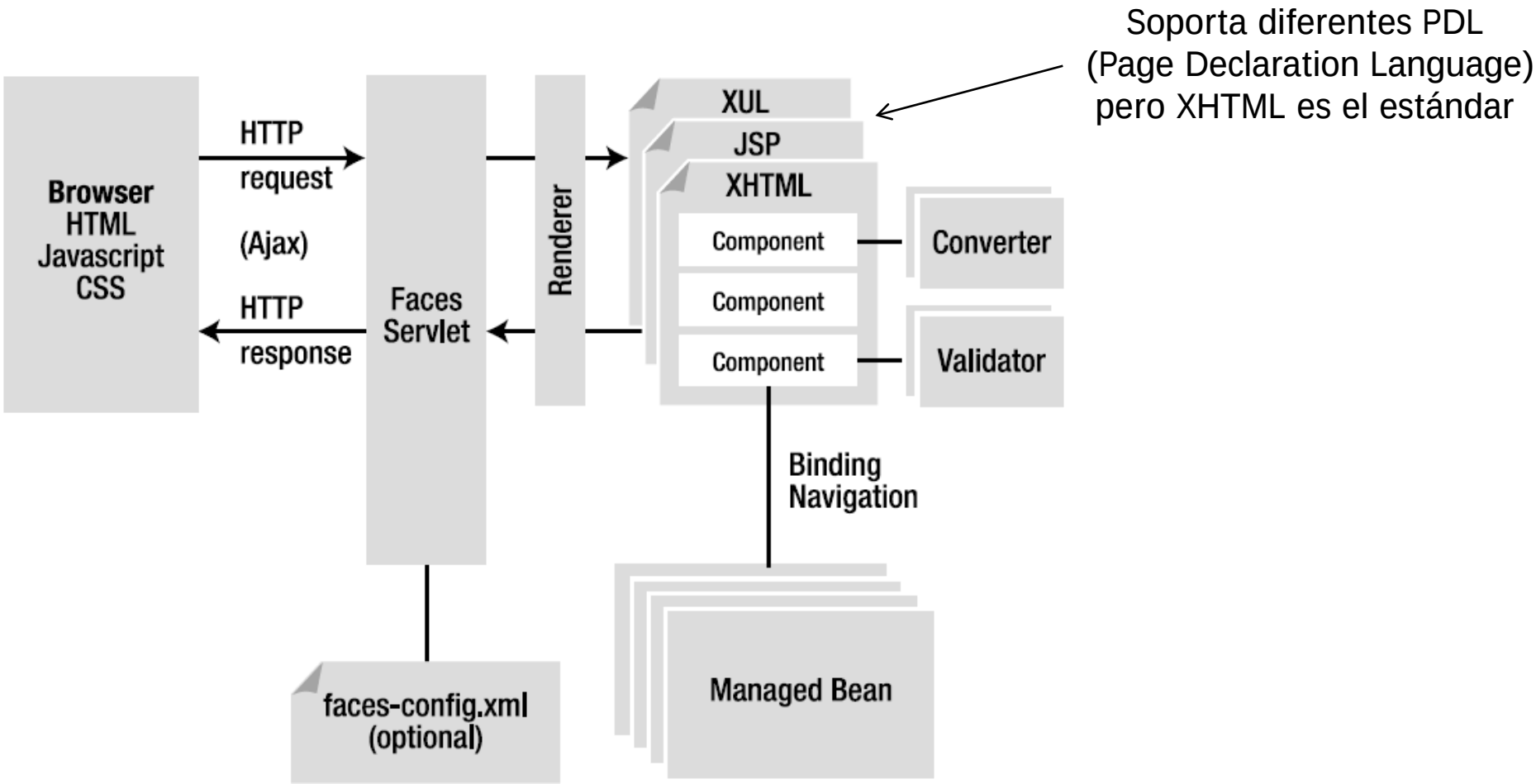
- JSF es un framework que implementa el patrón **MVC**
  - Vista: **Facelets** (Páginas xhtml)
    - Recogen/muestran información de/para el usuario y la conectan con el modelo
  - Controlador: **FacesServlet**
    - Interno al contenedor web
    - Recoge todas las invocaciones HTTP y las redirige al componente adecuado
  - Modelo: **Backing Beans**
    - Implementan la lógica de negocio
      - Directamente (no Java EE)
      - A través de invocaciones a EJBs, servicios web, etc. (Gateway)
    - Implementan la navegación entre los facelets



