DESARROLLO DE APLICACIONES WEB

Tema 5.- Java Entreprise Edition

Servlet: Controlador

Contenido

- Tecnologías Java
- Introducción JEE
- Servlets
- MVC: Controlador

Introducción

- Tecnologías Java

Java SE: Java Standard Edition

Conjunto de APIs y utilidades que permiten desarrollar aplicaciones Java de escritorio.

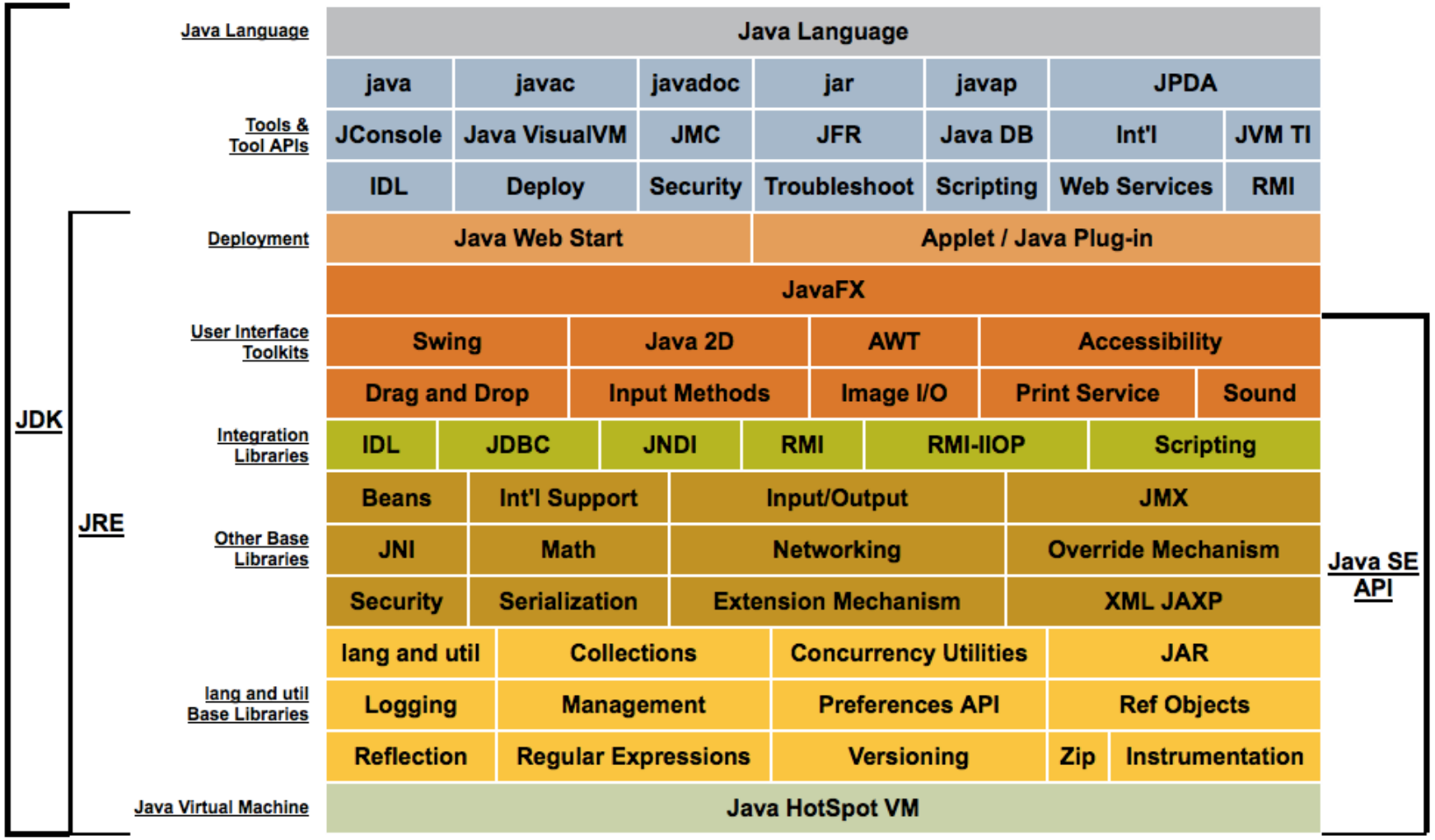
Java EE: Java Entreprise Edition

Conjunto de APIs que permiten desarrollar aplicaciones Web y aplicaciones empresariales. Toma como base el Java SE.

