

Taller de Sistemas de Información 2

Desarrollo Web

21 de Agosto de 2014



Instituto de
Computación

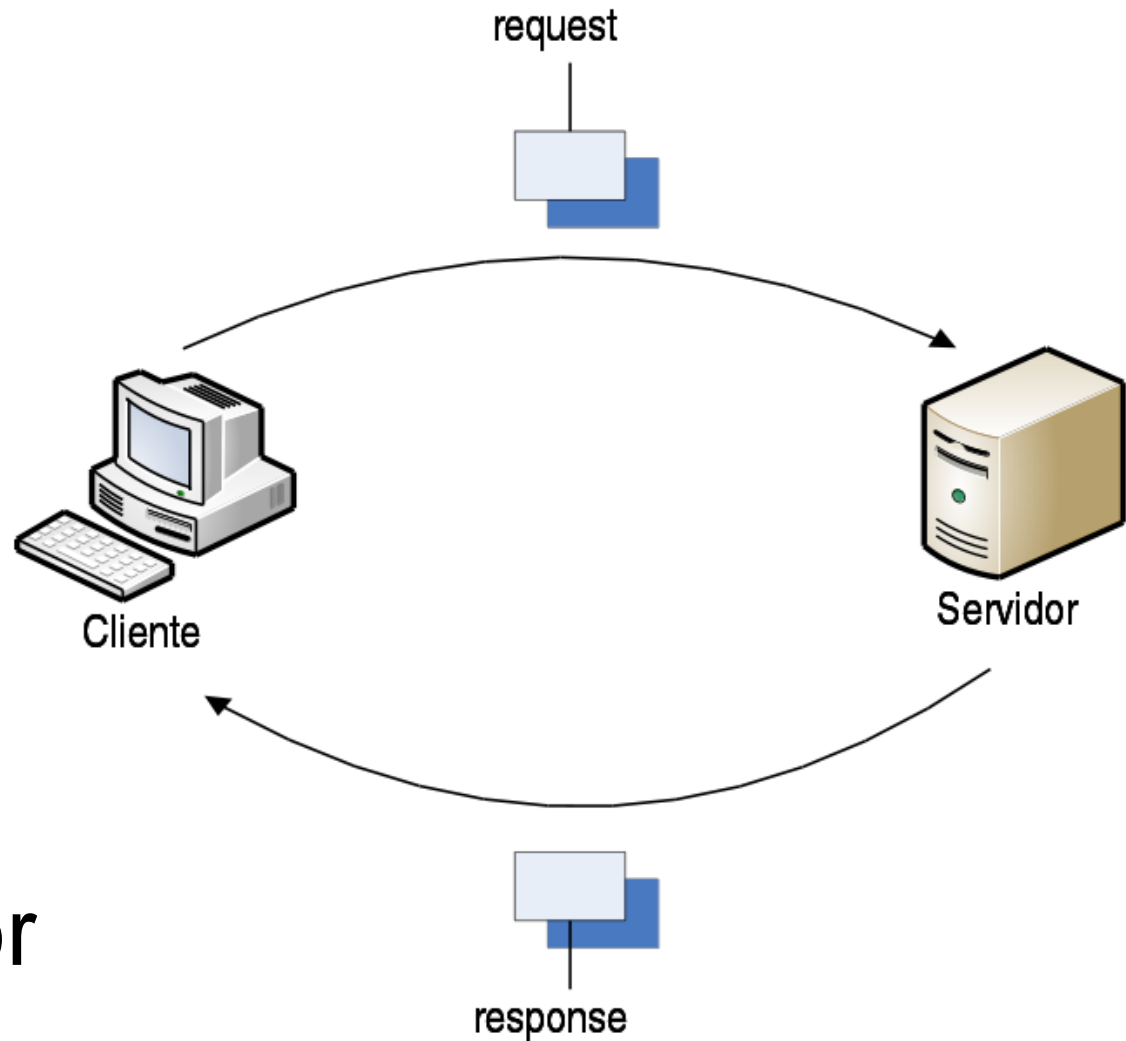


Facultad de
Ingeniería



Universidad de la
República de Uruguay

Como funciona la web?



Modelo
cliente / servidor

Como funciona la web?

- ❑ Protocolo HTTP
 - Hyper Text Transfer Protocol
- ❑ Basado en TCP/IP
 - encargado de mover datos entre cliente (browser) y servidor (web server)
- ❑ Compuesto por
 - HTTP Request (solicitud)
 - HTTP Response (respuesta)

Recursos / URI / URL

- ❑ Se solicitan recursos, se devuelve contenido
- ❑ Los recursos se identifican a través de direcciones denominadas URI
- ❑ Uniform Resource Identifier
- ❑ De la forma
 - <http://www.google.com/gmail>
 - <http://www.fing.edu.uy/inco/cursos/compil>

- ❑ Las URLs son un caso particular de URI
- ❑ Representan un recurso de cierto tipo que existe en un servidor
- ❑ Forma general
 - **Protocolo://Servidor:Puerto/Request–URI**

- ❑ Se envía al servidor una tira de texto con el siguiente contenido (a modo de ejemplo)
 - “GET /request-URI HTTP/version”
 - El envío se hace al servidor y puerto indicados en la URL
- ❑ En realidad no tiene porque ser un browser el que genera este paquete
- ❑ **Cualquier programa, en cualquier lenguaje puede generar un paquete HTTP**

- ❑ El servidor recibe la solicitud, busca el documento y retorna su contenido
- ❑ La respuesta tiene el siguiente formato:

HTTP/[VER] [CODE] [TEXT]

Field1: Value1

Field2: Value2

...Contenido del documento...

Respuesta

HTTP/1.0 200 OK

Server: Netscape-Communications/1.1

Date: Tuesday, 25-Nov-97 01:22:04 GMT

Last-modified: Thursday, 20-Nov-97 10:44:53 GMT

Content-length: 6372

Content-type: text/html

<!DOCTYPE HTML PUBLIC

"-//W3C//DTD HTML 3.2 Final//EN">

<HTML>

...contenido del documento...

- ❑ Todos los requests tienen esta estructura
[METH] [REQUEST-URI] HTTP/[VER]
[fieldname1]: [field-value1]
[fieldname2]: [field-value2]
[Cuerpo del request, si existe]

- ❑ METH representa el método HTTP utilizado
 - GET, POST, PUT, TRACE, HEAD, DELETE

□ GET

- Pide al servidor que le envíe un recurso

□ POST

- Envía datos al servidor para que sean procesados por el recurso especificado en la solicitud
- Los datos se incluyen en el cuerpo de la solicitud
- Este método podría crear un nuevo recurso en el servidor, o actualizar un recurso ya existente

□ PUT

- Envía un recurso determinado (por ejemplo un archivo) al servidor
- Este método crea una nueva conexión (socket) y la emplea para enviar el recurso
- Resulta más eficiente que enviarlo dentro del cuerpo del mensaje

□ DELETE

- Elimina el recurso especificado.

HTTP Response

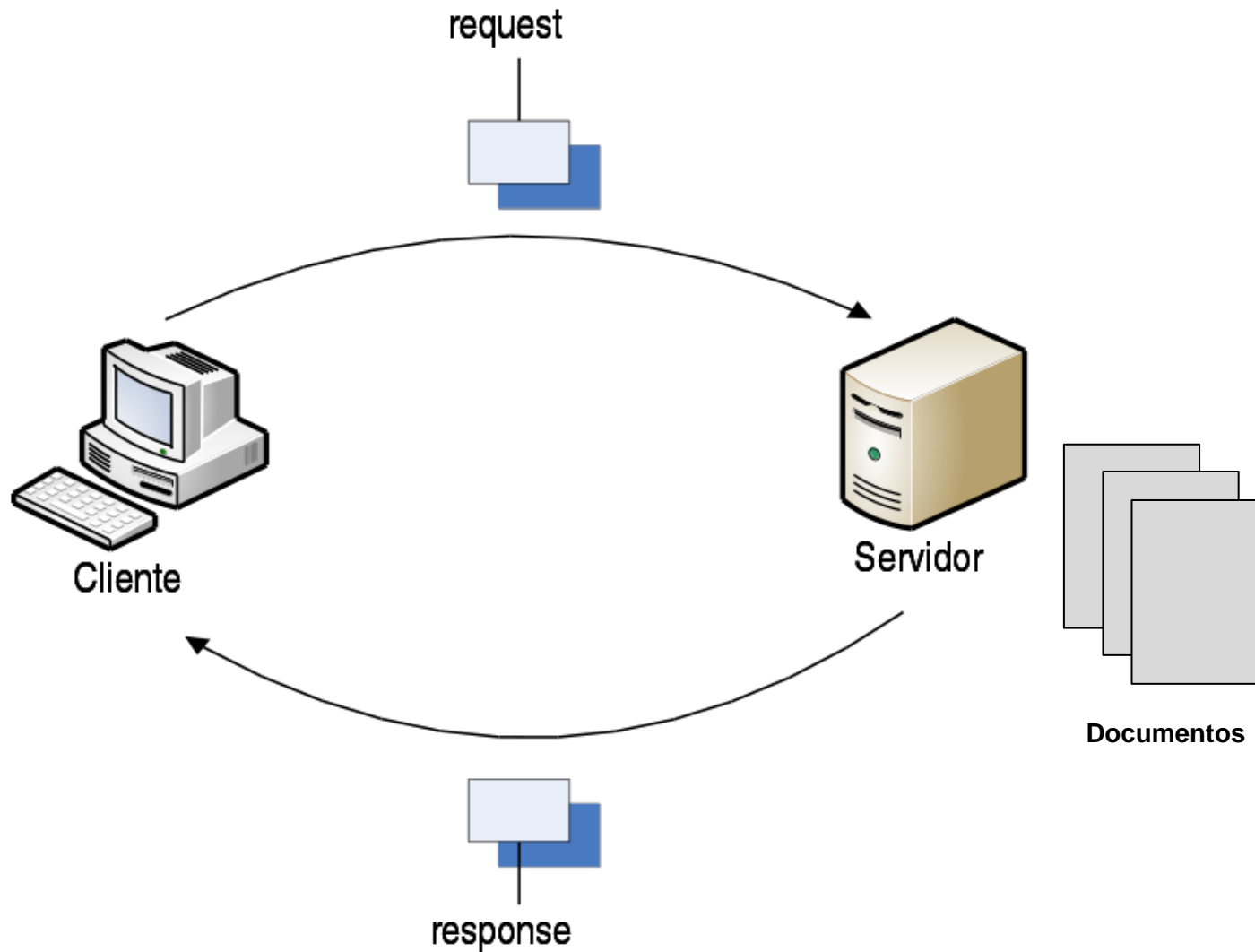
- ❑ El server retorna una fila con el status code, una serie de headers, una línea en blanco y luego el documento solicitado
 - HTTP/1.0 code text
 - Field1: Value1
 - Field2: Value2

 - Contenido del documento...

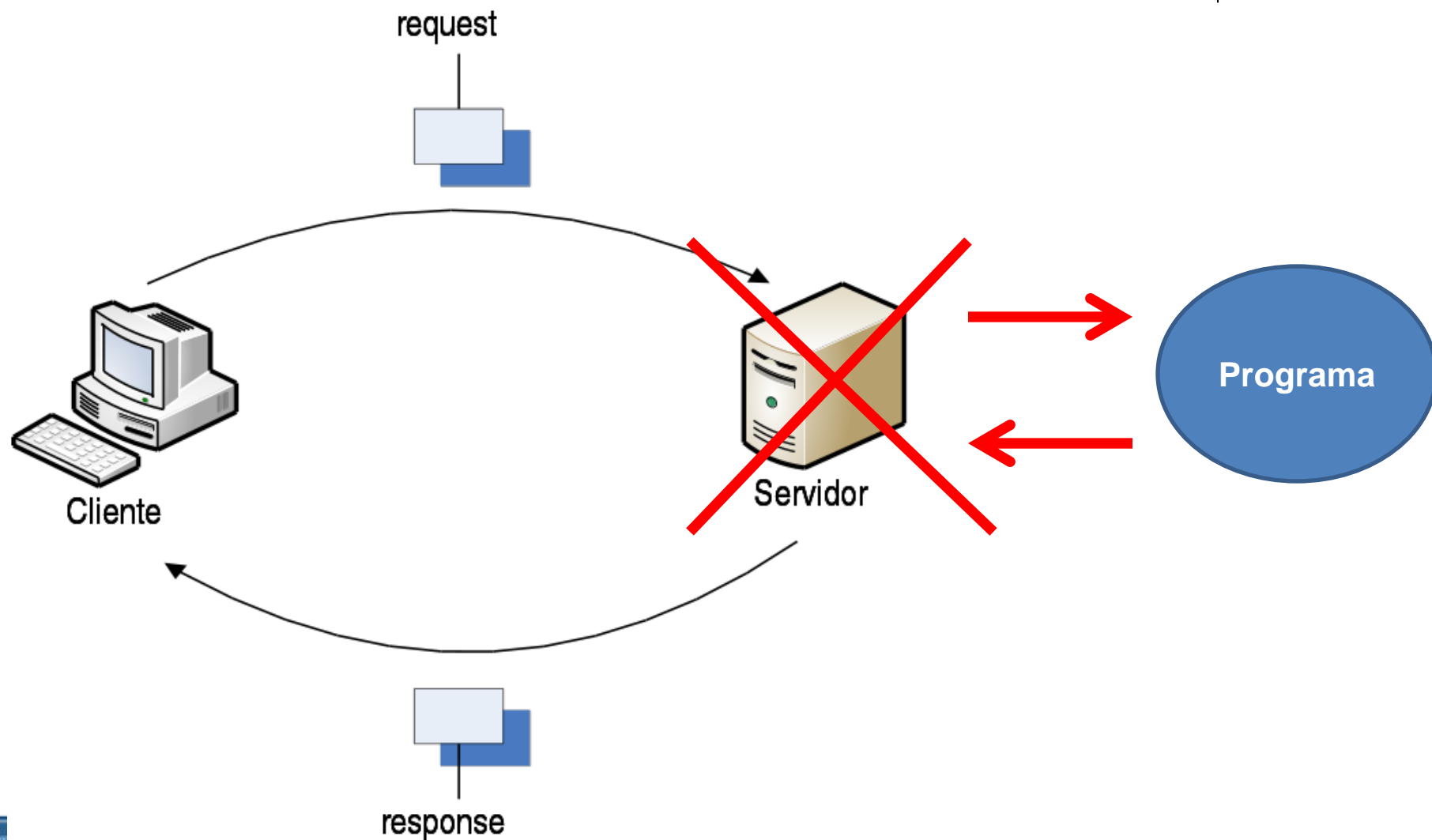
Códigos de estado

- ❑ Permiten devolver al cliente información de la solicitud, tanto en caso de éxito como de fallo
- ❑ 1xx: Informativos
- ❑ 2xx: Éxito
- ❑ 3xx: Dirección
- ❑ 4xx: Error del cliente
- ❑ 5xx: Error del servidor

