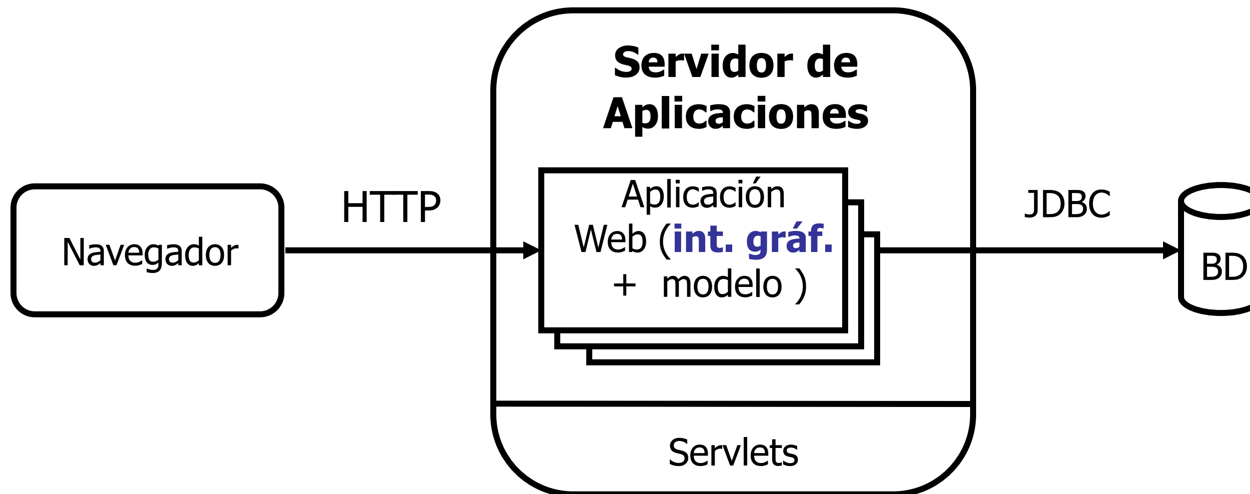
4.1 Tutorial de Servlets y JSPs

Índice

- Introducción
- Framework de Servlets
- JSPs
- Ejemplo **`pojo-servjsptutorial`**
- Empaquetado de una aplicación Web
  - Ficheros WAR y web.xml
- Frameworks POJO para interfaz Web

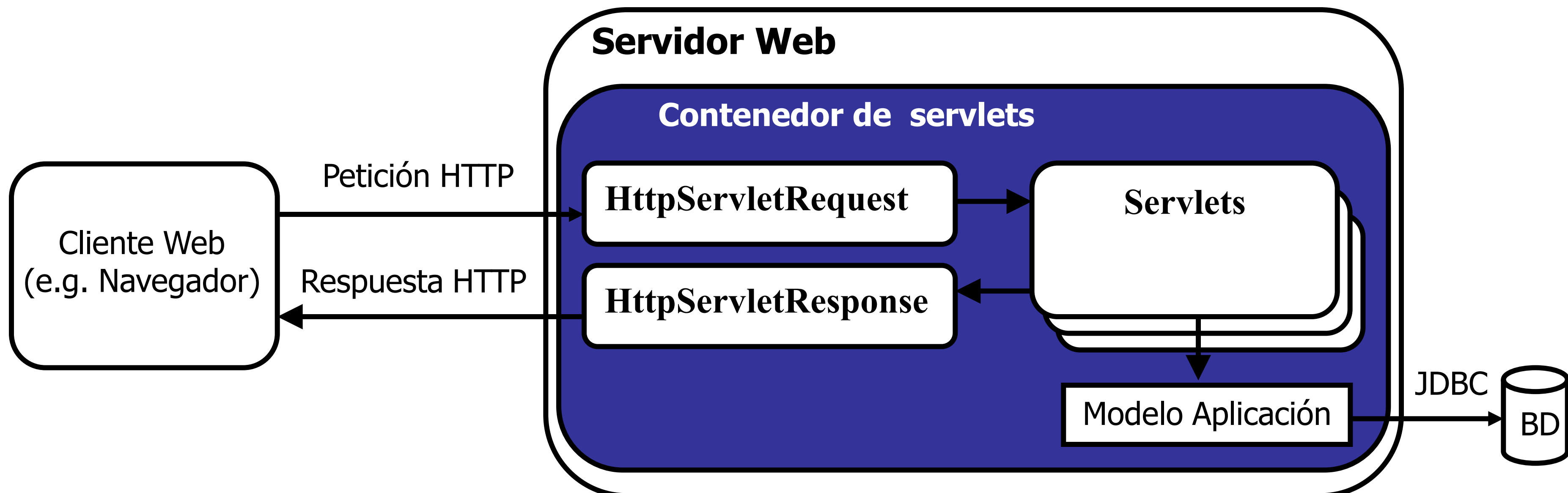
Objetivo

- Conocer la API de Servlets
  - JSP y cualquier framework Web Java se apoyan en esta API estándar



Introducción (1)

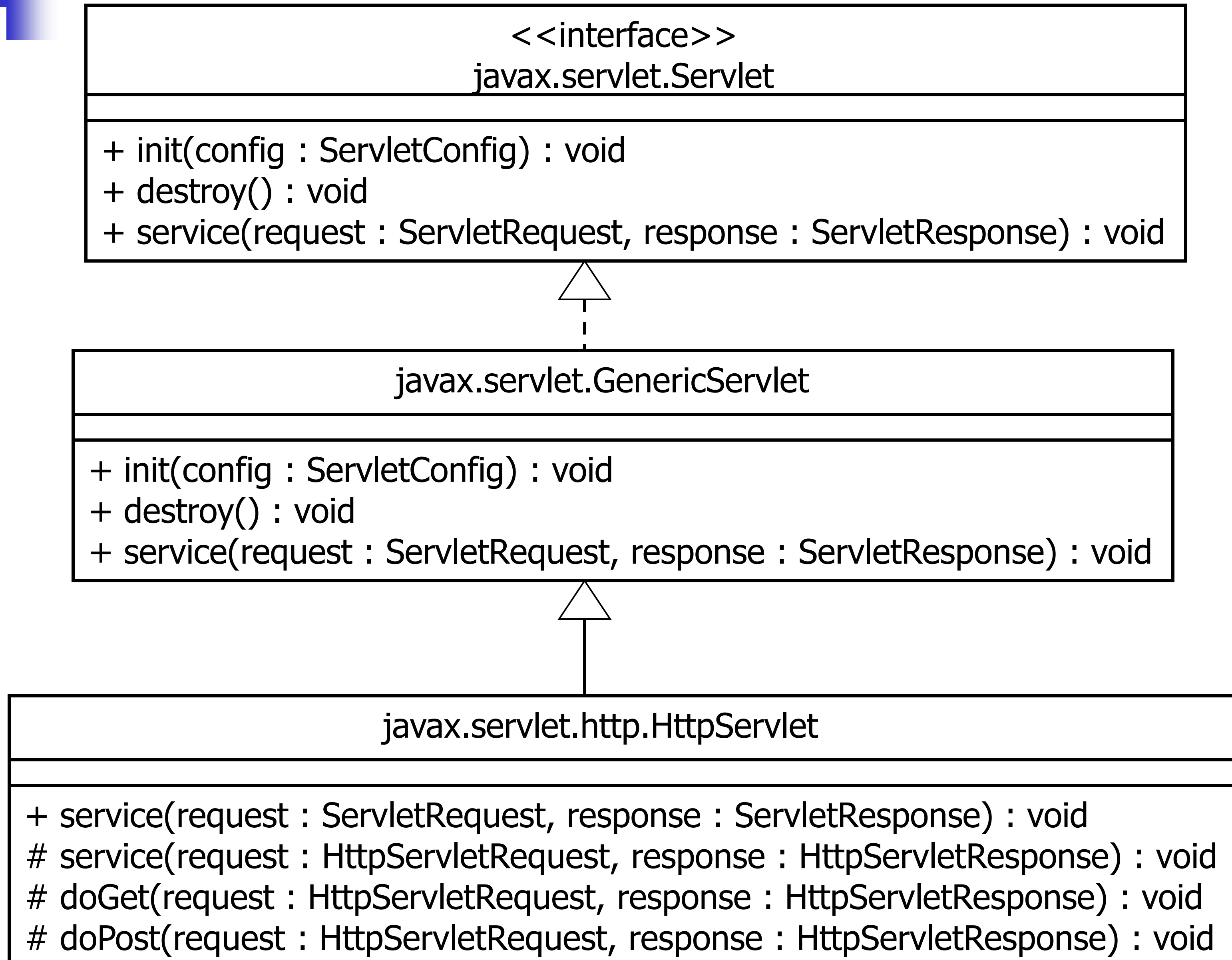
- Un **servlet** es una clase Java que puede recibir peticiones (normalmente HTTP) y generar una salida (normalmente HTML o XML)
  - Los servlets que conforman una aplicación Web se ejecutan en un servidor de aplicaciones Web (contenedor)
  - Cada servlet se puede asociar a una o más URLs
  - Paquetes **`javax.servlet`** y **`javax.servlet.http`**