* Figura de Goncalves (2010)

- Implementación de referencia: Mojarra (incluida en Glassfish)

JSF: Otra visión de arquitectura



JSF: Facelets

- Implementan el rol de “vista” en el patrón MVC de JSF, es decir, la interfaz de usuario en aplicaciones web Java EE
- Se desarrollan en lenguaje **xhtml**
  - XHTML = HTML con sintaxis XML estricta
- Son contenedores de componentes (no necesariamente gráficos) que sirven para la recogida y/o muestra información
  - Estilo SWING
  - Todos los componentes extienden a javax.faces.component.UIComponent
- Los componentes de un Facelet tienen estructura en árbol:
  - Elemento raíz de tipo javax.faces.component.UIViewRoot
  - Elementos anidados: Comandos, Entradas, Salidas, Selección, Tablas, Gráficos, etc.
- El Facelet es “renderizado” a HTML antes de enviarlo como respuesta

JSF: Backing Beans (Managed Beans)

- Implementan el rol de “modelo” en el patrón MVC de JSF
  - Sirven de enlace entre la vista y la lógica de negocio de la aplicación
    - La lógica pueden implementarla ellos directamente o a través de llamadas a la capa de negocio (EJBs, servicios web, etc.)
  - Gestionan la navegación entre Facelets
  - Mantienen los datos que se manejan en las vistas
    - Los datos se almacenan cómo atributos del bean
    - Los mismos beans/datos pueden ser compartidos por varias vistas o Facelets

- Un **Backing Bean** es una clase POJO:
  - Anotada como `@Named` (`javax.inject.Named`)
  - Con un scope asociado
  - Con, al menos, un constructor público sin parámetros
  - Con patrón get/set para todos aquellos atributos que se quieran vincular a los Facelets
  - Con **métodos de acción** públicos y retornando un String
    - Un método de acción es aquel que puede ser invocado desde un Facelet y que ejecuta lógica de negocio y navegación entre Facelets

JSF: Alcance (Scope) de Backing Beans

- El ciclo de vida de los objetos (atributos) que se crean dentro de un Managed Bean se puede configurar a través del scope
- Tipos de alcance (de mayor a menor):
  - 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();
      ```
  - View (@ViewScoped)
    - Disponibles mientras se está en la misma vista
  - Request (@RequestScoped)
    - Disponibles mientras se atiende la petición
    - Valor por defecto
  - Flash: Disponible temporalmente mientras se pasa de una vista a otra
  - Flow: (@FlowScoped) Disponibles mientras el usuario está en un flujo específico.

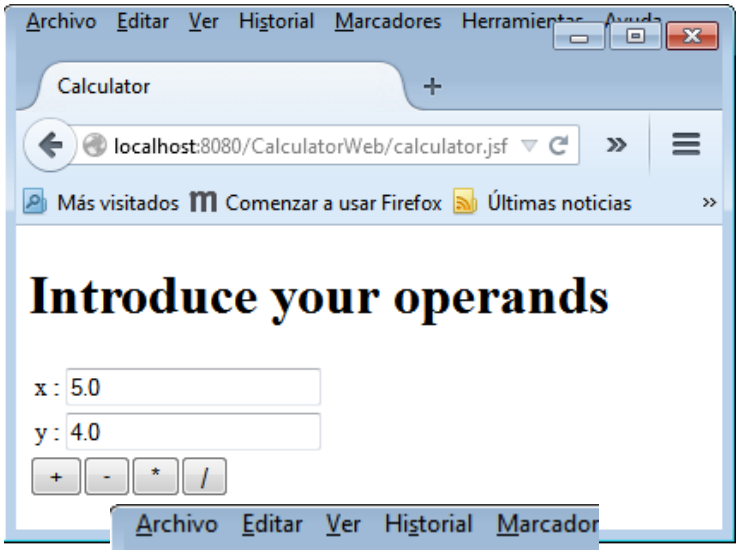
JSF: Expression Language (EL)