Java ME: Java Micro Edition

Conjunto de Apis que permiten desarrollar aplicaciones para dispositivos móviles.

Introducción: Java SE



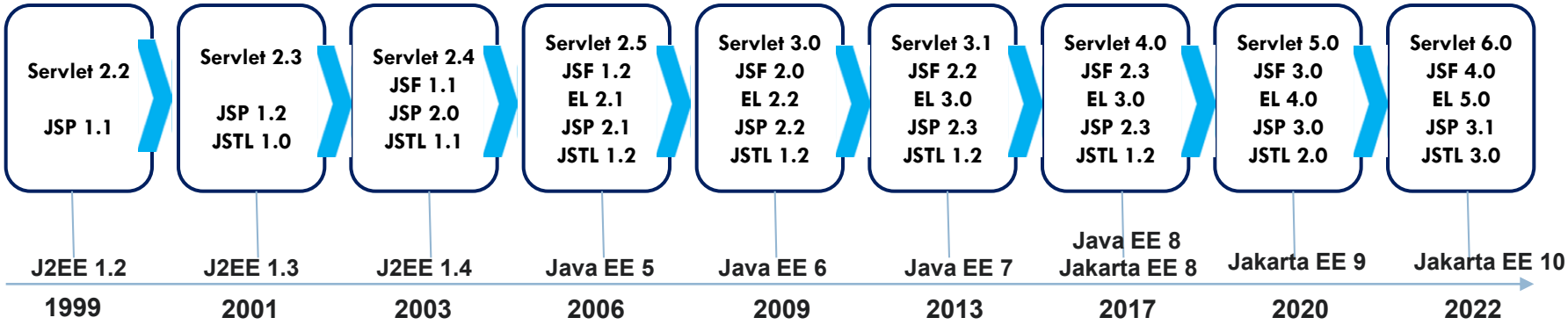
Introducción: Java SE

- Versiones

Version	Release date	End of Free Public Updates[1][5][6][7]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018 December 2026 for Azul[8]
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2030 for Oracle (non-commercial) December 2030 for Azul May 2026 for IBM Semeru[9] At least May 2026 for Eclipse Adoptium At least May 2026 for Amazon Corretto	December 2030[10]
Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	September 2026 for Azul October 2024 for IBM Semeru[9] At least October 2024 for Eclipse Adoptium At least September 2027 for Amazon Corretto At least October 2024 for Microsoft[11][12]	September 2026 September 2026 for Azul[8]
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK March 2023 for Azul[8]	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	September 2029 for Azul At least September 2027 for Microsoft At least TBA for Eclipse Adoptium	September 2029 or later September 2029 for Azul
Java SE 18	March 2022	September 2022 for OpenJDK	N/A
Java SE 19	September 2022	March 2023 for OpenJDK	N/A
Java SE 20	March 2023	September 2023 for OpenJDK	N/A
Java SE 21 (LTS)	September 2023	TBA	September 2031[10]

Legend: Old version | Older version, still maintained | **Latest version** | Future release

Introducción: Java/Jakarta EE

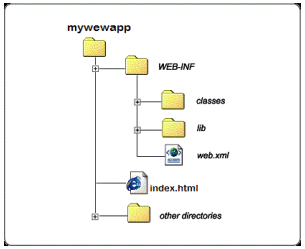


J2EE 1.2 (December 12, 1999)
J2EE 1.3 (September 24, 2001)
J2EE 1.4 (November 11, 2003)
Java EE 5 (May 11, 2006)
Java EE 6 (December 10, 2009)
Java EE 7 (May 28, 2013
Java EE 8 (August 31, 2017)
Jakarta EE 8 (September 10, 2019) - fully compatible with Java EE 8
Jakarta EE 9 (November 22 2020) - javax.* to jakarta.* namespace change.
Jakarta EE 9.1 (May 25 2021) - JDK 11 support
Jakarta EE 10 (September, 2022) – JDK 11, 17 o 21

Introducción: Jakarta EE

Estructura de una Aplicación Java Web

- Estructura de directorios



/ ---- Directorio raíz (HTML, CSS, JS, JSP, etc)

WEB-INF/ ---- No accesible desde web (Vistas JSP de MVC)

classes/ ---------- (clases: servlets, etc)

lib/ ---------- (otras librerías .jar)

web.xml ---------- (Opcional. Descriptor web)