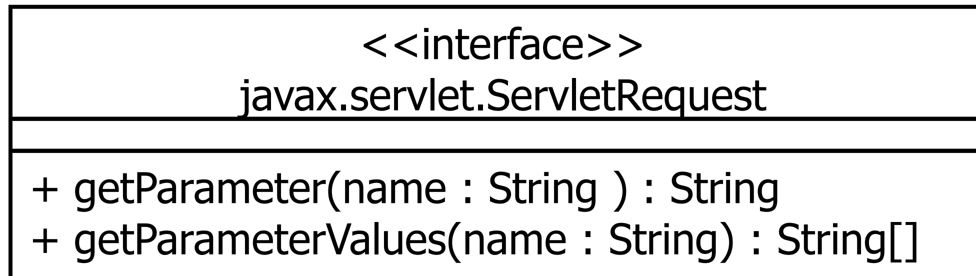
Introducción (y 2)

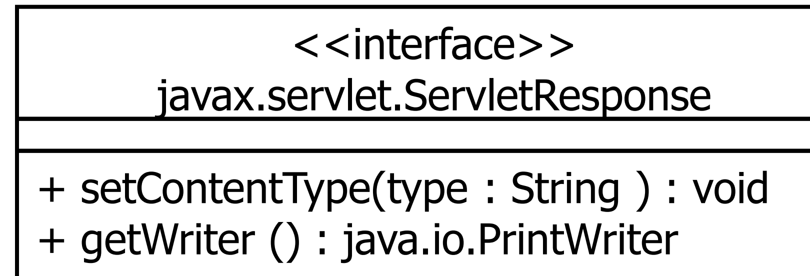
- Una **página JSP** (Java Server Page) es un tipo especial de servlet (**`javax.servlet.jsp`** y **`javax.servlet.jsp.tagext`**) orientado a generar el texto de la interfaz gráfica
  - Tiene el aspecto de una página HTML
  - Puede incluir scriptlets (scripts) para generar HTML dinámicamente
  - Típicamente los scriptlets se escriben en Java
- Hasta la aparición de JSP, el uso principal de los servlets era generar la vista de una aplicación Web
  - Recibir petición HTTP asociada a una URL
  - Leer parámetros (encapsulados en un objeto **`HttpServletRequest`**)
  - Invocar operación sobre el modelo
  - Generar salida HTML (generar respuesta HTTP, utilizando el objeto **`HttpServletResponse`**)

Visión global del framework de servlets (1)



Visión global del framework de servlets (2)





Visión global del framework de servlets (3)

- Cuando el servidor de aplicaciones Web decide cargar un servlet en memoria (e.g.: al arrancar, la primera vez que se accede a él) llama a la operación **`init`** de **`javax.servlet.Servlet`**
- Cuando el servidor de aplicaciones Web decide eliminar un servlet de memoria (e.g.: lleva cierto tiempo sin usarse), llama a la operación **`destroy`** de **`javax.servlet.Servlet`**
- La operación pública **`service`** de **`javax.servlet.http.HttpServlet`** llama a la operación protegida **`service`**
  - Es una operación plantilla (Template Method), que llama a **`doGet`**, **`doPost`**, **`doPut`**, **`doDelete`**, etc., según la petición HTTP sea un **`GET`**, **`POST`**, **`PUT`**, **`DELETE`**, etc.
- Normalmente, el programador extiende de **`javax.servlet.http.HttpServlet`** y redefine **`doGet`** y/o **`doPost`**

Visión global del framework de servlets (4)

- En una máquina virtual Java, sólo existe una instancia de cada servlet que se programe, y en consecuencia puede recibir peticiones concurrentes
  - **`doGet`**, **`doPost`**, etc. deben ser *thread-safe*
  - Normalmente no será necesario hacer nada especial, dado que la implementación de estas operaciones generalmente sólo hace uso de variables locales (pila) o de variables globales (**`static`**) de sólo lectura (típicamente caches)
  - Si modifican alguna estructura global (un atributo propio o alguna variable global), necesitan sincronizar su acceso
    - Sin embargo, en general, eso es mala idea, dado que una aplicación con estas características no será fácil que funcione en un entorno en cluster
    - En estos casos, es mejor usar una BD para las estructuras globales que sean de lectura/escritura

Visión global del framework de servlets (y 5)

- **`javax.servlet.http.HttpServletRequest`**
  - **`getParameter`**

    Permite obtener el valor de un parámetro univaluado
  - **`getParameterValues`**
    - Permite obtener el valor de un parámetro multivaluado
    - También se puede usar con parámetros univaluados
    - IMPORTANTE: cuando el usuario no selecciona ningún valor (e.g.: en una lista desplegable múltiple o una lista de checkboxes en HTML), el navegador no envía el parámetro, y en consecuencia, esta operación devuelve **`null`**
- **`javax.servlet.http.HttpServletResponse`**
  - **`setContentType`** debe llamarse antes de escribir en el **`PrintWriter`** que devuelve **`getWriter`**

Sesión

- La API de Servlets permite crear una sesión en el servidor por cada navegador que accede a una aplicación Web
  - **`request.getSession()`** devuelve la sesión actual; si no existe, crea una antes
- Cada cliente que accede al servidor (e.g. un navegador) dispone de su propio objeto **`javax.servlet.http.HttpSession`** en el servidor
- Es posible enganchar objetos a una sesión y recuperarlos
  - **`void setAttribute(String, Object)`**
  - **`Object getAttribute(String)`**
- Por motivos de escalabilidad y de que en HTTP no hay nada especial que indique que un navegador ha dejado de usar la aplicación Web, cada sesión dispone de un timeout (en minutos)
  - Si transcurre el timeout sin que el navegador acceda a la aplicación, el servidor destruye la sesión