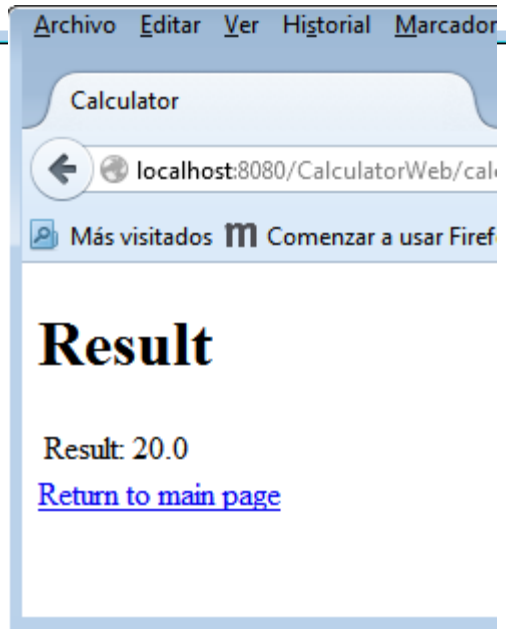
- El **Expression Language** (EL) permite vincular los atributos o métodos de acción de los Backing Beans a los componentes de las vistas (Facelets)

- 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 Managed bean:
    - `#{backingBeanName.aMethod}`
    - `#{backingBeanName.aMethod(parametro)}`
  - Ejemplo: `<h:outputLabel value="#{bookController.book.title}"/>`

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

Ejemplo de aplicación JSF: Calculadora





- Para implementar la aplicación se usa el EJB Calculator, que se encuentra ya desplegado en el servidor de aplicaciones y que implementa la siguiente interfaz:

```
public interface CalculatorEJBRemote {
   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
    - `CalculatorBean`
    - Enlace entre las páginas xhtml y entre las páginas y el EJB que implementa las operaciones de cálculo

Ejemplo de aplicación JSF: 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://java.sun.com/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="#{calculatorBean.op1}"/>
               <h:outputLabel value="y :"/>
              <h:inputText value="#{calculatorBean.op2}"/>
           </h:panelGrid>
           <h:commandButton value="+"action="#{calculatorBean.add}"/>
           <h:commandButton value="-"action="#{calculatorBean.subtract}"/>
           <h:commandButton value="*"action="#{calculatorBean.multiply}"/>
           <h:commandButton value="/"action="#{calculatorBean.divide}"/>
    </h:form>
</h:body>
</html>
```

Valores para los atributos op1 y op2 de CalculatorBean

Acciones sobre CalculatorBean

Ejemplo de aplicación JSF: Facelets

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://java.sun.com/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="#{calculatorBean.result}"/>
             </h:panelGrid>

             <h:commandLink value=“Return to Main Page“ action=“calculator.xhtml” />
        </h:form>
</h:body>
</html>
```

Valor del atributo result de CalculatorBean

Navegación a página principal (directa)

Ejemplo de aplicación JSF: CalculatorBean

Anotación @Named y @RequestScoped

```
import javax.ejb.EJB;
import javax.inject.Named; import javax.enterprise.context.RequestScoped;
import org.example.calculator.CalculatorEJBRemote;

