Desarrollo de Aplicaciones Web

JSP. JavaBeans.

USC UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Pablo García Tahoces

Código servlet

```
Public class pbi_servlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                HttpServletResponse response)
                throws IOException {

                // Lógica de negocio, incluyendo posible
                // acceso a DB

                // Redirigir la respuesta a un JSP
    }
}
```

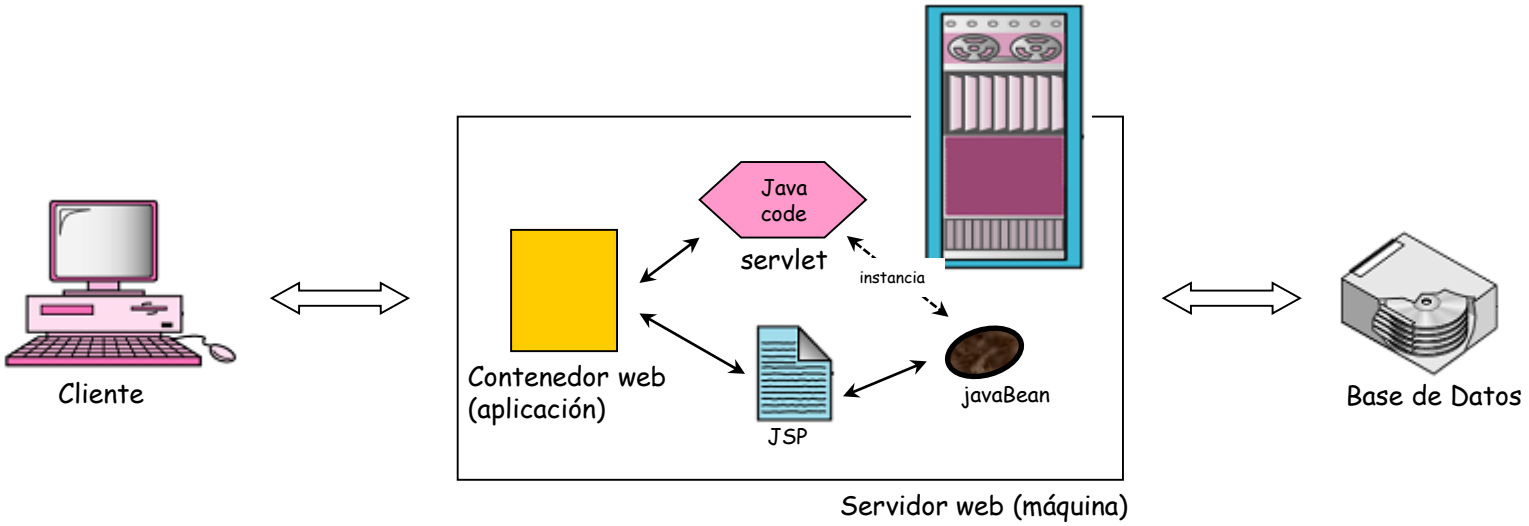
Lógica de negocio y presentación desacoplados!!!,
Pero lógica de negocio dentro del servlet → Dificulta reutilización del código java….

```
Public class pbi_servlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                HttpServletResponse response)
                throws IOException {

                // Servlet instancia JavaBean

                // Lógica de negocio, incluyendo posible
                // acceso a DB, en javaBean

                // Redirigir la respuesta a un JSP
    }
}
```

Servlet (Controlador) Lógica de negocio y datos (Modelo) presentación (Vista) desacoplados!!!,

Modelo Vista Controlador (MVC)

Java Server Pages (JSP)



Java Server Pages (JSP)

MVC

Modelo → Incluye lógica de negocios y el estado de los datos. Es la única parte del sistema que se comunica con la DB.

Vista → Responsable de la presentación de los datos.

Controlador → Controla la entrada, desde la petición (request) y decide cómo afecta ello al modelo, actualizándolo.

