Especialista en Aplicaciones y Servicios Web con Java Enterprise

JSF Sesión 2

Funcionamiento JSF

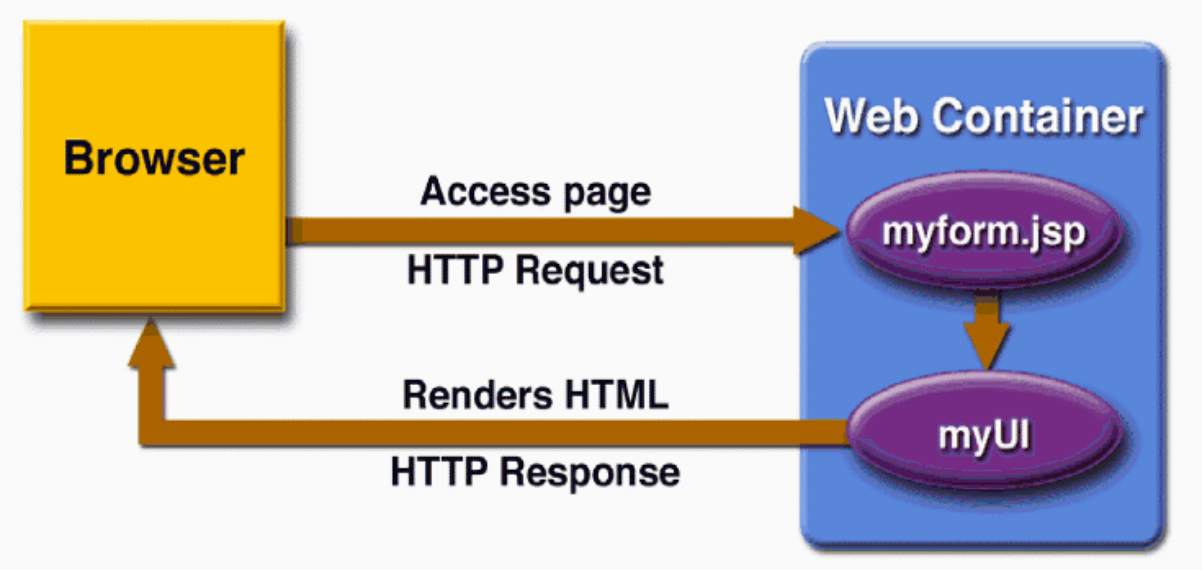
(y algún que otro extra ...)

Índice

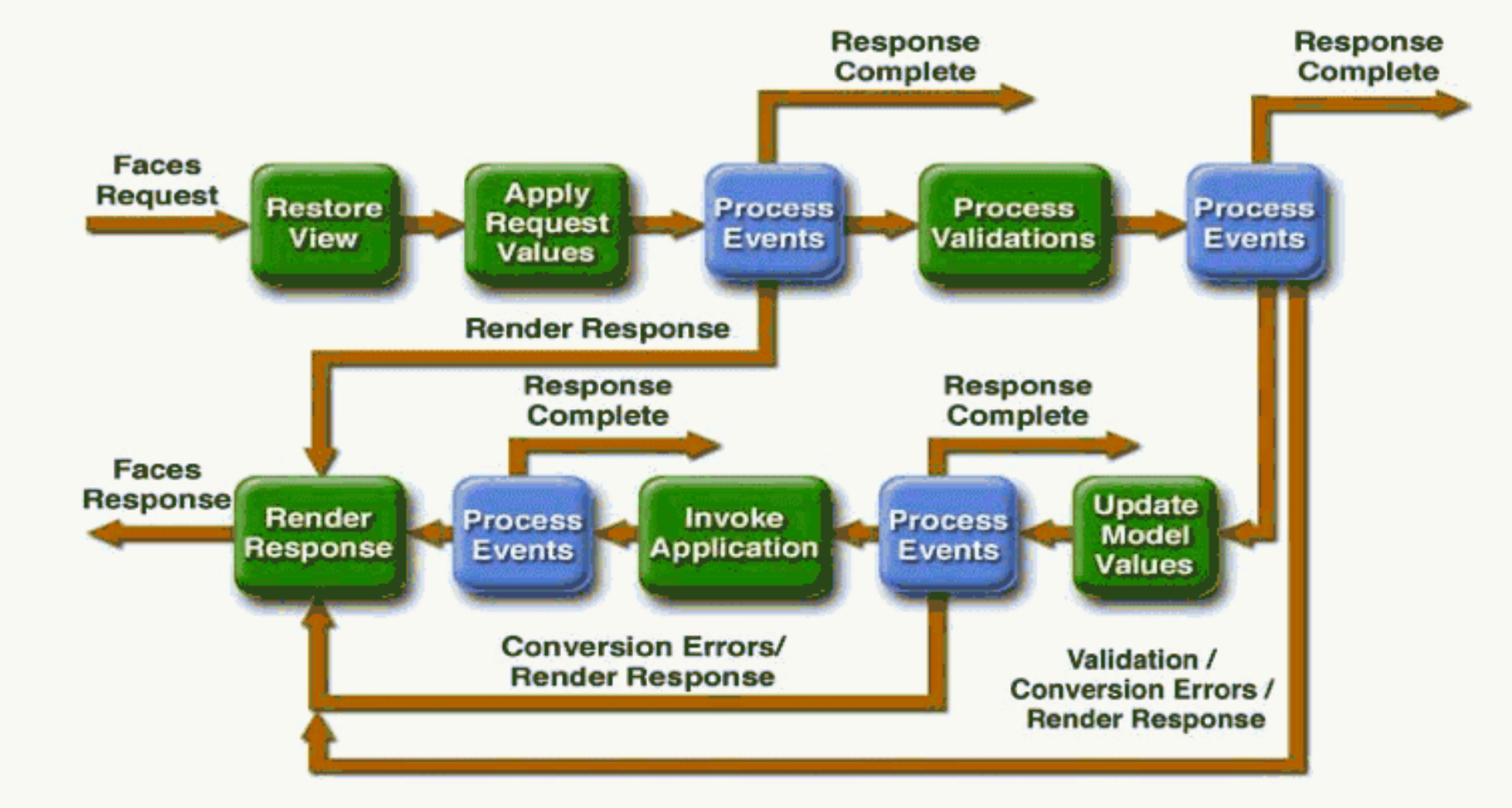
- El ciclo de vida de una petición
- Validaciones definidas en la aplicación
- Peticiones JSF
- Eventos en JSF

