



# CONSTRUÇÃO DE SISTEMAS DE SOFTWARE

## APLICAÇÕES WEB JAVA

### Exercícios



*Camada de Apresentação das Aplicações Web Java*

#### 1. Aplicação Web Java

- (a) Qual o estilo arquitectural destas aplicações?
- (b) Quais os dois principais componentes (programas) envolvidos e qual o papel de cada um deles?
- (c) Onde (i.e., em que máquinas) executa cada um dos componentes?
- (d) Que tipo de comunicação existe entre os dois?
- (e) A construção da camada de apresentação destas aplicações assenta em que tipo de elementos?
- (f) Nestas aplicações a parte estática da apresentação é suportada com *client side templates* ou *server side templates*? O que isso implica?
- (g) E como é suportada a definição da aparência do que é apresentado?
- (h) Explique o papel típico que assumem os diferentes componentes que se podem usar na construção destas aplicações.



## Exemplo: SaleSys

The screenshot shows a web browser window with two tabs. The top tab is titled 'Menu principal' and has the URL <http://localhost:8080/domain-model-jpa-web-v0/>. The bottom tab is titled 'Adicionar Cliente' and has the URL <http://localhost:8080/domain-model-jpa-web-v0/action/clientes/novoCliente>. The 'Adicionar Cliente' page contains fields for 'Designação' (Designation), 'Número de pessoa colectiva' (Collective Person Number), 'Telefone' (Phone), and a dropdown menu for 'Tipo de desconto' (Discount Type) which includes options like 'Sem desconto' (No discount), 'Percentagem do Total (acima de limiar)' (Percentage of Total (above threshold)), and 'Percentagem do Total Elegível' (Eligible Total Percentage). A 'Criar Cliente' button is also present. The browser's address bar shows the URL of the current tab.

## Exemplo: SaleSys

The screenshot shows a web browser window with two tabs. The top tab is titled 'Menu principal' and has the URL <http://localhost:8080/domain-model-jpa-web-v0/>. The bottom tab is titled 'SaleSys: efectuar venda' and has the URL <http://localhost:8080/domain-model-jpa-web-v0/action/vendas/novaVenda>. The 'Efectuar Venda' page contains a field for 'Número de pessoa colectiva' (Collective Person Number) and a 'Criar Venda' button. The browser's address bar shows the URL of the current tab.

## Exercícios

---

- (a) Qual o estilo arquitectural destas aplicações?
- (b) Quais os dois principais componentes (programas) envolvidos e qual o papel de cada um deles?
- (c) Onde (i.e., em que máquinas) executa cada um dos componentes?
- (d) Que tipo de comunicação existe entre os dois?
- (e) A construção da camada de apresentação destas aplicações assenta em que tipo de elementos?
- (f) Nestas aplicações a parte estática da apresentação é suportada com *client side templates* ou *server side templates*? O que isso implica?
- (g) E como é suportada a definição da aparência do que é apresentado?
- (h) Explique o papel típico que assumem os diferentes componentes que se podem usar na construção destas aplicações.

## Exercícios

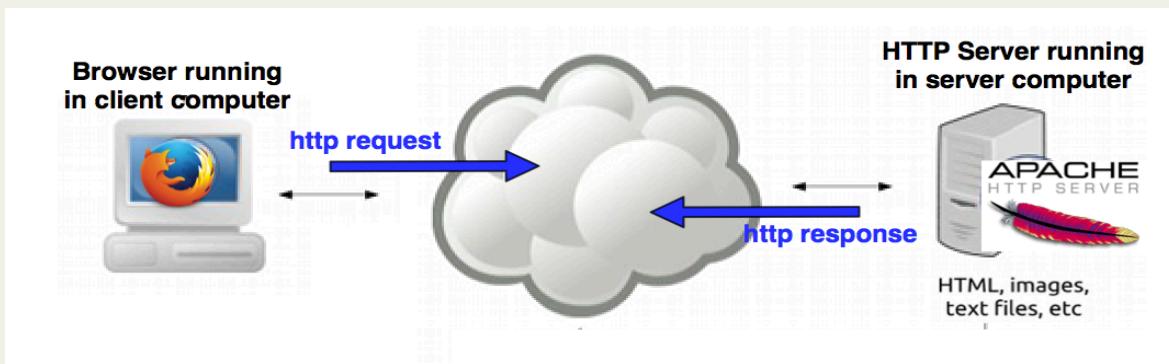
---

### 2. Web Server

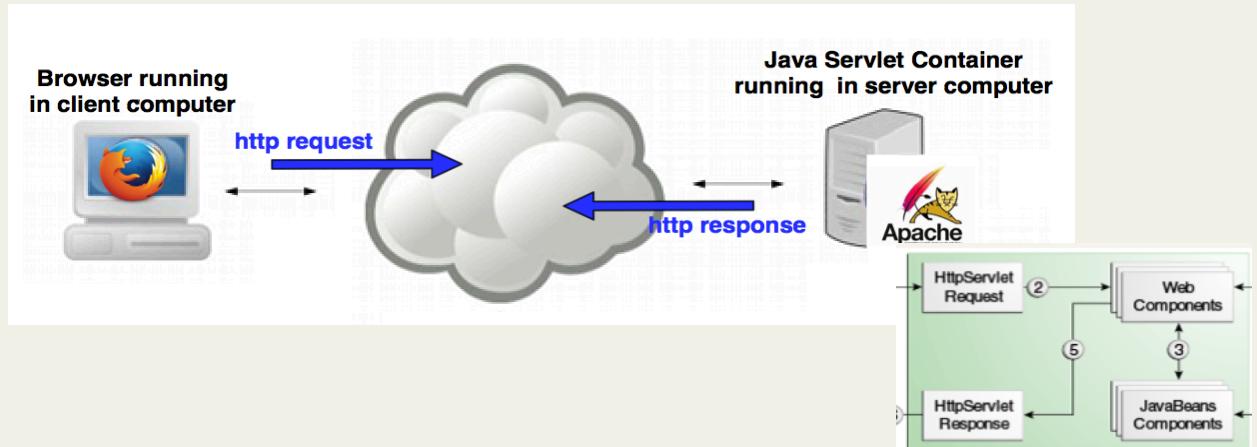
- (a) A que usualmente se chama um *Web Server*? E *Java Web Server*? Indique 2 exemplos de *Java Web Servers* que atualmente podemos usar para correr as nossas aplicações web Java.
- (b) Explique o que é um *web container* (ou *servlet container*) e o papel que cumpre no tratamento dos pedidos HTTP que chegam ao *Web Server*.