Cookies

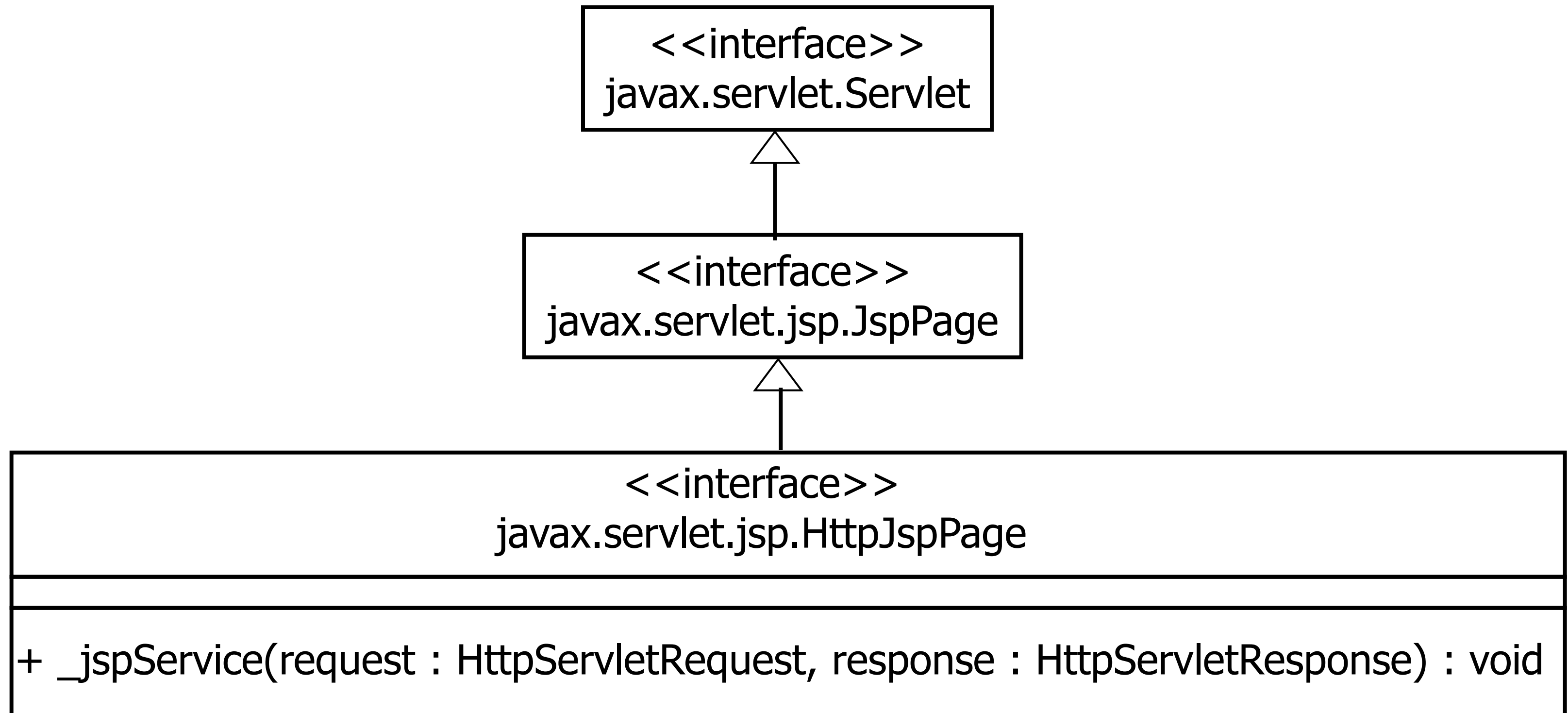
- La API de Servlets permite enviar cookies al navegador del usuario
  - Una cookie tiene un nombre y un valor asociado (cadena de caracteres)
    - `Cookie cookie = new Cookie("loginName", loginName);`
  - Cada navegador debería soportar alrededor de 20 cookies por cada sitio Web al que está conectado, 300 en total y puede limitar el tamaño de cada cookie a 4 Kbytes
- Para enviar una o varias cookies al navegador se incluyen en la **`response`**
  - `response.addCookie(cookie);`
- Cada vez que el navegador hace una petición, todas las cookies relativas a esa aplicación Web llegan en la **`request`**
  - `Cookie[] cookies = request.getCookies();`

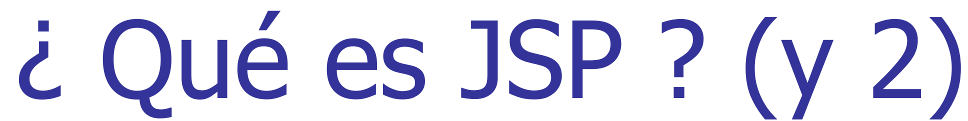
Mantenimiento de la Sesión

- HTTP es un protocolo sin estado
- Para que el servidor de aplicaciones pueda saber a qué sesión corresponde cada petición HTTP
  - Si el navegador acepta cookies, envía la cookie **`jsessionid`**
  - En otro caso, es posible construir URLs que lleven incrustado el **`jsessionid`**
- Los servidores de aplicaciones Web disponen de un soporte especial para contemplar el concepto de sesión en un cluster de máquinas

¿ Qué es JSP ? (1)

- En realidad, una página JSP es un tipo especial de servlet (**`javax.servlet.jsp`** y **`javax.servlet.jsp.tagext`**) orientado a generar el texto de la interfaz gráfica
  - Invocables por GET y POST





¿ Qué es JSP ? (y 2)

- ¿ Qué ocurre cuando se accede a una página JSP ?
  - Si es la primera vez, el servidor de aplicaciones genera un servlet (que implementa **`javax.servlet.jsp.HttpJspPage`**) a partir de la página JSP, lo compila y lo carga en memoria
  - Si no es la primera vez, le pasa la petición al servlet (ya compilado y creado en memoria)
  - Si la página se ha modificado desde la última compilación, el servidor se da cuenta, genera el nuevo servlet, lo compila y lo carga de nuevo

Modelo `pojo-servjsptutorial` (1)

- Para aprender a utilizar el framework de servlets y JSPs utilizaremos el módulo **`pojo-servjsptutorial`** de los ejemplos
- Implementa las capas modelo y Web de una aplicación bancaria mínima, con los siguientes casos de uso
  - Crear una cuenta
  - Buscar una cuenta a partir de su identificador
- Para simplificar la implementación de la capa modelo
  - Se ha optado por la realización de la persistencia en memoria, en lugar de utilizar una BD
  - Se ha definido la implementación del servicio directamente, sin crear una interfaz

Modelo `pojo-servjsptutorial` (y 2)

- Modelado de entidades
  - **`Account`**
  - Representa una cuenta bancaria, con la misma información que la comentada para **`pojo-minibank`**