Componentes en el servidor

- Una diferencia fundamental entre JSF y Struts/JSP es que en JSF la vista reside en el servidor en forma de componentes JSF



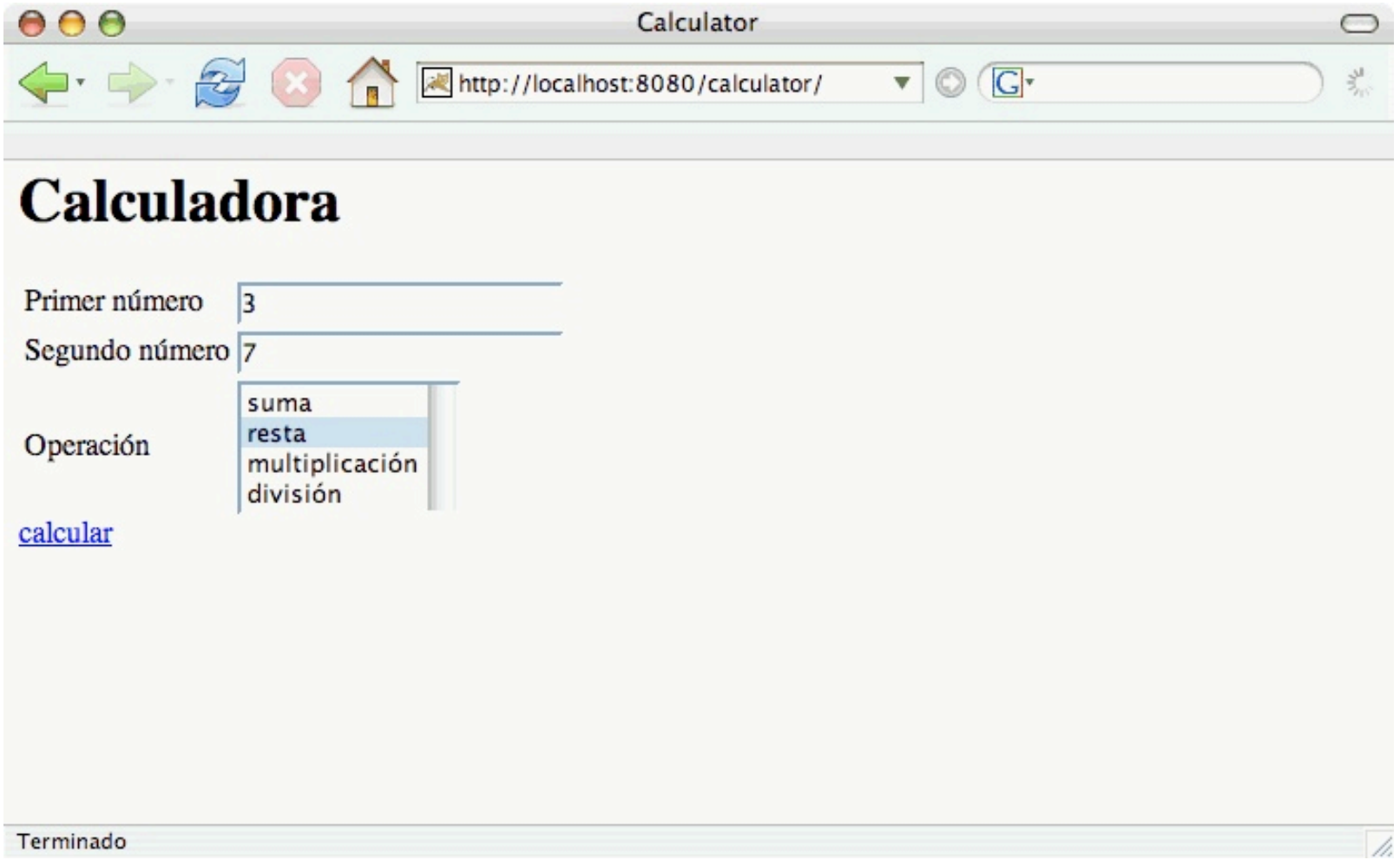
Ciclo de vida de una petición



Un ejemplo: calculator.jsp

```
<f:view>
     <h:form id="calcForm">
        <h:panelGrid columns="3">
           <h:outputLabel value="Primer número"/>
           <h:inputText id="firstNumber"
                        value="#{calcBean.firstNumber}"
                          required="true"/>
           <h:message for="firstNumber"/>
           ...

           <h:outputLabel value="Operación"/>
           <h:selectOneListbox id="operation" required="true"
                                 value="#{calcBean.operation}">