## Web Server



## Java Web Server



## Exercícios

---

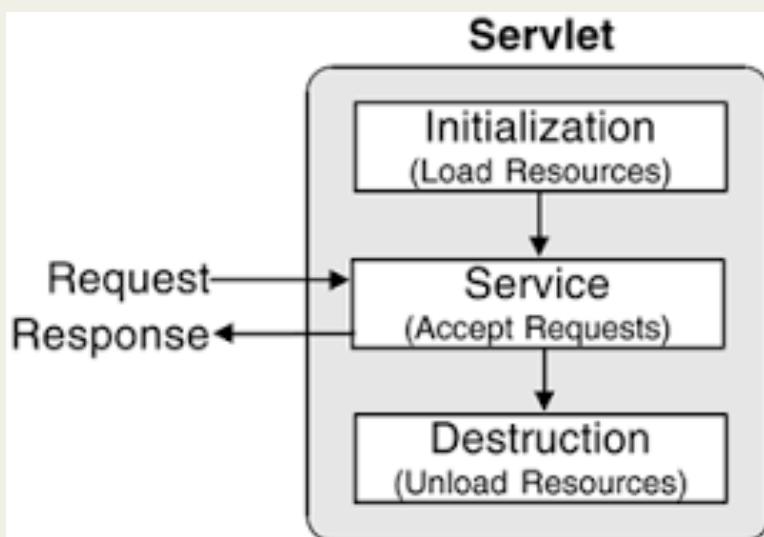
### 3. Servlets

- (a) Porque é que se diz que as *Servlets* permitem estender as capacidades de um *Web Server*?
- (b) Quais as suas principais características?
- (c) De que forma lhes é entregue a informação que foi enviada no pedido http?
- (d) Explique o papel do *web container* (ou *servlet container*) no seu ciclo de vida.



### ***Servlet Container: servlet life cycle***

---



## **Http Servlets**

---

- Classes que estendem a classe abstrata  
**javax.servlet.http.HttpServlet**
- Redefinem métodos correspondente aos pedidos que suportam

**doGet(HttpServletRequest req, HttpServletResponse res)**  
**doPost(HttpServletRequest req, HttpServletResponse res)**

...

- Através de uma anotação

**@WebServlet("...")**

podem definir que pedidos que lhe devem ser entregues

- subconjunto específico do espaço de URLs do servidor



## **GET e POST**

---

**void doGet(HttpServletRequest request, HttpServletResponse res)**

- **GET**

- envia a informação codificada com uma *string* que vai anexa ao url do pedido, depois de um **?**

<http://www.site.com/hello?key1=value1&key2=value2>

- a informação vai no **request** e pode ser acedida com método  
**getParameter(String)**

**request.getParameter("key1")**

- a operação deve ser *safe* e idempotente

- **POST**

- codifica a informação da mesma forma que o **GET** mas não a envia como texto no url
  - a informação vai numa mensagem separada, não estando sujeita a limites de tamanho e podendo ser acedida da mesma maneira



## **javax.servlet.RequestDispatcher**

---

### **RequestDispatcher**

- objetos que recebem pedidos num determinado caminho e os enviam para os correspondentes recursos —um *servlet*, um JSP ou ficheiro HTML— instalado no *web server*
- são criados pelo *container*
- podem ser obtidos num *servlet* a partir de um **ServletContext** (caminho tem se ser absoluto, ou melhor, relativo à raiz do contexto e começar com /) ou a partir de um **Request** (caminho pode ser relativo ao pedido corrente ou absoluto)

```
getServletContext().getRequestDispatcher("/helloWeb");  
request.getRequestDispatcher("/helloWeb");  
request.getRequestDispatcher("helloWeb");
```

