Universidad Zaragoza
1542

# Introducción a Servlets y JSP

## Grado en Ingeniería en Informática


## Curso 2024-2025

**Francisco Javier López Pellicer (fjlopez@unizar.es)**
**Carlos Tellería Orriols (telleria@unizar.es)**
**Fernando Tricas García (ftricas@unizar.es)**
**Raquel Trillo Lado (raqueltl@unizar.es)**
**Dpto. Informática e Ingeniería de Sistemas**

Universidad Zaragoza
1542

# Introduction to Web Technologies

## Servlets & JSP

Universidad Zaragoza
1542

# Guión

- Introduction
- Servlets
  - API classes and interfaces
  - Examples
- Java Server Pages (JSPs)
  - Examples
- Design patterns
- Applications packaging.
- Bibliography and References

Universidad Zaragoza

# Web servers history

- **CERN'S HTTPD server (1990, December, the origin of W3C)**
  - NCSA HTTPd (1993) University of Illinois at Urbana–Champaign by Robert McCool
  - APACHE Server (1995, Apache Software Foundation, Origin: NCSA HTTPd, rewritten later)
- **CGI (Common Gateway Interface) – Enables web servers to execute an external program (1993, NCSA)**
  - Standard input/output
  - Scripts, but also more complex programs
- **Processes' integration and multithreading**
  - Oracle Web Server (OWS) de Oracle
  - Servlet Specification ( Java software component that extends the capabilities of a server, May 1996, JavaOne Conference, Pavni Diwanji while she worked at Sun Microsystems)

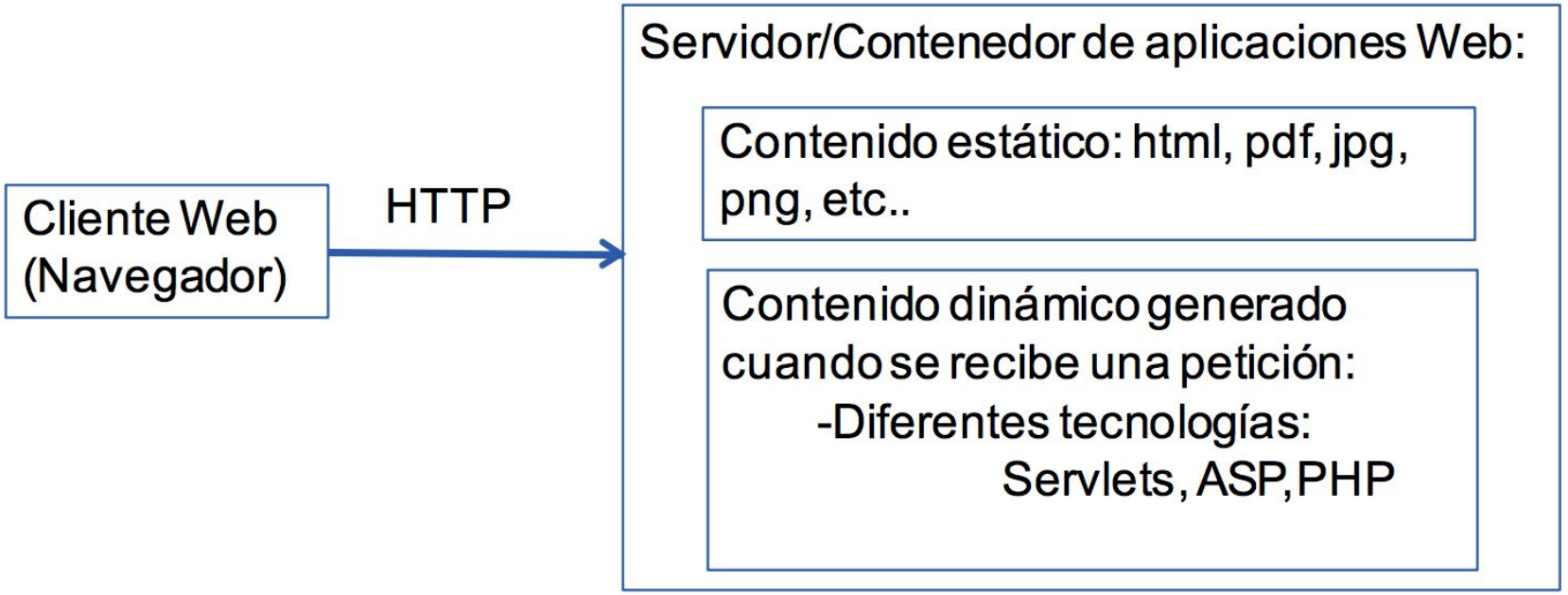
https://web.archive.org/web/20200815055946/https://community.oracle.com/blogs/driscoll/2005/12/10/servlet-history

Universidad Zaragoza

# Introduction

## Web Application

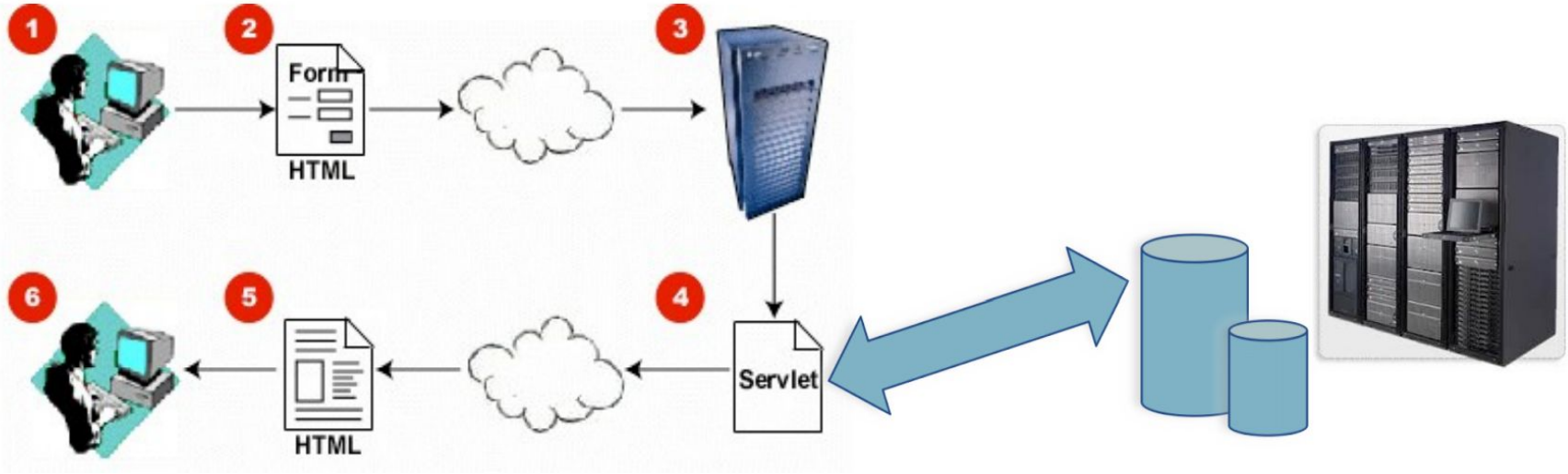
- A **web application** (or web app) is **application software** that is accessed using a web browser. :
  - Static content and Dynamic content (depending on users' requests)



# Servlet (Jakarta Servlet)

## Definition

- Java software component that extends the capabilities of a server:
  - Servlets could in principle communicate over any client–server protocol, but they are most often used with HTTP.
  - A Servlet is an object that receives a request, process some data, and generates a response based on that request. They can generate HTML (dynamic web page), but also other possibilities are available.