              <f:selectItem itemValue="+" itemLabel="suma"/>
              <f:selectItem itemValue="-" itemLabel="resta"/>
              <f:selectItem itemValue="*" itemLabel="multiplicación"/>
              <f:selectItem itemValue="/" itemLabel="división"/>
           </h:selectOneListbox>
           <h:message for="operation"/>
        </h:panelGrid>
        <h:commandLink action="#{calcBean.doOperation}">
            <h:outputText value="calcular"/>
         </h:commandLink>
     </h:form>
</f:view>
```

Pantalla


Calculator
http://localhost:8080/calculator/
Calculadora
Primer número
3
Segundo número
7
Operación
suma
resta
multiplicación
división
calcular
Terminado

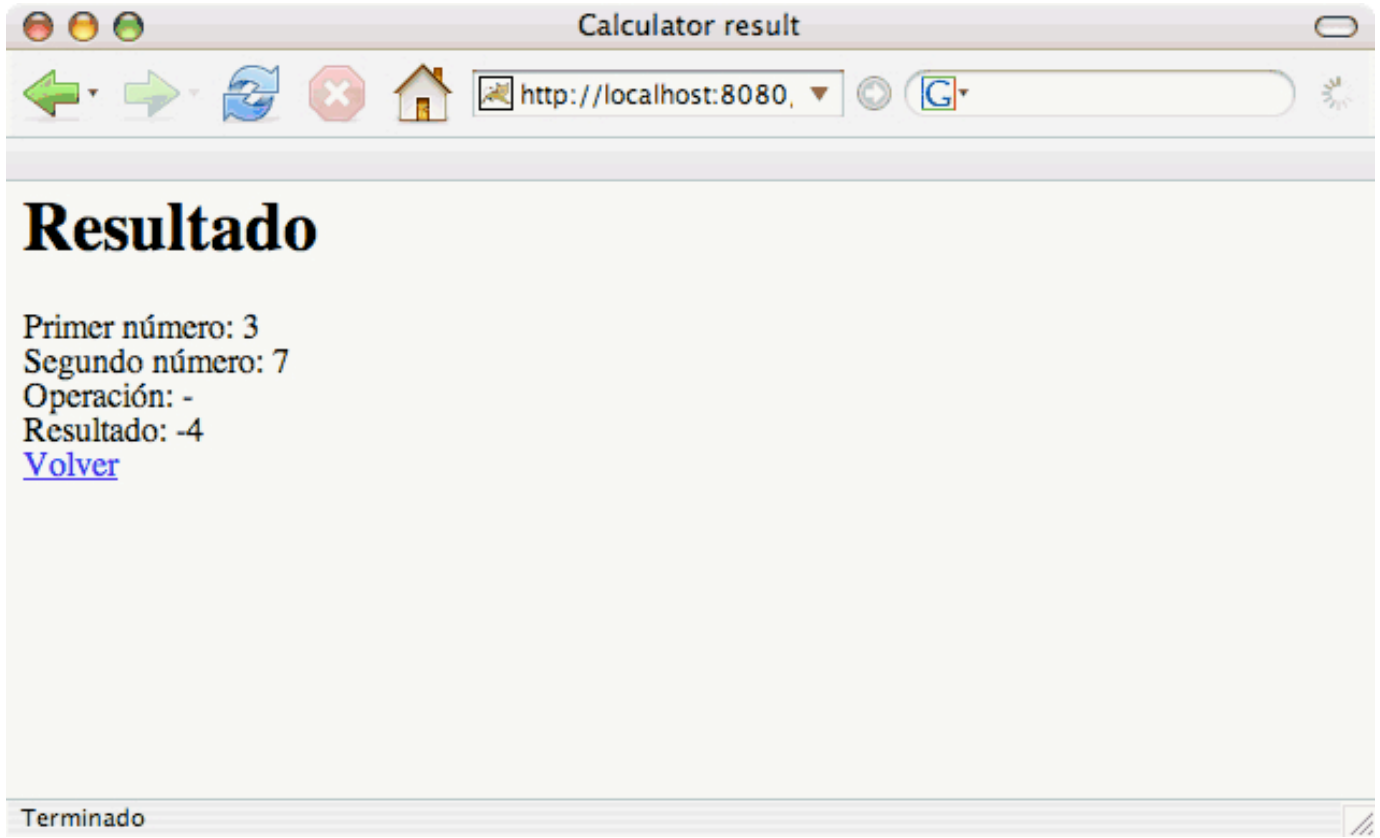
¿Cuándo se crea el árbol de componentes?