Existe apenas um objeto **ServletContext** por *web app/JVM*

## **Exercícios**

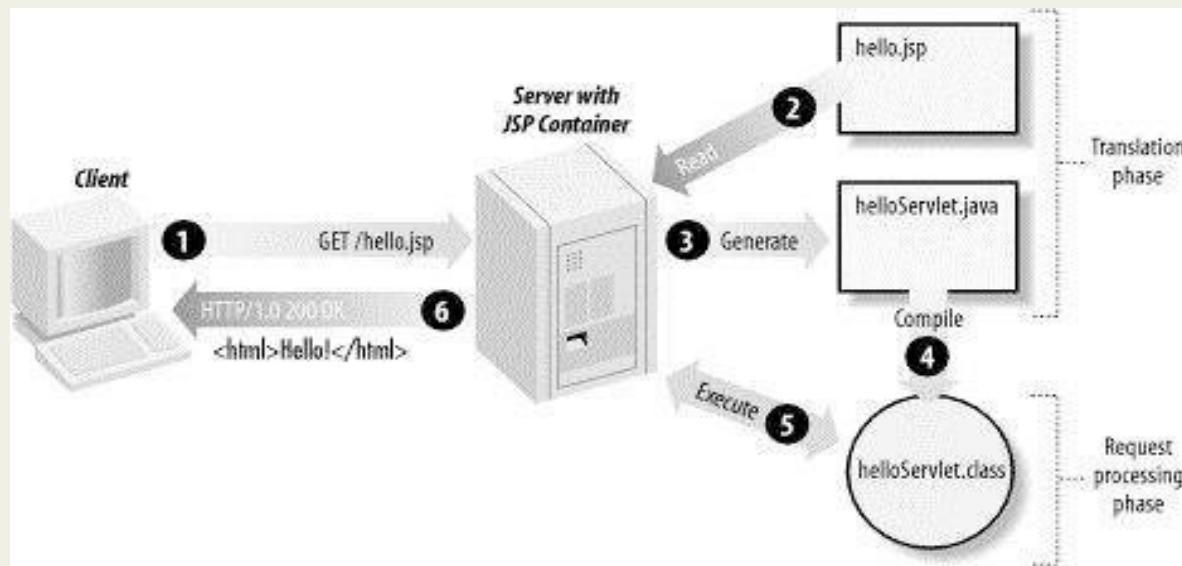
---

### *4. Java Server Pages (JSP)*

- (a) Que relação existe entre *Java Server Pages* e *Servlets*?
- (b) O que é a *Java Standard Tag Language* (JSTL) e que vantagens tem a sua utilização sobre os *scriptlets* no JSP?
- (c) O que são *Java Beans* e porque são úteis na escrita de JSP com a *Expression Language* (EL)?
- (d) De que forma podem ser usadas na escrita de páginas JSP?



# Java Server Pages (JSP)



# Java Server Pages (JSP)

```
<html>
<head><title>First JSP</title></head>
<body>
<%
    double num = Math.random();
    if (num > 0.95) {
%
        <h2>You'll have a luck day!</h2><p>(<%= num %>)</p>
<%
    } else {
%
        <h2>Well, life goes on ... </h2><p>(<%= num %>)</p>
<%
    }
%
    <a href="<%= request.getRequestURI() %>"><h3>Try Again</h3></a>
</body>
</html>
```

## JSP com *scriptlets*

Só para exemplificar tradução!

**resposta**

```
<html>
<h2>You'll have a luck day!</h2>
<p>(0.987)</p>
<a href="first.jsp"><h3>Try Again</h3></a>
</html>
```

**excerto do servlet**

```
out.write("<html>\r\n  ");
double num = Math.random();
if (num > 0.95) {
    out.write("<h2>You will have a luck day!");
    out.write("</h2><p>(");
    out.print( num );
    out.write("</p>\r\n");
} else {
    out.write("\r\n  ");
    out.write("<h2>Well, life goes on ... ");
    out.write("</h2><p>(");
    out.print( num );
    out.write("</p>\r\n");
}
out.write("<a href=\"\"");
out.print( request.getRequestURI() );
out.write("\">");
out.write("<h3>Try Again</h3></a>\r\n");
out.write("</html>\r\n");
```



## JSTL

---

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<html>
<head>
<title>Example c:forEach tag in JSTL</title>
</head>
<body>
<c:forEach var="counter" begin="1" end="10">
    <c:out value="${counter}" />
</c:forEach>
</body>
</html>
```



## JSTL tags: out, iteration, conditionals

---

- **c:out**  
avalia a expressão e faz output todo resultado para o objeto *JspWriter* corrente
- **c:forEach**

```
<c:forEach var="i" begin="1" end="5">
    Item <c:out value="${i}" /><p>
</c:forEach>
```

- **c:forTokens**
- **c:if**

```
<c:forTokens items="Zara,nuha,roshy" delims="," var="name">
    <c:out value="${name}" /><p>
</c:forTokens>

<c:if test="${salary > 2000}">
    <p>My salary is: <c:out value="${salary}" /><p>
</c:if>
```



## JSTL tags: + conditionals

c:choose

```
<c:set var="salary" scope="session" value="${2000*2}" />
<p>Your salary is : <c:out value="${salary}" /></p>
<c:choose>
    <c:when test="${salary <= 0}">
        Salary is very low to survive.
    </c:when>
    <c:when test="${salary > 1000}">
        Salary is very good.
    </c:when>
    <c:otherwise>
        No comment sir...
    </c:otherwise>
</c:choose>
```



Tag	Description
<c:out>	To write something in JSP page, we can use EL also with this tag
<c:import>	Same as <jsp:include> or include directive
<c:redirect>	redirect request to another resource
<c:set>	To set the variable value in given scope.
<c:remove>	To remove the variable from given scope
<c:catch>	To catch the exception and wrap it into an object.
<c:if>	Simple conditional logic, used with EL and we can use it to process the exception from <c:catch>
<c:choose>	Simple conditional tag that establishes a context for mutually exclusive conditional operations, marked by <c:when> and <c:otherwise>
<c:when>	Subtag of <c:choose> that includes its body if its condition evaluates to 'true'.
<c:otherwise>	Subtag of <c:choose> that includes its body if its condition evaluates to 'false'.
<c:forEach>	for iteration over a collection
<c:forTokens>	for iteration over tokens separated by a delimiter.
<c:param>	used with <c:import> to pass parameters
<c:url>	to create a URL with optional query string parameters



## JSTL + EL

```
17 <%-- Using JSTL forEach and out to loop a list and display items in table --%>
18 <table>
19 <tbody>
20 <tr><th>ID</th><th>Name</th><th>Role</th></tr>
21 <c:forEach items="${requestScope.empList}" var="emp">
22 <tr><td><c:out value="${emp.id}"></c:out></td>
23 <td><c:out value="${emp.name}"></c:out></td>
24 <td><c:out value="${emp.role}"></c:out></td></tr>
25 </c:forEach>
26 </tbody>
27 </table>
28 <br><br>
```

ID	Name	Role
1	Pankaj	Developer
2	Meghna	Manager

## JSP Expression Language (EL)

- Expressões **`\${el-expression-goes-here}`** são avaliada para um valor, o qual pode ser passado para uma ação JSP ou expresso como texto no *output* do JSP (enviado para o objecto *JSPWriter* corrente)

**`\${requestScope.empList}`**

**`\${employee.name}`**      ou      **`\${employee['name']}`**

- Para ter objetos que podem ser acedidos num JSP pela EL basta colocá-los em atributos no âmbito desejado com o método **setAttribute()**

- Para usar um Java Bean num JSP fazer a sua declaração com