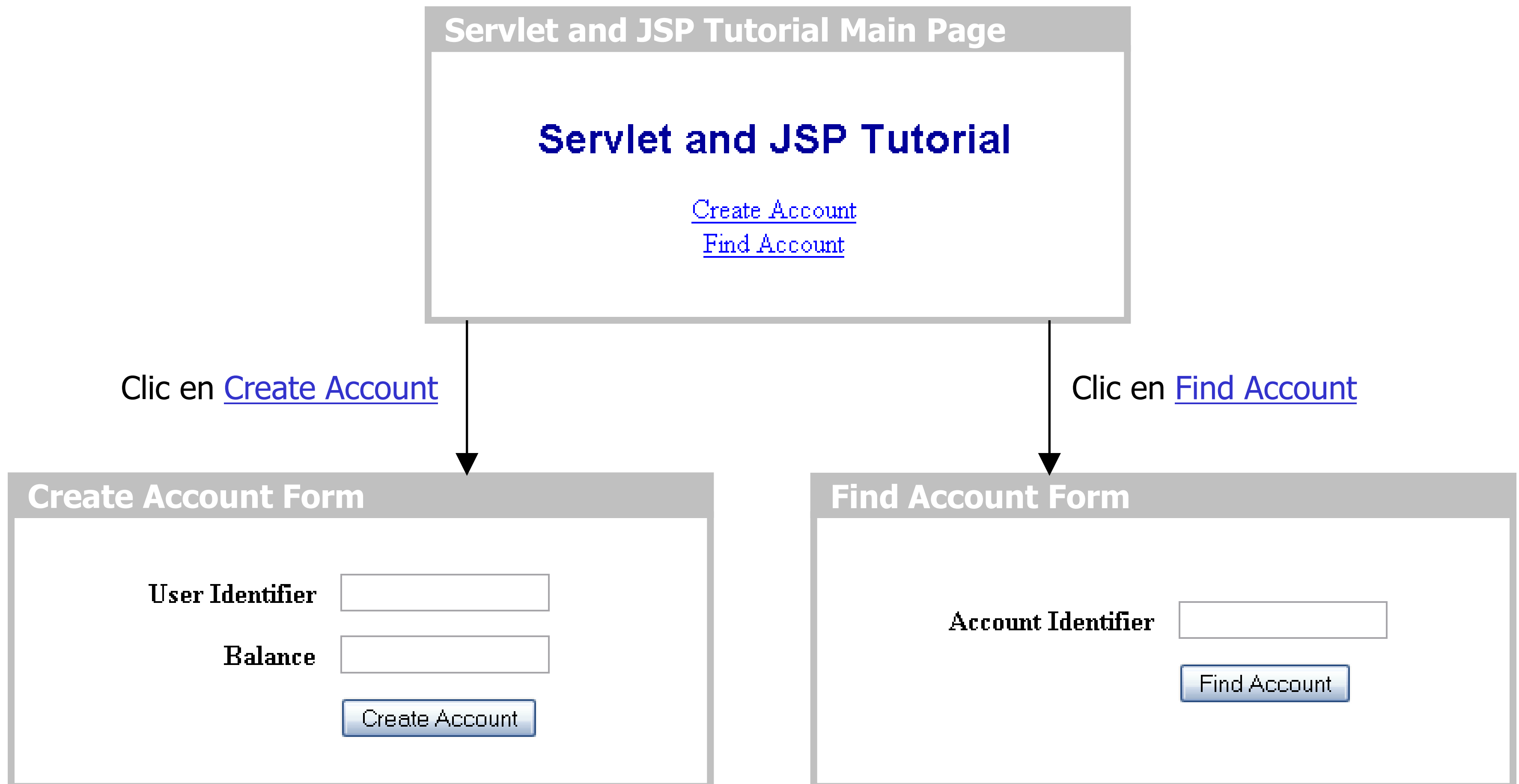
Account
- accountId : Long - userId : Long - balance : double
+ Constructores + métodos get/set

- Definición API modelo
  - **`AccountServiceImpl`**
  - Define los métodos **`createAccount`** y **`findAccount`**
  - No utiliza DAOs porque mantiene las cuentas en memoria en lugar de hacerlas persistentes

AccountServiceImpl
- lastAccountId : long - accounts: Map<Long, Account>
+ createAccount(account : Account) : Account + findAccount(accountId : long) : Account

Página principal del tutorial

- http://localhost:9090/pojo-servjsptutorial



Index.html

```
<html>
<head>
  <title>Servlet and JSP Tutorial Main Page</title>
</head>

<body text="#000000" bgcolor="#ffffff">

<div align="center">
  <p><font color="#000099" size="+2" face="Arial, Helvetica, sans-
   serif">
    <b>Servlet and JSP Tutorial</b></font><br>
  </p>
</div>

<div align="center">
<a href="CreateAccountForm.jsp">Create Account</a>
<br/>
<a href="FindAccountForm.jsp">Find Account</a>
</div>


</body>

</html>
```

Tipos de URLs en HTML (1)

- Las URLs que muestra el navegador empiezan por http://localhost:9090/pojo-servjsptutorial
  - Son URLs absolutas
  - **`localhost:9090`**: máquina y puerto en el que se ejecuta el servidor de aplicaciones (en este caso Jetty o Tomcat)
  - **`/pojo-servjsptutorial`**: nombre de la aplicación Web
    - En un servidor de aplicaciones pueden instalarse varias aplicaciones Web
- Las URLs que se usan en los ficheros HTML o en las respuestas generadas del ejemplo son de tipo path relativo, y en consecuencia no incluyen http://localhost:9090/pojo-servjsptutorial
  - Buena idea, permite instalar la aplicación en otro servidor bajo un nombre distinto

Tipos URLs en HTML (y 2)

- Si se desea escribir una URL de tipo path absoluto en un fichero HTML o una respuesta generada, ésta ha de empezar por **`/nombreAplicación`**

```
<a href="/pojo-servjsptutorial/CreateAccountForm.jsp">
Create Account</a>
```

  - Es posible generar automáticamente la parte del nombre de la aplicación
    - Permite instalar la aplicación con otro nombre, sin que haya que realizar modificaciones
      - **`request.getContextPath()`** devuelve **`/nombreAplicación`**
- Volveremos a hablar sobre tipos de URLs más adelante

Demo: Creación de una Cuenta

- http://localhost:9090/pojo-servjsptutorial/CreateAccountForm.jsp

Create Account Form

User Identifier

Balance

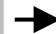
Create Account

Create Account Form

User Identifier 1

Balance 12345.67

Create Account



Created Account Data

Account number 1 created sucessfully

Home

Demo: Control de Errores

- El identificador de usuario y el balance son obligatorios

Create Account Form

User Identifier [ ] Mandatory field

Balance [ ] Mandatory field

[Create Account]

- El identificador de cuenta debe ser un número entero y >=0

Create Account Form

User Identifier [xyz] Incorrect integer value

Balance [100]

[Create Account]

- El balance debe ser un número real y >=0

Create Account Form

User Identifier [1]

Balance [abc] Incorrect real value

[Create Account]

Demo: Búsqueda de una cuenta

- http://localhost:9090/pojo-servjsptutorial/FindAccountForm.jsp

Find Account Form

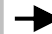
Account Identifier [ ]

[Find Account]

Find Account Form

Account Identifier [1]

[Find Account]



Account Data

Account Information

Account Identifier	1
User Identifier	1
Balance	12345.67

Home

Demo: Control de Errores

- El identificador de cuenta es obligatorio

Find Account Form

Account Identifier [ ] Mandatory field

[Find Account]

- El identificador de cuenta debe ser un número entero y >=0

Find Account Form

Account Identifier [xyz] Incorrect integer value

[Find Account]

- Debe existir alguna cuenta con el identificador introducido

Account Data

Account Information

Account not found

Home

- Se utilizarán páginas JSP para la generación de la vista de la aplicación
  - Visualización de formularios y de mensajes de error tras las validaciones
  - Visualización de resultados
- Se utilizará un servlet para procesar cada formulario
  - El servlet validará la corrección de los parámetros introducidos
    - Si los parámetros son válidos
      - invocará al modelo para obtener los resultados asociados a la acción realizada y
      - pasará el control a la página JSP que visualiza el resultado de la acción
    - Si los parámetros no son válidos
      - creará un mapa con los errores detectados
      - pasará el control a la página JSP que genera el formulario para permitir al usuario corregir el valor de los parámetros