- Contexto (web context): Ruta donde se aloja la app en el servidor
- Anotaciones versus descriptor web.xml

- Fichero WAR

  Comprimen la aplicación Web en un único fichero, como los jar una aplicación Java o rar un conjunto de ficheros genéricos

Descriptor Web o de Aplicación: web.xml

- Documento XML:

<web-app> … </web-app>: Etiqueta Principal

<display-name> … </display-name> : Nombre que mostrara el contenedor

<description> … </description> : Descripción de la aplicación

<servlet> … </servlet> : Existe una para cada servlet de la aplicación, con:

<servlet-name> … </servlet-name> : Nombre del servlet

<servlet-class> … </servlet-class> : Clase que ejecutará el servlet

<servlet-mapping> … </servlet-mapping> : Una para cada servlet con:

<servlet-name> … </servlet-name> : Nombre del servlet

<url-pattern> … </url-pattern> : Ruta URL para invocar el servlet

Contenido

- Tecnologías Java
- Introducción JEE
- **Servlets**
- MVC: Controlador

Web Servlets

Un servlet es un componente Java alojado en un contenedor de servlets (o servidor de aplicaciones).

Los clientes web interactúan con un servlet usando un patrón de solicitud / respuesta.

El contenedor es responsable del ciclo de vida del servlet, de recibir las peticiones y enviar las respuestas, además de realizar cualquier otra acción requerida.

Los servlets están implementados en el paquete jakarta.servlet.

En concreto, los Web Servlet se implementan en jakarta.servlet.http

Ver javadoc API

Este cuadro, en las transparencias, proporciona un enlace al JavaDoc API del paquete, la interface o la clase que se presenta, para que el alumno pueda profundizar más en cada una de ellas.

En este caso es un enlace al paquete jakarta.servlet y jakarta.servlet.http

Ver javadoc API

Introducción – Ejemplo Servlet

Servlets:

Clase Java que atiende peticiones HTTP y genera una respuesta HTML

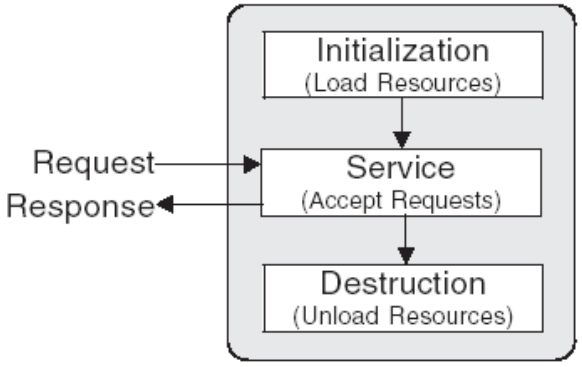
```
//  servletHolaMundo.java
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
@WebServlet(urlPatterns = {"/holaMundo"})
public class servletHolaMundo extends HttpServlet {
    @Override
    public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException  {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Hola Mundo!</title>");
      out.println(<meta charset="UTF-8" /></head>");
      out.println("<body>");
      out.println("<h1>Hola Mundo!</h1>");
      out.println("</body>");
      out.println("</html>");
  }
}
```

Servlets

Ver javadoc API

- **Ciclo de vida:** Un servlet pasa por tres etapas (interface *jakarta.servlet.Servlet*)

  Inicialization: Creación e inicialización de los recursos del servlet. Método **init()**
  Service: Atiende las peticiones de los clientes y envía las respuestas.
  La interface Servlet anterior define un método **service()** con dos parámetros:
  Request: Petición del cliente al servidor
  Response: Respuesta del servidor para el cliente.
  Destruction: Borra el objeto servlet cuando no es necesario.
  La interface Servlet define un método **destroy()** con ese objetivo.