```
<jsp:useBean id = "instanceName" class = "package.className"
              scope = "page|request|session|application" ... />
```

```
<jsp:useBean id = "date" class = "java.util.Date" />
```

## Variáveis implícitas EL

- Além das variáveis definidas pelo programador, há várias variáveis predefinidas que podem ser usadas nas expressões EL de uma página

### **param**

– A Map of all request parameters on the current request

- Exemplos

EL

```
-----  
${param.foo}  
${paramValues.foo}  
${header['user-agent']}  
${pageContext.request.contextPath}  
${cookie.somename}
```

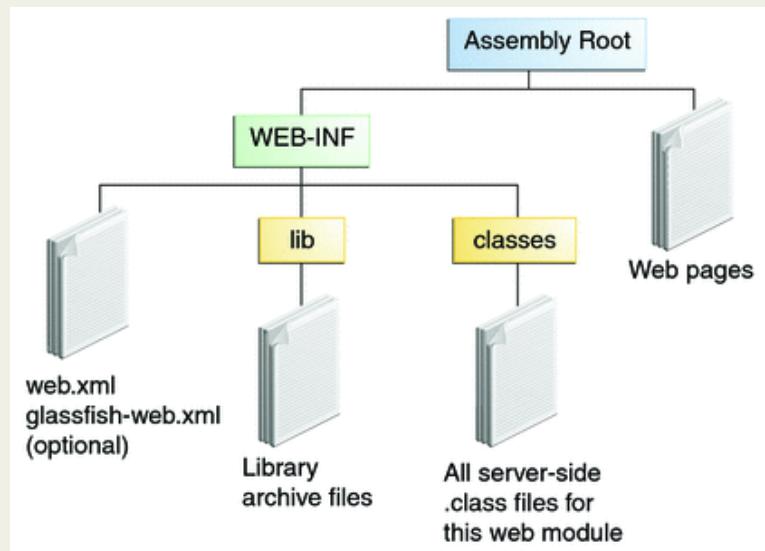
Scriptlet (out.print and null checks omitted!)