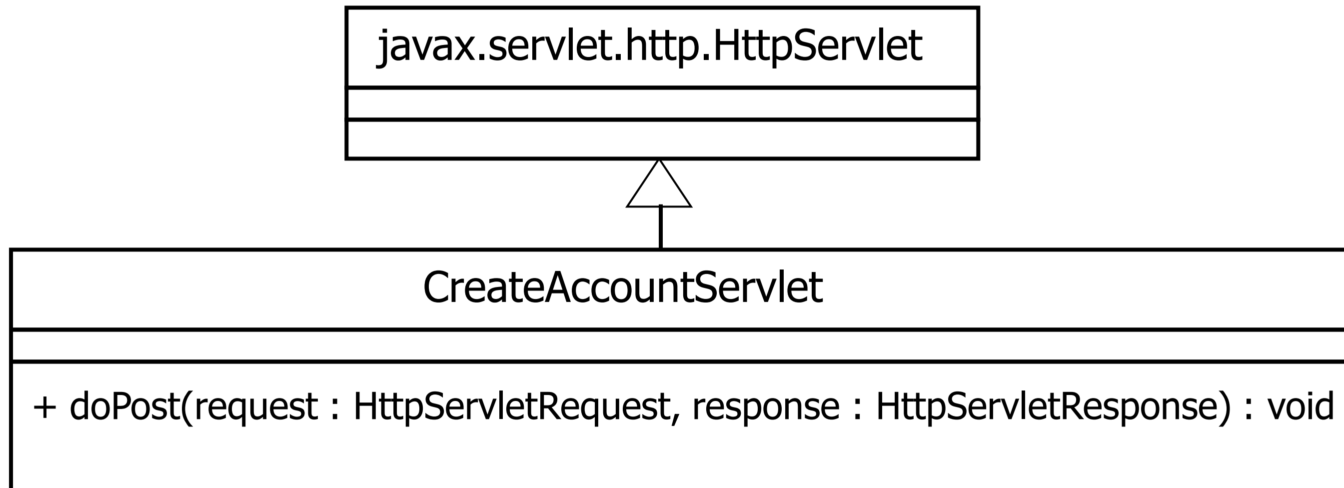
es.udc.pojo.servjsptutorial.web.CreateAccountServlet (1)

```
...
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
...
import es.udc.pojo.servjsptutorial.web.util.PropertyValidator;
import es.udc.pojo.servjsptutorial.web.util.WebUtil;

@SuppressWarnings("serial")
public class CreateAccountServlet extends HttpServlet {

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

        Map<String, String> errors = new HashMap<String, String>();

        String userId = request.getParameter("userId");
        String balance = request.getParameter("balance");

        long userIdAsLong =
            PropertyValidator.validateLong(errors,"userId",
                userId, true, 0, Long.MAX_VALUE);
```

es.udc.pojo.servjsptutorial.web.CreateAccountServlet (y 2)

```
        double balanceAsDouble =
            PropertyValidator.validateDouble(errors, "balance",
                balance, true, 0, Double.MAX_VALUE);

        if (!errors.isEmpty()) {
            request.setAttribute("errors", errors);
            WebUtil.forwardTo(request, response,
                "CreateAccountForm.jsp");
        } else {

            Account account =
                new Account(userIdAsLong, balanceAsDouble);

            /* Insert the Account in the database
             */
            Account insertedAccount = new AccountServiceImpl()
                    .createAccount(account);

            response.sendRedirect("ShowCreatedAccount.jsp?accountId="
                    + insertedAccount.getAccountId());

        }
    }
}
```

Creación de una Cuenta (1)



- Extiende **`javax.servlet.HttpServlet`**
  - Implementa el método **`doPost`**, para atender peticiones HTTP **`POST`**
- Este servlet espera recibir como parámetros el identificador de usuario y el balance de la cuenta a ser creada
- Obtiene los parámetros asociados a la petición utilizando el método **`getParameter`** del objeto **`HttpServletRequest`**
- NOTA: Este servlet está asociado a la URL indicada en el campo **`action`** del formulario de creación de cuentas

Creación de una Cuenta (2)

- Valida los parámetros de entrada usando la clase **`es.udc.pojo.servjsptutorial.web.util.PropertyValidator`**
  - Los métodos **`validateLong`** y **`validateDouble`** verifican si se ha especificado valor para el parámetro validado (si su argumento **`mandatory`** es **`true`**) y si el valor se encuentra en un rango de valores determinado de los números enteros o reales, respectivamente
  - Los métodos **`validateXXX`** reciben un mapa (**`errors`**) al que añaden una entrada en el caso de detectar un error en el parámetro validado
    - El mapa tiene como clave el nombre del parámetro y como valor el mensaje de error
  - En el ejemplo se valida que el identificador de usuario exista y sea un entero positivo, y que el balance también se haya especificado y sea un número real positivo

Creación de una Cuenta (y 3)

- Tras la validación de parámetros, si se ha detectado algún error (mapa **`errors`** no está vacío)
  - Se utiliza el método **`setAttribute`** del objeto **`HttpServletRequest`** para añadir el mapa de errores a la request
  - Se pasa el control a la página JSP que muestra el formulario de creación de una cuenta (**`CreateAccountForm.jsp`**), que tendrá accesible los errores, para informar al usuario
    - **`WebUtil.forwardTo`**
- Si los parámetros son correctos
  - Se crea el objeto **`Account`**
  - Se invoca el método **`createAccount`** del modelo
  - Se redirige a la página JSP que muestra que se ha creado la cuenta correctamente (**`ShowCreatedAccount.jsp`**)
    - Se especifica como parámetro el identificador asignado a la nueva cuenta, para que pueda ser visualizado
    - **`response.sendRedirect`**

es.udc.pojo.servjsptutorial.web.util.WebUtil (1)

```
...
import javax.servlet.RequestDispatcher;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class WebUtil {

    public final static void forwardTo(HttpServletRequest request,
           HttpServletResponse response, String url)throws IOException,
           ServletException {

           RequestDispatcher requestDispatcher =
                request.getRequestDispatcher(url);

           requestDispatcher.forward(request, response);

    }

}
```

Redirecciones (1)

- En el ejemplo se utiliza **`sendRedirect`** y **`forward`**
- **`sendRedirect`**
  - Le decimos al navegador que haga una nueva petición a otra URL
  - La barra de navegación muestra la URL
- **`forward`**
  - Se pasa el control a otro recurso (servlet o JSP) dentro del servidor
  - El navegador no es consciente de los forwards (la barra de navegación sigue mostrando la URL correspondiente a la petición HTTP original)
  - Se conserva todo lo que había en la request
  - Útil para tratar errores en formularios
    - El servlet de procesamiento engancha el atributo **`errors`** (un **`Map`**) en la request y hace un forward a la página JSP que genera el formulario
    - La página JSP que genera el formulario comprueba si la request incluye el atributo **`errors`**

Redirecciones (2)

- Más sobre tipos de URLs
  - Los tipos de URLs que hemos visto al principio de este apartado se refieren a las URLs en ficheros HTML, páginas JSP o respuestas generadas
  - ¿Qué tipos de URLs acepta **`sendRedirect`**?
    - Los tres tipos que conocemos hasta ahora
    - Lógico, pues un **`sendRedirect`** se usa para informar al navegador que nos haga una petición a otra URL
  - ¿Qué tipos de URLs acepta **`forward`**?
    - De tipo path relativo (sin salir de la aplicación)
    - De tipo path relativo a contexto
      - Empiezan por "/" y no incluyen el nombre de la aplicación
      - e.g.: **`/Index.html`**

Redirecciones (y 3)

- ¿Cuándo usar **`forward`** y cuándo **`sendRedirect`**?
  - En principio, un **`forward`** siempre es más rápido (ocurre en el servidor)
  - Un **`forward`** es preciso cuando queremos enganchar atributos a la **`request`**
    - e.g.: Tratamiento de errores en formularios
  - Para el resto de situaciones, es mejor usar un **`sendRedirect`**, dado que **`forward`** no cambia la URL que muestra la caja de diálogo del navegador (el navegador no se entera de que se hace un **`forward`**), lo que será confuso si el usuario decide recargar la página (se invocará a la URL antigua que todavía muestra la caja de diálogo del navegador)