@Named
@RequestScoped
public class CalculatorBean {
    @EJB
    private CalculatorEJBRemote 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 EJBs

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

Descriptor web.xml

```xml
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns="http://java.sun.com/xml/ns/javaee"
         xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
                             http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
         id="WebApp_ID" version="3.0">

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

  <servlet>
    <servlet-name>Faces Servlet</servlet-name>
    <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
  </servlet>

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

</web-app>
```

Página principal de la aplicación

Se declara el Faces Servlet y se mapea a la clase que lo implementa (interna)

Indica que FacesServlet responde a las peticiones con extensión .jsf (.xhtml lo proceso por defecto)

Descriptor faces-config.xml

- La navegación entre Facelets se puede centralizar en el descriptor faces-config.xml

```
@Named
@RequestScoped
public class CalculatorBean {
     @EJB
     private CalculatorEJBRemote myCalculator;

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


    // Métodos de acción
     public String add()  {
        result = myCalculator.add(op1, op2);
        return “success”;
     }

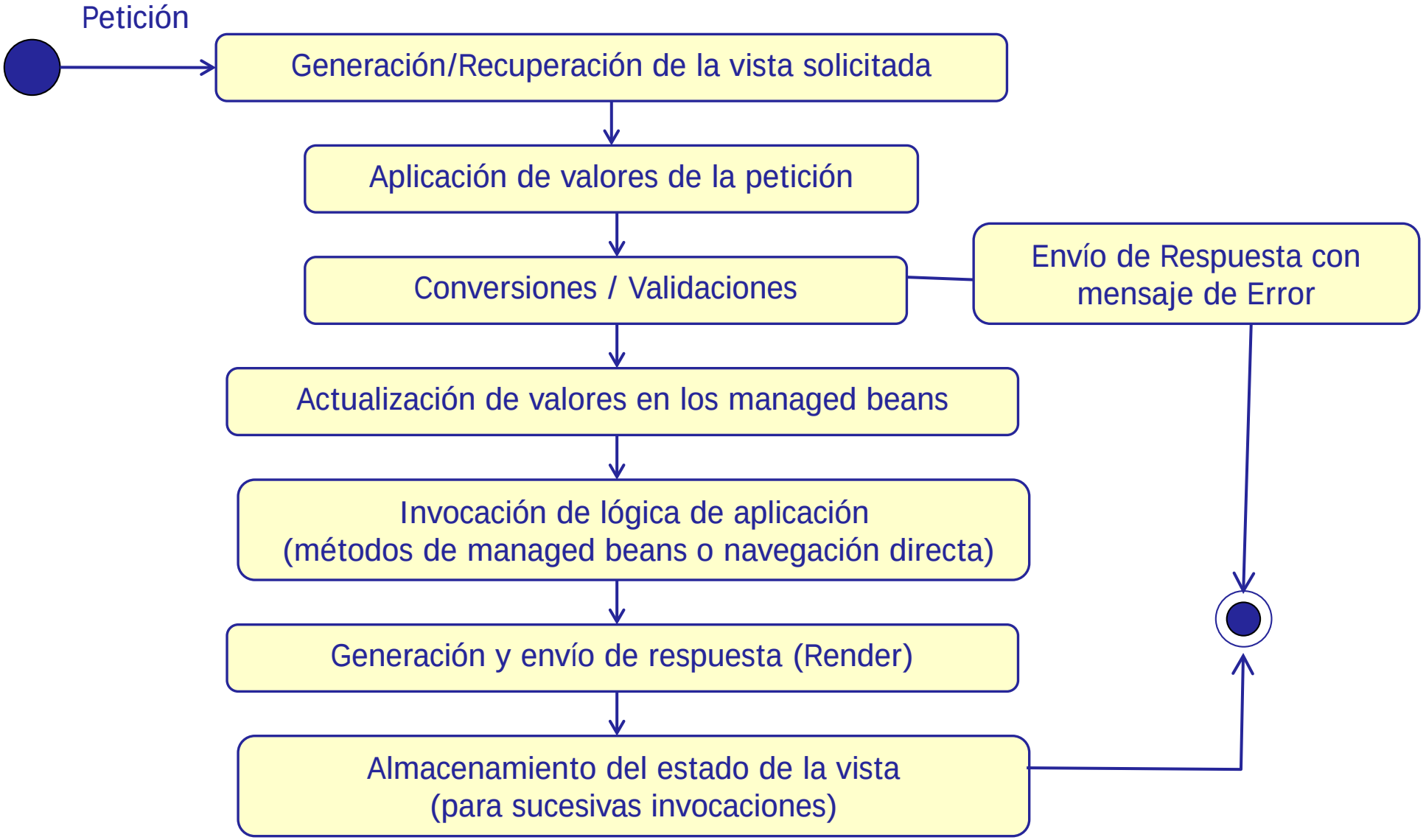
     public String subtract()  {
        result=  myCalculator.subtract(op1, op2);
        return “success";
     }
     ...
}
```

```
<?xml version="1.0" encoding="UTF-8"?>
<faces-config xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
    http://xmlns.jcp.org/xml/ns/javaee/web-facesconfig_2_2.xsd"
    id="WebApp_ID" version="3.0">

     <navigation-rule>
         <from-view-id>calculator.xhtml</from-view-id>
         <navigation-case>
             <from-outcome>success</from-outcome>
            <to-view-id>calculatorResult.xhtml</to-view-id>
         </navigation-case>
     </navigation-rule>

</faces-config>
```