```
-----  
request.getParameter("foo");  
request.getParameterValues("foo");  
request.getHeader("user-agent");  
request.getContextPath();  
Too verbose (start with request.getCookies())
```

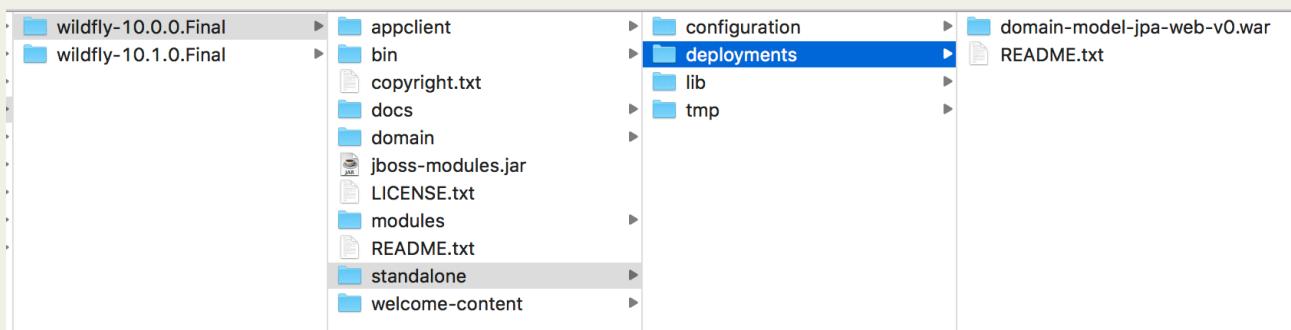
## Exercícios

- De que forma têm de ser empacotados os elementos usados na construção de uma aplicação web Java para serem instalados num Web Server?
- Indique de que forma está organizado o *web application archive* de uma aplicação web Java.

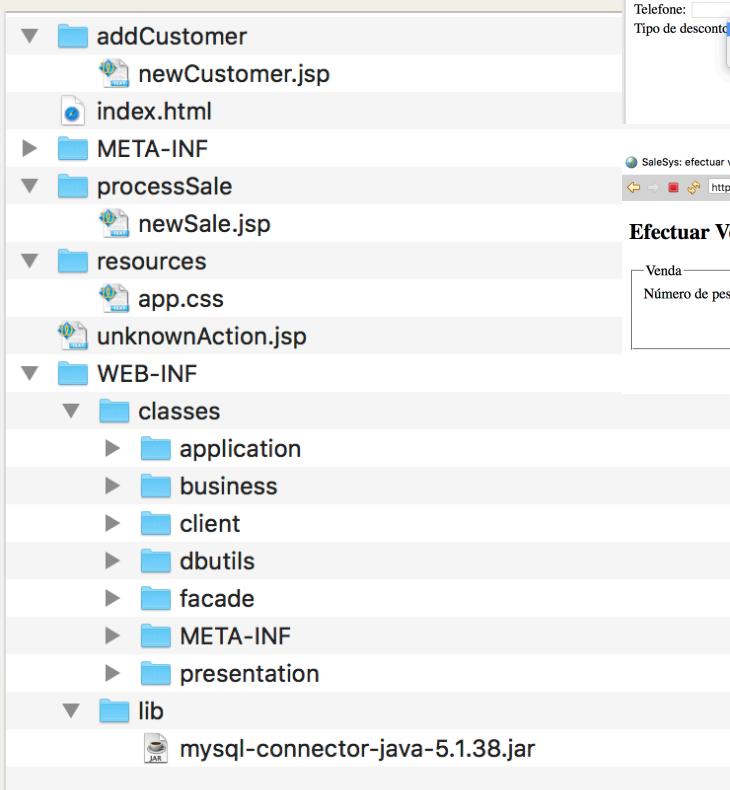
## Exercícios



## SaleSys: Exemplo



## SaleSys: Exemplo



Adicionar Cliente

Designação:

Número de pessoa colectiva:

Telefone:

Tipo de desconto:  Sem desconto  
Percentagem do Total (acima de limitar)  