- En la primera petición (*http://localhost:8080/calculator*) se crea el árbol de componentes a partir del fichero calculator.jsp
- El árbol de componentes (vista) se guarda en el servidor.
- En la segunda petición (cuando el usuario pincha en el enlace “calcular” y se envía el formulario al servidor) JSF obtiene el árbol creado anteriormente, lo guarda en la petición y le aplica el ciclo de vida a la petición.

result.jsp

```
<f:view>
     <h:form id="calcResult">
        Primer número:
           <h:outputText id="firstNumber"
                         value="#{calcBean.firstNumber}"/> <br />
        Segundo número:
           <h:outputText id="secondNumber"
                         value="#{calcBean.secondNumber}"/> <br />
        Operación:
            <h:outputText id="operation"
                         value="#{calcBean.operation}"/> <br />
        Resultado:
            <h:outputText id="result"
                         value="#{calcBean.result}"/> <br />
        <h:commandLink action="OK">
            <h:outputText value="Volver"/>
         </h:commandLink>
     </h:form>
   </f:view>
```

Pantalla


Calculator result
http://localhost:8080,
Resultado
Primer número: 3
Segundo número: 7
Operación: -
Resultado: -4
Volver
Terminado

calculator.controller.CalculatorController

Backing Bean

¿Controller?

```
package calculator.controller;
import calculator.model.Calculato

public class CalculatorController {
    private Calculator calculator = new Calculator();
    private int firstNumber = 0;
    private int secondNumber = 0;
    private String operation = "";
    private int result = 0;

    public CalculatorController() {}

    public int getFirstNumber() {
        return firstNumber;
    }

    public void setFirstNumber(int firstNumber) {
        this.firstNumber = firstNumber;
    }

    // get-set int secondNumber
    // get-set String operation
    // get-set int result
    ...
```

calculator.controller.CalculatorController

Método de acción

```
...
 public String doOperation() {
       if (operation.equals("+"))
           result = calculator.add(firstNumber, secondNumber);
       else if (operation.equals("-"))
           result = calculator.substract(firstNumber, secondNumber);
       else if (operation.equals("*"))
           result = calculator.multiply(firstNumber, secondNumber);
       else if (operation.equals("/"))
           result = calculator.divide(firstNumber, secondNumber);
       return "OK";
   }
```

Programa ejemplo

- Ejecutar el programa ejemplo.

Validaciones definidas por el usuario

- Las conversiones y validaciones de valores de componentes se realizan en la fase 3 (con excepción de aquellos componentes con el atribute “inmediate” a true, en los que se realizan en la fase 2).
- Además de las validaciones y conversiones definidas por JSF, es posible ampliar el framework con las nuestras propias.
- Para ello debemos:
  - 1. Crear una clase que implemente la interfaz javax.faces.validator.Validator e implementar el método validate() de esa interfaz.
  - 2. Registrar el validador propio en el fichero “faces-config.xml”
  - 3. Usar la etiqueta <f:validator/> en las páginas JSP.

Validaciones definidas por el usuario

- Probar el ejemplo en la aplicación “calculator”.