JSP. JavaBeans.

Limitaciones de los *scriptlets*:

- A medida que la página JSP se complica, los *scriptlets* insertados se hacen largos y complicados.
- La funcionalidad del *scriplet* está ligada a la JSP para la que ha sido desarrollada → Complica reutilización del código.

Solucion.- mover datos y funcionalidad presentes en los *scriplets* a componentes (*JavaBeans*), lo cual permite:

- Reutilización del código
- Separación de papeles (diseñador, desarrollador).

JSP. JavaBeans.

JavaBeans.- son componentes que exponen parte de sus datos, llamados propiedades. Los datos son almacenados por los JavaBeans y se pueden recuperar posteriormente.

JavaBeans (definición formal).- clase java que mantiene algunos datos (propiedades), sigue unas determinadas convenciones de codificación y recupera/asigna valores a las propiedades mediente métodos get*xxx*/set*xxx*.

JavaBeans.- los métodos get*xxx*/set*xxx*, son invocados mediante lenguaje de etiquetas.

Ej.-

```
<jsp:setProperty name="miCoche" property="fabricante" value="Renault"/>
```

JSP. JavaBeans.

Construcción de un Java Bean:

```
public class CarBean {

        public CarBean() {
        }

        private String fabricante= “Fiat”;

        public String getFabricante() {
                return fabricante;
        }

        public void setFabricante(String fabricante) {
                this.fabricante= fabricante;
        }

}
```

JSP. JavaBeans.

Uso de un JavaBean mediante *scriplets*:

```
<html>
    <head>
        <title> Uso de JavaBeans </title>
    </head>
    <body>
        <% CarBean miCoche= new CarBean(); %>

        Mi coche es un <%= miCoche.getFabricante() %> <br/>

        <% miCoche.setFabricante("Renault"); %>

        Ahora mi coche es un <%= miCoche.getFabricante() %>
    </body>
</html>
```

JSP. JavaBeans.

Etiquetas JavaBean.- mecanismo de llamada a JavaBeans, mediante etiquetas análogas a las del HTML. Existen tres etiquetas:

- <jsp:useBean> .- localiza y hace una llamada a un JavaBean.

  Ej.- <jsp:useBean id="miCoche" class="CarBean" />

- <jsp:setProperty> .- establece el valor de una propiedad de un componente.

  Ej.- <jsp:setProperty name="miCoche" property="fabricante" value="Renault" />

- <jsp:getProperty> .- obtiene el valor de una propiedad almacenada en un componente.

  Ej.- <jsp:getProperty name="miCoche" property="fabricante" />

JSP. JavaBeans.

Uso de un JavaBean mediante etiquetas:

```
<html>
    <head>
        <title> Uso de JavaBeans </title>
    </head>
    <body>
        <jsp:useBean id="miCoche" class="CarBean"/>

        Mi coche es un <jsp:getProperty name="miCoche" property="fabricante" /><br/>

        <jsp:setProperty name="miCoche" property="fabricante" value="Renault" />

        Ahora mi coche es un <jsp:getProperty name="miCoche" property="fabricante" />
    </body>
</html>
```

JSP. JavaBeans.

<jsp:useBean>, también permite la creación de un *bean* en caso de no existir.

```
<jsp:useBean id="miCoche" class="pbi.CarBean" scope="request">
```

```
pbi.CarBean miCoche= null;

Syncronized(request){

    miCoche=  (pbi.CarBean)_jpx_page_context.getAttribute("miCoche",
                                        PageContext.REQUEST_SCOPE);

    if (miCoche == null){
            miCoche= new  pbi.CarBean();
            _jpx_page_context.setAttribute("miCoche", miCoche,
                                        PageContext.REQUEST_SCOPE);
}
```

Nota.- *al instanciar se crea un atributo automáticamente, lo cual permite luego intercambiar datos fácilmente entre diferentes JSPs.*