CreateAccountForm.jsp (1)

```
<%@ page import="java.util.Map" %>

<html>
<head>
  <title>Create Account Form</title>
</head>

<body text="#000000" bgcolor="#ffffff">

<%-- Get errors. --%>

<%
    String userIdErrorMessage = "";
    String balanceErrorMessage = "";
    Map<String, String> errors =
        (Map<String, String>) request.getAttribute("errors");

    if (errors != null) {

        String errorHeader = "<font color=\"red\"><b>";
        String errorFooter = "</b></font>";

        if (errors.containsKey("userId")) {
            userIdErrorMessage = errorHeader + errors.get("userId") +
                errorFooter;
        }
```

CreateAccountForm.jsp (2)

```
        if (errors.containsKey("balance")) {
            balanceErrorMessage = errorHeader + errors.get("balance") +
                errorFooter;
        }

    }

    String userId = request.getParameter("userId");
    if (userId==null) {
        userId="";
    }
    String balance = request.getParameter("balance");
    if (balance==null) {
        balance="";
    }

%>

<form method="POST" action="CreateAccount">

<table width="100%" border="0" align="center" cellspacing="12">
```

CreateAccountForm.jsp (3)

```
<%-- User Identifier --%>
    <tr>

        <th align="right" width="50%">
            User Identifier
        </th>
        <td align="left">
            <input type="text" name="userId"
                value="<%= userId %>"
                size="16" maxlength="16">
            <%= userIdErrorMessage %>
        </td>
    </tr>

<%-- Balance --%>
    <tr>
        <th align="right" width="50%">
            Balance
        </th>
        <td align="left">
            <input type="text" name="balance"
                value="<%= balance %>"
                size="16" maxlength="16">
            <%= balanceErrorMessage %>
        </td>
    </tr>
```

CreateAccountForm.jsp (y 4)

```
<%-- Create button --%>
    <tr>
        <td width="50%"></td>
        <td align="left" width="50%">
            <input type="submit" value="Create Account">
        </td>
    </tr>

</table>

</form>

</body>

</html>
```

Formulario de Creación de una Cuenta (1)

- Página JSP que muestra el formulario de creación de una cuenta bancaria
- Utiliza etiquetas HTML para crear el formulario
- Usa **`<%= expresión %>`** para incluir expresiones Java
  - La expresión es evaluada en tiempo de ejecución y convertida a un **`String`**
- Usa scriptlets para incluir código Java
  - **`<% ... %>`**
- Objetos implícitos
  - **`request`**: **`javax.servlet.http.HttpServletRequest`**
  - **`response`**: **`javax.servlet.http.HttpServletResponse`**
  - **`session`**: **`javax.servlet.http.HttpSession`**
  - **`out`**: **`javax.servlet.jsp.JspWriter`**
  - Algunos más

Formulario de Creación de una Cuenta (2)

- **`<%@ ... %>`**
  - Directiva
  - En particular, la directiva **`page`** tiene varios atributos, entre otros
    - **`import`** (opcional): permite importar una o más clases (separadas por comas)

- **`<%-- ... --%>`**
  - Comentarios
  - No aparecen en la respuesta generada, a diferencia de los comentarios HTML

Formulario de Creación de una Cuenta (y 3)

- Puede llegarse a la página **`CreateAccountForm.jsp`** con una petición directa HTTP o como resultado de la validación de parámetros de una petición al servlet **`CreateAccountServlet`**
  - Utiliza scriptlets para comprobar si existe un mapa de nombre **`errors`** enganchado al objeto **`request`**
  - Inicializa las variables **`userIdErrorMessage`** y **`balanceErrorMessage`** con los mensajes asociados a los campos **`userId`** y **`balance`** en el mapa de errores, respectivamente
  - Utiliza expresiones al lado de los campos "User Identifier" y "Balance", para mostrar los mensajes de error (**`userIdErrorMessage`** y **`balanceErrorMessage`**), si existen
  - El envío del formulario genera una petición **`POST`** al servlet **`CreateAccountServlet`**

ShowCreatedAccount.jsp

```
<%@ page import="es.udc.pojo.servjsptutorial.model.account.Account" %>

<html>
<head>
  <title>Created Account Data</title>
</head>

<body text="#000000" bgcolor="#ffffff">
<div align="center">
  <p>
    <font color="#000099" face="Arial, Helvetica, sans-serif">
      <b>Account number <%= request.getParameter("accountId") %>
   created sucessfully</b>
    </font>
  </p>
</div>

<br/>
<a href="Index.html">Home</a>
<br/>
</body>

</html>
```

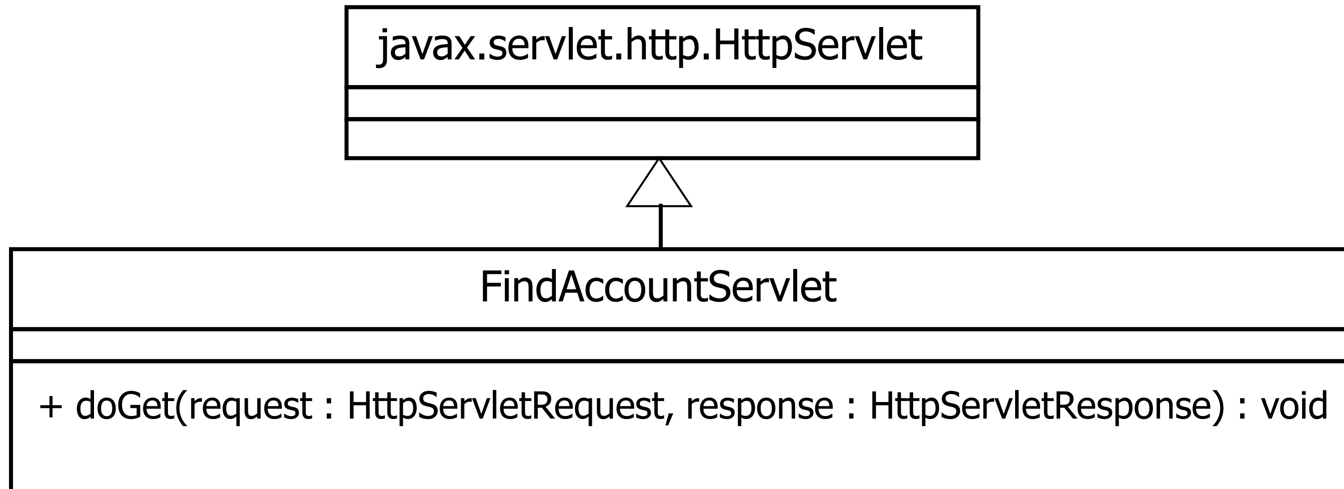
Resultado de Creación de una Cuenta

- JSP que recibe como parámetro el identificador de la cuenta que ha sido creada
- Utiliza una expresión para visualizar el identificador de cuenta

es.udc.pojo.servjsptutorial.web.FindAccountServlet

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

        Map<String,String> errors = new HashMap<String,String>();
        String accountId = request.getParameter("accountId");
        long accountIdAsLong =
            PropertyValidator.validateLong(errors, "accountId",
                accountId, true, 0, Long.MAX_VALUE);