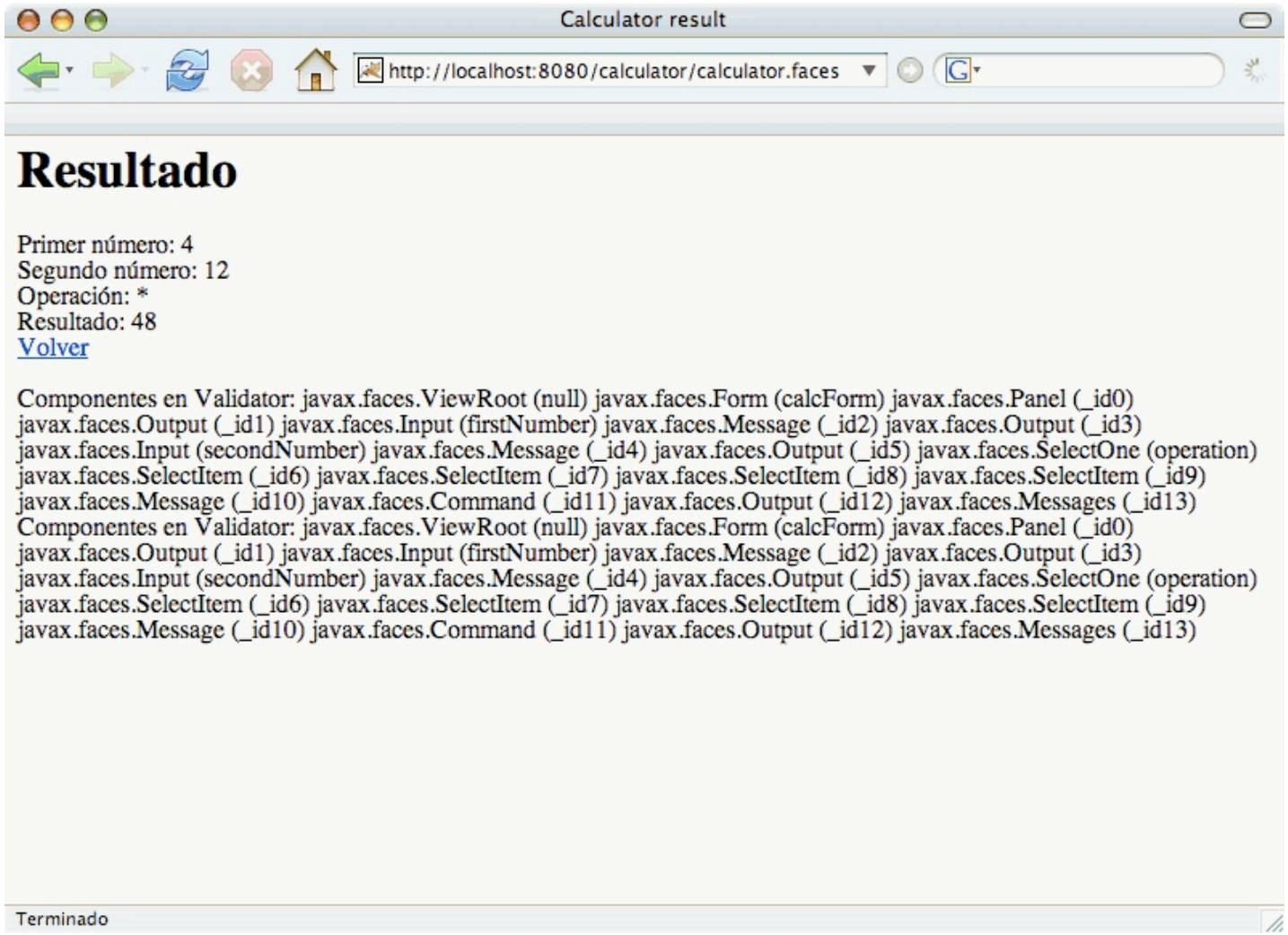
Peticiones JSF

- Una petición JSF tiene asociado un contexto, en forma de una instancia de FacesContext.
- Esta clase define un conjunto de métodos que nos permiten obtener y modificar sus elementos:
  - La cola de mensajes
  - El árbol de componentes
  - Objetos de configuración de la aplicación
  - Métodos de control del flujo del ciclo de vida