# Servlet

## Servlets and CGIs: differences

- **CGI Scripts (Common Gateway Interface):**
  - They will create a **new process** for each request (more resources and execution time needed).
  - Platform dependent
  - Any language

Universidad Zaragoza
1542

# CGI example

```
#!/usr/bin/perl
print "Content-Type: text/html\n\n";
print "<html><body>";
print "<h1>Hello, CGI World!</h1>";
print "</body></html>";
```

**Source: https://www.geeksforgeeks.org/common-gateway-interface-cgi/**

```
print("Content-Type: text/html")    # HTML is following
print()                             # blank line, end of headers
print("<TITLE>CGI script output</TITLE>")
print("<H1>This is my first CGI script</H1>")
print("Hello, world!")
```

https://docs.python.org/3/library/cgi.html

Universidad Zaragoza
1542

# CGI example

```
#!/usr/bin/perl

print "Content-type: text/html\n\n";

print "<font size=+1>Environment</font>\n";

foreach (sort keys %ENV) {

   print "<b>$_</b>: $ENV{$_}<br>\n";

}
```

Source: https://www.tutorialspoint.com/perl/perl_cgi.htm

Universidad Zaragoza

# CGI example

**URL:** http://www.test.com/cgi-bin/hello.cgi?key1=value1&key2=value2

This information is passed using **QUERY_STRING** header and will be accessible in your **CGI Program** through **QUERY_STRING** environment variable.

```
# Read in text

$ENV{'REQUEST_METHOD'} =~ tr/a-z/A-Z/;

if ($ENV{'REQUEST_METHOD'} eq "GET") {

   $buffer = $ENV{'QUERY_STRING'};

}

# Split information into name/value pairs
@pairs = split(/&/, $buffer);
```

Source: https://www.tutorialspoint.com/perl/perl_cgi.htm

Universidad Zaragoza
1542

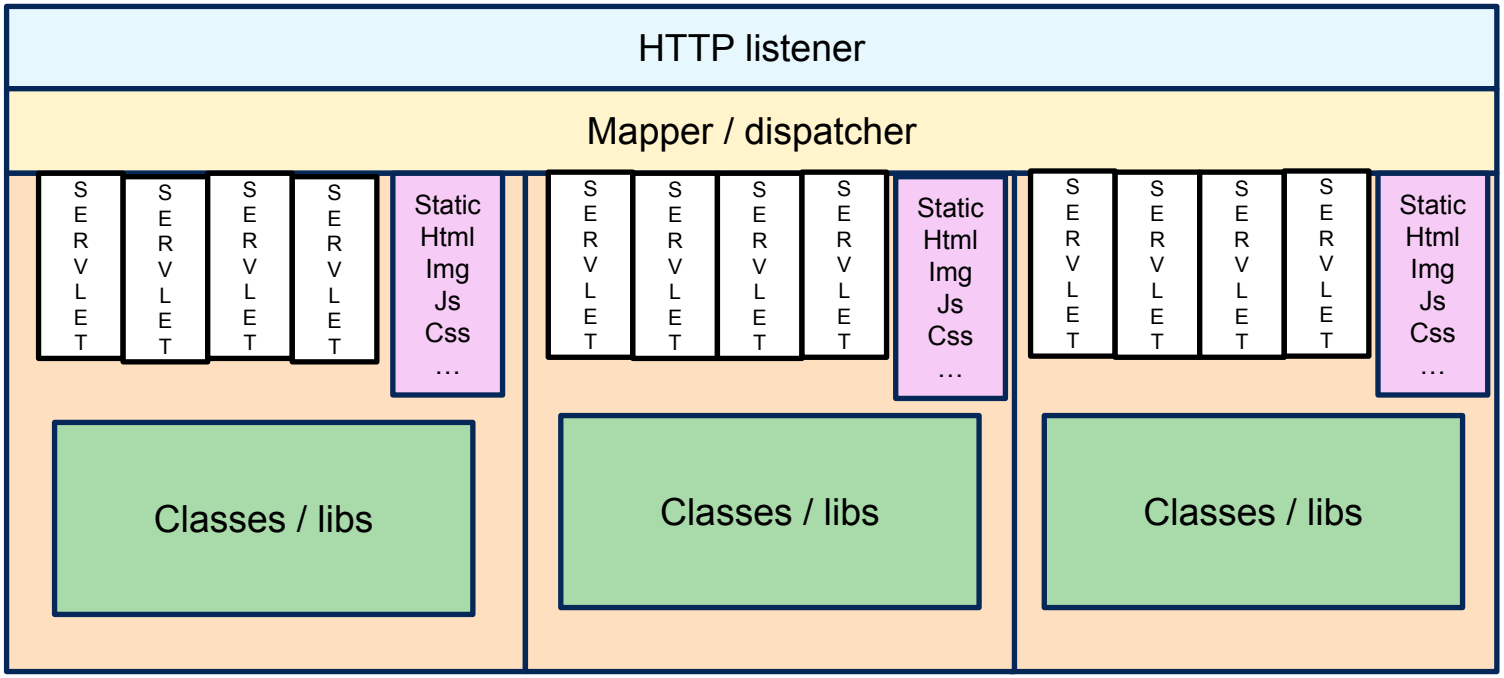
# Servlet

## Servlets and CGIs: differences

- **CGI Scripts (Common Gateway Interface):**
  - They will create a **new process** for each request (more resources and execution time needed).
  - Platform dependent
  - Any language
- **Servlets:**
  - they will create **new threads** in the server process.
  - Platform independent.
  - Java