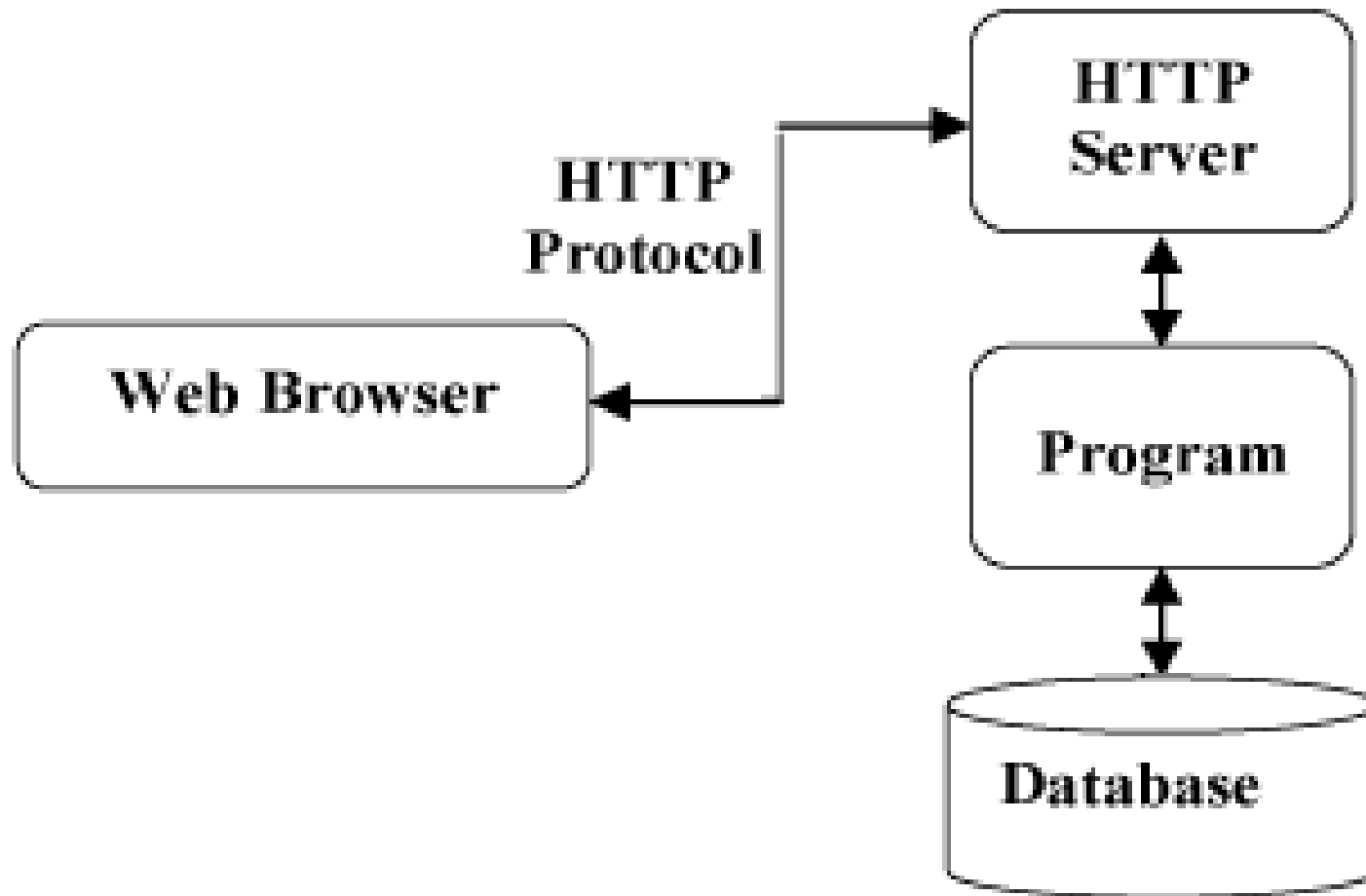
Programación server side



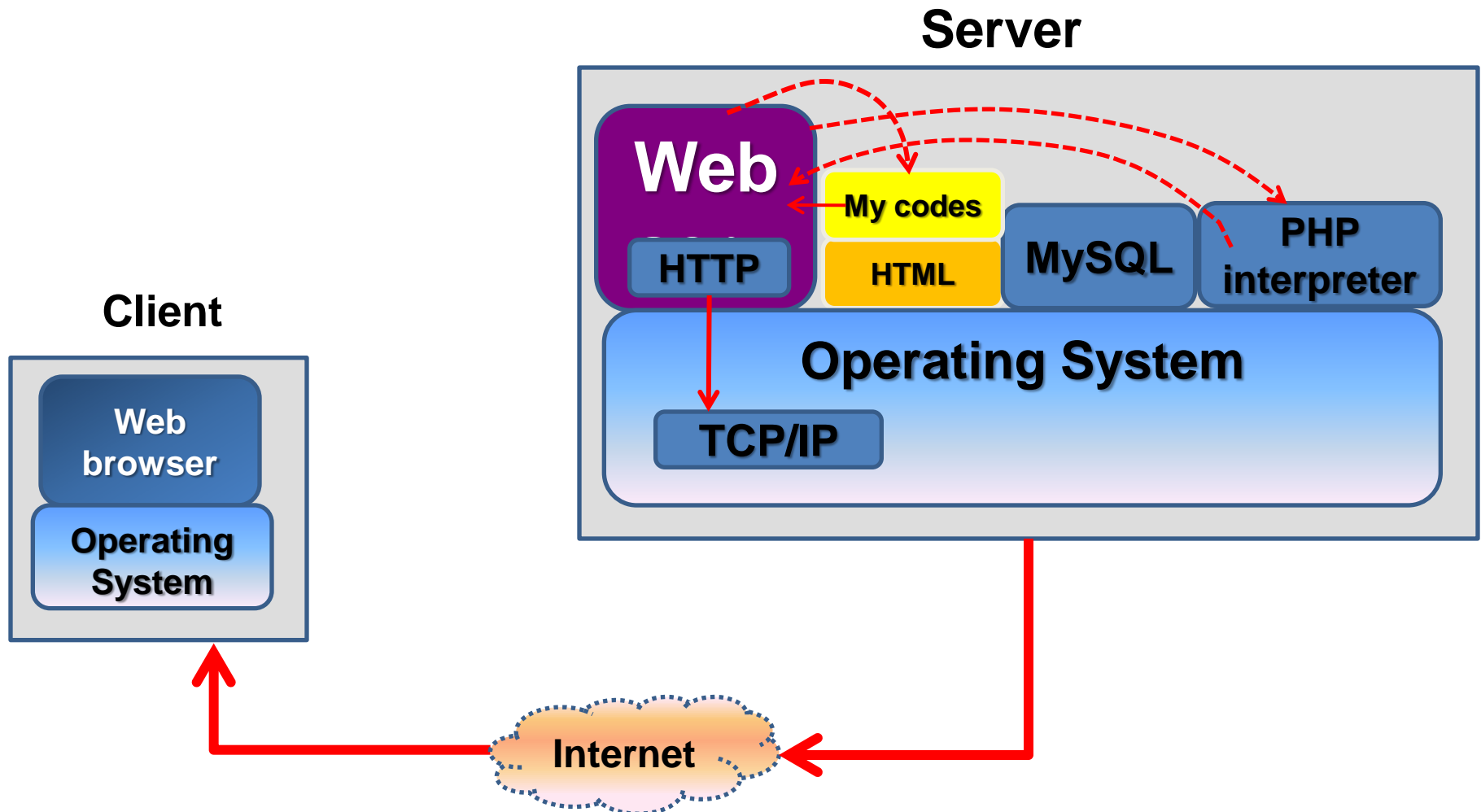
Programación server side

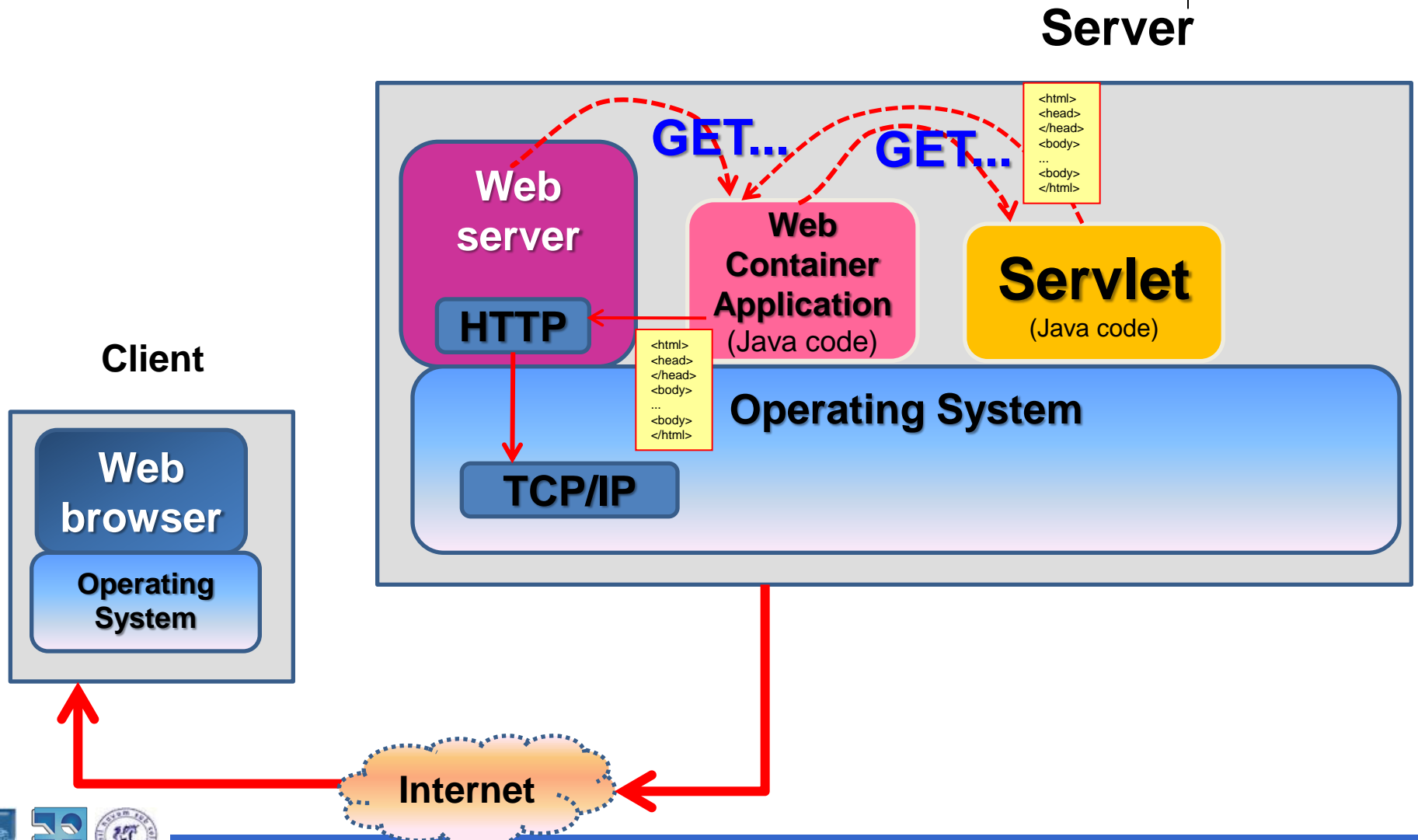


Programación server side



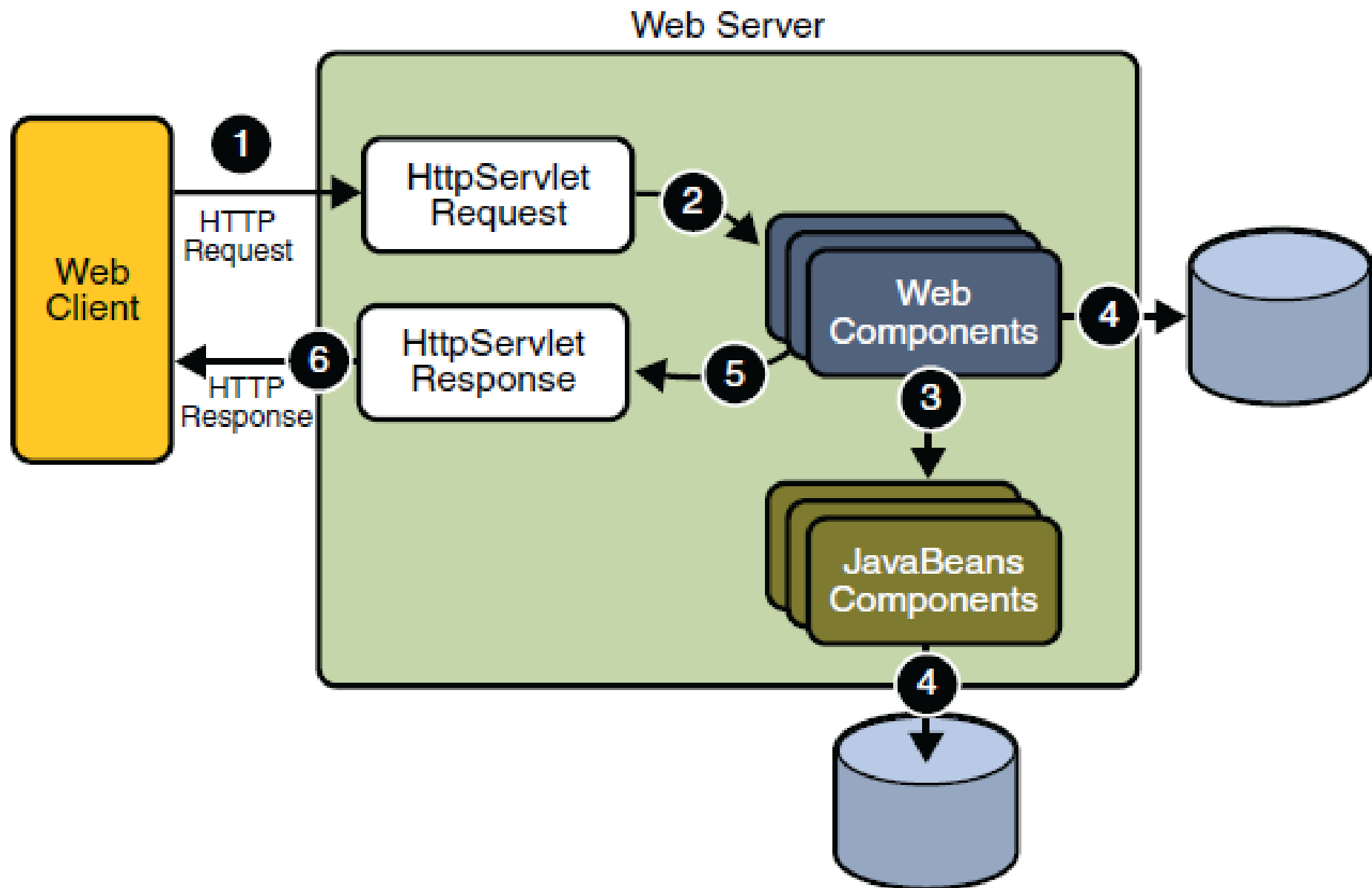
PHP / MySQL



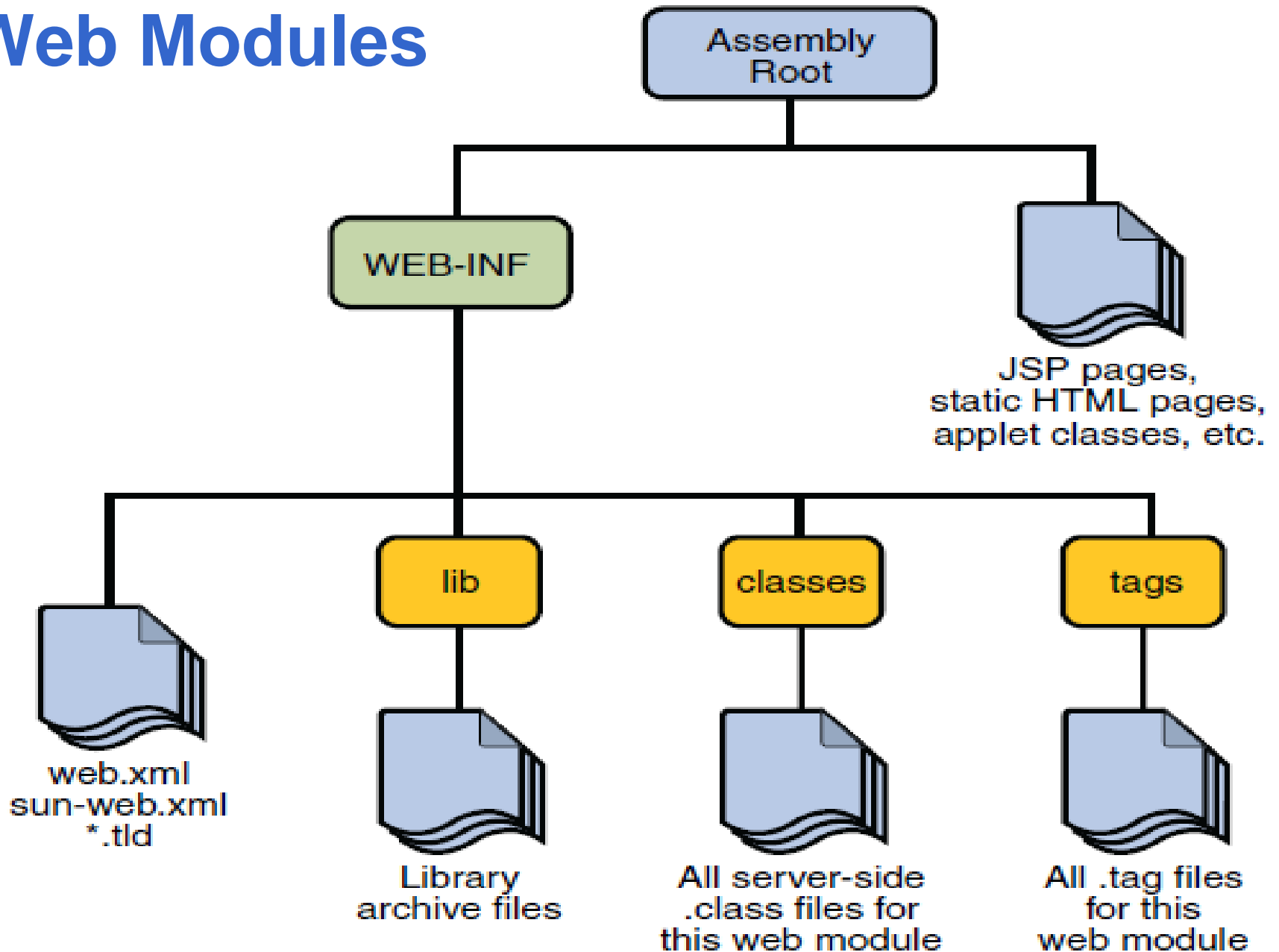


SERVLETS & JSPS

- En Java, la programación server-side se logra a través de los **componentes web**
- Estos componentes web participan del protocolo HTTP, interceptando el request, obteniendo valores del mismo y generando una respuesta
- Estos componentes web, ejecutan dentro de un ambiente, denominado Web Container



Web Modules



- ❑ Los Servlets son componentes que ejecutan en el servidor, que generan código HTML dinámicamente
- ❑ Participan del protocolo HTTP
- ❑ Están desarrollados utilizando el lenguaje Java, y ejecutan dentro de un Web Container, el cual se encuentra presente dentro de un Application Server

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet
    extends HttpServlet {

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

        PrintWriter out = response.getWriter();
        out.println("Hello World");

    }
}
```



```

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

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

        out.println(
            "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
            \"          Transitional//EN\">\n" +
            "<HTML>\n" +
            "    <HEAD><TITLE>Hello World</TITLE></HEAD>\n" +
            "    <BODY>\n" +
            "        <H1>Hello World</H1>\n" +
            "    </BODY>\n" +
            "</HTML>"
        );
    }
}

```

Servlet Declarations (web.xml)

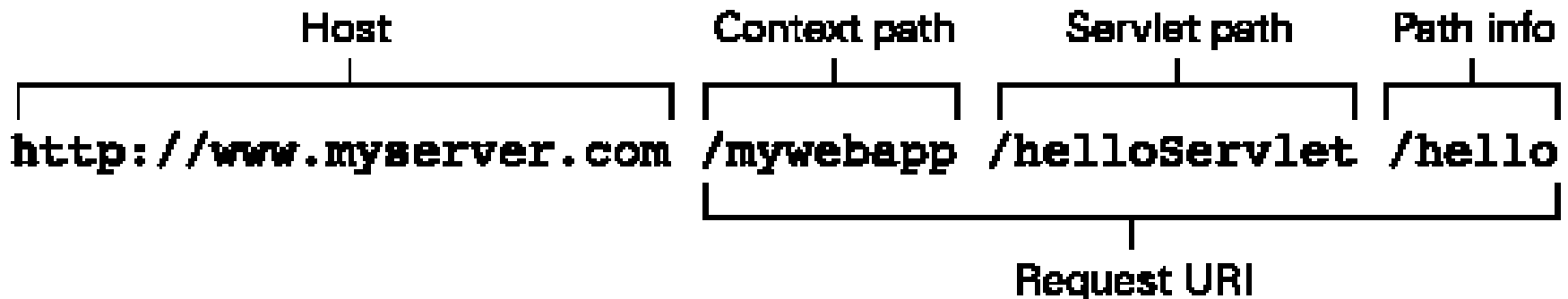
```
<servlet>
  <servlet-name>HelloWorldServlet</servlet-name>
  <servlet-class>test.HelloWorld</servlet-class>
  <init-param>
    <param-name>cant</param-name>
    <param-value>5</param-value>
  </init-param>
</servlet>

<servlet>
  <servlet-name>LoginServlet</servlet-name>
  <servlet-class>test.LoginServlet</servlet-class>
</servlet>
```

Servlet Mappings (web.xml)

```
<servlet-mapping>  
  <servlet-name>HelloWorldServlet</servlet-name>  
  <url-pattern>/helloServlet/*</url-pattern>  
</servlet-mapping>
```

```
<servlet-mapping>  
  <servlet-name>LoginServlet</servlet-name>  
  <url-pattern>/loginServlet/*</url-pattern>  
</servlet-mapping>
```



- ❑ Especifica la configuración necesaria para una clase de un Servlet

```
@WebServlet(  
    name = "HelloAnnotationServlet",  
    urlPatterns = {"/hello", "/helloanno"},  
    initParams = {  
        @WebInitParam(name = "name", value = "admin"),  
        @WebInitParam(name = "param1", value = "value1"),  
        @WebInitParam(name = "param2", value = "value2")  
    }  
)  
public class HelloAnnotationServlet extends HttpServlet
```

Interactuando con el Servlet

- ❑ Las paginas HTML pueden enviar información a un Servlet, a través de formularios HTML

```
<form method="POST"  
      action="/Usuarios/loginServlet">
```

Nombre:

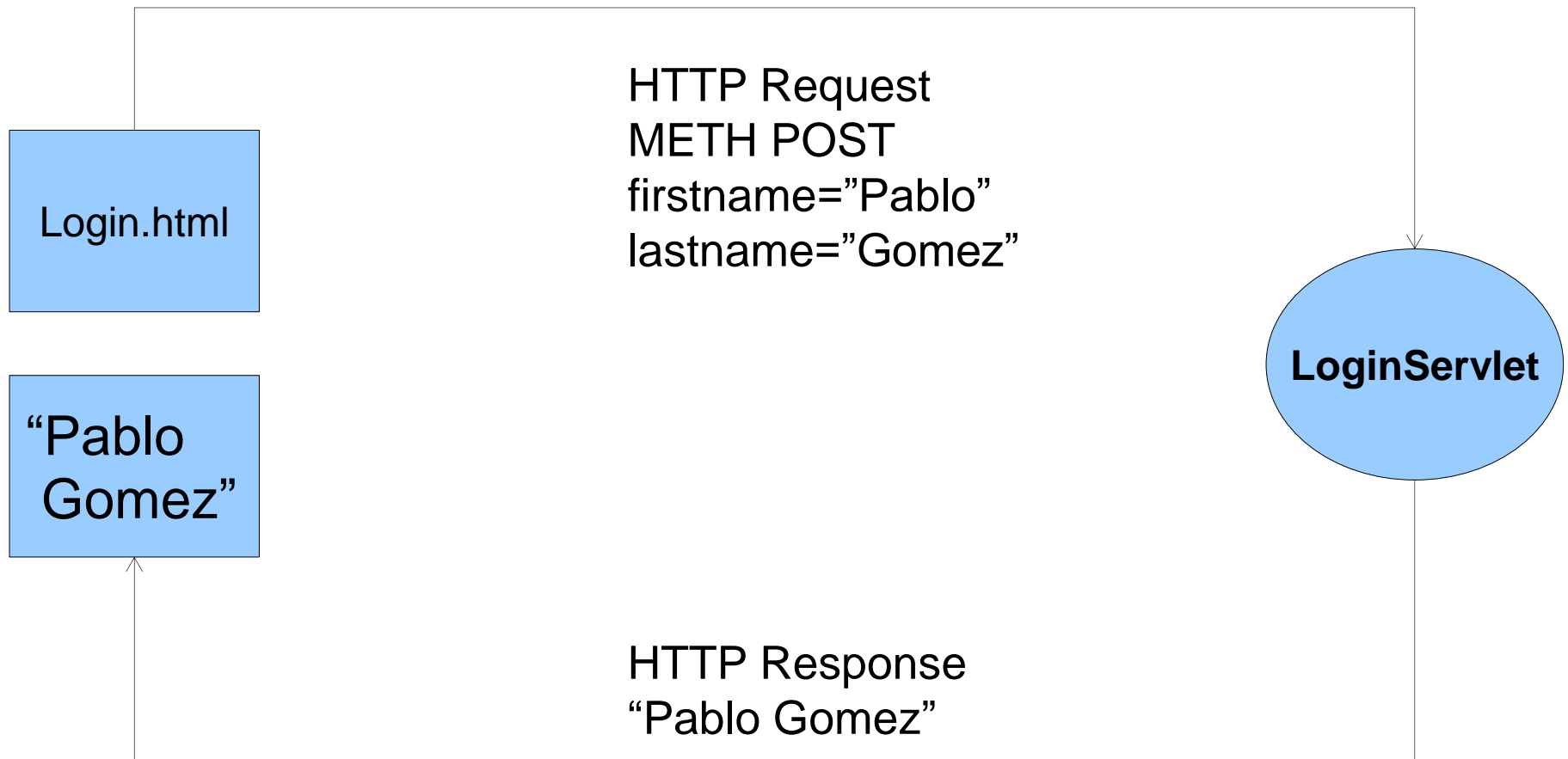
```
<input type="text" name="firstname"/>
```

Apellido