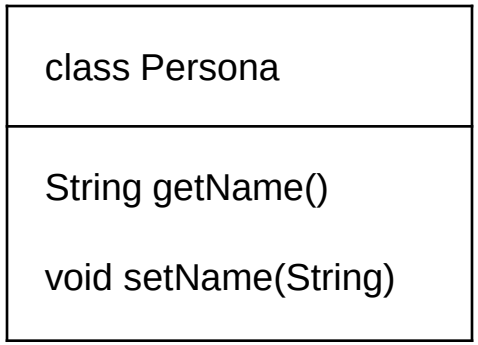
JSP. JavaBeans.

<jsp:useBean>, con un cuerpo. *Setting* condicional.

```
<jsp:useBean id="miCoche" class="CarBean" scope="page"/>
    <jsp:setProperty name="miCoche" property="fabricante" value="Audi"/>
</ jsp:useBean >
```

JSP. JavaBeans.

Referencias polimórficas

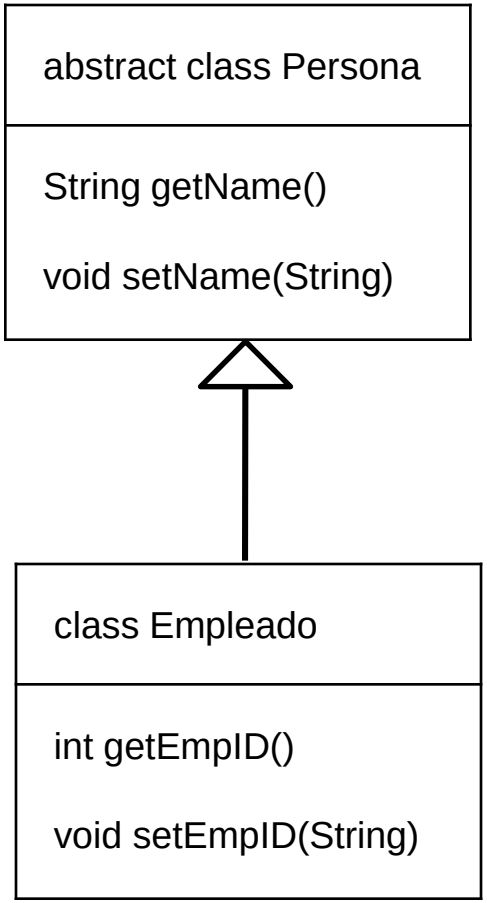



```
Persona persona= null;
    if (persona == null)
        persona= new Persona();
```

```
<jsp:useBean id="persona" class="Persona" scope="page"/>
```

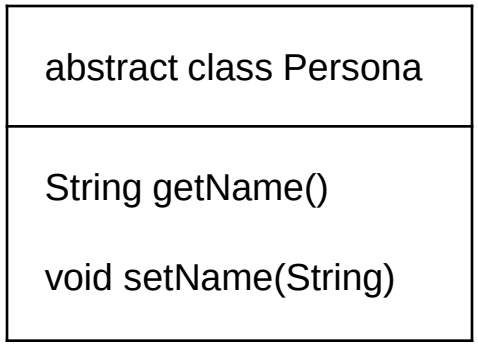
JSP. JavaBeans.

Referencias polimórficas



JSP. JavaBeans.