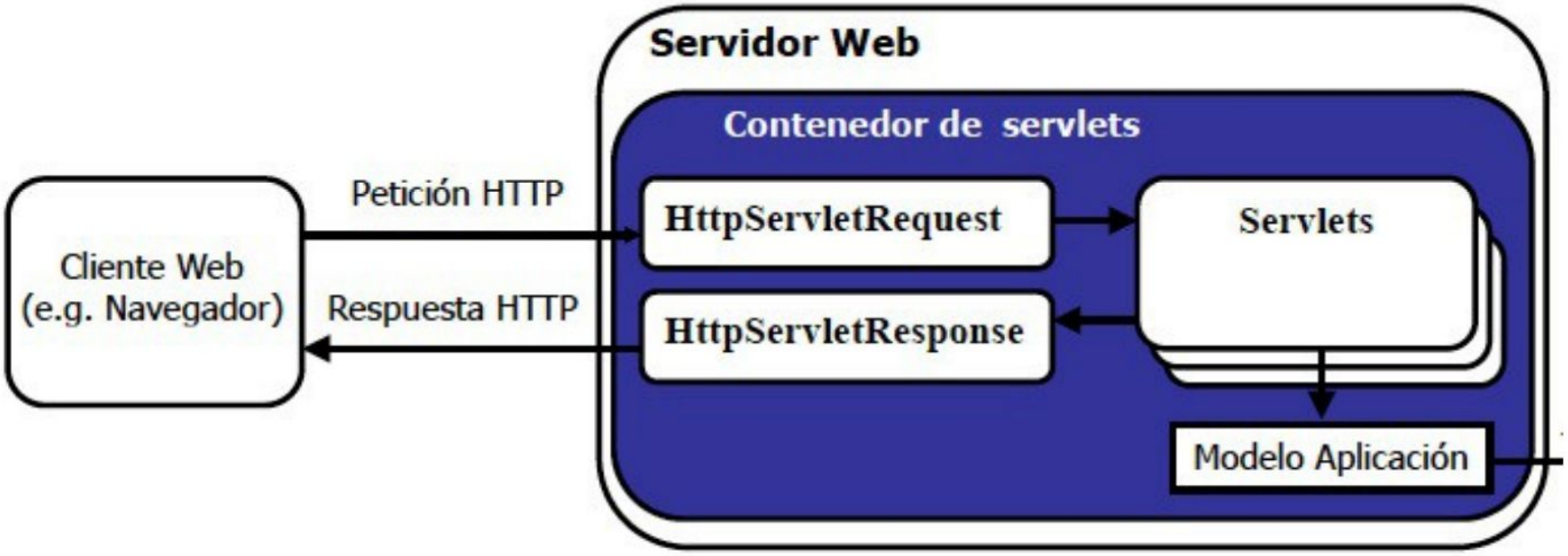
Universidad Zaragoza

# Application server basic structure



# Servlet

- **Each servlet can be associated to one or more URLs:**
  - The application container will associate URLs to servlets