Ejemplo de código

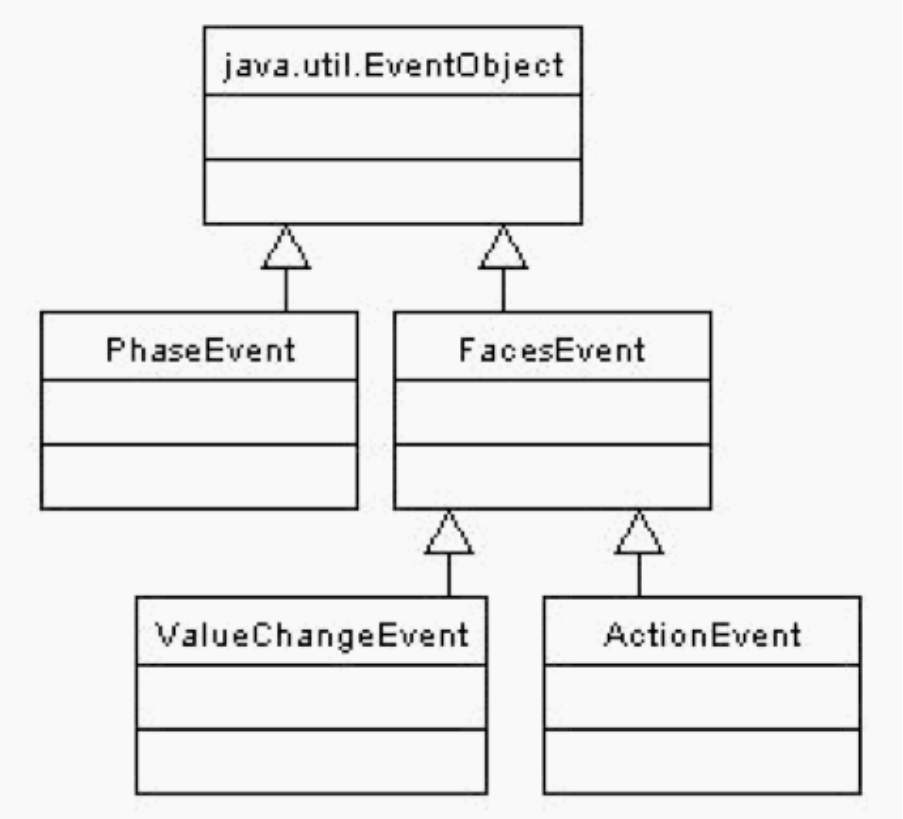
```
package calculator.validator;
...
import javax.faces.component.UIComponentBase;
import javax.faces.component.UIViewRoot;
import javax.faces.context.FacesContext;

public class PairNumberValidator implements Validator {
    public void validate(FacesContext arg0,
                         UIComponent component,
                            Object value)
            throws ValidatorException {
        FacesContext context = FacesContext.getCurrentInstance();
        UIViewRoot viewRoot = context.getViewRoot();
        String ids = getComponentIds(viewRoot);
        FacesMessage message = new FacesMessage("Componentes: "+ ids);
        context.addMessage(null,message);
        ...
        }
    }

}
```

Pantalla


Calculator result
http://localhost:8080/calculator/calculator.faces
Resultado
Primer número: 4
Segundo número: 12
Operación: *
Resultado: 48
Volver
Componentes en Validator: javax.faces.ViewRoot (null) javax.faces.Form (calcForm) javax.faces.Panel (_id0) javax.faces.Output (_id1) javax.faces.Input (firstNumber) javax.faces.Message (_id2) javax.faces.Output (_id3) javax.faces.Input (secondNumber) javax.faces.Message (_id4) javax.faces.Output (_id5) javax.faces.SelectOne (operation) javax.faces.SelectItem (_id6) javax.faces.SelectItem (_id7) javax.faces.SelectItem (_id8) javax.faces.SelectItem (_id9) javax.faces.Message (_id10) javax.faces.Command (_id11) javax.faces.Output (_id12) javax.faces.Messages (_id13)
Componentes en Validator: javax.faces.ViewRoot (null) javax.faces.Form (calcForm) javax.faces.Panel (_id0) javax.faces.Output (_id1) javax.faces.Input (firstNumber) javax.faces.Message (_id2) javax.faces.Output (_id3) javax.faces.Input (secondNumber) javax.faces.Message (_id4) javax.faces.Output (_id5) javax.faces.SelectOne (operation) javax.faces.SelectItem (_id6) javax.faces.SelectItem (_id7) javax.faces.SelectItem (_id8) javax.faces.SelectItem (_id9) javax.faces.Message (_id10) javax.faces.Command (_id11) javax.faces.Output (_id12) javax.faces.Messages (_id13)
Terminado

Tipos de eventos en JSF

- En JSF se define tres grandes tipos de eventos:
  - ValueChangeEvent
  - ActionEvent
  - PhaseEvent



Definir un manejador de ValueChangeEvent

- Definir una clase que implemente la interfaz ValueChangeListener
- Asociar esa clase con el componente que queremos vigilar usando la etiqueta

```
<f:valueChangeListener type="nombre de la
clase"/>
```