```
<input type="text" name="lastname"/>
```

```
</form>
```

Interactuando con el Servlet



Interactuando con un Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

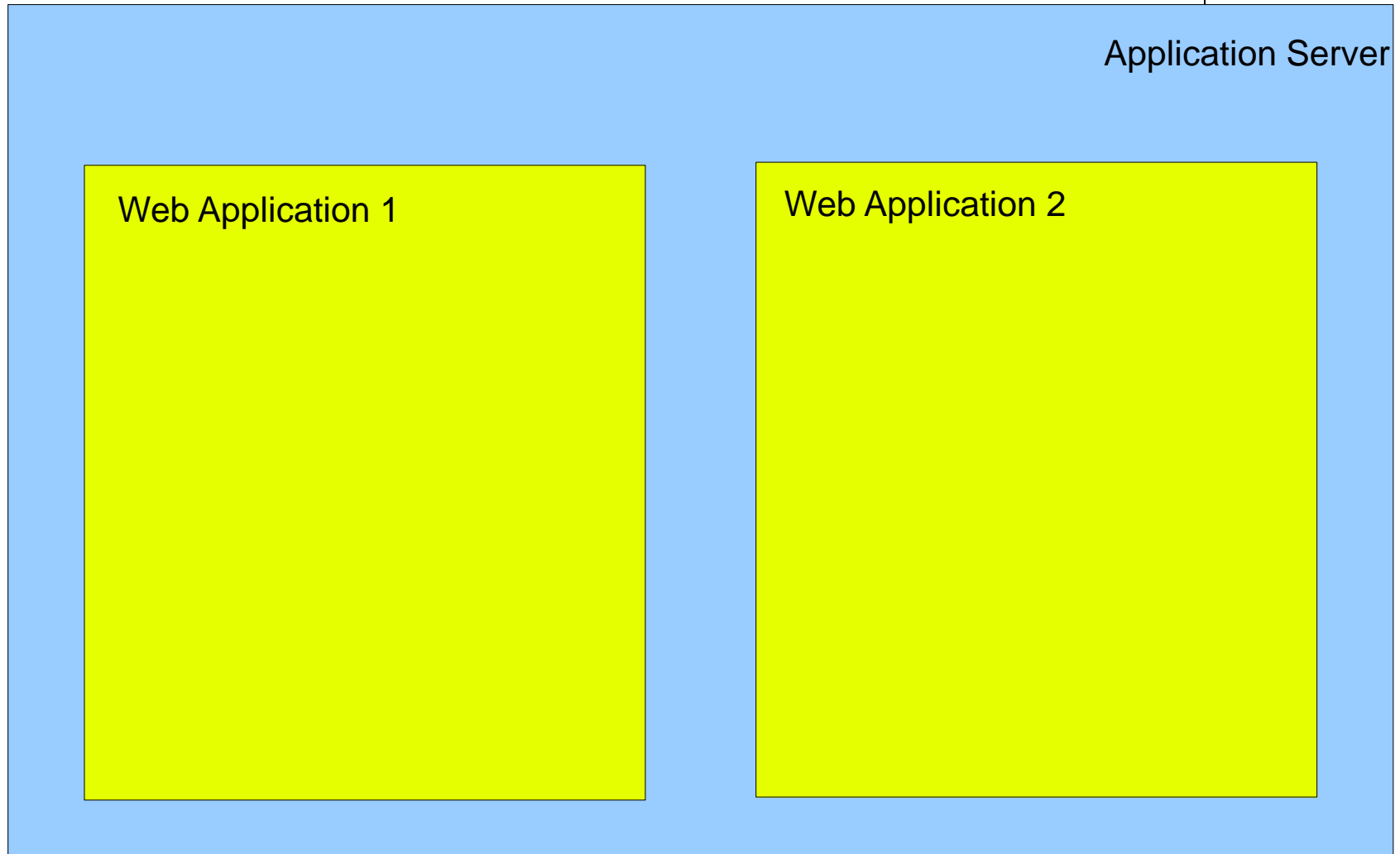
public class HelloWorld extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

        String firstName = request.getParameter("firstname");
        String lastName = request.getParameter("lastname");

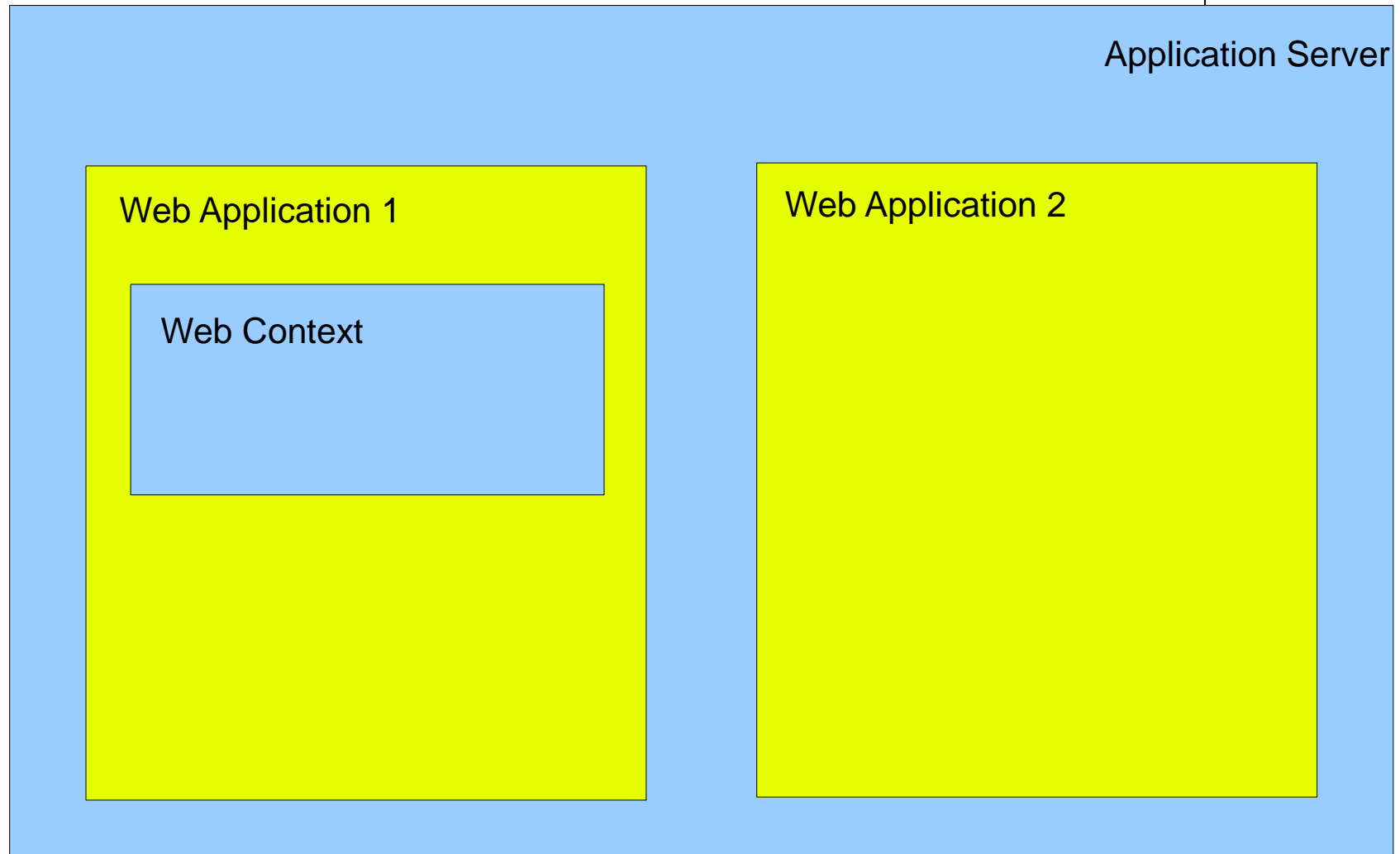
        PrintWriter out = response.getWriter();
        out.println(firstName + " " + lastName);
    }
}
```

- ❑ Los **web contexts** permiten que los componentes web puedan compartir e intercambiar información
- ❑ Supongamos que tenemos una aplicación web con un carrito de compras
- ❑ Un servlet se puede encargar de actualizar el contenido del objeto “carrito”
- ❑ Otro servlet genera el HTML necesario para mostrar el contenido de dicho “carrito”