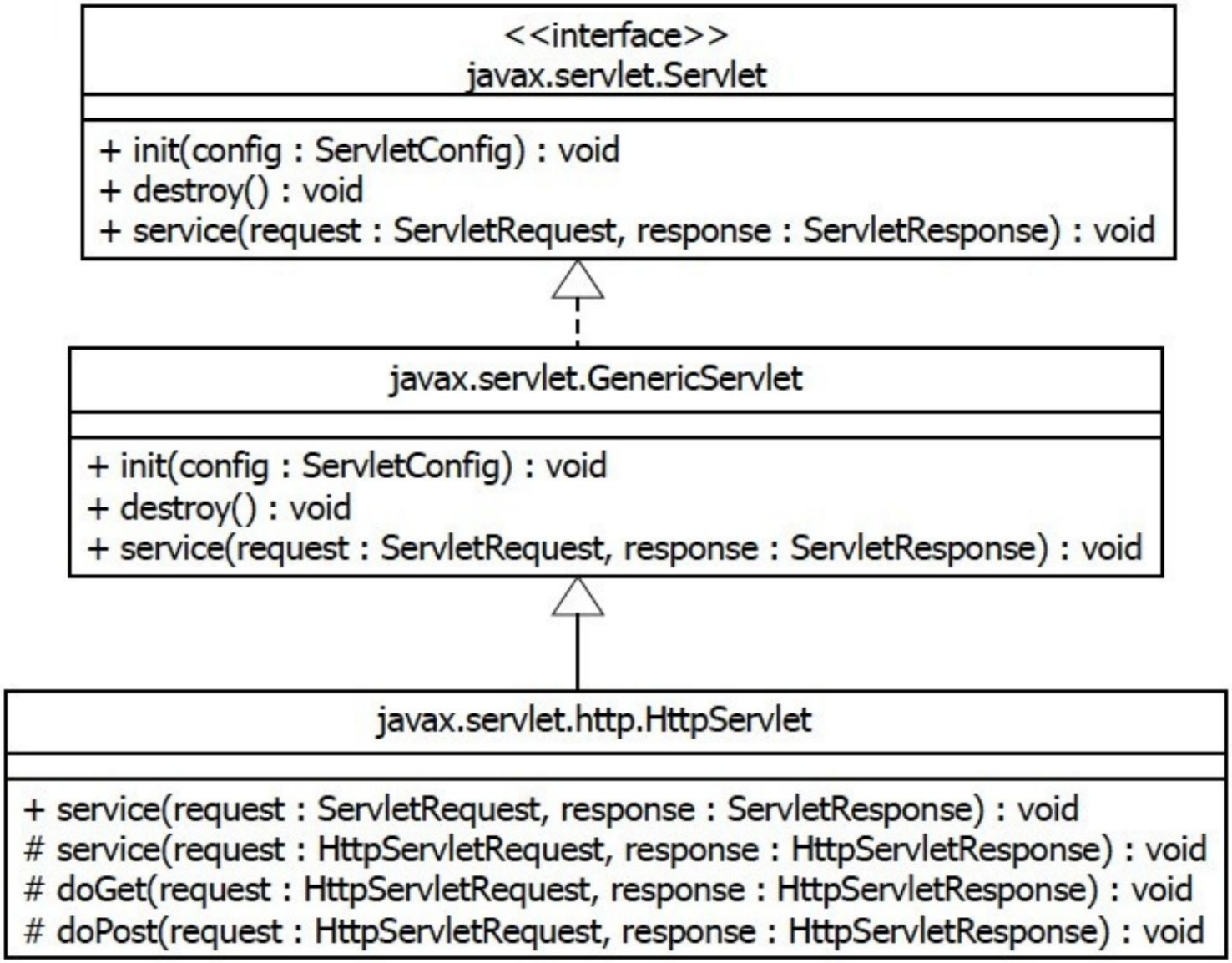



  - **Java packages**: javax.servlet y javax.servlet.http

https://docs.oracle.com/javaee/7/api/javax/servlet/package-summary.html

https://docs.oracle.com/javaee/7/api/javax/servlet/http/package-summary.html

# Paquete: javax.servlet (architecture)



Universidad Zaragoza
1542

# API classes and interfaces

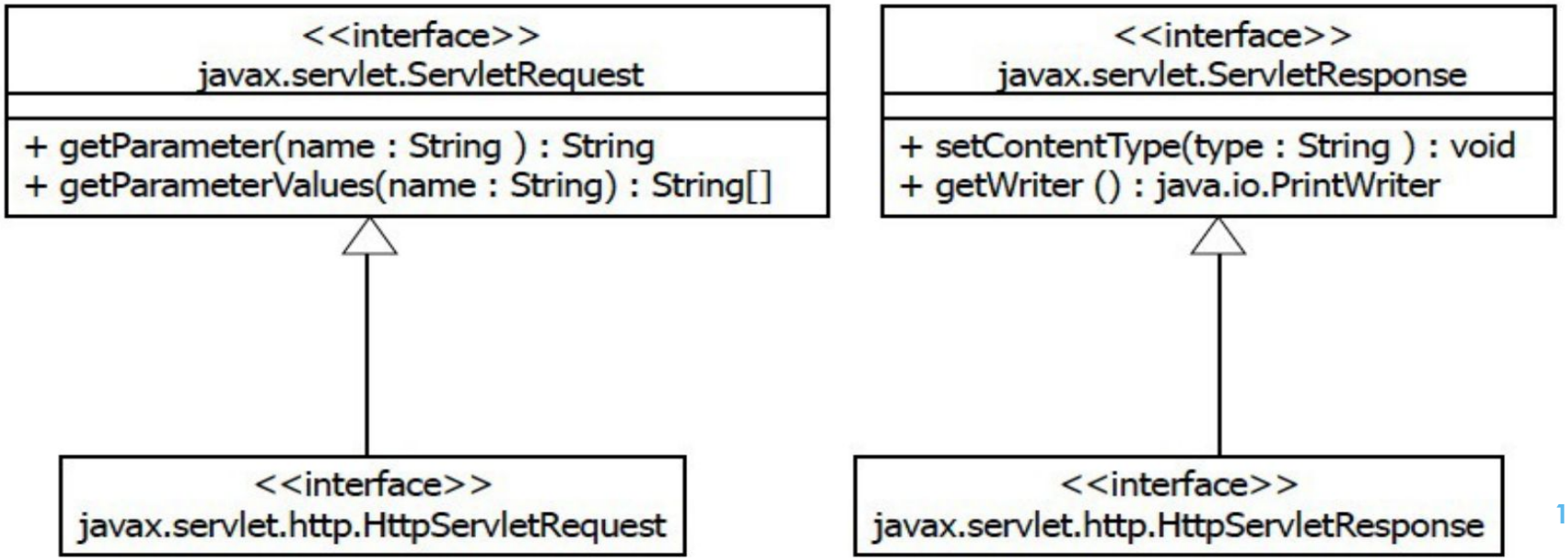
- **init method is executed when the web application server loads the servlet in memory.:**
  - At server starting time.
  - The first time it is requested
  - When some component of the web application is accessed.
- **destroy method is executed when the web application server decides to eliminate it from memory:**
  - When the server is shutdown.
  - When the web application is shutdown.
  - When some time without using it has pased

**API: Application Programming Interface**

Universidad Zaragoza

# API classes and interfaces

- **service method is executed when the service provided by the servlet is requested:**
  - When some request associated to the URLs related to the servlet:
    - Parameters are encoded in they **ServletRequest** object.
    - Response that will be sent to the client is encoded in the **ServletResponse** object.



Universidad Zaragoza
1542

# API classes and interfaces

- Public method **service** can be implemented calling the protected operation service, which is a template operation which will call the method ***doGet*** or ***doPost*** depending on the type of request **(GET or POST)**
  - In general, we will create a new class extending the class javax.servlet.http.HttpServlet and redefining the methods doGet and/or doPost
  - In general, both methods are not implemented, but only one of them is developed and in the other a call is made to the method that has been developed (if we want that behavior)
    - **¿Which one, doGet or doPost, will we implement?**

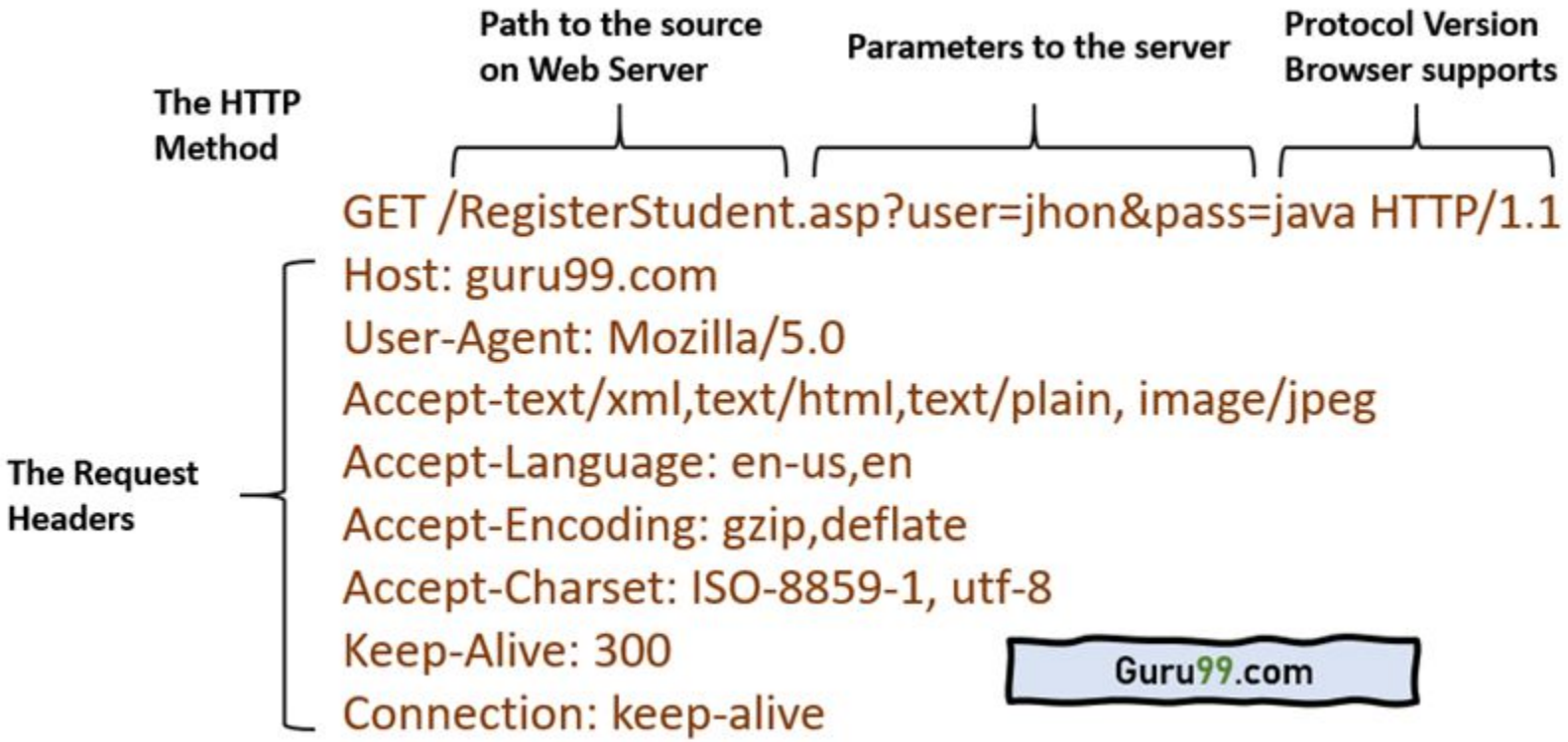
Universidad Zaragoza
1542

# API classes and interfaces. doGet

- **GET**
  - Idempotent petitions (same result each time). No lateral effects (reading).
    - Example: http://www.amazon.com/search?**item=earphone**.
  - Requests can be shared, bookmarked, …
  - Request data visible in URL. Size limitation (255)