Ciclo de vida de una página JSF



Facelets: Librerías de tags JSF

URI	Common Prefix	Description
`http://java.sun.com/jsf/html`	`h`	This tag library contains components and their HTML renderers (`h:commandButton`, `h:commandLink`, `h:inputText`, etc.).
`http://java.sun.com/jsf/core`	`f`	This library contains custom actions that are independent of any particular rendering (`f:selectItem`, `f:validateLength`, `f:convertNumber`, etc.).
`http://java.sun.com/jsf/facelets`	`ui`	Tags in this library add templating support.
`http://java.sun.com/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)
  - Botón: `<h:commandButton>`
    - `<h:commandButton value="A submit button“ action=“#{myManagedBean.method}”/>`
    - `<h:commandButton value=“Return to home“ action=“index.xhtml”/>`
  - Hiperenlace: `<h:commandLink>`
    - `<h:commandLink action=“#{myManagedBean.method}”>Submit</h:commandLink>`
    - `<h:commandLink value=“Return to home“ action=“index.xhtml”/>`

- **Entradas**: Permiten introducir/mostrar 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”/>`

Facelets: Componentes básicos

- **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>
      ```

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

Facelets: Componentes básicos

- **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

- **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

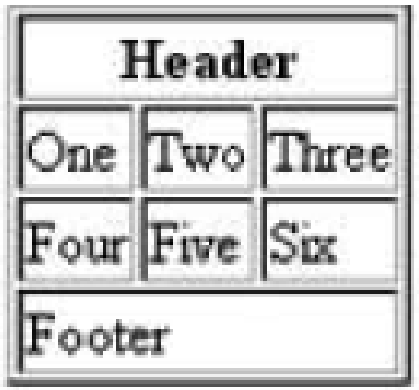
```
<h:dataTable value="#{curso.alumnos}" var=“alumno“ border="0" cellpadding="4"
             cellspacing="0“ rules="all" style="border:solid 1px">
   <h:column>
        <f:facet name="header">
              <h:outputText value="Nombre"/>
        </f:facet>
        <h:outputText value="#{alumno.nombre}"/>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

- **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>
```



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, monedas 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>`

Ejemplo de convertidor

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 Managed 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>

Ejemplo de validador

ISBNValidator.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

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

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

Plantillas en páginas JSF

- En general, todas las páginas web de una aplicación comparten una estructura común (cabecera, pies de página, etc.)