Percentagem do Total Elegível

Criar Cliente

Efectuar Venda

Venda

Número de pessoa colectiva:

Criar Venda

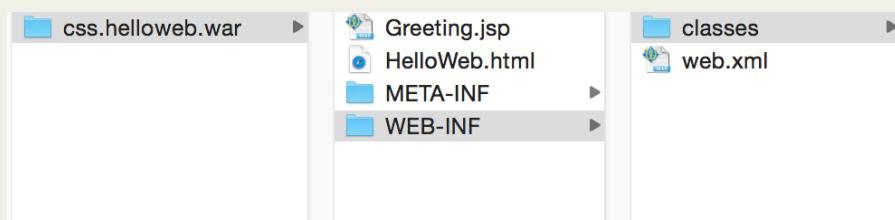
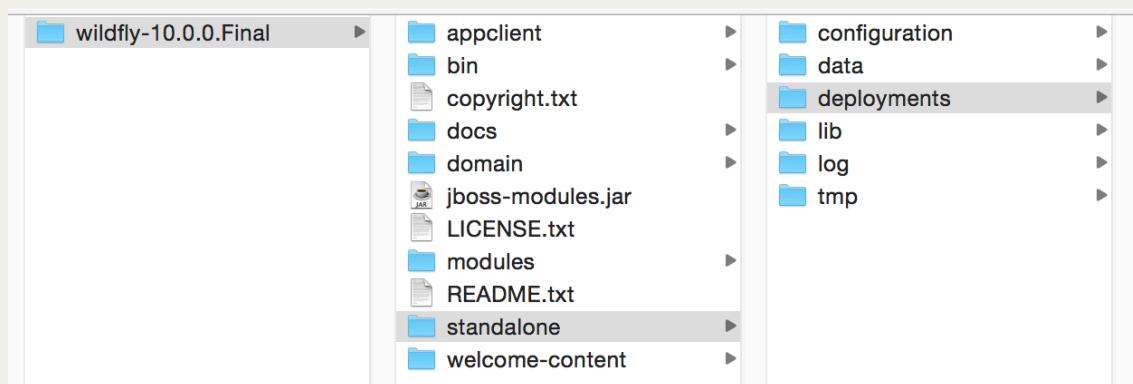
## Exercícios

6. Pretende-se programar a camada de apresentação de uma aplicação web Java muito simples utilizando *Servlets* e JSP+JSTL+EL. O objetivo principal é utilizar num caso concreto muitos dos conceitos cobertos nas perguntas anteriores.



Comece com uma versão simples em que não há necessidade de qualquer validação de negócio e gera um número de pontos aleatório. Modifique posteriormente essa versão de forma a passar a validar os dados de entrada (i.e., que o nome e o título fornecidos são conhecidos pelo