# API classes and interfaces. doGet

- GET



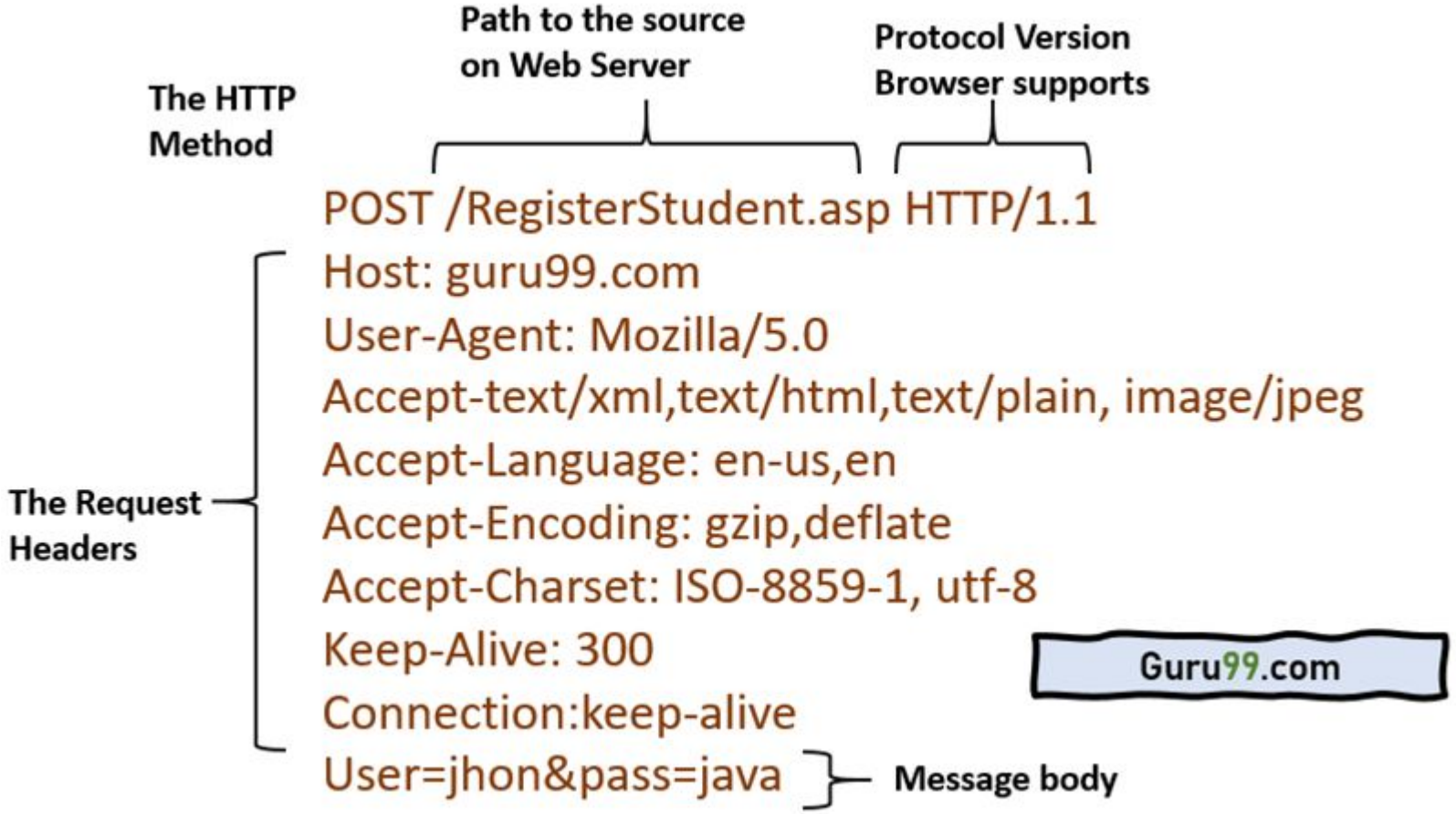
Universidad Zaragoza
1542

# API classes and interfaces. doPost

- **POST**
  - With side effects (writing/updating)..
  - URL depends on data which is not there.
    - Parameters are not visible, they won't be in your browsing history either.
  - Request data hidden in the HTTP request header

**Images: https://www.guru99.com/difference-get-post-http.html**

# API classes and interfaces. doPost




application/x-www-form-urlencoded

Universidad Zaragoza

# Servlets API classes and interfaces.

- Public methods in **HttpServletRequest**:
  - **getParameter** Returns the value of a request parameter as a String, or null if the parameter does not exist. (**monovalued** attribute).
  - **getParameterValues** Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist. (If the parameter has a single value, the array has a length of 1.)
- Public method in **HttpServletResponse:**
  - **setContentType** Sets the content type of the response being sent to the client, if the response has not been committed yet.The response's character encoding is only set from the given content type if this method is called before **getWriter** is called.