Referencias polimórficas



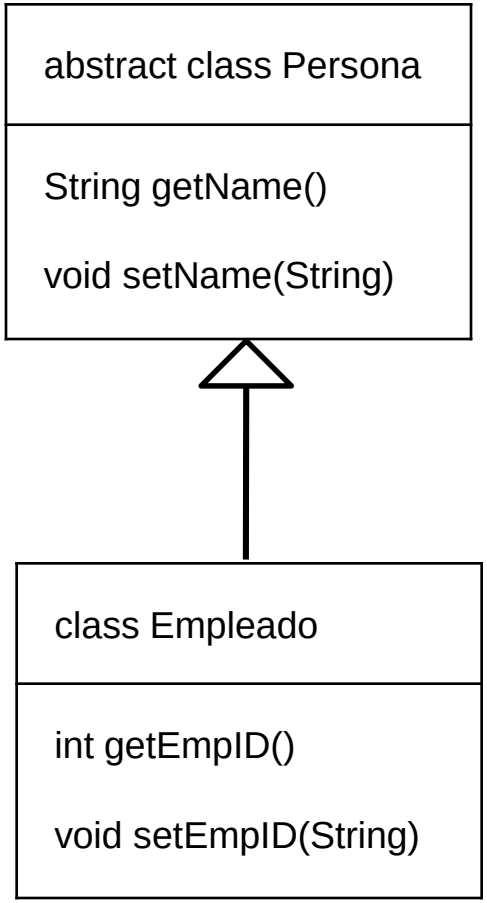

```
Persona persona= null;
    if (persona == null)
        persona= new Persona();
```

<jsp:useBean id="persona" class="Persona" scope="page"/>

ERROR → *InstantalizationException*

JSP. JavaBeans.

Referencias polimórficas

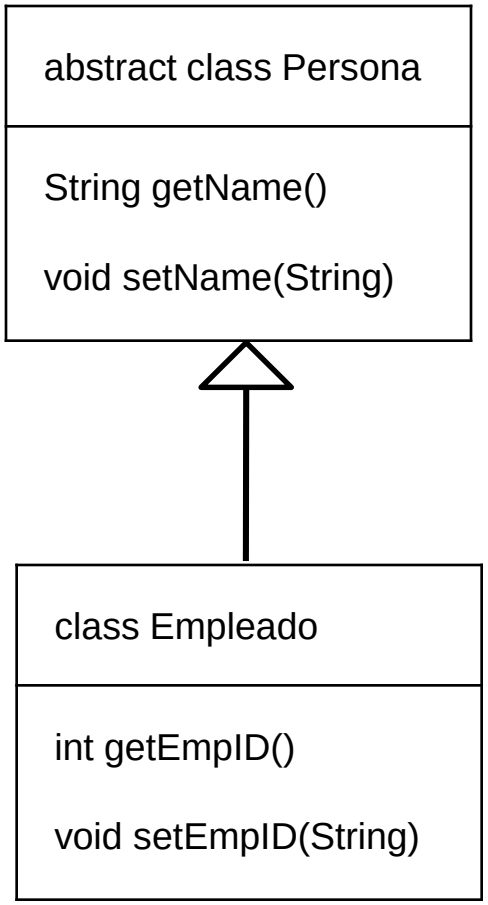



```
Persona persona= null;
    if (persona == null)
        persona= new Empleado();
```

```
<jsp:useBean id="persona" type="Persona"
    class=Empleado scope="page"/>
```

JSP. JavaBeans.

Formularios:




```
public abstract class Persona {
        private String name;

        public void setName (String name) {
                this.name=name;
        }

        public String getName() {
                return name;
        }
}
```

```
public class Empleado extends Persona {
        private int empID;

        public void setEmpID (int empID) {
                this.empID=empID;
        }

        public int getEmpID() {
                return empID;
        }
}
```

JSP. JavaBeans.

Formularios:

```
<html>
    <body>
        <form action="miBean.jsp">
            nombre: <input type="text" name="userName">
            ID#: <input type="text" name="userID">
            <input type= "submit">
        </form>
    </body>
</html>
```

```
<jsp:useBean id="persona" type="Persona"class=Empleado/>
<% persona.setName(request.getParameter("userName")); %>
```

JSP. JavaBeans.

Formularios:

```
<jsp:useBean id="persona" type="Persona"class=Empleado/>
    <jsp: setProperty name="persona" property="name"
        value="<%=request.getParameter("userName")); %>" />
</ jsp:useBean>

<jsp:useBean id="persona" type="Persona"class=Empleado/>
    <jsp: setProperty name="persona" property="name"
        param="userName" />
</ jsp:useBean>
```

JSP. JavaBeans.