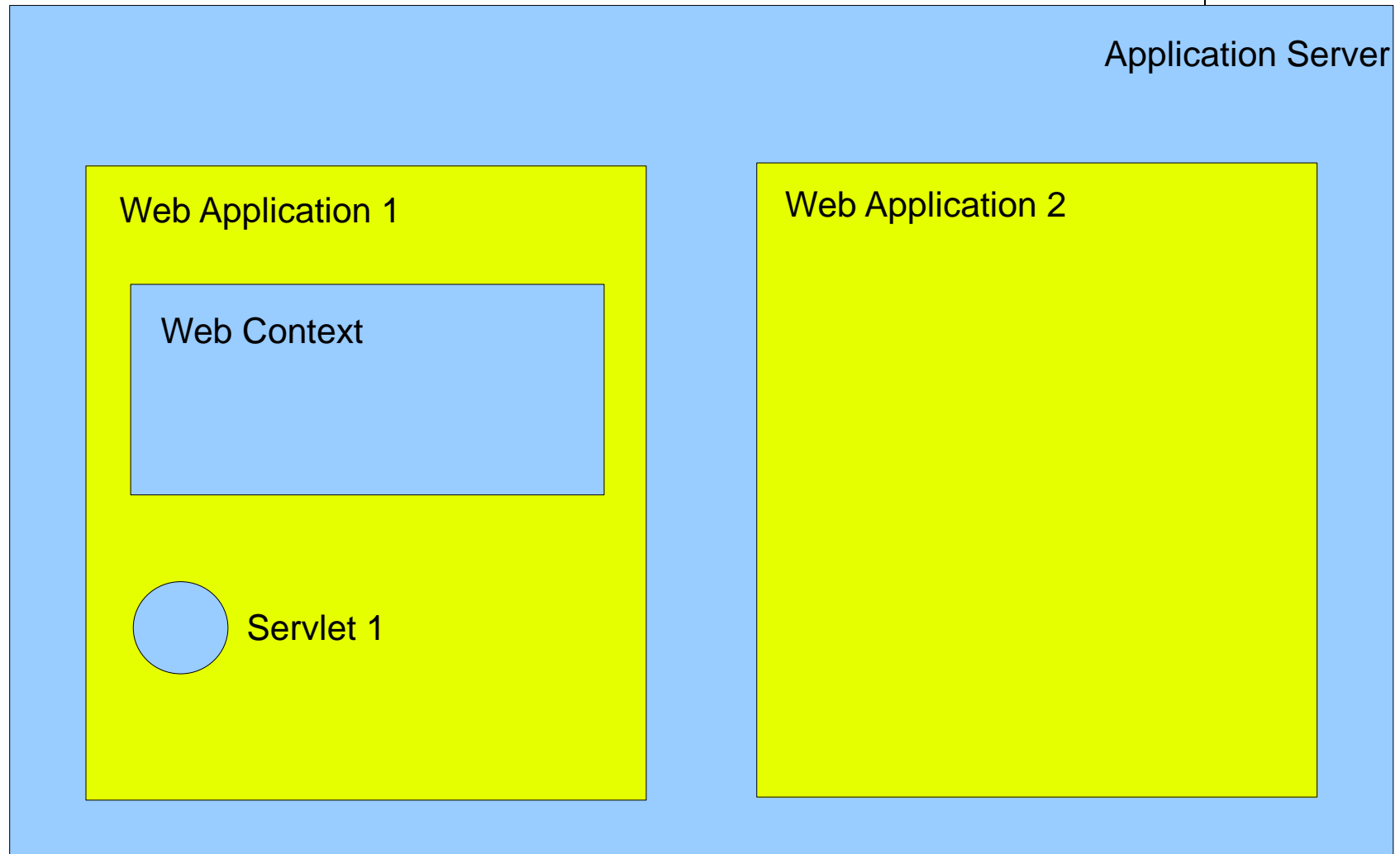
Web Contexts



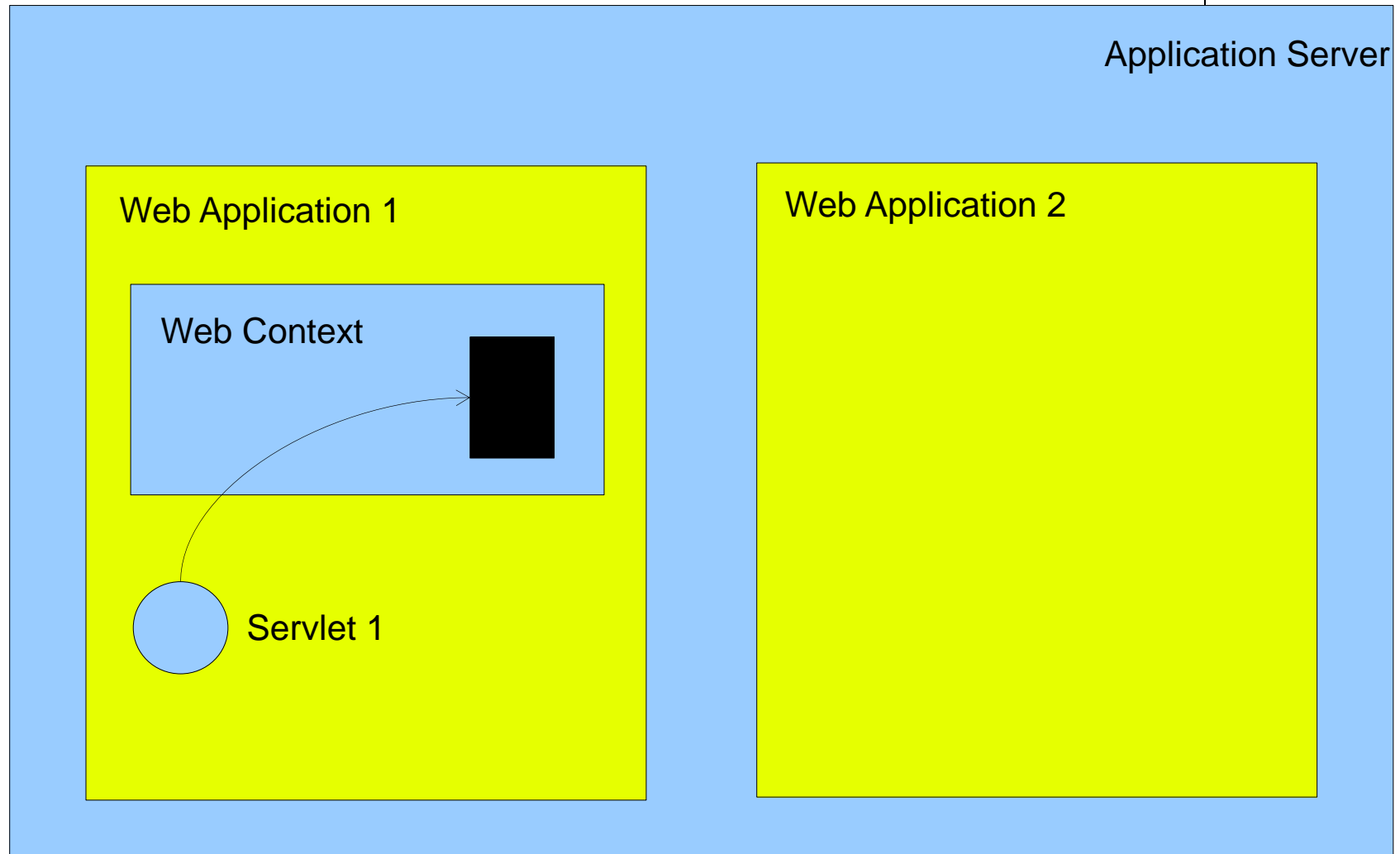
Web Contexts



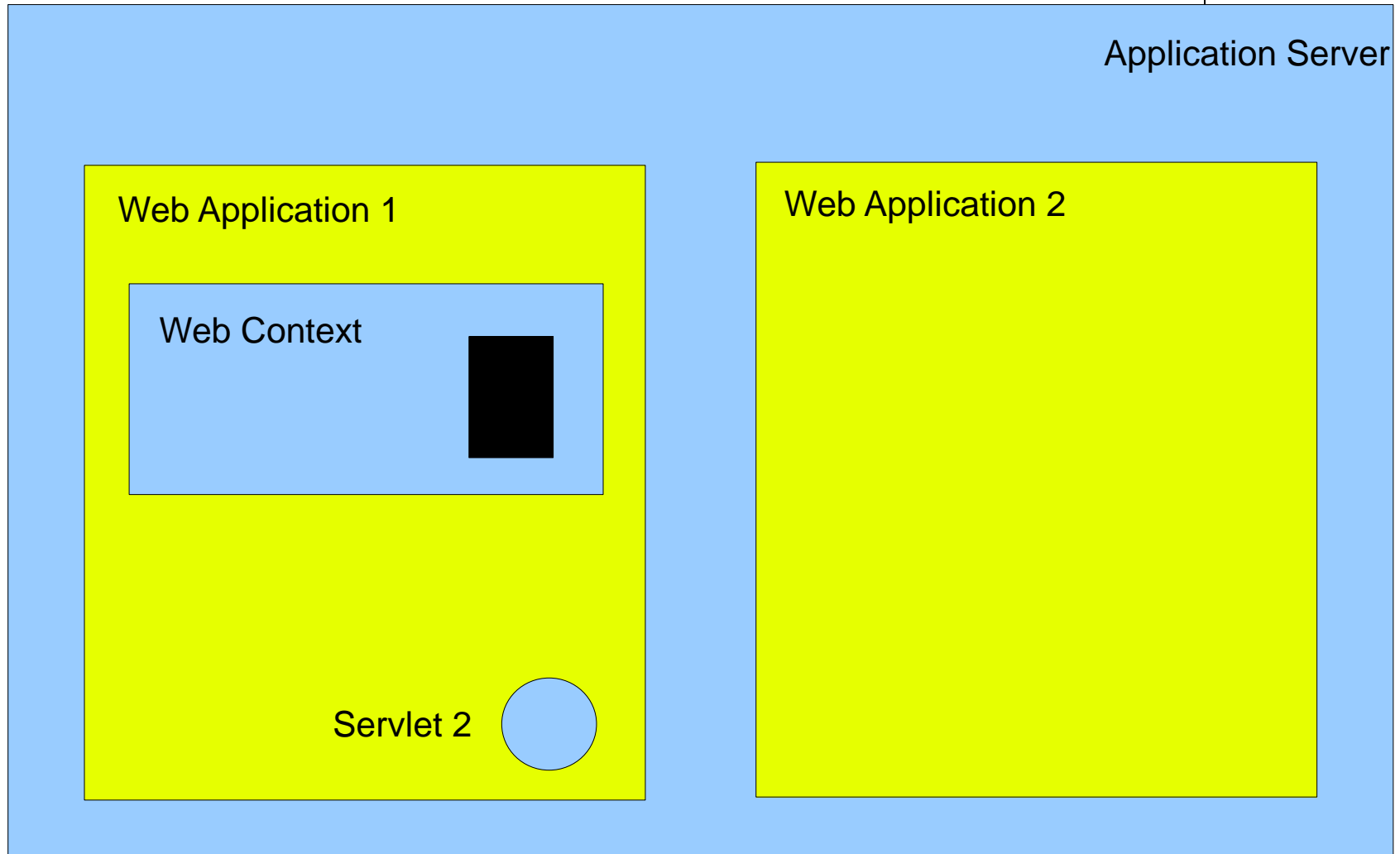
Web Contexts



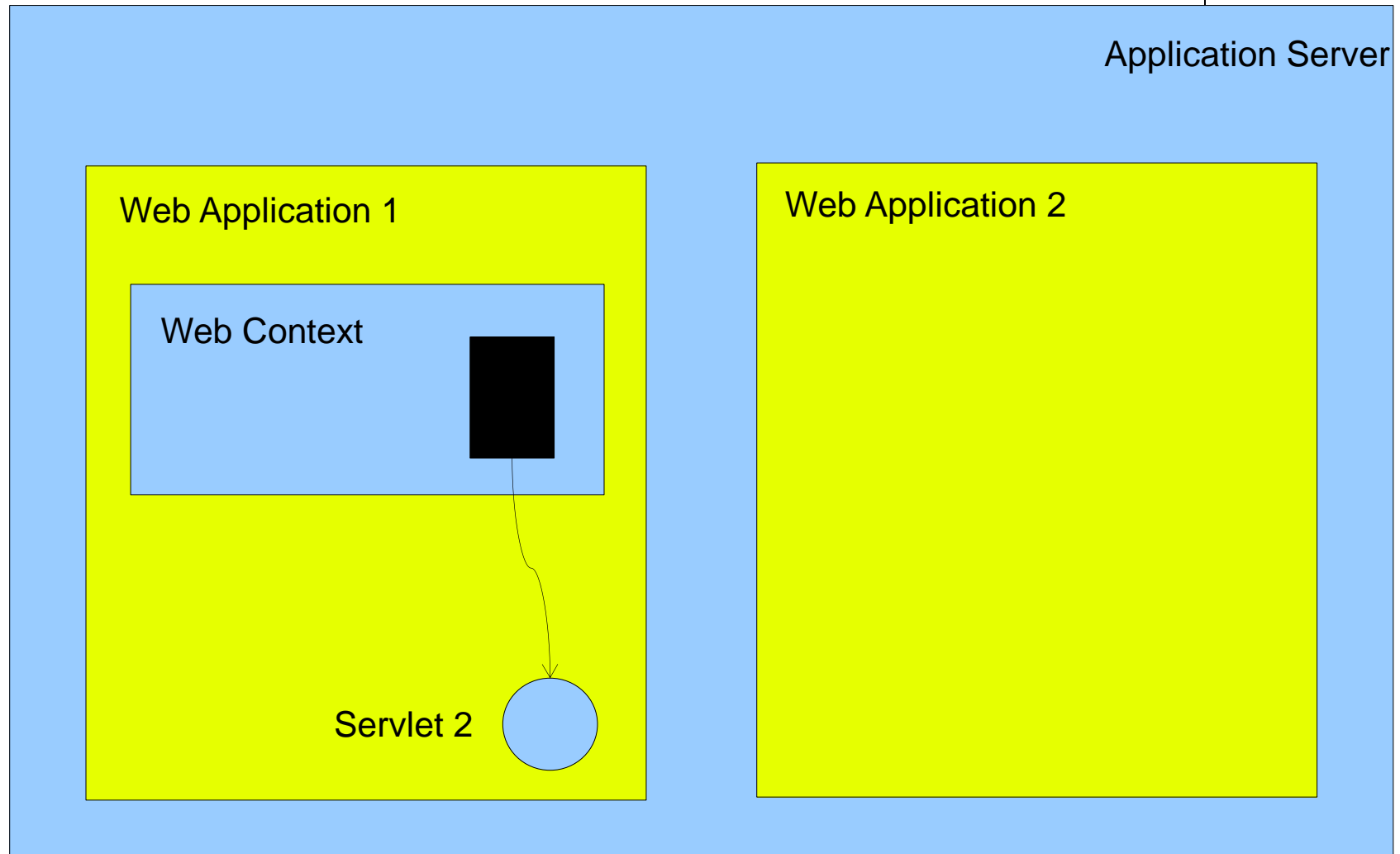
Web Contexts



Web Contexts



Web Contexts



- ❑ Tenemos cuatro posibles contextos en una aplicación web
- ❑ Application context
- ❑ Session context
- ❑ Request context
- ❑ Page context
 - Solo aplica a paginas JSP

- ❑ javax.servlet.ServletContext
- ❑ Desde cualquier Servlet, podemos obtener una referencia al contexto invocando el método:
 - ServletContext ctx = this.getServletContext();
- ❑ Provee los métodos
 - Object getAttribute(String key);
 - void setAttribute(String key, Object value);

ApplicationContext

```
public class HolaMundoServlet extends HttpServlet {  
  
    public void doGet(  
        HttpServletRequest request,  
        HttpServletResponse response) {  
  
        ServletContext ctx = getServletContext();  
        ctx.setAttribute("fechaProceso", new Date());  
  
        ... construccion de la respuesta ...  
    }  
}
```

ApplicationContext

```
public class ChauMundoServlet extends HttpServlet {  
  
    public void doGet(  
        HttpServletRequest request,  
        HttpServletResponse response) {  
  
        ServletContext ctx = getServletContext();  
        Date fecha =  
            (Date)ctx.getAttribute("fechaProceso");  
        ... construccion de la respuesta ...  
    }  
}
```

- ❑ HttpSession es el objeto que representa a la sesión del usuario
- ❑ Puede ser obtenido a partir del objeto Request en cada servlet
 - `HttpSession session = request.getSession();`
- ❑ Provee los métodos:
 - `Object getAttribute(String key);`
 - `void setAttribute(String key, Object value);`

❑ Ejemplos:

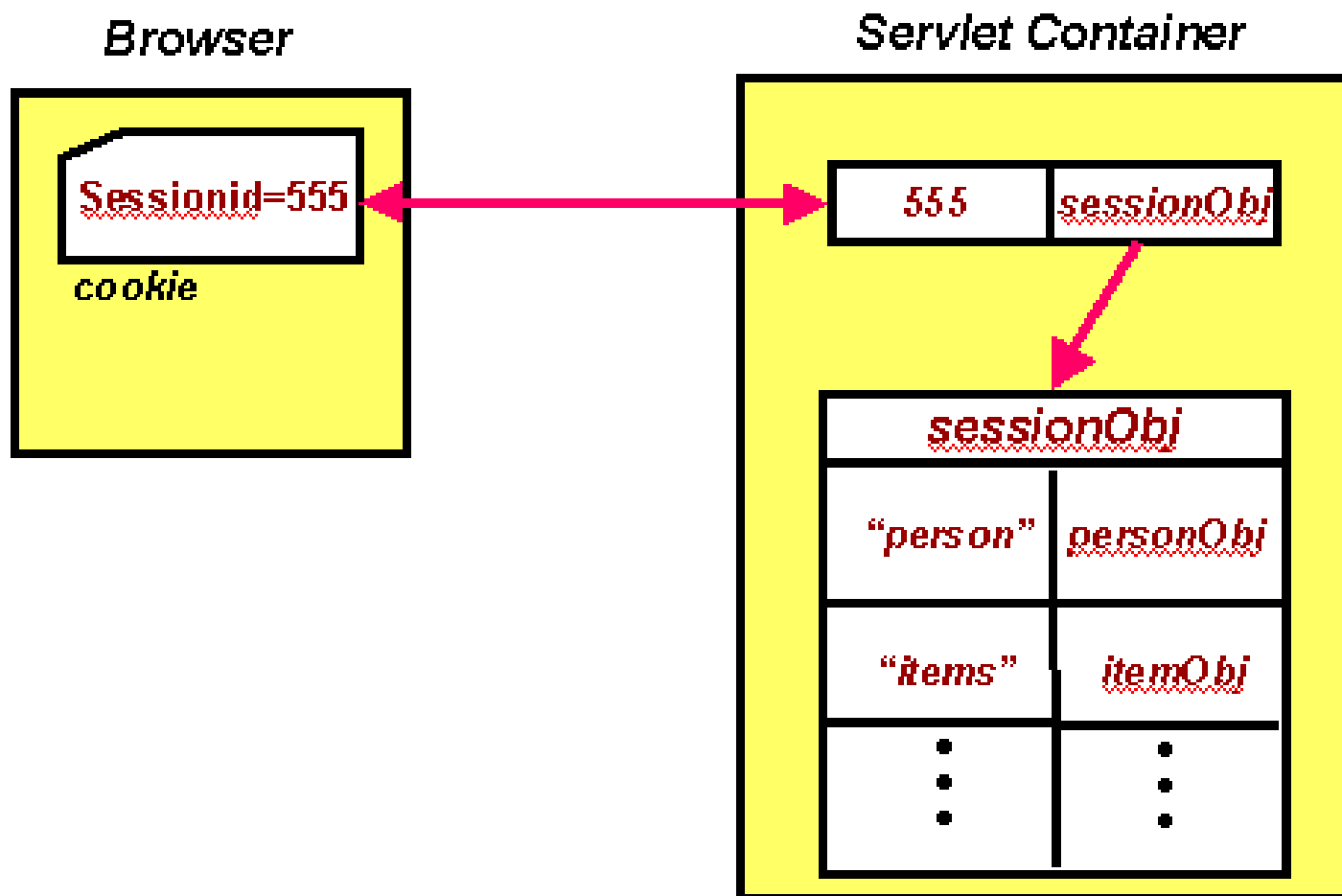
- Carrito carrito = new Carrito();
- HttpSession session = request.getSession();
- session.setAttribute("carrito", carrito);

❑ En otro servlet...

- Carrito carrito =
(Carrito)session.getAttribute("carrito");

- ❑ Los session context sirven para mantener información (estado) asociado a cada uno de los clientes conectados a una aplicación web
- ❑ Debido a que HTTP es un protocolo stateless, los requests seguidos de un mismo cliente no pueden relacionarse
- ❑ El web container necesita implementar un mecanismo de tracking que le permita identificar pedidos que vienen desde un mismo cliente

Session Context



- ❑ Un componente web puede delegar el procesamiento de un `HttpRequest` a otro componente web
- ❑ Usamos `javax.servlet.RequestDispatcher`
- ❑ Esta provee 2 métodos:
 - `void forward(ServletRequest request, ServletResponse response);`
 - `void include(ServletRequest request, ServletResponse response);`

- Las paginas JSP son documentos HTML, que pueden contener código Java embebido entre sus tags
- En los lugares de la pagina en los que deba generarse contenido dinámico, se colocan tags especiales, que indican la presencia de dicho contenido
- En general son de la forma `<% y %>`, `<%= y %>`, etc.

- Por ejemplo, podemos tener lo siguiente en una pagina JSP:

<html>

<body>

La hora actual es:

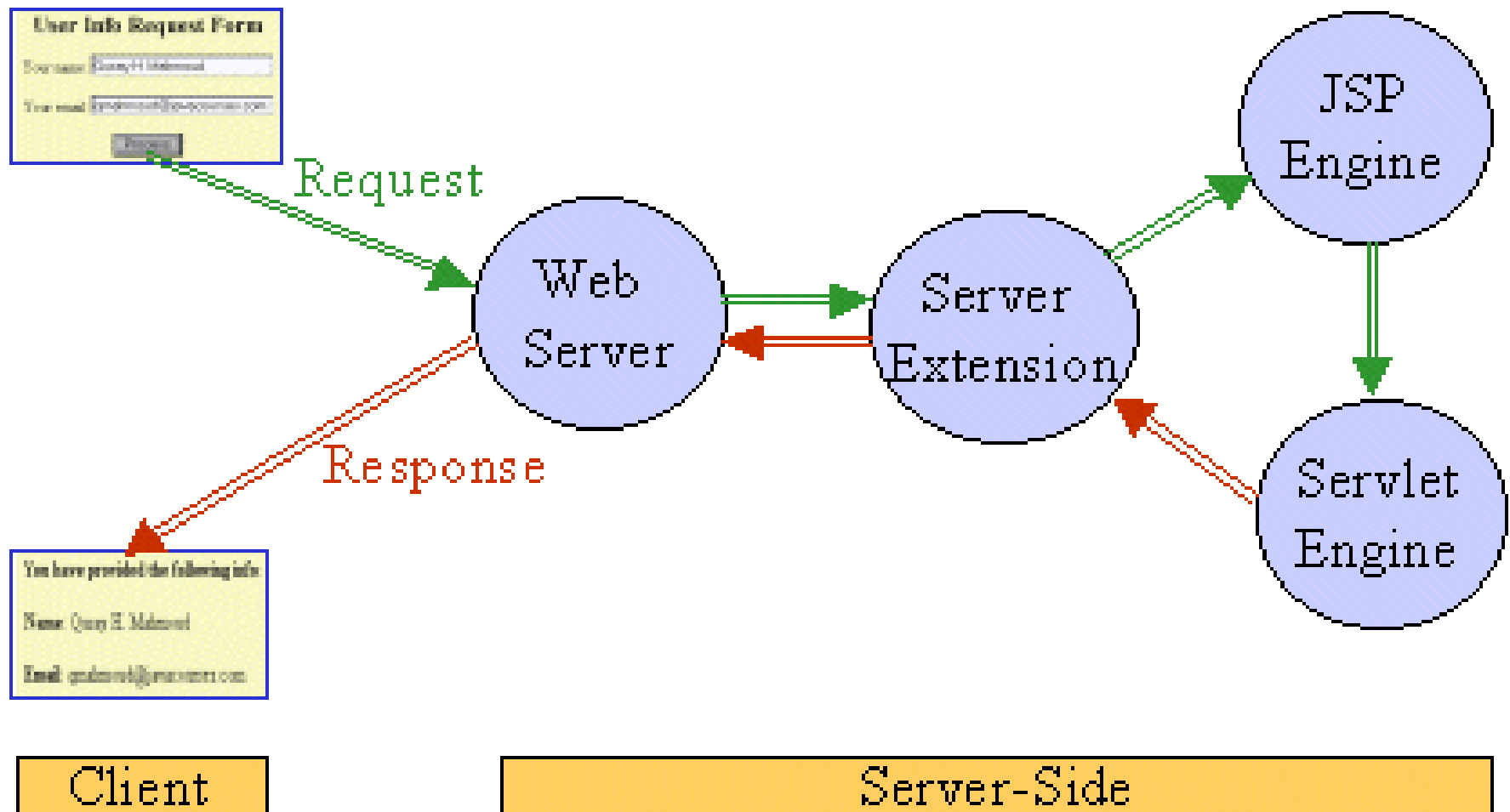
<%= new Date().toString() %>

</body>

</html>

- En general las paginas JSP reciben la extensión .jsp, pudiendo ser colocadas en cualquier lugar de la aplicación web
 - No requieren mappings
 - No tienen porque ir dentro del WEB-INF
- En general las paginas JSP tienen mas la apariencia de paginas HTMLs que de Servlets
 - Son document-oriented

Funcionamiento



Elementos de scripting

- Dentro de la pagina JSP podemos embeber diferentes construcciones en Java, para dar dinamismo a la pagina
- Los elementos de scripting permiten insertar código Java en el cuerpo del servlet generado
- Tenemos tres tipos de elementos:
 - Expressions de la forma `<%= %>`
 - Scriptlets de la forma `<% %>`
 - Declarations de la forma `<%! %>`

- Es utilizada para colocar el valor de una expresión Java directamente en la salida del servlet generado
- Es de la forma:
 - `<%= Java Expression %>`
- La expresión es evaluada, convertida a String (a través del método **toString**) y luego colocada en la salida del servlet

- La expresión es evaluada en tiempo de ejecución (al momento de hacer el request de la pagina)
- Por este motivo, la expresión tiene acceso a toda la información de dicho request
- Por ejemplo:
 - **Current time: `<%= new java.util.Date() %>`**

- Para simplificar las expresiones, existen una serie de objetos implícitos
 - request: HttpServletRequest
 - response: HttpServletResponse
 - session: HttpSession
 - out: PrintWriter asociado al response
- Por ejemplo:

Your hostname: <%= request.getRemoteHost() %>

JSP Scriptlets

- Si necesitamos realizar una tarea mas compleja que una expresión, entonces podemos insertar código Java arbitrario a través de un scriptlet
- Los scriptlets tienen la siguiente forma:

<%

int x = 0;

int y = 10;

out.print("La suma es: " + (x+y));

%>

JSP Scriptlets

- El código dentro de un scriptlet es insertado tal cual como fue escrito
- Cualquier código HTML estático antes o después del scriptlet es insertado sin cambios

```
<% if (Math.random() < 0.5) { %>
```

```
Have a <B>nice</B> day!
```

```
<% } else { %>
```

```
Have a <B>lousy</B> day!
```

```
<% } %>
```

- Una declaración permite colocar métodos y campos en el cuerpo principal del servlet generado
- Tienen la siguiente forma:
<%! Java Code %>
- Por ejemplo:
<%! private int accessCount = 0; %>
Accesses to page since server reboot:
<%= ++accessCount %>

JAVA SERVER FACES

Que es/provee JSF?

- ❑ Es un framework (estándar) para el desarrollo de aplicaciones web en la plataforma Java EE
 - Controles de UI
 - Independiente de la tecnología de renderización
 - Basado en MVC
 - Soporte de AJAX
 - Múltiples implementaciones y bibliotecas de extensión

Ejemplo...