Universidad Zaragoza

# Servlet: Examples

- We are going to create a servlet for processing some data provided by some client in the following form. It will reply if everything went fine, or the errors, if any:
  - - Login
  - - Clave
  - - Clave (repetir para verificar)
  - - Nombre
  - - Apellidos
  - - E-mail:
  - - Idioma
  - - País
  - Incluirá botón de restablecer

Universidad Zaragoza
1542

# Servlet: Ejemplos

- **First step:** Create a HTML document that can generate the previous form
- Then, associate the following URL to the action:

  http://localhost:8080/MiPrimeraAplicacion/InsertarUsuario

Universidad Zaragoza
1542

# Servlet: Ejemplos

- **Second step:** Create the servlet that will manage the requests from http://localhost:8080/ MiPrimeraAplicacion/InsertarUsuario

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

public class InsertUserServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
                       throws IOException, ServletException {
    String login = request.getParameter(”login");
    String password = request.getParameter(“pw");
    String repassword = request.getParameter(”repw");
    String name = request.getParameter(“name");
    String surname = request.getParameter(“surname");
    String email = request.getParameter(“email");

...
```

Universidad Zaragoza
1542

# Servlet: Examples

| UserVO |
| --- |
| - login: String<br>- name: String<br>- surname: String<br>- encryptedPassword: String<br>- email: String<br>- language: String<br>- country: String |
| + UserVO (login, name, surname, encryptedPass, email, lang, country: String): UserVO<br>/*métodos get para cada uno de los atributos privados de la clase*/<br>/*métodos set para cada uno de los atributos privados de la clase excepto el atributo login*/ |

| ModelFacade |
| --- |
| |
| + insertUser (user: UserVO): void<br>+ findUser (login: String): UserVO<br>/*El método insertUser puede lanzar la excepción DuplicatedUser*/<br>/*El método findUser devuelve null si no encuentra un usuario con ese login en el sistema*/<br>/*Todos los métodos pueden lanzar la excepción InternalErrorException*/ |

Universidad Zaragoza

# Servlet: Example

- **Third step:** Data validation and insertion in the database

```
…
   public void doPost(HttpServletRequest request, HttpServletResponse response)
         throws IOException, ServletException {
            …...
   //p.e.: todos los campos obligatorios  deben ser distintos de null,  la clave
   // debe ser igual a la clave tecleada por segunda vez,  la dirección de
   //correo debe ser una dirección válida
   // Si los parámetros son correctos se crea el objeto usuario  y se realiza
   // una llamada a una clase del modelo que realiza la inserción en la bd

   UserVO user = new UserVO (login, password, name, surname, email, language,
         country)
   try {
         ModelFacade model = new ModelFacade();
         model.insertUser(user);
   } catch (Exception e) {
         response.sendRedirect("paginaError.html")
   } …
}
```

Universidad Zaragoza

# Servlet: Example

- **Fourth step:** If there is some unexpected error, we will redirect the user to some error page.

```
…
    public model = new ModelFacade();
            model.insertUser(user);
        } catch Exception e {
            response.sendRedirect("paginaError.html")
        } …
}
```

Universidad Zaragoza

# Servlet: Examples

- **Fifth step:** Reply page if everything went OK

```
…

response.setContentType(“text/html; charset=ISO-8859-1”);

PrinterWriter out = response.getWriter();

out.println(“<html><head><title>Inserción del
usuario</title></head>”);

out.println(“<body><p> El usuario con login “ + login
       +   “se ha registrado correctamente </p>”);

out.println(“</body></html>”);
```

Universidad Zaragoza
1542

# Servlets API classes and interfaces.

- The **sendRedirect** method can be used to redirect response to another resource, it may be servlet, jsp or html file.
- Sometimes it is useful to go to another element in the server instead of sending a reply.
  - In these cases the **forward** method is used:

```
RequestDispatcher dispatcher = request.getRequestDispatcher(url);

dispatcher.forward(request, response)
```

  - **Control is passed to another element within the server.** The web container handles all processing internally and the client or browser is not involved. The request and response objects are passed, so our old request object is present on the new resource which is going to process our request. It is transparent to the user.

Universidad Zaragoza
1542

# My first Servlet

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HolaMundo extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws IOException, ServletException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Clasico Hola mundo!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hola mundo de servlets!</h1>");
        out.println("</body>");
        out.println("</html>");
    }
}
```

Universidad Zaragoza

# Mapping Configuration (web.xml)

```
<!-- SERVLET DE HOLA MUNDO -->
<servlet>
    <servlet-name>HolaMundo</servlet-name>
    <servlet-class>HolaMundo</servlet-class>
</servlet>

<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
    <servlet-name>HolaMundo</servlet-name>
    <url-pattern>/HolaMundo</url-pattern>
</servlet-mapping>
```

Universidad Zaragoza
1542

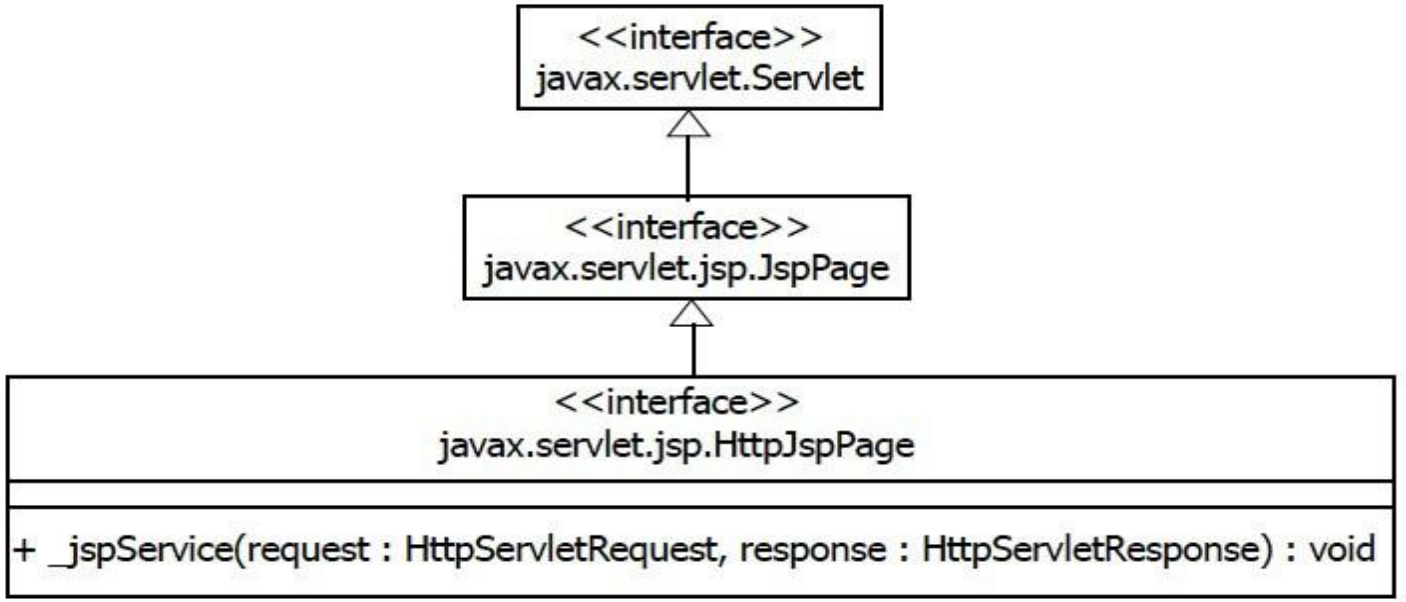
# Paradigm evolution

- **From Servlet to…**
- **If most of the content is static HTML and only some part is dynamic…**
- **Instead of code generating HTML -> HTML with embedded code**
  - JSP (Java)
  - ASP (Microsoft)
  - PHP (Hypertext preprocessor – Open source)
  - OSP (Oracle)
- **Other models: OPA, React, Angular, Web Services, REST, GraphQL, microservices based architectures…**

# Jakarta Server Pages (JSPs)

## Introduction

- **A JSP page is a special type of Servlet, composed of:**
  - static data, which can be expressed in any text-based format (such as HTML),
  - JSP elements (scriptlets written in Java), which construct dynamic content.