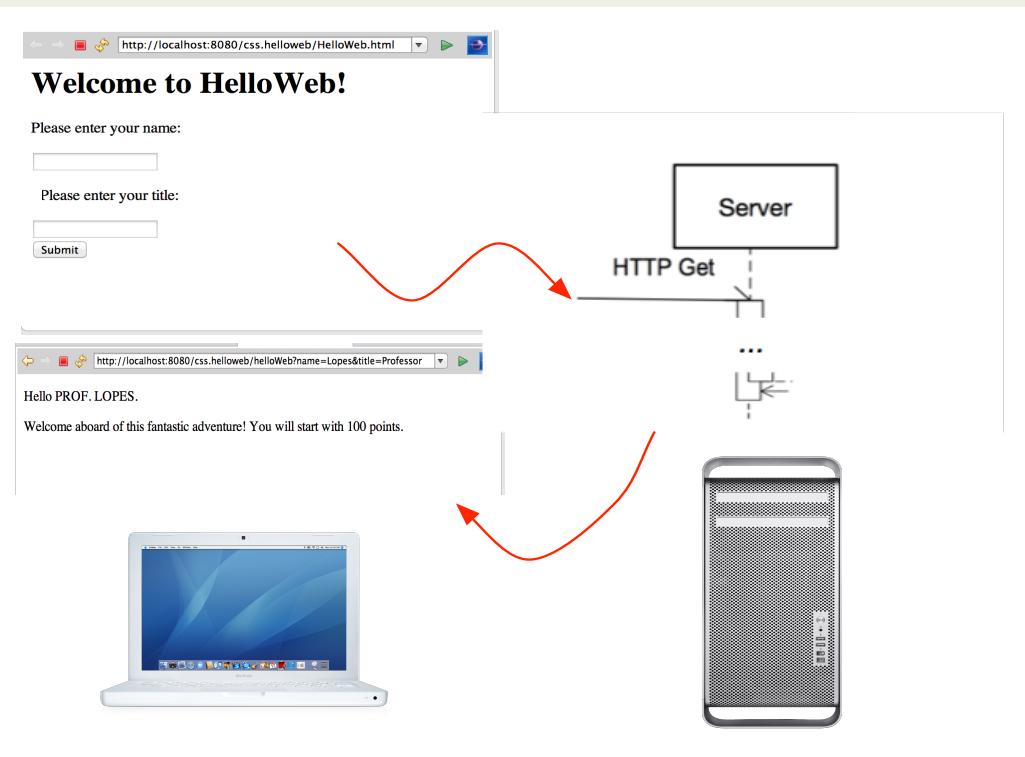
## Exercícios



Ciências  
ULisboa

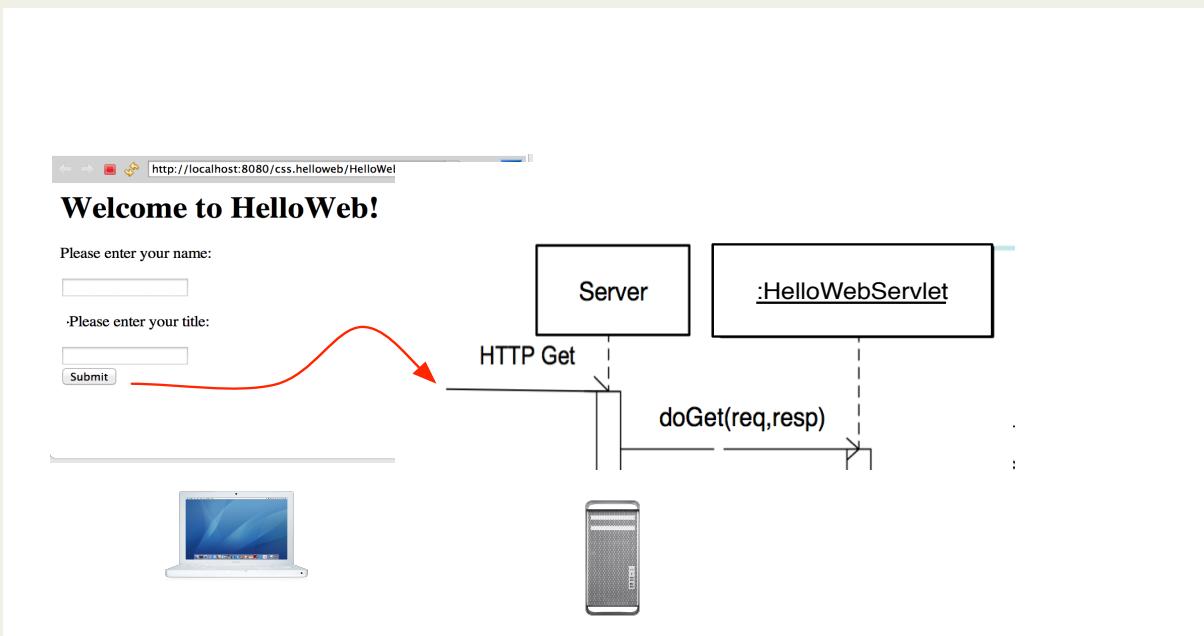
| Informática

## Exemplo: HelloWeb

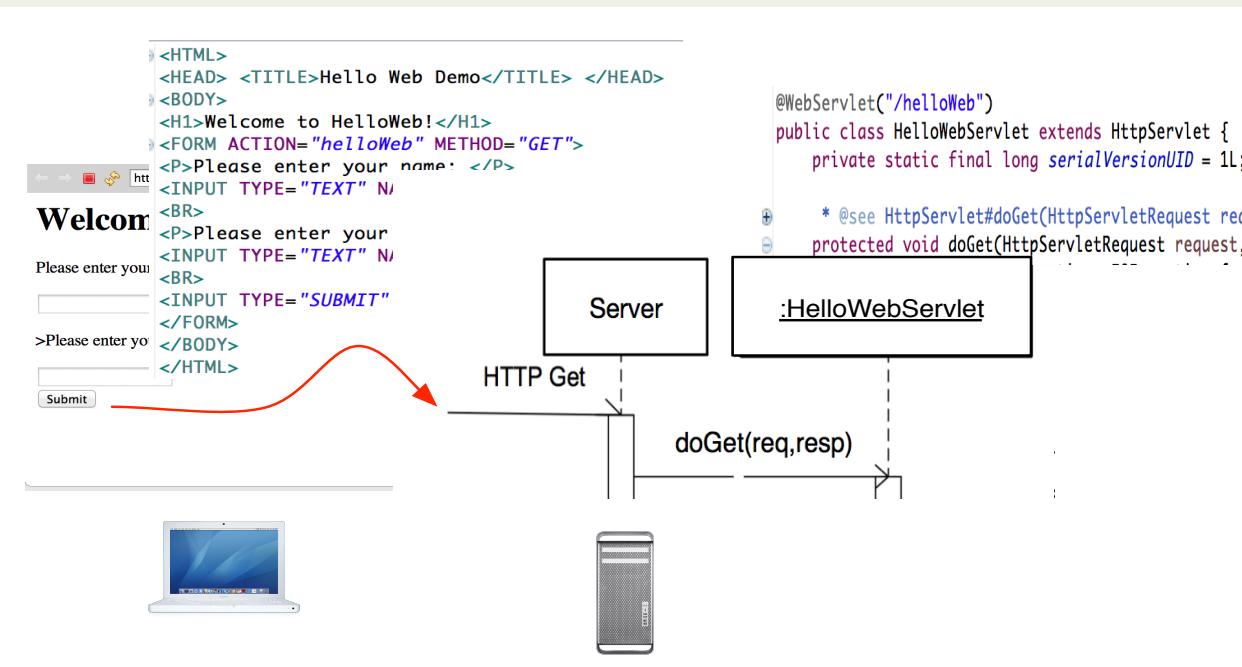


Ciências  
ULisboa

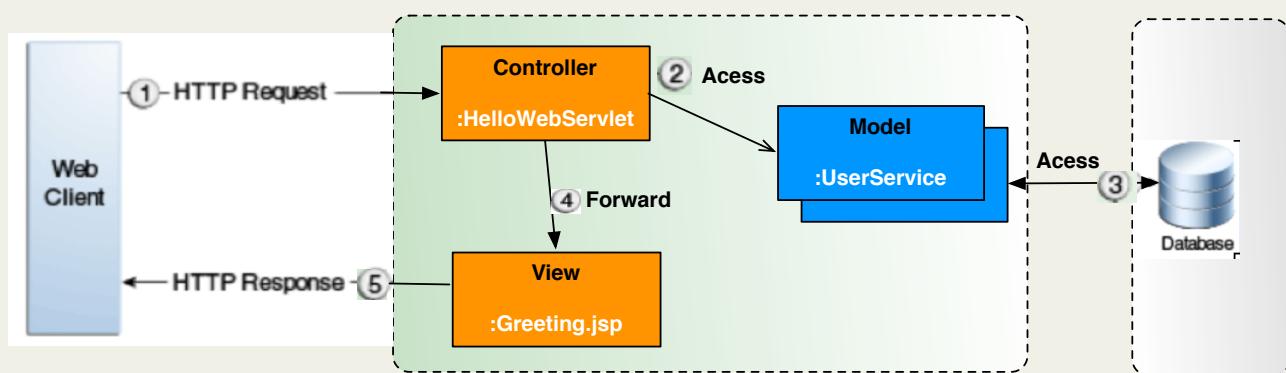
## Exemplo HelloWeb: Controller = Servlet



## Exemplo HelloWeb: Controller = Servlet

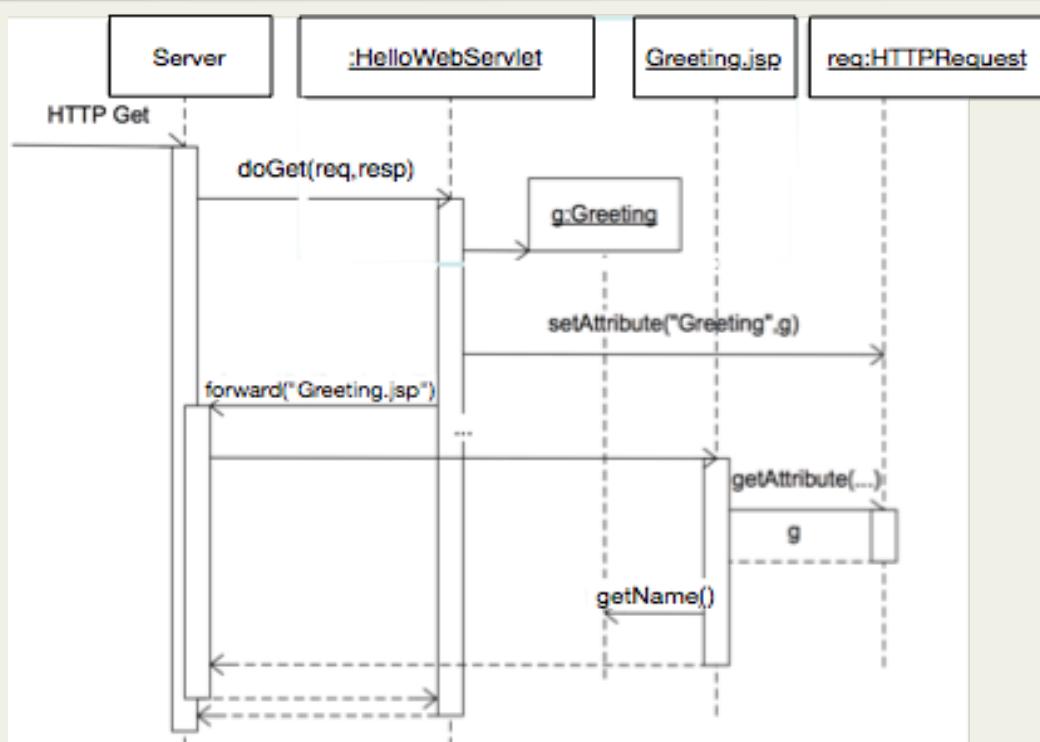


## Exemplo: Controller + View + Model



- Dados obtidos pelo *Controller* do *UserService* para serem mostrados na *View* são carregados num *JavaBean* do tipo **Greeting** e colocados num atributo do pedido

## Exemplo: Controller -> View



## Exemplo: *Controller* (versão simples; não usa UserService)