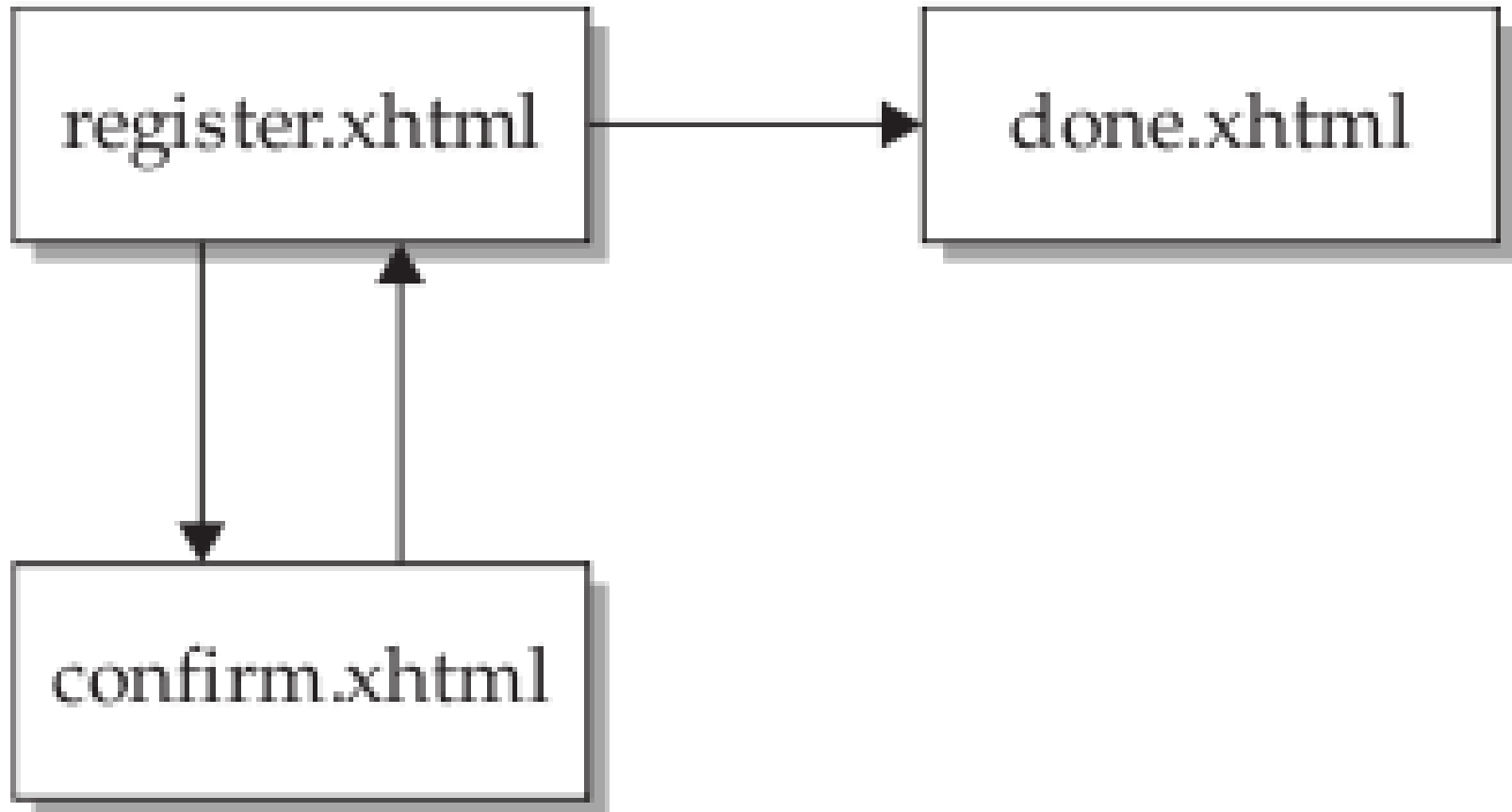
```
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBeanAjax.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBeanAjax.password}"/><br/>
  <h:commandButton value="Show Current Balance"
    action="#{bankingBeanAjax.showBalance}">
    <f:ajax execute="@form" render="ajaxMessage1"/>
  </h:commandButton>
  <br/>
  <h2><h:outputText value="#{bankingBeanAjax.message}"
    id="ajaxMessage1"/></h2>
</h:form>
```

Ejemplo...

```
<h:form>
  Customer ID:
  <h:inputText value="#{bankingBeanAjax.customerId}"/><br/>
  Password:
  <h:inputSecret value="#{bankingBeanAjax.password}"/><br/>
  <h:commandButton value="Show Current Balance"
    action="#{bankingBeanAjax.showBalance}">
    <f:ajax execute="@form" render="ajaxMessage1"/>
  </h:commandButton>
  <br/>
  <h2><h:outputText value="#{bankingBeanAjax.message}"
    id="ajaxMessage1"/></h2>
</h:form>
```

```
@ManagedBean
public class BankingBeanAjax extends BankingBeanBase {
    private String message = "";
    public String getMessage() {
        return(message);
    }
    public void setMessage(String message) {
        this.message = message;
    }
    public String showBalance() {
        if (!password.equals("secret")) {
            message = "Incorrect password";
        } else {
            CustomerLookupService service =
                new CustomerSimpleMap();
            customer = service.findCustomer(customerId);
            if (customer == null) {
                message = "Unknown customer";
            } else {
                message =
                    String.format("Balance for %s %s is $%,.2f", customer.getFirstName(),
                                customer.getLastName(),
                                customer.getBalance());
            }
        }
        return(null);
    }
}
```

Un ejemplo



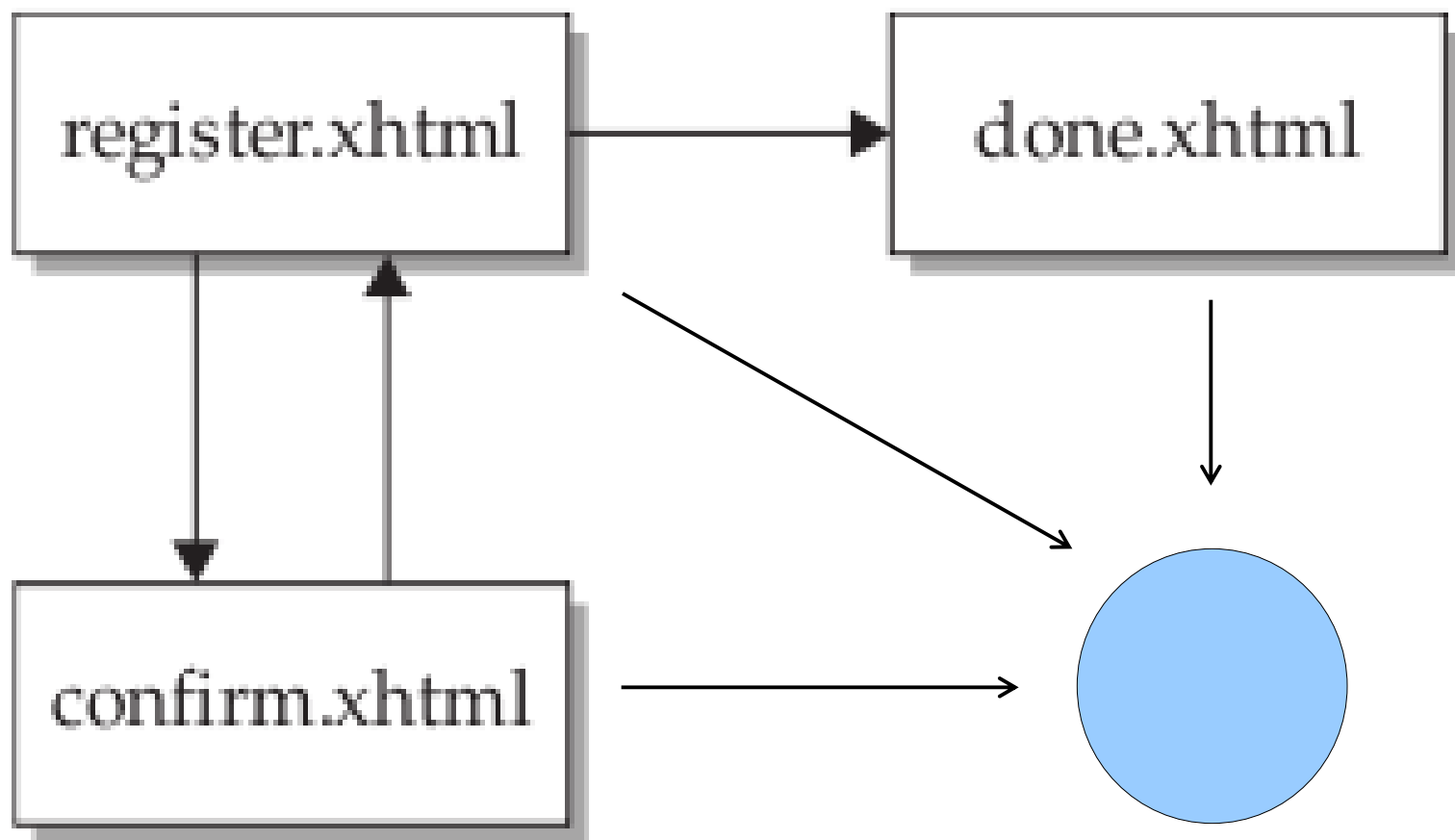
Elementos de la aplicación

- ❑ web.xml
 - Descriptor general de la aplicación web, se usa para establecer las paginas de bienvenida de la aplicación
- ❑ register.xhtml
 - Contiene el formulario de registro de usuario
- ❑ confirm.xhtml
 - Contiene los datos ingresados antes, luego de un proceso de validación En caso de error, se vuelve a la pagina anterior

Elementos de la aplicación

- ❑ done.xhtml
 - Muestra un mensaje de éxito al finalizar el ingreso del usuario al sistema
- ❑ UserBean.java
 - Contiene los datos ingresados por el usuario, almacenados temporalmente por la aplicación
 - Contiene también unos métodos simples de validación sobre los datos ingresados

Estructura de la aplicación



UserBean : JavaBean@ServerSide

JSF Registration App

Registration Form

First Name:

Last Name:

Sex: ☐ ☐

Date of Birth: (mm-dd-yy)

Email Address:

Service Level:

register.xhtml

```
<tr>
  <td>First Name:</td>
  <td><h:inputText label="First Name" id="fname" value="#{userBean.firstName}"
    required="true"/>
    <h:message for="fname" /></td>
</tr>
<tr>
  <td>Last Name:</td>
  <td><h:inputText label="Last Name" id="lname" value="#{userBean.lastName}"
    required="true"/>
    <h:message for="lname" /></td>
</tr>
<tr>
  <td>Sex:</td>
  <td><h:selectOneRadio label="Sex" id="sex" value="#{userBean.sex}" required="true">
    <f:selectItem itemLabel="Male" itemValue="male" />
    <f:selectItem itemLabel="Female" itemValue="female" />
  </h:selectOneRadio>
  <h:message for="sex" /></td>
</tr>
```

```

<tr>
  <td>Date of Birth:</td>
  <td><h:inputText label="Date of Birth" id="dob" value="#{userBean.dob}"
        required="true">
    <f:convertDateTime pattern="MM-dd-yy" />
  </h:inputText> (mm-dd-yy)
  <h:message for="dob" />
</td>
</tr>
<tr>
  <td>Email Address:</td>
  <td><h:inputText label="Email Address" id="email" value="#{userBean.email}"
        required="true" validator="#{userBean.validateEmail}"/>
  <h:message for="email" />
</td>
</tr>
<tr>
  <td>Service Level:</td>
  <td><h:selectOneMenu label="Service Level" value="#{userBean.serviceLevel}">
    <f:selectItem itemLabel="Medium" itemValue="medium" />
    <f:selectItem itemLabel="Basic" itemValue="basic" />
    <f:selectItem itemLabel="Premium" itemValue="premium" />
  </h:selectOneMenu>
</td>
</tr>

```


UserBean (Managed bean)

```
@ManagedBean
@SessionScoped
public class UserBean {
    protected String firstName;
    protected String lastName;
    protected Date dob;
    protected String sex;
    protected String email;
    protected String serviceLevel = "medium";
}
```

UserBean (Managed bean)

```
public String addConfirmedUser() {  
    boolean added = BusinessService.addUser(this.fir  
        ....);  
  
    FacesMessage doneMessage = null;  
    String outcome = null;  
    if (added) {  
        doneMessage = new FacesMessage("Successfu  
        outcome = "done";  
    } else {  
        doneMessage = new FacesMessage("Failed to  
        outcome = "register";  
    }  
    FacesContext.getCurrentInstance().addMessage(nu  
    return outcome;  
}
```


JSF Registration App

Registration Form

First Name: First Name: Validation Error: Value is required.

Last Name: Last Name: Validation Error: Value is required.

Sex: ☐ ☐ Sex: Validation Error: Value is required.

Date of Birth: (mm-dd-yy) Date of Birth: 'as-df-23' could not be understood as a date. Example: 04-08-09

Email Address: Email Address: Validation Error: Value is required.

Service Level:

- First Name: Validation Error: Value is required.
- Last Name: Validation Error: Value is required.
- Sex: Validation Error: Value is required.
- Date of Birth: 'as-df-23' could not be understood as a date.
- Email Address: Validation Error: Value is required.

JSF Registration App

Registration Confirmation

First Name: John

Last Name: Doe

Sex: male

Date of Birth: Thu Jan 31 01:00:00 CET 1980

Email Address: jdoe@foo.com

Service Level: medium

Edit

Confirm

confirm.xhtml

- ❑ Los datos son desplegados utilizando el componente `<h:outputText>`
 - `<h:outputText value="#{userBean.firstName}"/>`
 - `<h:outputText value="#{userBean.lastName}"/>`
- ❑ Los botones con las acciones disparadas desde la pagina son codificadas con el componente `<h:commandButton>`
 - `<h:commandButton value="Edit" action="register" />`
 - `<h:commandButton value="Confirm"`
`action="#{userBean.addConfirmedUser}" />`

JSF Registration App

Registration Confirmation

- Successfully added new user

First Name: John

Last Name: Doe

Sex: male

Date of Birth: Thu Jan 31 01:00:00 CET 1980

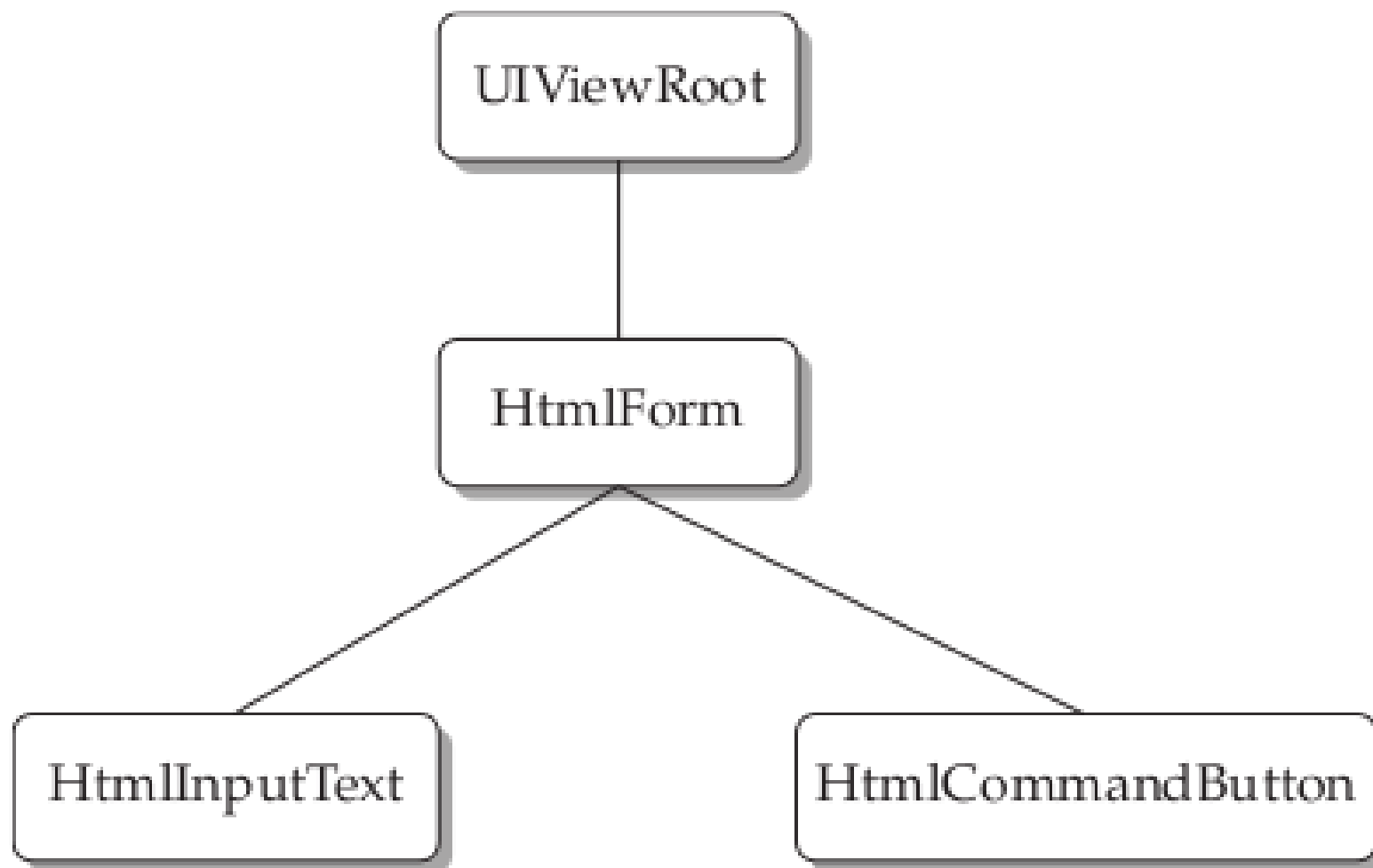
Email Address: jdoe@foo.com

Service Level: medium

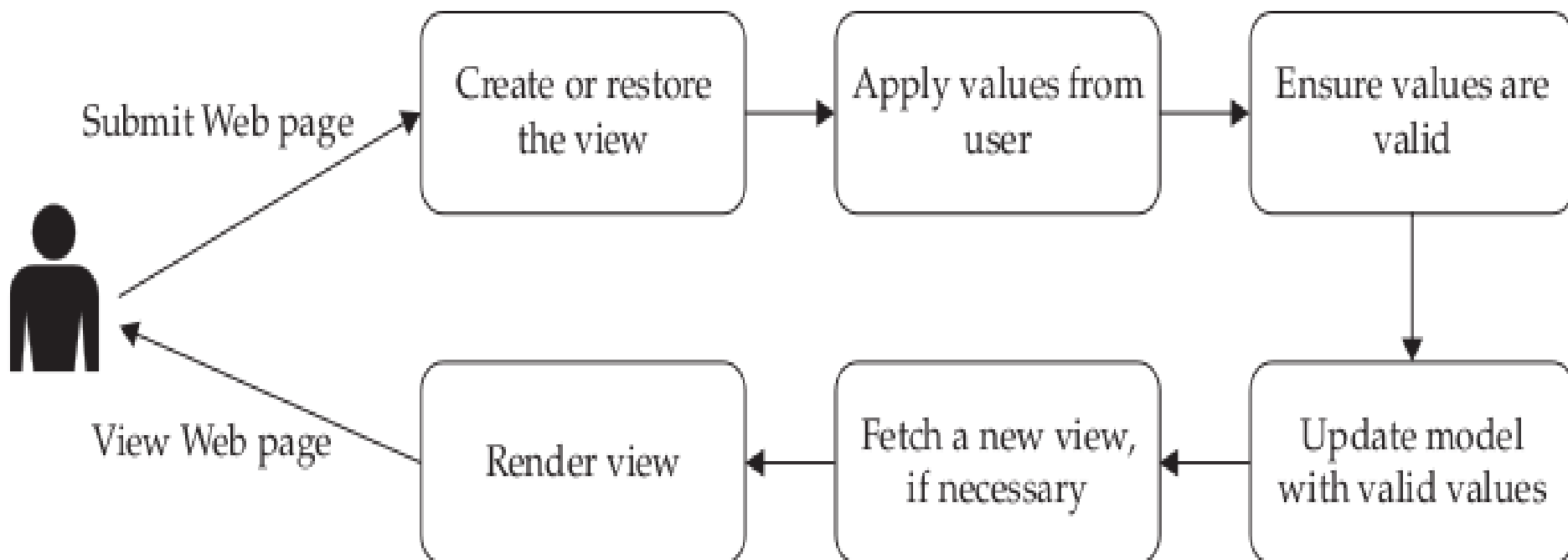
Entonces...si tenemos esta pagina

```
<html xmlns=http://www.w3.org/1999/xhtml
      xmlns:h=http://java.sun.com/jsf/html
      xmlns:f="http://java.sun.com/jsf/core">
  <body>
    <h:form>
      <h2>A Simple JSF Page</h2>
      <h:inputText value="#{modelBean.username}"/>
      <h:commandButton value="Click Here"/>
    </h:form>
  </body>
</html>
```

Árbol de componentes...



Ciclo de vida



- ❑ Basado en la idea de un conjunto de “reglas de navegación”
- ❑ Estas reglas definen
 - Una pagina de inicio (from-view-id)
 - Uno o mas casos de navegación, los cuales tienen
 - Un String denominado “outcome”
 - Una pagina destino (to-view-id)

Regla de navegación

```
<navigation-rule>  
  <from-view-id>/page1.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/page2.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

Regla de navegación

```
<navigation-rule>  
  <from-view-id>/page1.xhtml</from-view-id>  
  <navigation-case>  
    <from-outcome>success</from-outcome>  
    <to-view-id>/page2.xhtml</to-view-id>  
  </navigation-case>  
  <navigation-case>  
    <from-outcome>failure</from-outcome>  
    <to-view-id>/page3.xhtml</to-view-id>  
  </navigation-case>  
</navigation-rule>
```

Y el outcome?