Universidad Zaragoza
1542

# Jakarta Server Pages (JSPs)

## Introduction

- **When the user access to a JSP webpage:**
  - It it is the first time, the web application server translates it to a servlet, compiles it and it is loaded in memory.
  - If it is not the first time, the request is passed to the associated servlet
  - If the JSP page has been modified since the compilation, a new servlet will be generated.

**Jakarta Server Pages, formerly Java Server Pages**

# Jakarta Server Pages (JSPs)

## Example

- JSP page creation to process data provided by the client in the following search form. It will show the available data for the client, if it exists (buscar.html):


- **Exercise:** Create a HTML web page for this form (action http://localhost:8080/MiPrimeraAplicacion/MostrarUsuario.jsp)

Universidad Zaragoza

# Jakarta Server Pages (JSPs): Example

```
<%@ page import="miaplicacion.userVO">
<html>
<%-- Comentarios en las páginas JSP --%>
<head><title>Show a user </title></head>
<body>
<%
     String login = request.getParameter("login");
     try { if ((login != null) && (!login.trim().equals(""))){
           ModelFacade model = new ModelFacade();
           UserVO user;
           user = model.findUser(login);}else{
           response.sendRedirect("buscar.html");}
     } catch Exception e{ response.sendRedirect("paginaError.html")}
      if (user == null) response.sendRedirect("paginaError.html")
%>
     <h1>Información del usuario buscado (<%=login%>) </h1>
      <p>Nombre: <%= user.getLogin() %> </p>
      <p>Apellidos: <%= user.getSurname() %> </p>
…
</body> </html>
```

Universidad Zaragoza
1542

# Jakarta Server Pages (JSPs): Example

Response that the client will get:

```
<html>
<head><title>Show a user </title></head>
<body>
   <h1>Información del usuario buscado (Pepe) </h1>
    <p>Nombre: Pepe> </p>
    <p>Apellidos: Lopez </p>
     …
</body> </html>
```

# Jakarta Server Pages (JSPs)

## Example

- JSP pages use <% ….%> to include Java code (scriptlets)
- JSP pages use the directive <%@ … %> to import classes
- JSP pages use <%--….--%> to generate comments (no HTML comments)
- JSP pages have the following implicitly defined objects::
  - **request**: javax.servlet.http.HttpServletRequest
  - **response**: javax.servlet.http.HttpServletResponse
  - **session**: javax.servlet.http.HttpSession
  - **out**: javax.servlet.jsp.JspWriter
- JSP pages use <%= expression %> to include Java expressions which can return String objects or objects that can be converted to String (by calling toString() method)

Universidad Zaragoza
1542

# JSP standard tags (JSTL)

JSTL provides a way to embed logic within a JSP page without using embedded Java code directly. Standardized tag set leads to more maintainable code and enables separation of concerns between the development of the application code and user interface.

```
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<!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>Lista Demo</title>
</head>
<body>
     <table>
     <c:forEach var="demo" items="${listaDemo}">
          <tr><td>
               <c:out value="${demo.id}" />
          </td><td>
               <c:out value="${demo.name}" />
          </td></tr>
     </c:forEach>
     </table>
</body>
</html>
```

Universidad Zaragoza

# Dynamic web or web of transition (Web 1.5)

- **Session:** Time period during which a particular user interacts with a website.
  - The state or session must be implemented by each application (HTTP is a stateless protocol).
  - Different alternatives. You can include the session *id* in:
    - *cookies*
    - Each URL.
    - Hidden fields in the forms
    - Heuristics: based on parameters of the HTTP request (IP, time,, etc.)

Universidad Zaragoza

# Servlets and JSP together

- JSP for the **V**iew of the application:
  - Form visualization and error messages after validation of parameters in the server
  - Result visualization for some operation (e.g.: search operation)
- Servlets for form processing and application layer:
  - If the parameters are valid it will perform the operation and the control will be passed to the visualization JSP page.
  - It they are not correct, it will pass the control to the JSP that allows the user to correct errors

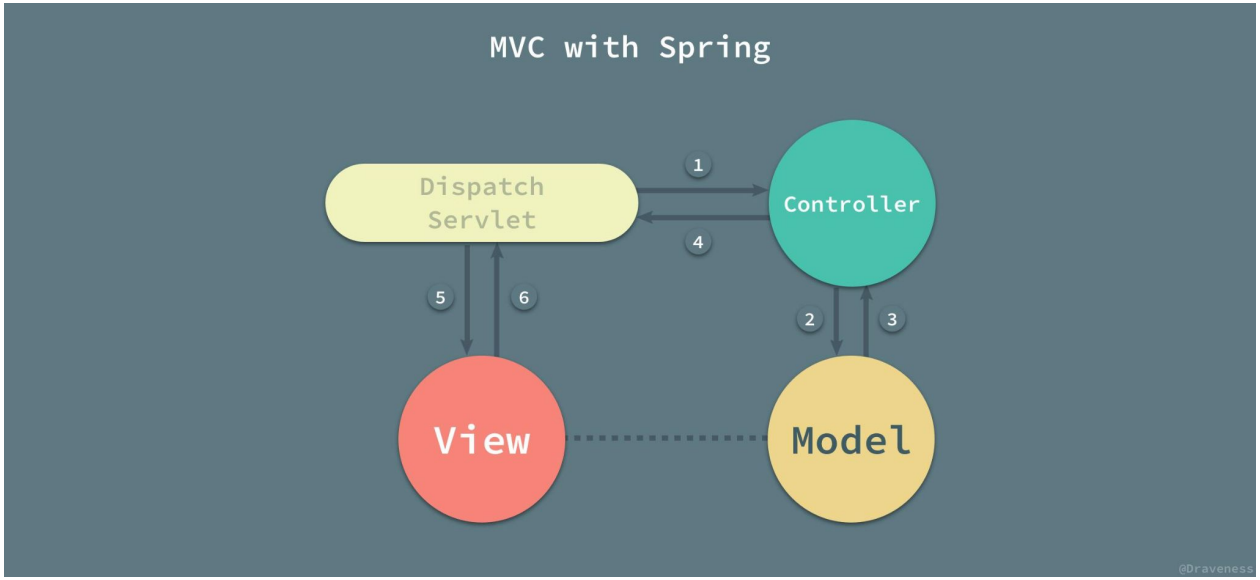
Universidad Zaragoza

# Desing patterns for web (and mobile) architectures.

- **MVC - Model View Controller**
  - Struts, Spring MVC, PHP …
- **MVP - Model View Presenter**
  - JSF, ASP … Android, iOS
- **MVVM - Model View ViewModel**
  - Android (p.ej., Swift), Angular, React