Ejemplo en la aplicación

- Probar el ejemplo en la aplicación

Procesamiento de eventos

- Los eventos ValueChangeEvent se procesan después de la fase de “Process Validations”
- El posible evento ActionEvent se procesa en la fase “Invoke Application”
  - Se llama al método de acción definido en el evento y se obtiene la cadena resultante
  - Se busca en el fichero “faces-config.xml” la regla a aplicar
  - Se “carga” la nueva vista: se desecha el árbol de componentes actual y se crea el nuevo árbol de componentes
- Si un componente de entrada o de acción tiene el atributo “inmediate” a “true”, el evento asociado se procesa en la primera fase “Apply Request Values”

Eventos PhaseEvent

- Se producen antes y después de todas las fases del ciclo de vida de la petición
- Un manejador de este evento debe implementar la interfaz PhaseListener, con los métodos beforePhase() y afterPase()
- Se puede declarar un manejador propio en el fichero faces-config.xml:

```
<lifecycle>
   <phase-listener>
      calculator.controller.CalculatorPhaseListener
   </phase-listener>
</lifecycle>
```

Ejemplo en la aplicación

- Probar ejemplo

Extra: Un par de ejemplos más

- Probar el proyecto `jsf-invoices`
- Ejemplo completo definido en el libro 'Mastering JavaServer Faces'

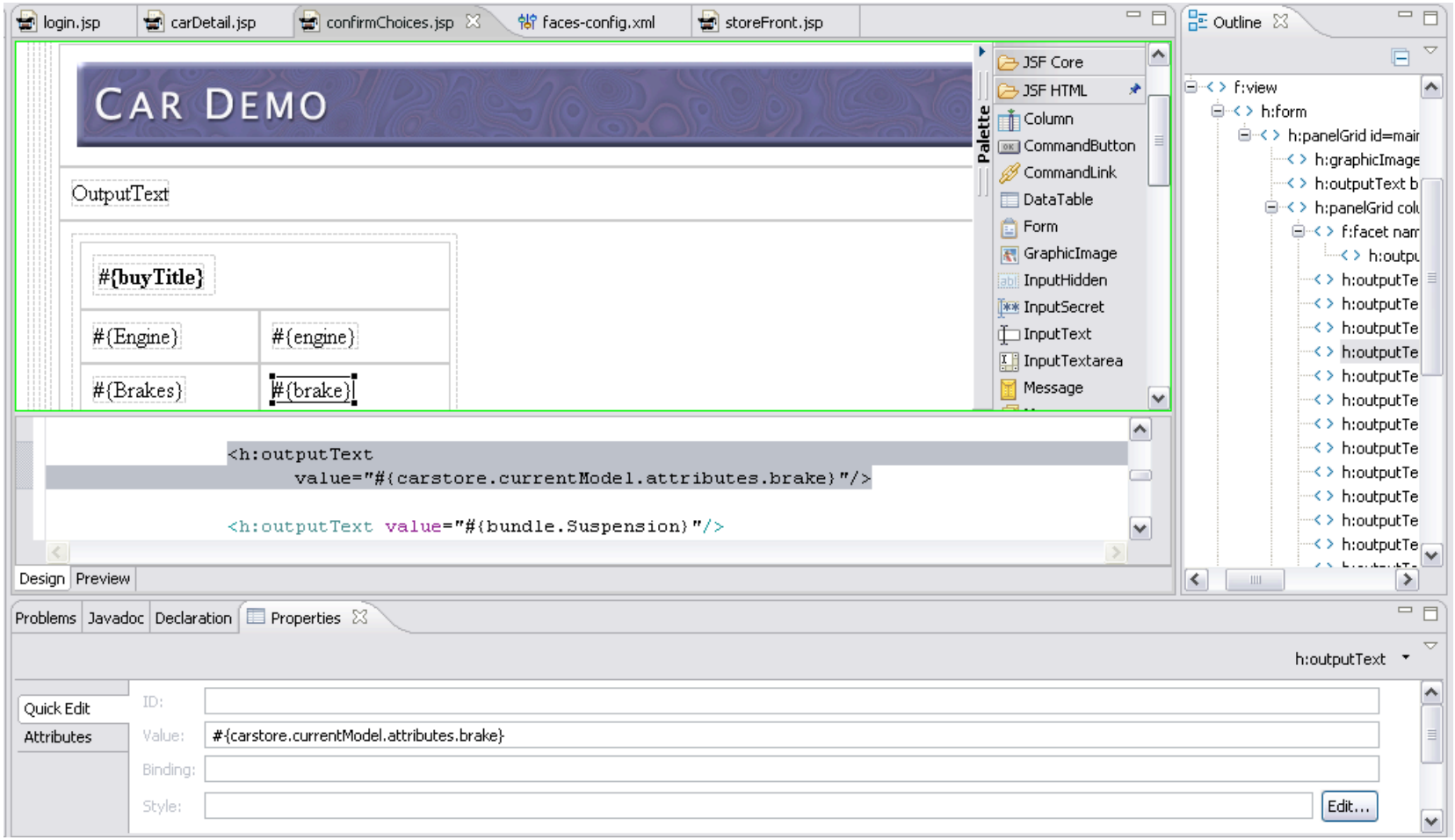
- Probar todos los componentes proporcionados por MyFaces Tobago:

```
tobago-example-demo-1.0.9.war
```