```
public String addConfirmedUser() {  
    boolean added = BusinessService.addUser(this.fir  
        ....);  
  
    FacesMessage doneMessage = null;  
    String outcome = null;  
    if (added) {  
        doneMessage = new FacesMessage("Successfu  
        outcome = "done";  
    } else {  
        doneMessage = new FacesMessage("Failed to  
        outcome = "register";  
    }  
    FacesContext.getCurrentInstance().addMessage(nul  
    return outcome;  
}
```

Estructura de una pagina

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//E  
"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>
```

```
...
```

```
</h:head>
```

```
<h:body>
```

```
...
```

```
<h:form>
```

```
...
```

```
</h:form>
```

```
...
```

```
</h:body>
```

```
</html>
```

Siempre son en formato XHTML

La dirección de acceso, es diferente a la ruta física

Se acceden con extensión *.jsf

Pero se escriben como paginas XHTML, *.xhtml

Managed Beans (@Server)

@ManagedBean

```
public class SomeBean {  
    private String someProperty;  
  
    public String getSomeProperty() { ... }  
    public void setSomeProperty() { ... }  
    public String actionControllerMethod() {  
        ...  
    }  
    // Other methods  
}
```

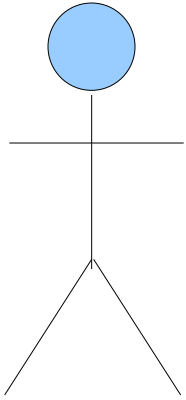
Son JavaBeans (POJOs)

Siempre para cada propiedad, tenemos un getter y setter

La visión desde afuera de las propiedades, se hace en base al setter y getter

Flujo de control / Ciclo de vida

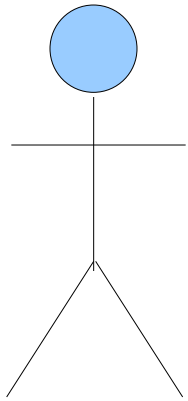
- ❑ Es el conjunto de pasos ordenados que se llevan a cabo desde que ejecutamos una tarea en una pagina, y obtenemos un resultado en el browser
- ❑ Implica toda una serie de pasos que ocurren entre el cliente y el servidor
- ❑ El objetivo es que sea transparente para el usuario



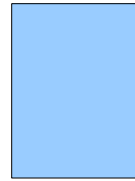
pagina.xhtml

<h:commandButton

action="#{someBean.someMethod}"

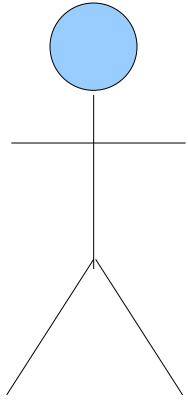


GET pagina.jsf



pagina.xhtml

```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```

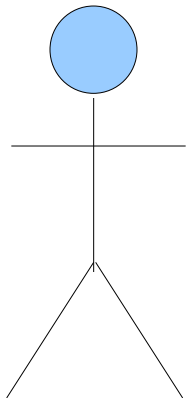
HTML dinámico

Correspondiente a la pagina JSF



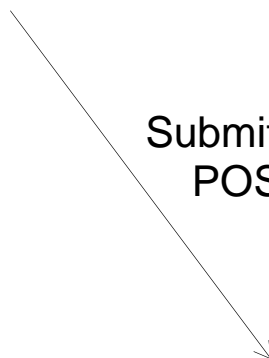
pagina.xhtml

```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```

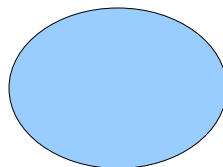


pagina.xhtml

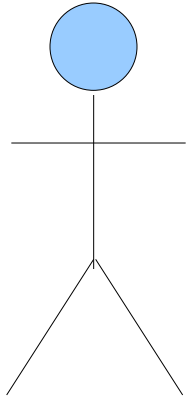
```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```



Submit del formulario
POST pagina.jsf



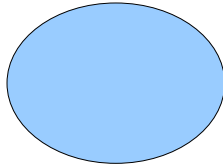
Instanciamos someBean



pagina.xhtml

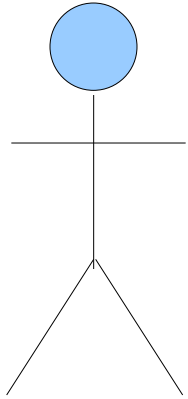
```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```

Submit del formulario
POST pagina.jsf



Instanciamos someBean

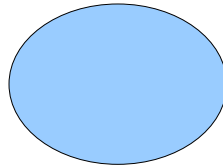
Ejecutamos la acción
(someMethod)



pagina.xhtml

```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```

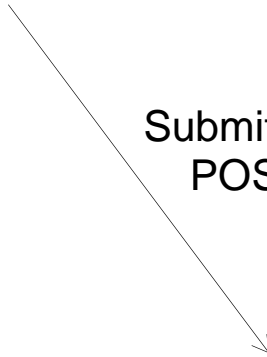
Submit del formulario
POST pagina.jsf

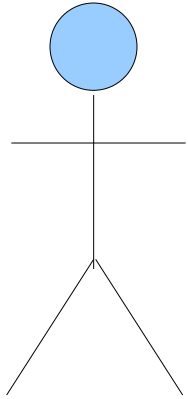


Instanciamos someBean

Ejecutamos la acción
(someMethod)

Acceso a lógica de negocio
(EJB, WebService, BD)

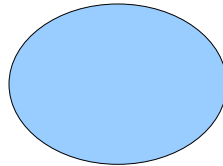




pagina.xhtml

```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```

Submit del formulario
POST pagina.jsf



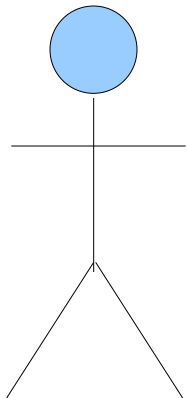
Instanciamos someBean



Ejecutamos la acción
(someMethod)



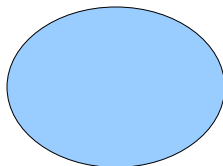
Valor
retornado



pagina.xhtml

```
<h:commandButton ....  
    action="#{someBean.someMethod}"
```

Submit del formulario
POST pagina.jsf



Instanciamos someBean

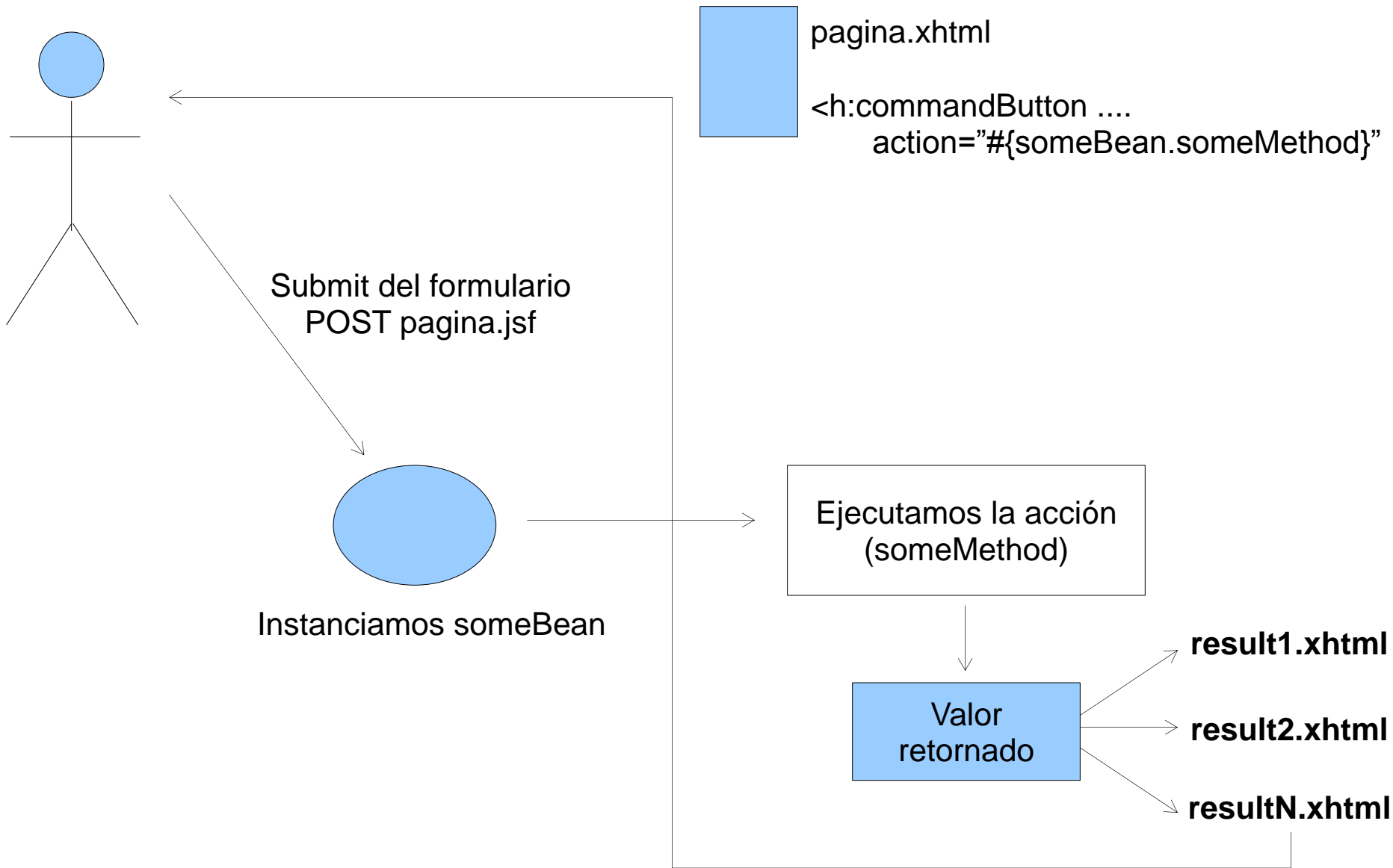
Ejecutamos la acción
(someMethod)

Valor
retornado

result1.xhtml

result2.xhtml

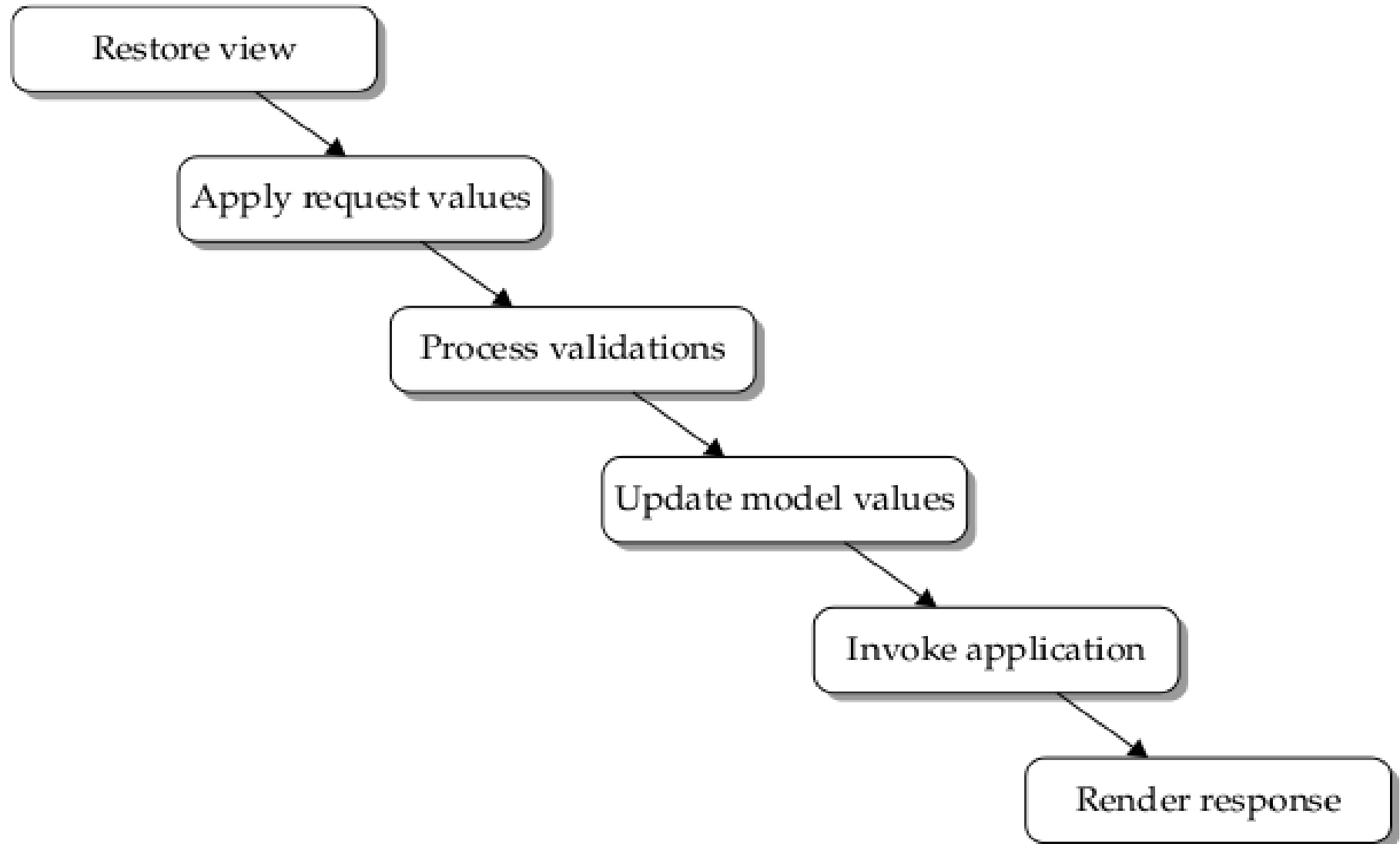
resultN.xhtml



Resultado del submit. Corresponde a la pagina ResultN.xhtml

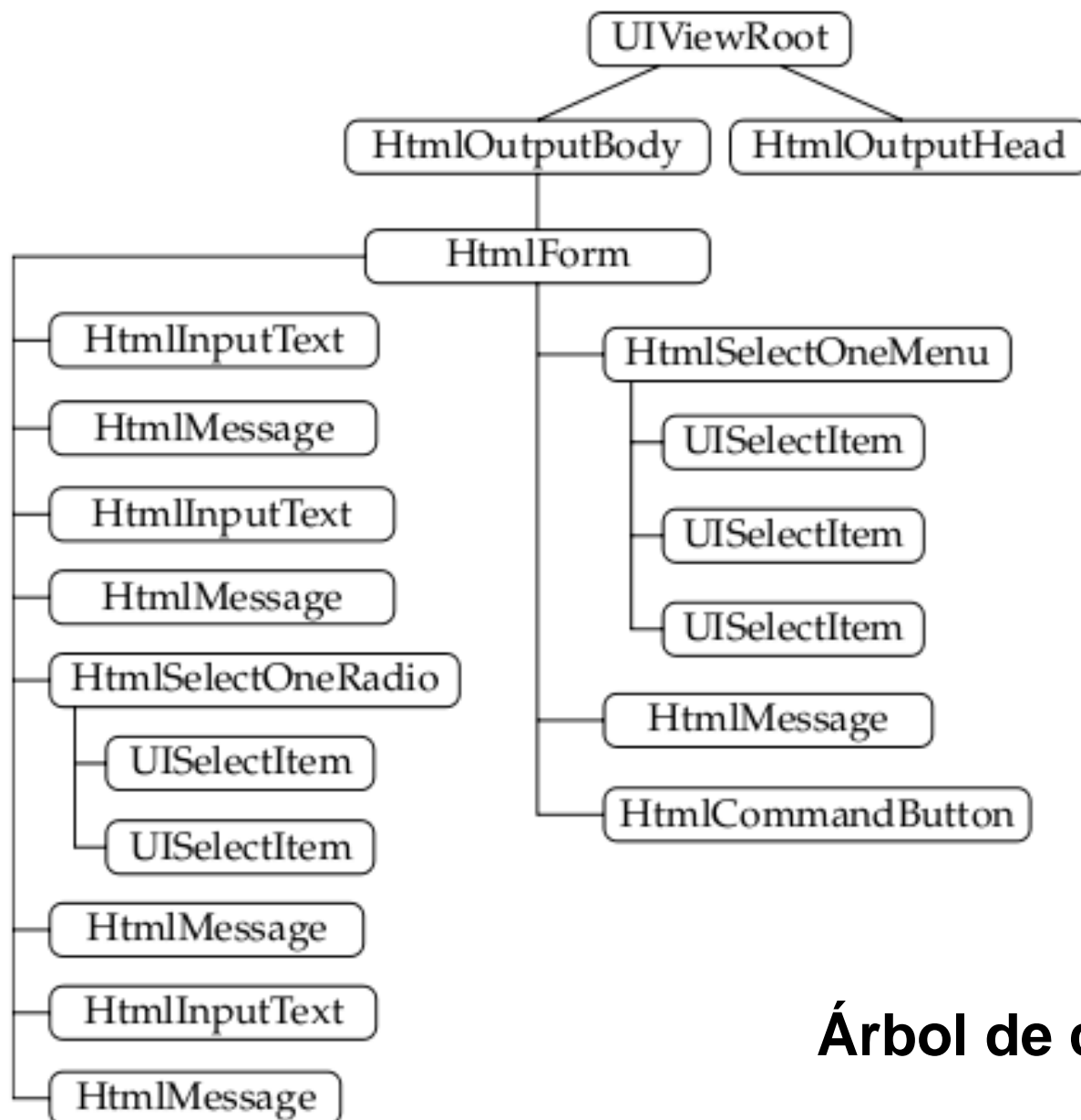
- ❑ El ciclo de vida se encuentra dividido en fases, cada una de las cuales se encarga de una tarea específica del proceso server-side
 - Restore View
 - Apply Request Values
 - Process Validations
 - Update Model Values
 - Invoke Application
 - Render Response

Fases



Restore View

- ❑ La Faces View, es una representación server side de los componentes que forman la interfaz de usuario
- ❑ Es un árbol de componentes de UI
- ❑ Es tarea de esta etapa, restaurar este árbol de componentes de una interacción anterior, o crear un nuevo árbol para un nuevo request



Árbol de componentes

Apply Request Values

- ❑ El objetivo de este paso es procesar los diferentes parámetros que puedan venir dentro del request
- ❑ Cada componente de la View esta listo en este momento para recibir una actualización de los valores que puedan venir del cliente
- ❑ Estos valores siempre llegan de la forma name-value, donde el name y el value son de tipo String

JSF Registration App

Registration Form

First Name:

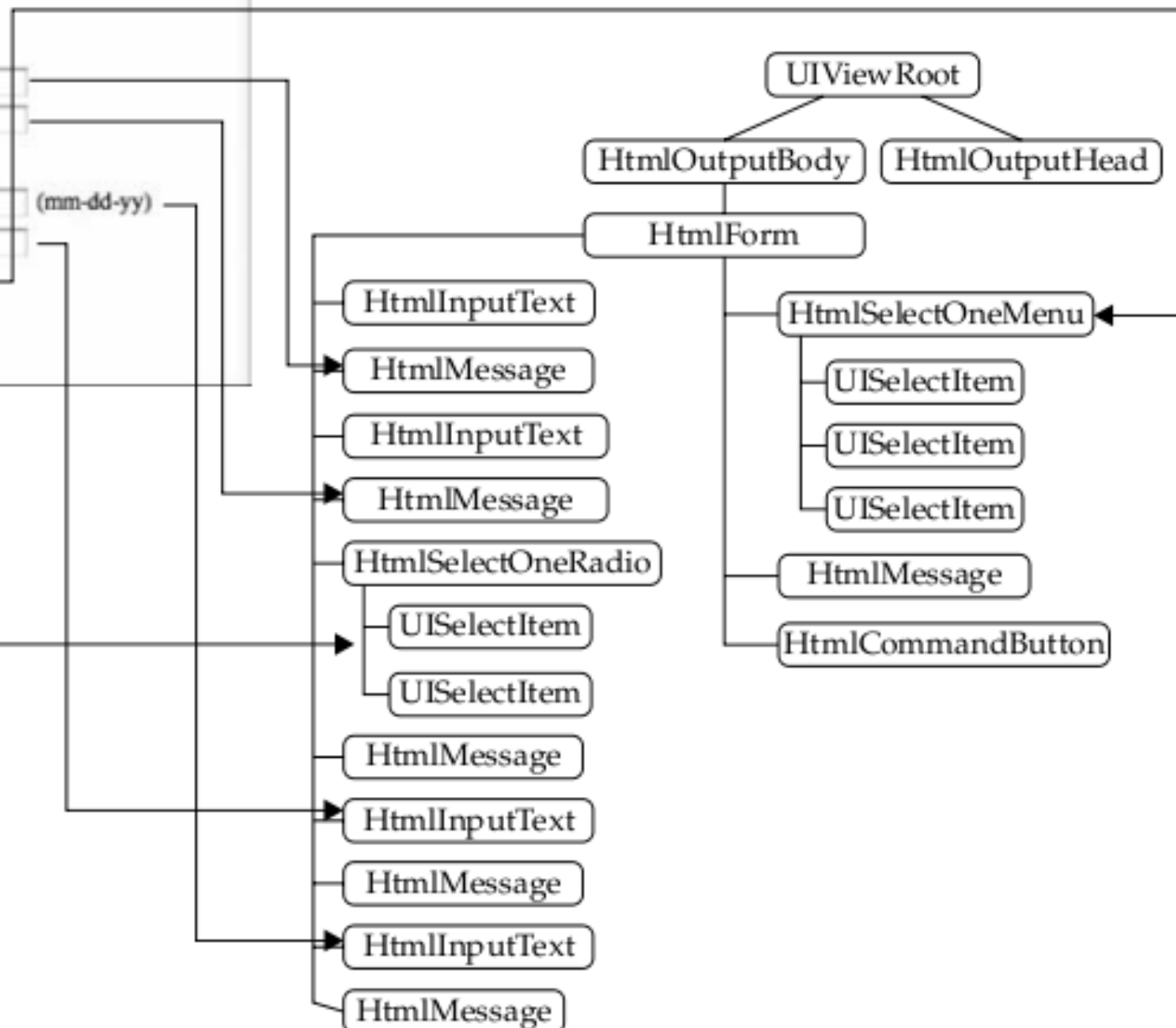
Last Name:

Sex: ☒ ☐

Date of Birth: (mm-dd-yy)

Email Address:

Service Level:



- ❑ Esta etapa se encarga de la validación y conversión de los datos
- ❑ Los componentes pueden tener asociados validadores y convertidores
- ❑ Todo componente que falle en su validación
 - Tendrá su atributo valid en false
 - Tendrá un mensaje asociado (FacesMessage) el cual sera encolado en el FacesContext
 - Cuando la respuesta sea enviada, estos mensajes podrán ser desplegados

Process Validations

A Simple JavaServer Faces Registration Application

http://localhost:8080/jsfreg/faces/register.xhtml

JSF Registration App

Registration Form

First Name: First Name: Validation Error: Value is required.