        if (!errors.isEmpty()) {
            request.setAttribute("errors", errors);
            WebUtil.forwardTo(request, response,
                "FindAccountForm.jsp");
        } else {
            try {
                Account account = new AccountServiceImpl()
                    .findAccount(accountIdAsLong);
                request.setAttribute("account", account);
            } catch (InstanceNotFoundException e) {
            }
            WebUtil.forwardTo(request, response, "ShowAccount.jsp");
        }
    }
}
```

Búsqueda de una Cuenta (1)



- Extiende **`javax.servlet.HttpServlet`**
  - Implementa el método **`doGet`**, para atender peticiones HTTP GET
- Este servlet espera recibir como parámetro el identificador de la cuenta a mostrar
- NOTA: Este servlet está asociado a la URL indicada en el campo **`action`** del formulario de búsqueda de cuentas

Búsqueda de una Cuenta (y 2)

- Como en el ejemplo anterior, utiliza la clase **`PropertyValidator`** para validar el parámetro que recibe (**`accountId`**)
  - En caso de error, pasa el control a la página que muestra el formulario de búsqueda, estableciendo previamente el mapa con los errores en el objeto **`HttpServletRequest`**
    - **`WebUtil.forwardTo`**
  - Si no existen errores de validación de parámetros
    - Invoca el modelo para recuperar el objeto **`Account`** a partir de su identificador (**`findAccount`**)
    - Engancha el objeto **`Account`** en la request, con el nombre **`account`**, si el modelo lo ha localizado
    - Pasa el control a la página JSP que mostrará los datos de la cuenta (**`ShowAccount.jsp`**)
      - **`WebUtil.forwardTo`**

FindAccountForm.jsp (1)

```
<%@ page import="java.util.Map" %>

<html>
<head>
  <title>Find Account Form</title>
</head>

<body text="#000000" bgcolor="#ffffff">

<%-- Get errors. --%>

<%
    String accountIdErrorMessage = "";
    Map<String, String> errors = (Map<String, String>)
   request.getAttribute("errors");

    if (errors != null) {

        String errorHeader = "<font color=\"red\"><b>";
        String errorFooter = "</b></font>";

        if (errors.containsKey("accountId")) {
            accountIdErrorMessage = errorHeader +
                errors.get("accountId") + errorFooter;
        }
    }
```

FindAccountForm.jsp (2)

```
    String accountId = request.getParameter("accountId");
    if (accountId==null) {
        accountId="";
    }

%>

<form method="GET" action="FindAccount">

<table width="100%" border="0" align="center" cellspacing="12">

<%-- Account Identifier --%>

    <tr>
        <th align="right" width="50%">
            Account Identifier
        </th>
        <td align="left">
            <input type="text" name="accountId"
                value="<%= accountId %>"
                size="16" maxlength="16">
            <%= accountIdErrorMessage %>
        </td>
    </tr>
```

FindAccountForm.jsp (y 3)

```
<%-- Search button --%>

    <tr>
        <td width="50%"></td>
        <td align="left" width="50%">
            <input type="submit" value="Find Account">
        </td>
    </tr>

</table>

</form>

</body>

</html>
```

Formulario de Búsqueda de una Cuenta

- Página JSP que muestra el formulario de búsqueda de una cuenta bancaria
  - Al igual que el formulario de creación de una cuenta, comprueba si en la request hay un mapa con errores
  - Si hay un error asociado al parámetro **`accountId`**, lo muestra en el formulario, al lado del campo
- El envío del formulario genera una petición **`GET`** al servlet **`FindAccountServlet`**

ShowAccount.jsp (1)

```
<%@ page import="es.udc.pojo.servjsptutorial.model.account.Account" %>

<html>
<head>
  <title>Account Data</title>
</head>

<body text="#000000" bgcolor="#ffffff">

<%
    Account account = (Account) request.getAttribute("account");
%>

<div align="center">
  <p>
    <font color="#000099" face="Arial, Helvetica, sans-serif">
      <b>Account Information</b>
    </font>
  </p>
</div>

<div align="center">
```

ShowAccount.jsp (2)

```
<%
if (account!=null) {
%>

    <table border="1" align="center" width="35%">

        <tr>
            <th width="60%">Account Identifier</th>
            <td width="30%" align="center"><%= account.getAccountId()
%></td>
        </tr>

        <tr>
            <th width="60%">User Identifier</th>
            <td width="30%" align="center"><%= account.getUserId()
%></td>
        </tr>

        <tr>
            <th width="60%">Balance</th>
            <td width="30%" align="center"><%= account.getBalance()
%></td>
        </tr>

    </table>
```

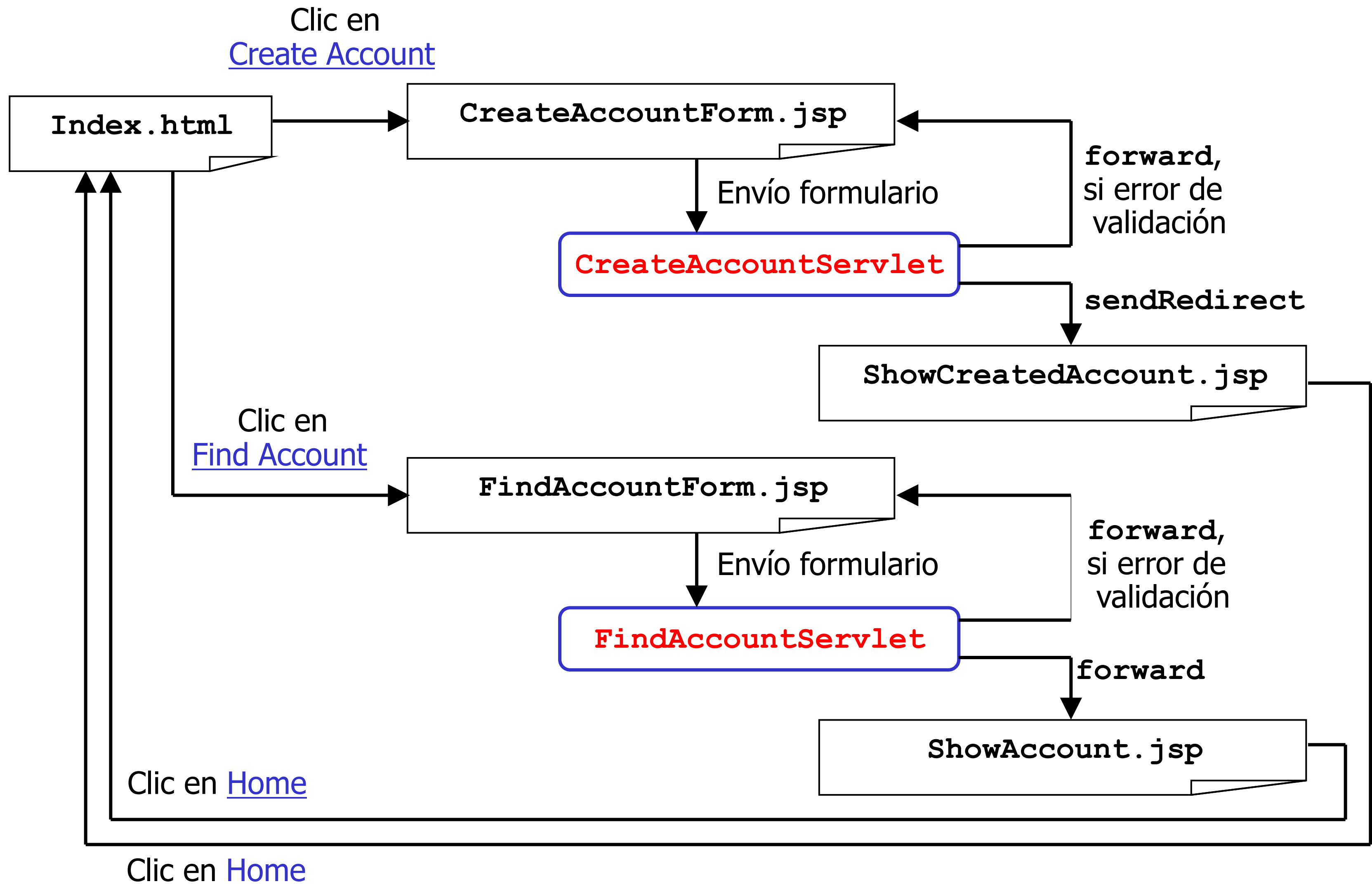
ShowAccount.jsp (y 3)