```
@WebServlet("/helloWeb")
public class HelloWebServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    /**
     * @see HttpServlet#doGet(HttpServletRequest request,
     *      HttpServletResponse response)
     */
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        Greeting g = new Greeting();
        request.setAttribute("Greeting", g);

        g.setName(request.getParameter("name"));
        g.setTitle(request.getParameter("title"));

        g.setPoints(100);

        request.getRequestDispatcher("Greeting.jsp").forward(request, response);
    }
}
```



## Exemplo: *View*

Greeting.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

<jsp:useBean id="Greeting" class="css.Greeting" scope="request"/>

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">

<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>HelloWeb Welcome message</title>
    </head>

    <body>
        Hello <c:out value ="${Greeting.title}"></c:out>  <c:out value ="${Greeting.name}"></c:out>. </p>
        <p> Welcome aboard of this fantastic adventure!
        You will start with <c:out value ="${Greeting.points}"></c:out> points. </p>
    </body>

</html>
```

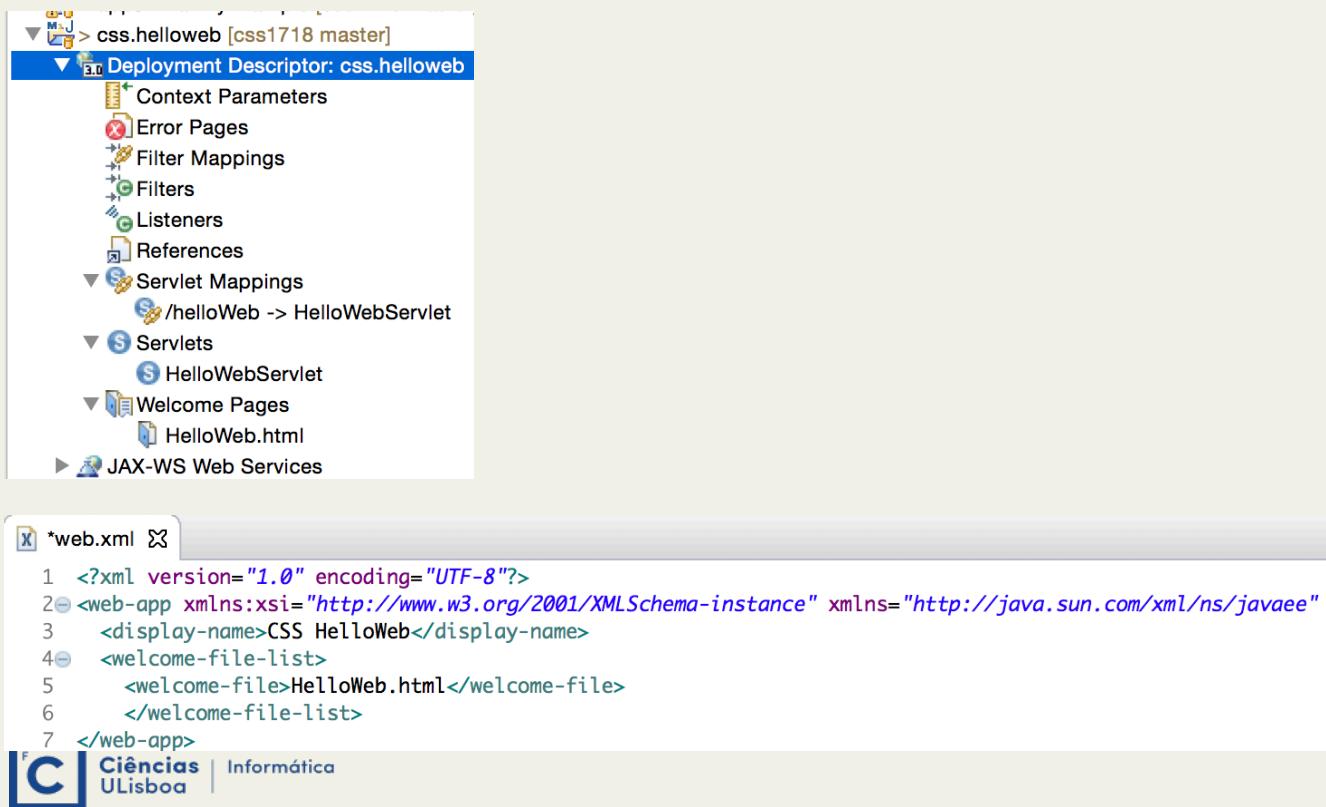


## Exemplo: View Model

```
public class Greeting implements Serializable{  
    private String name;  
  
    public String getName() {...}  
  
    public String getTitle() {...}  
  
    private String title;  
  
    public void setName(String name) {...}  
  
    public void setTitle(String title) {...}  
  
    private int points;  
  
    public void setPoints(int points) {...}  
  
    public int getPoints() {...}  
}
```

Greeting.java  
(Java Bean)

## Deployment Descriptor



The screenshot shows the Eclipse IDE interface with the following details:

- Project Explorer:** Shows a project named "css.helloweb [css1718 master]".
- Deployment Descriptor:** A tree view under "Deployment Descriptor: css.helloweb" containing:
  - Context Parameters
  - Error Pages
  - Filter Mappings
  - Filters
  - Listeners
  - References
  - Servlet Mappings
    - /helloWeb -> HelloWebServlet
  - Servlets
    - HelloWebServlet
  - Welcome Pages
    - HelloWeb.html
- Content Editor:** Displays the content of the web.xml file:

```
<?xml version="1.0" encoding="UTF-8"?>  
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns="http://java.sun.com/xml/ns/javaee"  
    <display-name>CSS HelloWeb</display-name>  
    <welcome-file-list>  
        <welcome-file>HelloWeb.html</welcome-file>  
    </welcome-file-list></web-app>
```
- Page Footer:** Features the ULisboa logo and the text "Ciências ULisboa | Informática".

## Projeto Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  <modelVersion>4.0.0</modelVersion>
  <groupId>pt.ulisboa.ciencia.di</groupId>
  <artifactId>css.helloweb</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>war</packaging>

<dependencies>
  <!-- First declare the APIs we depend on and need for compilation. All
      of them are provided by JBoss WildFly -->

  <!-- Import the Servlet API -->
  <dependency>
    <groupId>org.jboss.spec.javax.servlet</groupId>
    <artifactId>jboss-servlet-api_3.1_spec</artifactId>
    <scope>provided</scope>
  </dependency>

  <!-- Import the JSTL API -->
  <dependency>
    <groupId>org.jboss.spec.javax.servlet.jsp</groupId>
    <artifactId>jboss-jstl-api_1.2_spec</artifactId>
    <scope>provided</scope>
  </dependency>
</dependencies>
```

## Projeto Maven

```
<dependencyManagement>
  <dependencies>
    <!-- JBoss distributes a complete set of Java EE 7 APIs including a Bill
        of Materials (BOM). A BOM specifies the versions of a "stack" (or a collection)
        of artifacts. We use this here so that we always get the correct versions
        of artifacts. Here we use the jboss-javae-7.0-with-tools stack (you can
        read this as the JBoss stack of the Java EE 7 APIs, with some extras tools
        for your project, such as Arquillian for testing) -->
    <dependency>
      <groupId>org.wildfly.bom</groupId>
      <artifactId>jboss-javae-7.0-with-tools</artifactId>
      <version>${version.jboss.bom}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
```



## Correr a aplicação