Last Name: Last Name: Validation Error: Value is required.

Sex: ☐ Male ☐ Female
Sex: Validation Error: Value is required.

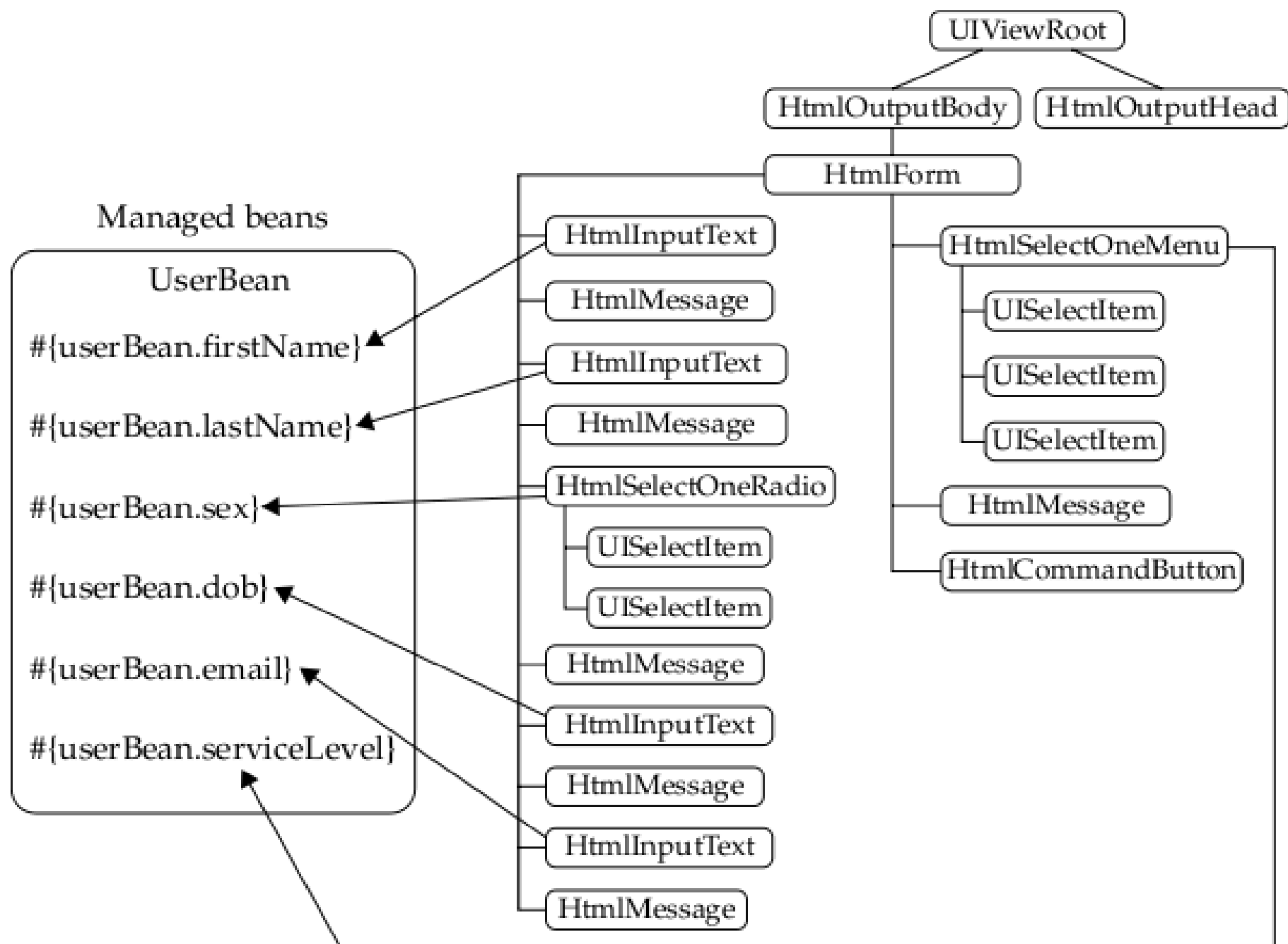
Date of Birth: (mm-dd-yy) Date of Birth: Validation Error: Value is required.

Email Address: Email Address: Validation Error: Value is required.

Service Level:

Update Model Values

- ❑ Asumiendo que la validación y conversión fue superada, es hora de colocar la información en el modelo
- ❑ Este proceso se realiza para los objetos que han sido asociados (value binding) a los valores de los componentes de UI
- ❑ Esta es una de las etapas mas interesantes, ya que aquí es donde se produce parte de la magia de JSF



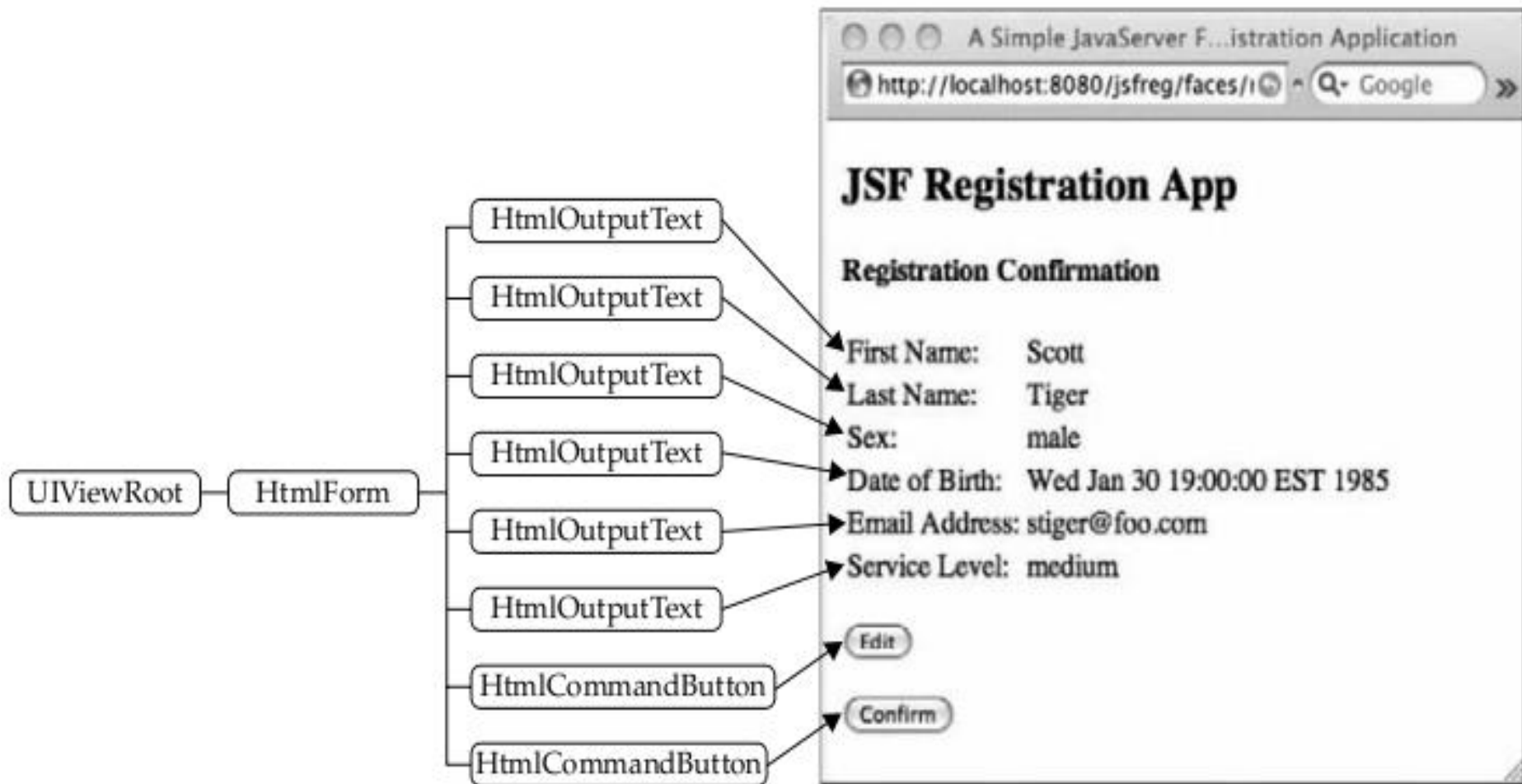
Invoke Application

- ❑ En esta etapa se invoca el código asociado a los botones o links en la pagina
- ❑ El código de los beans a los cuales esta asociado un componente de pantalla es ejecutado
- ❑ En esta etapa también ocurren las navegaciones a otras paginas
 - Cuando los Action Methods devuelven un String

Render Response

- ❑ Es la etapa final del proceso, cuando la respuesta es construida y entregada al cliente
- ❑ En esta etapa también se almacena el estado de la View, de forma de poder restaurarla en subsiguientes requests
- ❑ En esta etapa es donde se realiza la unión del código estático en la vista, junto con el código dinámico de la misma
 - En general el desarrollador no debe intervenir en esta etapa

Render Response



Actions y ActionListeners

- ❑ Los componentes que disparan eventos en JSF, son los `commandButtons` (botones) y los `commandLinks`
- ❑ Ambos tienen los atributos `actionListener` y `action`

```
<h:commandButton value="Search"  
                  actionListener="#{flight.confirm}"  
                  action="#{flight.search}"/>
```

Actions y ActionListener

- ❑ Cuando se presiona el `commandButton`, JSF invoca el `ActionListener` durante la fase `Invoke Application`
- ❑ En este podemos realizar cualquier procesamiento relacionado con el botón que presionamos
- ❑ Firma del método `Action Listener`
 - `public void <Nombre>(ActionEvent evt)`

```
<h:commandButton id="submitButton"  
    value="Submit" action="#{normal.outcome}"  
    actionListener="#{normal.printIt}" />
```

```
@ManagedBean(name="normal")  
@SessionScoped  
public class NormalBean{  
  
    public String buttonId;  
  
    public void printIt(ActionEvent event){  
        buttonId = event.getComponent().getClientId();  
        // Hacer algo con el componente  
    }  
  
    public String outcome(){  
        return "result";  
    }  
}
```

Actions y ActionListeners

```
<h:commandButton id="submitButton"
    value="Submit" action="#{normal.outcome}" >
    <f:actionListener type="com.mkyong.NormalActionListener" />
</h:commandButton>
```

```
import javax.faces.event.AbortProcessingException;
import javax.faces.event.ActionEvent;
import javax.faces.event.ActionListener;
```

```
public class NormalActionListener implements ActionListener {

    @Override
    public void processAction(ActionEvent event) throws AbortProcessingException {
        System.out.println("Any use case here?");
    }
}
```


Immediate

A Simple JavaServer Faces Registration Application

http://localhost:8080/jsfreg/faces/register.xhtml

JSF Registration App

Registration Form

First Name: First Name: Validation Error: Value is required.

Last Name: Last Name: Validation Error: Value is required.

Sex: ☐ Male ☐ Female
Sex: Validation Error: Value is required.

Date of Birth: (mm-dd-yy) Date of Birth: Validation Error: Value is required.

Email Address: Email Address: Validation Error: Value is required.

Service Level:

- ❑ Si Immediate esta en true, entonces el ciclo de vida se saltea cualquier etapa de validación y conversión, navegando directamente a donde nos interesa
- ❑ `<h:commandButton value="Cancel" action="main" immediate="true" />`

- ❑ Es un componente que permite interceptar las etapas del ciclo de vida de JSF
- ❑ Es muy similar a un Filter en el caso de Servlets
- ❑ El listener permite ejecutar código antes y después de un cambio de etapa del ciclo de vida de JSF

```
public class MyListener implements PhaseListener {  
  
    public PhaseId getPhaseId() {  
        return PhaseId.RENDER_RESPONSE;  
    }  
  
    public void beforePhase(PhaseEvent event) {  
        ...  
    }  
  
    public void afterPhase(PhaseEvent event) {  
        ...  
    }  
}
```

- ❑ Debemos configurarlo en el web.xml

```
<lifecycle>  
  <phase-listener>  
    com.test.application.MyListener  
  </phase-listener>  
</lifecycle>
```

Managed Bean

```
import javax.faces.bean.ManagedBean
import javax.faces.bean.SessionScoped

@ManagedBean
@SessionScoped
public class UserBean {

    ...

}
```

Managed Bean

- ❑ El nombre del bean determina como las expresiones EL accederán a la información de este
- ❑ El scope determina el tiempo/lugar de vida del bean, para otros componentes de la aplicación

Managed Bean

- Lo anterior es equivalente a:

```
import javax.faces.bean.ManagedBean
import javax.faces.bean.SessionScoped

@ManagedBean(name="userBean")
@SessionScoped
public class UserBean {

    ...

}
```


□ none o @NoneScoped

- Son instanciados en demanda por otro managed bean
- Seguirán vivos en el sistema, mientras el bean que los referencee siga vivo

□ request o @RequestScoped

- Las instancias de los managed beans con este scope existirán mientras el HTTP request del pedido actual exista

- ❑ view o @ViewScoped
 - Las instancias de los managed beans estarán asociadas a la vista actual
 - Si el usuario navega fuera de la vista actual, los beans registrados serán eliminados
- ❑ session o @SessionScoped
 - Los managed beans estarán asociados a la sesión del usuario
- ❑ application o @ApplicationScoped
 - Como antes, pero las instancias se almacenan en el application scope

Value Expressions

- ❑ Son la forma más común de uso de las expresiones EL
- ❑ Permiten obtener valores evaluados dinámicamente o establecer el valor de una propiedad de un bean, usando siempre una expresión compacta
- ❑ Por ejemplo:
 - `<h:outputText value="#{userBean.firstName}"/>`

- ❑ Pueden ser usadas también para actualizar las propiedades de los managed beans
- ❑ Por ejemplo, si un componente implementa EditableValueHolder, y esta asociado a una propiedad:
 - `<h:inputText value="#{userBean.firstName}"/>`
- ❑ En el momento del postback al servidor, si el valor ha cambiado, sera aplicado al managed bean

Method Expressions

- ❑ Son utilizadas para invocar métodos públicos no estáticos de los managed beans
- ❑ Por ejemplo:
 - `<h:commandButton value="Confirm" action="#{userBean.addConfirmedUser}" />`
- ❑ Si es necesario, pueden pasarse parámetros a los métodos del bean

`<p>`

The text after the colon comes from the invocation of a method via the EL:

`#{model.generateSentance(param.word1, param.word2)}`

`</p>`

Modelo de navegación

- ❑ Es uno de los aspectos mas elegantes de JSF
- ❑ Permite especificar el flujo de paginas completo de la aplicación
- ❑ Tenemos dos tipos de Navegación
 - Navegación implícita
 - Navegación basada en reglas XML

Navegación implícita

```
<!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:f="http://java.sun.com/jsf/core">
<h:head>
  <title>A Simple JavaServer Faces Registration Application</title>
</h:head>
<h:body>
  <h:form>
    <h2>JSF Registration App</h2>
    <h4>Registration Form</h4>
    <table>...Irrelevant content not shown...</table>
    <p><h:messages /></p>
    <p><h:commandButton value="Register" action="confirm" /></p>
  </h:form>
</h:body>
</html>
```

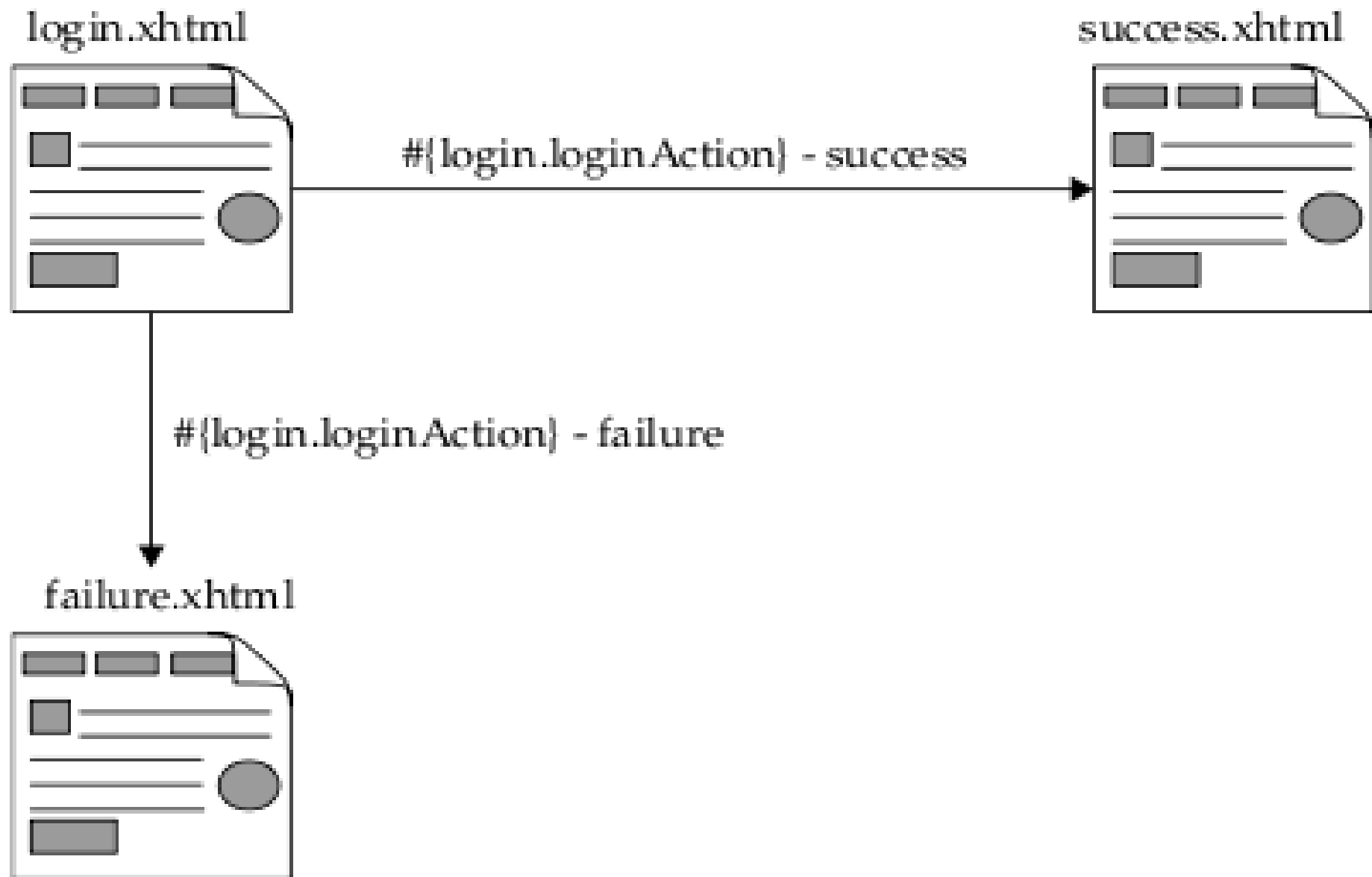
^

Navegación implícita

```
public String addConfirmedUser() {  
    boolean added = true; // actual application may fail to add user  
    FacesMessage doneMessage = null;  
    String outcome = null;  
    if (added) {  
        doneMessage = new FacesMessage("Successfully added new user");  
        outcome = "done";  
    } else {  
        doneMessage = new FacesMessage("Failed to add new user");  
        outcome = "register";  
    }  
    FacesContext.getCurrentInstance().addMessage(null, doneMessage);  
    return outcome;  
}
```