Formularios:

```
<html>
    <body>
        <form action="miBean.jsp">
            nombre: <input type="text" name="name">
            ID#: <input type="text" name="userID">
            <input type= "submit">
        </form>
    </body>
</html>
```

```
<jsp:useBean id="persona" type="Persona" class=Empleado/>
    <jsp: setProperty name="persona" property="name"  />
</ jsp:useBean>
```

JSP. JavaBeans.

Formularios:

```
<html>
    <body>
        <form action="miBean.jsp">
            nombre: <input type="text" name="name">
            ID#: <input type="text" name="empID">
            <input type= "submit">
        </form>
    </body>
</html>
```

```
<jsp:useBean id="persona" type="Persona" class=Empleado/>
    <jsp: setProperty name="persona" property="*"  />
</ jsp:useBean>
```

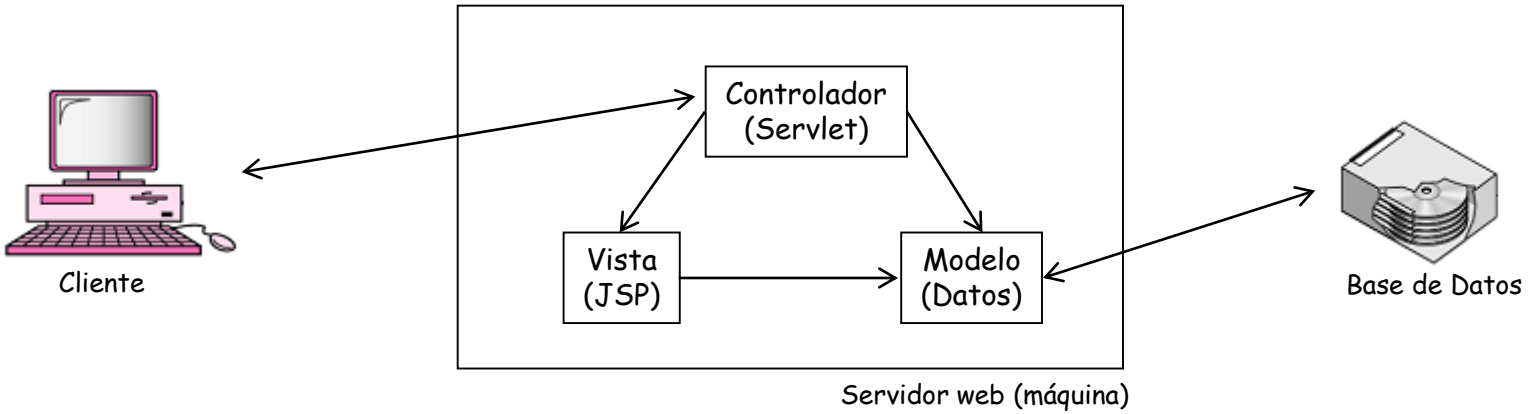
Java Server Pages (JSP)

MVC: Modelo Vista Controlador

	Propósito	Descripción
Modelo	Mantenimiento de los datos	Lógica de negocio más una o más estructuras de datos tales como una base de datos relacional.
Vista	Visualización de todos o parte de los datos	Interfaz de usuario que muestra información sobre el modelo al usuario.
Controlador	Manejo de los eventos que afectan tanto al modelo como a la vista	El mecanismo de control del flujo mediante el cual el usuario interactúa con la aplicación.

Java Server Pages (JSP)

MVC: Modelo Vista Controlador



Modelo → Datos y estado de la aplicación

Vista → Presentación a través de la Interfaz de Usuario

Controlador → Interfaz entre *Vista* y *Modelo*, interceptando los datos de las diferentes peticiones (request).