- JSF permite definir esta estructura común a través 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 después serán 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://java.sun.com/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://java.sun.com/jsf/html"
      xmlns:ui="http://java.sun.com/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="#{calculatorBean.op1}" />

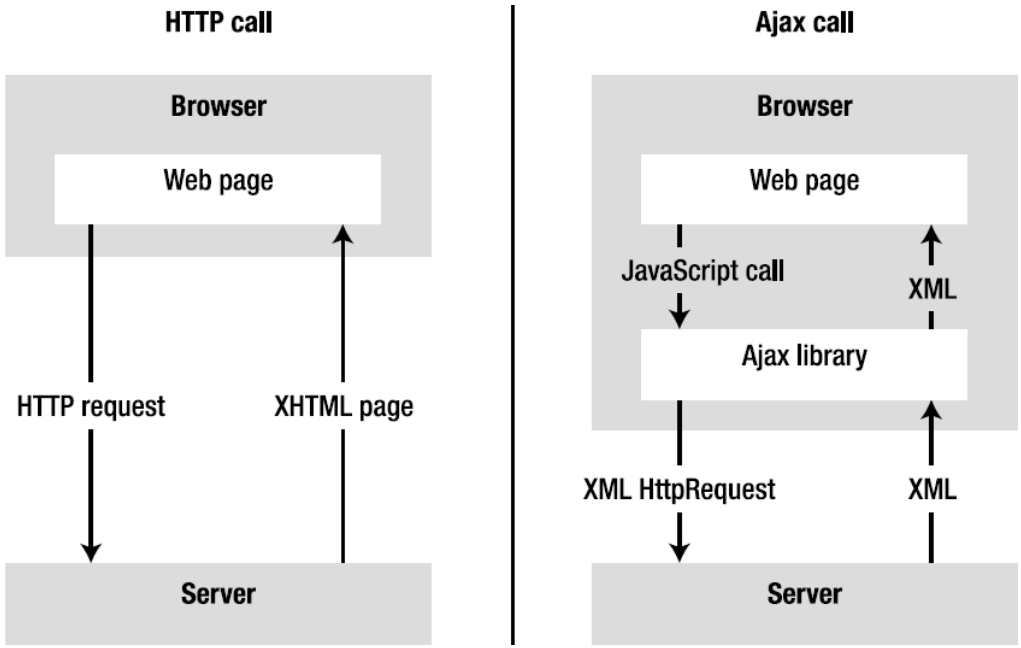
               <h:outputLabel value="y :" />
               <h:inputText value="#{calculatorBean.op2}" />
             </h:panelGrid>
             <h:commandButton value="+"
                              action="#{calculatorBean.add}" />
             <h:commandButton value="-"
                              action="#{calculatorBean.subtract}"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

Soporte para Ajax en JSF

- Ajax (Asynchronous JavaScript and XML) es una técnica de desarrollo web usada para crear aplicaciones web interactivas

- Permite a las aplicaciones web actualizar parte de sus datos, a través de interacciones asíncronas con el servidor
  - Evitando la recarga completa de la página

- Se basa en la utilización del objeto **XMLHttpRequest**



Soporte para Ajax en JSF

- JSF 2.0 proporciona soporte nativo para Ajax
  - No hace falta usar JavaScript ni usar directamente el objeto XMLHttpRequest, aunque sí una librería JavaScript predefinida

- La librería jsf.js proporciona una serie de funciones que permiten realizar las peticiones o envíos asíncronos al servidor

- El **modo** en que se realiza una petición Ajax en una página JSF es el siguiente:

```
<h:commandButton value="Create a book" action="#{bookController.doCreateBook}">
    <f:ajax execute="@form" render=":booklist"/>
</h:commandButton>
```

- Ver ejemplo Books (disponible en Moodle)