Navegación basada en reglas

```
<navigation-rule>
  <from-view-id>from_page.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{ManagedBean.actionMethod}</from-action>
    <from-outcome>condition 1</from-outcome>
    <to-view-id>/to_page1.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{ManagedBean.actionMethod}</from-action>
    <from-outcome>condition 2</from-outcome>
    <to-view-id>/to_page2.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```



```

<!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:f="http://java.sun.com/jsf/core">
<h:head>
  <title>Login</title>
</h:head>
<h:body>
  <h:form>
    <h:panelGrid border="1" columns="2">

      <h:outputText value="Userid:" />
      <h:inputText id="userid" value="#{login.userid}" />

      <h:outputText value="Password:" />
      <h:inputSecret id="password" value="#{login.password}" />

      <h:outputText value="&nbsp;" />
      <h:commandButton value="Login" action="#{login.loginAction}" />
    </h:panelGrid>
  </h:form>

</h:body>
</html>

```

```
<!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">  
  <head><title>Success!</title></head>  
  <body>  
    <p>Success!</p>  
  </body>  
</html>
```

```
<!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">  
  <head><title>Failure!</title></head>  
  <body>  
    <p>Failure!</p>  
  </body>  
</html>
```

```
import javax.faces.bean.ManagedBean;
import javax.faces.bean.RequestScoped;
```

```
@ManagedBean
```

```
@RequestScoped
```

```
public class Login {
```

```
    String userid;
```

```
    String password;
```

```
    public Login() { }
```

```
    public String getUserid() {
        return userid;
    }
```

```
    public void setUserid(String userid) {
        this.userid = userid;
    }
```

```
    public String getPassword() {
        return password;
    }
```

```
    public void setPassword(String password) {
        this.password = password;
    }
```

```
    // Action Method
```

```
    public String loginAction() {
        if (userid.equals("guest") && password.equals("welcome")) {
            return "success";
        } else {
            return "failure";
        }
    }
}
```

```
<?xml version='1.0' encoding='UTF-8'?>

<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-facesconfig_2_0.xsd"
  version="2.0">

  <navigation-rule>
    <from-view-id>/login.xhtml</from-view-id>
    <navigation-case>
      <from-action>#{login.loginAction}</from-action>
      <from-outcome>success</from-outcome>
      <to-view-id>/success.xhtml</to-view-id>
    </navigation-case>
    <navigation-case>
      <from-action>#{login.loginAction}</from-action>
      <from-outcome>failure</from-outcome>
      <to-view-id>/failure.xhtml</to-view-id>
    </navigation-case>
  </navigation-rule>

</faces-config>
```

```
<navigation-rule>
  <from-view-id>/login.xhtml</from-view-id>
  <navigation-case>
    <from-action>#{login.loginAction}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/success.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{login.loginActionCheckBalance}</from-action>
    <from-outcome>success</from-outcome>
    <to-view-id>/showbalance.xhtml</to-view-id>
  </navigation-case>
  <navigation-case>
    <from-action>#{login.loginAction}</from-action>
    <from-outcome>failure</from-outcome>
    <to-view-id>/failure.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Wildcards

```
<navigation-rule>
  <from-view-id> * </from-view-id>
  <navigation-case>
    <description>
      Global login rule for any page with a Login button/link
      or any action method
    </description>
    <from-outcome>login</from-outcome>
    <to-view-id>/login.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```


Wildcards

```
<navigation-rule>
  <from-view-id>/register/* </from-view-id>
  <navigation-case>
    <description>
      Login Required error condition for any /register/* page.
    </description>
    <from-action>#{Login.checkLogin}</from-action>
    <from-outcome>login required</from-outcome>
    <to-view-id>/login-required.xhtml</to-view-id>
  </navigation-case>
</navigation-rule>
```

Redirects

```
<navigation-case>
  <from-action>#{Login.loginAction}</from-action>
  <from-outcome>success</from-outcome>
  <to-view-id>/success.xhtml</to-view-id>
  <redirect/>
</navigation-case>
```

- ❑ En el ejemplo original para el login, cuando el usuario se logueaba y procedía a la pagina success.xhtml, la url en el browser aun decía “http://...login.xhtml...”
- ❑ Esto se debe a que el contenido de la pagina success.xhtml era enviado al browser, sin que el browser solicitara directamente esa pagina
- ❑ Al insertar el <redirect/>, estamos obligando al browser a buscar esa pagina, con lo que sera desplegada en la URL

Controles HTML

column	Columna de datos en una tabla HTML
commandButton	<code><input type='{submit', 'reset', 'image'}></code>
commandLink	<code></code>
dataTable	<code><table>...</code>
form	<code><form>...</code>
graphicImage	<code>...</code>
inputHidden	<code><input type='hidden'...></code>
inputSecret	<code><input type='password'...></code>
inputText	<code><input type='text'...></code>
inputTextArea	<code><textArea>...</code>
message	Mensaje localizado (<code>...</code> si se usan estilos)
messages	Mensajes localizados (<code>...</code> si se usan estilos)
...	...

Controles HTML

outputFormat	Despliega un mensaje localizado (texto plano)
outputLabel	Despliega un componente como un label (<label>...)
outputLink	<a>...
outputText	Despliega una línea de texto
panelGrid	Despliega un elemento <table>, con elementos <tr> y <td>
panelGroup	Agrupar un conjunto de componentes bajo un padre
selectBooleanCheckbox	<input type='checkbox'...>
selectItem	Representa un item (<option>) dentro de una lista
selectItems	Representa una lista de items (<option>)
selectManyCheckbox	Conjunto de elementos <input type='checkbox'...>
selectManyListbox	Permite seleccionar un conjunto de elementos (<select>...)
selectManyMenu	Permite seleccionar un conjunto de elementos (<select>...)
selectOneListbox	Permite seleccionar un conjunto de elementos (<select>...)
selectOneMenu	Permite seleccionar un item (<select>...)
selectOneRadio	Permite seleccionar un item (<input type='radio'>...)

UIForm – Form

```
<h:form id="jsftags">  
  ...  
</h:form>
```

```
<form id="jsftags" method="post"  
  action="/jsftags/faces/pages/tags.jsp"  
  enctype="application/x-www-form-urlencoded">  
  ...  
  <input type="hidden" name="jsftags"  
    value="jsftags" />  
  <input type="hidden" name="jsftags:link" />  
</form>
```

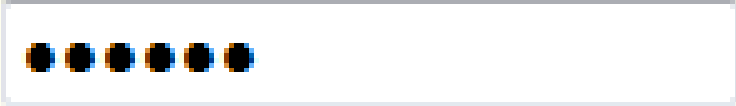
UIInput – inputText

```
<h:inputText id="address"  
  value="#{jsfexample.address}" />
```

```
<input type="text" name="jsftags:_id1"  
  value="123 JSF Ave" />
```

UIInput – inputSecret

```
<h:inputSecret redisplay="false"  
  value="#{jsfexample.password}" />
```



```
<input id="jsftags:password"  
  type="password"  
  name="jsftags:password"  
  value="secret" />
```

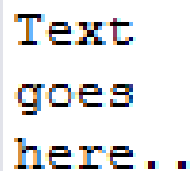

UIInput – inputHidden

```
<h:inputHidden id="hidden"  
    value="userPreference" />
```

```
<input id="jsftags:hidden"  
    type="hidden"  
    name="jsftags:hidden"  
    value="userPreference" />
```

UIInput – inputTextArea

```
<h:inputTextarea id="textArea"  
    rows="4" cols="7"  
    value="Text goes here.."/>
```



Text
goes
here..

```
<textarea id="jsftags:textArea"  
    name="jsftags:textArea"  
    cols="5" rows="3">  
    Text goes here..  
</textarea>
```

UIOutput – outputText

```
<h:outputText  
  value="#{jsfexample.zipCode}"/>
```

10032

10032

UIOutput – outputLabel

```
<h:outputLabel for="address">  
  <h:outputText id="addressLabel"  
    value="User Home Address"/>  
</h:outputLabel>
```

User Home Address

```
<span id="jsftags:addressLabel">  
  User Home Address</span>
```

UIOutput – outputLink

```
<h:outputLink  
  value="#{msg['jsfstudio.home.url']}">  
  <f:verbatim>JSF Studio</f:verbatim>  
</h:outputLink>
```

JSF Studio

```
<a href="http://www.jsf-studio.com">  
  JSF Studio  
</a>
```

UIOutput – outputFormat

```
<h:outputFormat  
  value="#{msg.jsfstudioThankYou}">  
  <f:param value="Joe Blow"/>  
  <f:param id="productName"  
    value="#{msg['jsfstudio.label']}" />  
</h:outputFormat>
```

Thank you, Joe Blow, for trying Exadel JSF Studio!

Thank you, Joe Blow, for trying Exadel JSF Studio!

UIMessage – UIMessages

```
Enter address:  
<h:message style="color: red"  
  for="useraddress" />  
<h:inputText id="useraddress"  
  value="#{jsfexample.address}"  
  required="true"/>  
<h:commandButton action="save" value="Save"/>
```

Enter address: Validation Error: Value is
required.

Save

```
Enter address:  
<span style="color: red">  
  Validation Error: Value is required.  
</span>  
<input id="jsftags:useraddress"  
  type="text"  
  name="jsftags:useraddress" value="" />  
<input type="submit" name="jsftags:_id1"  
  value="Save" />
```

UISelectBoolean selectBooleanCheckbox

```
<h:selectBooleanCheckbox  
  title="emailUpdates"  
  value="#{jsfexample.wantsEmailUpdates}" >  
</h:selectBooleanCheckbox>  
<h:outputText  
  value="Would you like email updates?"/>
```

☒ Would you like email updates?

```
<input type="checkbox"  
  name="jsftags:_id6" checked  
  title="emailUpdates" />  
Would you like email updates?
```


UISelectMany selectManyCheckboxList

```
<h:selectManyCheckbox id="cars"  
  value="#{carsBean.car}">  
  <f:selectItems  
    value="#{carBean.carList}"/>  
</h:selectManyCheckbox>
```

☐ Honda
Accord

☐ Toyota
4Runner

☐ Nissan
Z350

UISelectOne – selectOneMenu

```
<h:selectOneMenu id="selectCar"  
  value="#{carBean.currentCar}">  
  <f:selectItems  
    value="#{carBean.carList}" />  
</h:selectOneMenu>
```



Honda Accord ▼

```
<select id="jsftags:selectCar"  
  name="jsftags:selectCar" size="1">  
  <option value="accord">Honda Accord</option>  
  <option value="4runner">Toyota 4Runner</option>  
  <option value="nissan-z">Nissan Z350</option>  
</select>
```

UIPanel – panelGrid

```
<h:panelGrid columns="4" footerClass="subtitle"
  headerClass="subtitlebig" styleClass="medium"
  columnClasses="subtitle,medium">
  <f:facet name="header">
    <h:outputText value="Table with numbers"/>
  </f:facet>
  <h:outputText value="1" />
  <h:outputText value="2" />
  <h:outputText value="3" />
  <h:outputText value="4" />
  <h:outputText value="5" />
  <h:outputText value="6" />
  <h:outputText value="7" />
  <f:facet name="footer">
    <h:panelGroup>
      <h:outputText value="one row" />
      <h:outputText value=" " />
      <h:outputText
        value="grouped with panelGroup" />
    </h:panelGroup>
  </f:facet>
</h:panelGrid>
```

UIPanel – panelGrid

Table with numbers

1	2	3	4
5	6	7	

one row grouped with panelGroup

UIData / UIColumn – dataTable

```
<h:dataTable id="books"
  columnClasses="list-column-center,
  list-column-right, list-column-center,
  list-column-right" headerClass="list-header"
  rowClasses="list-row" styleClass="list-
  background" value="#{BookStore.items}" var="store">
  <h:column>
    <f:facet name="header">
      <h:outputText value="#{msg.storeNameLabel}"/>
    </f:facet>
    <h:outputText value="#{store.name}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <Subject
    </f:facet>
    <h:outputText value="#{store.subject}"/>
  </h:column>
  <h:column>
    <f:facet name="header">
      <h:outputText value="#{msg.storePriceLabel}"/>
    </f:facet>
    <h:outputText value="#{store.price}"/>
  </h:column>
</h:dataTable>
```

UIData / UIColumn – dataTable

Title	Subject	Price (\$)
JSF For Dummies	JSF	25.0
Struts For Dummies	Struts	22.95

- ❑ Es el proceso que nos permite determinar que un conjunto de datos tiene el tipo correcto
- ❑ En general, convertimos valores de tipo String en otros tipos, como ser Date, Float, Integer, etc.
- ❑ Podemos usar convertidores estándar, o convertidores customizados

Convertidores estándar

- ❑ Si hacemos un binding contra un Integer o un int, entonces la conversión ocurre automáticamente
- ❑ Por ejemplo:

```
<%-- age --%>  
<h:outputLabel value="Age" />  
<h:inputText id="age" size="3"  
    value="#{contactController.contact.age}">  
</h:inputText>
```


Convertidores estándar

- ❑ Por ejemplo, si queremos manejar una fecha como MM/yyyy, entonces podemos hacer esto:

```
<h:outputLabel value="Birth Date" />
<h:inputText id="birthDate"
    value="#{contactController.contact.birthDate}">
    <f:convertDateTime pattern="MM/yyyy" />
</h:inputText>

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

Convertidores customizados

- ❑ Queremos convertir un campo con información de la hora, minutos y segundos, en un entero con la cantidad de segundos que eso representa
- ❑ Por ejemplo, “00:01:00” equivale a 60 segundos
- ❑ Creamos la clase `hr_mi_se_Converter` que implementa la interfaz `Converter`

```
import javax.faces.component.UIComponent;
import javax.faces.context.FacesContext;
import javax.faces.convert.*;
import javax.faces.application.*;

public class hr_mi_se_Converter implements Converter {

    public Object getAsObject(FacesContext facesContext,
                              UIComponent uiComponent,
                              String param) {

        ...
    }

    public String getAsString(FacesContext facesContext,
                              UIComponent uiComponent,
                              Object obj) {

        ...
    }
}
```

```
public Object getAsObject(FacesContext facesContext,
                          UIComponent uiComponent,
                          String param) {

    try {
        String hr_mi_se[] = param.split(":");
        int seconds =
            Integer.parseInt(hr_mi_se[0])*60*60 +
            Integer.parseInt(hr_mi_se[1])*60+
            Integer.parseInt(hr_mi_se[2]);
        return new Integer(seconds);
    } catch (Exception exception) {
        throw new ConverterException(exception);
    }

}
```

```
public String getAsString(FacesContext facesContext,  
                          UIComponent uiComponent,  
                          Object obj) {
```

```
try {  
    int total_seconds = (int) ((Integer) obj).intValue();  
    int hours = (total_seconds) / (60 * 60);  
    int rem = (total_seconds) % (60 * 60);  
    int minutes = rem / 60;  
    int seconds = rem % 60;  
    String str_hours = "" + hours;  
    String str_minutes = "" + minutes;  
    String str_seconds = "" + seconds;  
    if (hours < 10) str_hours = "0" + hours;  
    if (minutes < 10) str_minutes = "0" + minutes;  
    if (seconds < 10) str_seconds = "0" + seconds;  
    return str_hours + ":" + str_minutes + ":" + str_seconds;  
} catch (Exception exception) {  
    throw new ConverterException(exception);  
}  
}
```

faces-config.xml

```
<?xml version="1.0"?>

<faces-config>
    .....
    .....
    <converter>
        <converter-id>
            hr_mi_se_Converter
        </converter-id>
        <converter-class>
            hr_mi_se_Converter
        </converter-class>
    </converter>
    .....
    .....
</faces-config>
```

Validación (estándar)

- ❑ DoubleRangeValidator
 - El valor debe ser double, y debe estar en los rangos especificados
- ❑ LongRangeValidator
 - El valor debe ser long, y debe estar en los rangos especificados
- ❑ LengthValidator
 - El valor debe ser String, y su largo debe estar en los rangos especificados

Validación estándar

```
<%-- age --%>
<h:outputLabel value="Age" />
<h:inputText id="age" size="3"
    value="#{contactController.contact.age}">
    <f:validateLongRange minimum="0"
        maximum="150" />
</h:inputText>
<h:message errorClass="errorClass" />
```


Validación estándar

```
<%-- First Name --%>
<h:outputLabel value="First Name"/>
<h:inputText id="firstName" required="true"
  value="#{contactController.contact.firstName}"
  size="10" >
  <f:validateLength minimum="2"
    maximum="25" />
</h:inputText>
<h:message for="firstName" />
```

- ❑ Podemos crear componentes de validación customizados, que pueden ser reutilizados entre aplicaciones
- ❑ Para esto, debemos seguir una serie de pasos
- ❑ Crear una clase que implementa la interfaz `javax.faces.validator.Validator`
- ❑ Implementar el método `validate()`

Validación customizada

- ❑ Registrar el validador en el archivo faces-config.xml
- ❑ Utilizar el tag `<f:validate/>` en las interfaces
- ❑ Por ejemplo, creamos un validador para controlar el código postal ingresado...

```

public class ZipCodeValidator implements Validator {
    private static final String ZIP_REGEX = "[0-9]{5}";
    private static final String
        PLUS4_OPTIONAL_REGEX = "([ |-]{1}[0-9]{4})?";
    private static Pattern mask = null;
    static {
        mask = Pattern.compile(ZIP_REGEX + PLUS4_OPTIONAL_REGEX);
    }
    public void validate(FacesContext context,
                        UIComponent component, Object value)
        throws ValidatorException {
        String zipField = (String) value;
        Matcher matcher = mask.matcher(zipField);
        if (!matcher.matches()) {
            FacesMessage message = new FacesMessage();
            message.setDetail("Zip code not valid");
            message.setSummary("Zip code not valid");
            message.setSeverity(FacesMessage.SEVERITY_ERROR);
            throw new ValidatorException(message);
        }
    }
}

```

Registramos el validador

```
<validator>
  <validator-id>
    zipCode
  </validator-id>
  <validator-class>
    validators.ZipCodeValidator
  </validator-class>
</validator>
```

Usamos el validador

```
<%-- zip --%>
<h:outputLabel value="Zip" />
<h:inputText id="zip" size="5"
    value="#{controller.contact.zip}">
    <f:validator validatorId="zipCode"/>
</h:inputText>
<h:message for="zip"
    errorClass="errorClass" />
```