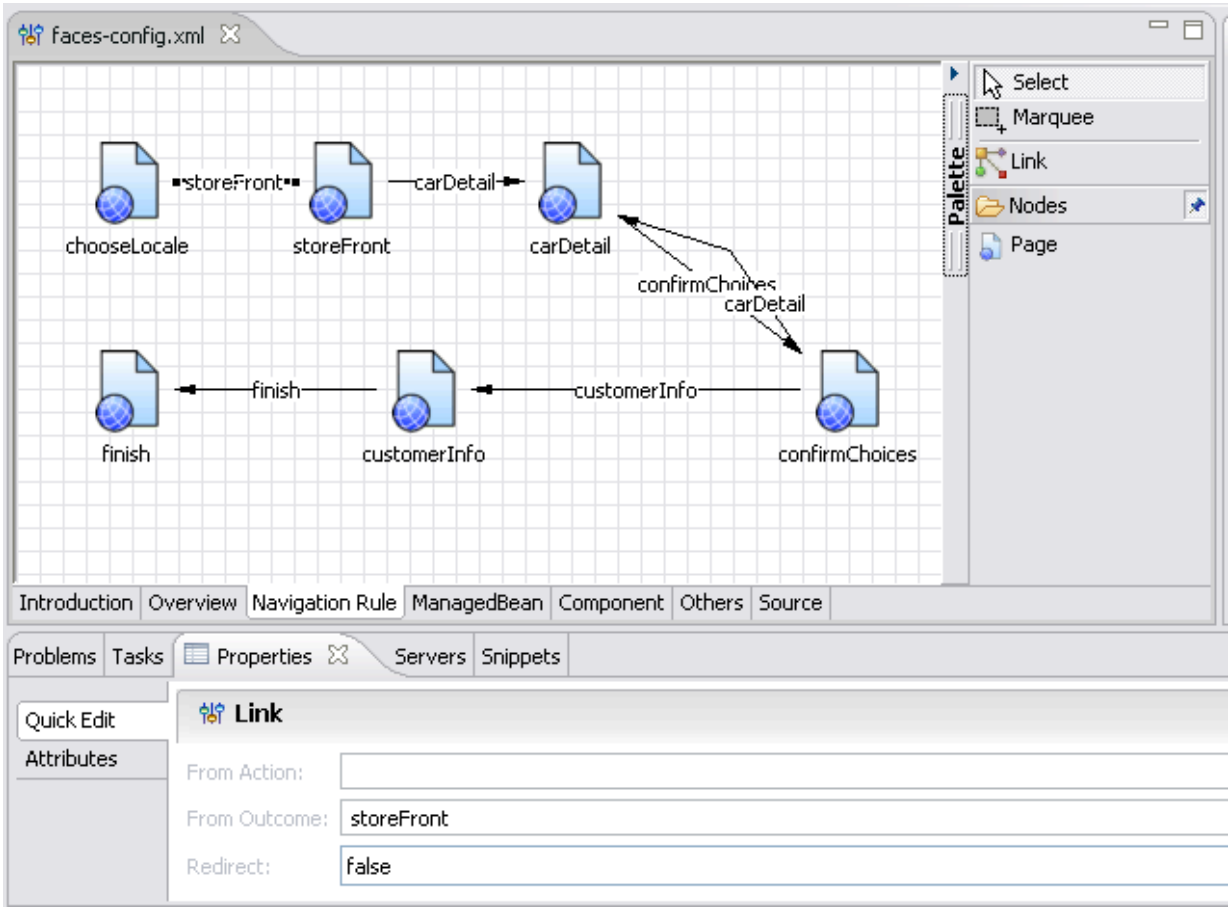
Extra: Herramienta en Eclipse para JSF

- Lo hemos “escondido” durante toda la mañana
- Sí que existe una herramienta en Eclipse para trabajar con JSF
- Está incluida en Eclipse WebTools, pero hay que “activarla” (se trata de una versión 0.5)
- Help > Software Updates > Find and Install

Entorno de trabajo



Vista de faces-config.xml



¿Preguntas?