```
<%
} else {
%>

        <font color="#000099" face="Arial, Helvetica, sans-serif">
          <b>Account not found</b>
        </font>
<%
}
%>

</div>

<br/>
<a href="Index.html">Home</a>
<br/>
</body>

</html>
```

Resultado de Búsqueda de una Cuenta

- Página JSP que muestra los datos de una cuenta bancaria
  - Importa el objeto **`Account`**
  - Utiliza scriptlets para recuperar el objeto **`Account`** enganchado a la request (atributo **`account`**)
    - Si existe una **`Account`**, utiliza expresiones para mostrar sus campos en una tabla HTML
    - Si no existe, muestra el mensaje de “Account not found”

Resumen pojo-servjsptutorial



Empaquetamiento de una aplicación Web (1)

- Una aplicación Web se empaqueta en un fichero **`.war`**
- **`jar cvf aplicacionWeb.war directorio`**
  - Opciones similares al comando Unix **`tar`**
  - El nombre de una aplicación Web no tiene que coincidir con el de su fichero **`.war`**
    - El nombre se decide al instalar el fichero **`.war`** en el servidor Web
- La fase **`package`** de Maven genera un fichero **`.war`** cuando el tipo del proyecto Maven es **`war`**
- Estructura de un fichero **`.war`**
  - Directorio **`WEB-INF/classes`**
    - Ficheros **`.class`** que conforman la aplicación Web, agrupados en directorios según su estructura en paquetes
    - ¡Sin ficheros fuente!
  - Directorio **`WEB-INF/lib`**
    - Ficheros **`.jar`** de librerías que usa la aplicación
    - ¡Sin ficheros fuente!

Empaquetamiento de una aplicación Web (2)

- Estructura de un fichero **`.war`** (cont)
  - **`WEB-INF/web.xml`**
    - Configuración estándar de la aplicación Web
  - Directorio raíz y subdirectorios (excepto **`WEB-INF`**)
    - Vista de la aplicación (e.g.: ficheros HTML, páginas JSP, imágenes, etc.)
    - Visible a los navegadores
      - Lo que hay debajo de **`WEB-INF`** sólo es visible a los servlets y páginas JSP de la aplicación
- Un fichero **`.war`** se puede instalar (deployment) en cualquier servidor de aplicaciones Web que implemente la API de Servlets (en la práctica, cualquier servidor de aplicaciones Java)

Empaquetamiento de una aplicación Web (y 3)

- Un contenedor de aplicaciones Web usa un cargador de clases distinto para cada **`war`** instalado => independencia entre aplicaciones Web
  - Distintas aplicaciones Web pueden usar versiones distintas de una misma clase
  - Dos aplicaciones Web nunca podrán compartir una variable global
    - Existe una instancia de un Singleton por cada aplicación que lo use

jar tvf pojo-servjsptutorial.war

```
CreateAccountForm.jsp
FindAccountForm.jsp
Index.html
ShowAccount.jsp
ShowCreatedAccount.jsp
WEB-INF/classes/es/udc/pojo/servjsptutorial/model/account/Account.class
WEB-INF/classes/es/udc/pojo/servjsptutorial/model/accountservice/
   AccountServiceImpl.class
WEB-INF/classes/es/udc/pojo/servjsptutorial/web/
   CreateAccountServlet.class
WEB-INF/classes/es/udc/pojo/servjsptutorial/web/
   FindAccountServlet.class
WEB-INF/classes/es/udc/pojo/servjsptutorial/web/util/
   PropertyValidator.class
WEB-INF/classes/es/udc/pojo/servjsptutorial/web/util/WebUtil.class
WEB-INF/lib/pojo-modelutil-1.0.jar
WEB-INF/web.xml
META-INF/MANIFEST.MF
META-INF/maven/es.udc.pojo/pojo-servjsptutorial/pom.xml
META-INF/maven/es.udc.pojo/pojo-servjsptutorial/pom.properties
```

WEB-INF/web.xml (1)

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

<!DOCTYPE web-app
        PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

  <distributable/>

  <!-- ========================= Servlets ======================== -->

  <servlet>
    <servlet-name>CreateAccount</servlet-name>
    <servlet-class>es.udc.pojo.servjsptutorial.web.CreateAccountServlet
        </servlet-class>
  </servlet>

  <servlet>
    <servlet-name>FindAccount</servlet-name>
    <servlet-class>es.udc.pojo.servjsptutorial.web.FindAccountServlet
        </servlet-class>
  </servlet>
```

WEB-INF/web.xml (2)

```
  <!-- ================= Servlet mapping ========================= -->

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

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

  <!-- ======================== Session ========================== -->

  <session-config>
    <session-timeout>30</session-timeout>
  </session-config>

  <!-- ====================== Welcome page ======================= -->

  <welcome-file-list>
    <welcome-file>Index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

WEB-INF/web.xml (3)

- Hasta la versión 2.3 de la especificación de servlets, los tags que se podían usar dentro del fichero **`web.xml`** estaban especificados en una DTD
- Desde la versión 2.4 están especificados en un XML-Schema
- Los ejemplos y Tapestry sólo requieren la API de Servlets 2.3, por eso no se ha especificado una versión superior

WEB-INF/web.xml (y 4)

- El anterior fichero **`web.xml`** sólo muestra algunos tags típicos
  - Existe un gran número de tags (la mayoría opcionales) que permiten expresar muchas opciones de configuración
- **`distributable`**
  - La aplicación puede funcionar en cluster
  - Nuestras aplicaciones siempre deberían estar diseñadas e implementadas para que puedan funcionar en cluster
- **`servlet`**
  - Declara cada clase servlet (**`servlet-class`**) con un nombre (**`servlet-name`**)
- **`servlet-mapping`**
  - Define la URL asociada (**`url-pattern`**) a cada servlet definido (**`servlet-name`**)
- **`session-config`**
  - **`session-timeout`** especifica el tiempo máximo de mantenimiento para una sesión que haya dejado de usarse
- **`welcome-file-list`**
  - **`welcome-file`** indica la página devuelta por el servidor cuando se accede a "**`/nombreAplicación`**"

Frameworks Web Java (1)

- El ejemplo ilustra un enfoque de implementación de aplicaciones Web orientado a procesar peticiones HTTP individuales (acciones)
- Existen frameworks de más alto nivel que siguen este mismo enfoque (frameworks orientados a acción)
  - Struts
  - Spring MVC
- Otros frameworks (frameworks orientados a componentes) siguen un enfoque distinto: modelan cada página Web como un componente que puede reaccionar a diversos eventos
  - Tapestry
  - Wicket
  - JSF
  - Seam

Frameworks Web Java (y 2)

- Todos estos frameworks (orientados a acción o a componentes)
  - Requieren como mínimo la API de Servlets
    - Tapestry, el framework Web que usaremos, de hecho, sólo requiere la API de Servlets
  - Proporcionan soporte para facilitar aspectos tales como: internacionalización de aplicaciones, gestión de layouts, aplicación de políticas globales al procesamiento de peticiones HTTP, etc.