Universidad Zaragoza
1542

# MVC

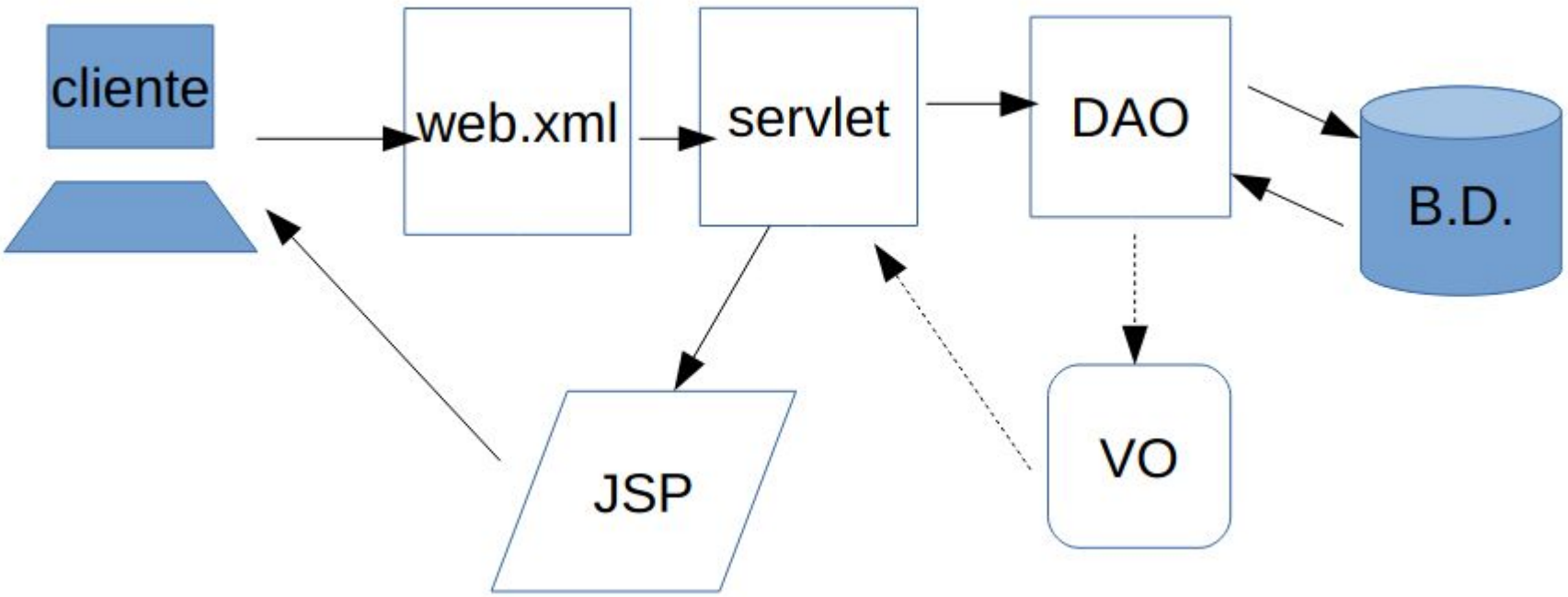



- **VIEW**
  - Visualization layer.
- **MODEL**
  - Model layer (data and entities). DAO/VO + database
- **CONTROLLER**
  - It manages traffic between the client (browser) and the model; it also manages traffic between model and view.

**DAO: Data Access Object VO: *V*alue Object.**

Universidad Zaragoza
1542

# MVC con J2EE



Universidad Zaragoza
1542

# Web application packaging

- How to make available a web application?
  - Installation/deploying in the web server
  - When using Java, web applications servers/containers use war files.
  - Jar -cvf myFirstApplication.war directory
- File .war structure:
  - Folder WEB-INF/classes: contains web application .class files grouped according to their own packages structure.
  - Folder WEB-INF/lib: contains the .jar files with the libraries needed in the application (e.g.: jdbc  driver for DB connections)
  - Folder root and subfolders will contain view files for the application: HTML pages, CSS, images, JSP pages, …
  - Browsers (clients) cannot access WEB-INF content (only servlets and JSP can view the content)

Universidad Zaragoza

# Web application packaging

```
<?xml version="1.0" encoding="UTF-8"?>
  <web-app version="2.5" 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-app_2_5.xsd">

<display-name>My first Web Application</display-name>

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

<servlet>
   <servlet-name>InsertUser</servlet-name>
   <servlet-class>miprimeraaplicacion.ejemplo1.InsertUserServlet</servl
   et-class>
</servlet>
```

Universidad Zaragoza
1542

# Web application packaging

```
<!-- ============ Servlets Mapping======= -->
<servlet-mapping>
  <servlet-name>InsertUser</servlet-name>
  <url-pattern>/InsertarUsuario</url-pattern>
</servlet-mapping>

<!-- ============ Welcome page ========= -->
  <welcome-file-list>
     <welcome-file>Index.html</welcome-file>
  </welcome-file-list>

</web-app>
```

Universidad Zaragoza
1542

# Web application packaging

In the example, only a few common tags are shown

- **Display-name**: Web Application name
- **Servlet**: Declares each servlet that is part of the application (**servlet-class**) and assigns a name to it (**servlet- name**)
- **Servlet-mapping**: Defines URLs associated to each servlet (**url-pattern**) previously defined (**servlet-name**)
- **Welcome-file-list**: Shows the webpage the server will return when the user accesses the application (**welcome-file**)

Universidad Zaragoza

# Servlet declaration with annotations

Optionally, instead of declaring the Servlets in the web.xml file, we can do so directly by annotating the corresponding classes with the @WebServlet element.

```
@WebServlet(value = "/Simple", initParams = {
   @WebInitParam(name = "foo", value = "Hello "),
   @WebInitParam(name = "bar", value = " World!")
})
public class Simple extends HttpServlet {

   private static final long serialVersionUID = 1L;

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

...

# Bibliography and references


- Mark Andrews "***Story of a Servlet: An instantTutorial***": http://www.oracle.com/technetwork/java/tutorial-138750.html [Fecha de último acceso: 18 de septiembre de 2017]

Universidad Zaragoza
1542

# Sistemas de información

## Grado en Ingeniería en Informática

## Curso 2024-2025


Francisco Javier López Pellicer (fjlopez@unizar.es)

Fernando Tricas García (ftricas@unizar.es)

Raquel Trillo Lado (raqueltl@unizar.es)

Carlos Tellería Orriols (telleria@unizar.es)

Dpto. Informática e Ingeniería de Sistemas