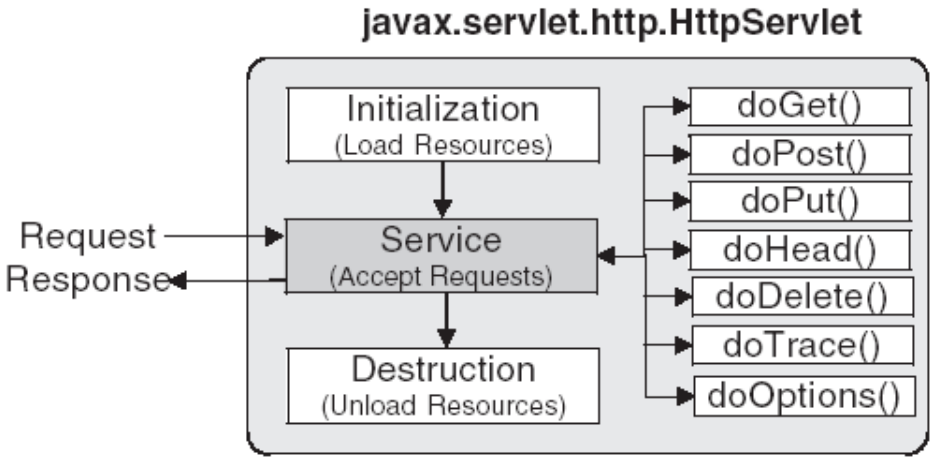


HTTP Servlets

Ver javadoc API

- La clase HttpServlet implementa los HTTP Servlets (*jakarta.servlet.http.HttpServlet*)
- Esta clase implementa la interface Servlets, añadiéndole funcionalidad propia del protocolo HTTP al método service().
- El método service() discrimina el tipo de petición del cliente y para ello define 7 métodos abstractos que el desarrollador debe implementar. Los tipos de petición y el método asociado para atenderla son:

Petición	Método
GET	doGet()
POST	doPost()
PUT	doPut()
HEAD	doHead()
OPTIONS	doOptions()
DELETE	doDelete()
TRACE	doTrace()



- Los dos parámetros de estos métodos son: HttpServletRequest (petición del cliente) y HttpServletResponse (respuesta para el cliente).

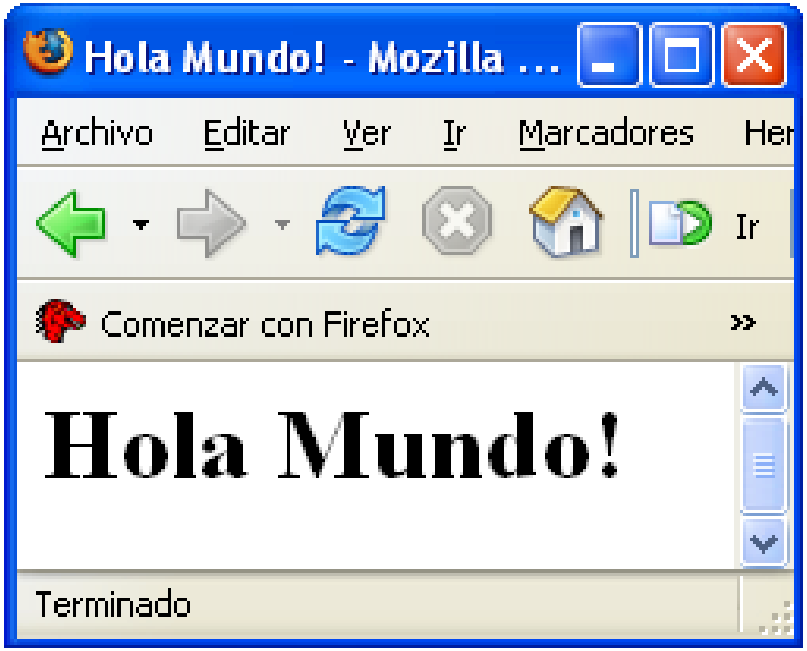
Servlets: Un ejemplo

```
//  servletHolaMundo.java
import java.io.*;
import jakarta.servlet.*;
import jakarta.servlet.http.*;
@WebServlet(urlPatterns = {"/holaMundo"})
public class servletHolaMundo extends HttpServlet {
   @Override
   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>Hola Mundo!</title>");
        out.println("</head>");
        out.println("<body>");
        out.println("<h1>Hola Mundo!</h1>");
        out.println("</body>");
        out.println("</html>");
   }
}
```

Invocando desde el navegador

- Invocación

  http://*HostServidor:port***/ContextPath/url_pattern**



Ver javadoc API

HttpServletRequest (hereda ServletRequest)

- Objeto generado desde la petición del cliente
- Contiene
  - Ruta del Servlet
    - request.getServletPath()
  - Cabeceras de la petición
  - Parámetros enviados por el usuario
    - Query Parameters: (GET) http://host:port/ContextPath/url_pattern?par1=val1
    - Path Parameters: (GET) http://host:port/ContextPath/url_pattern/pathparam
    - FormData: (POST) Datos enviados desde un formulario
  - Cookies y Sesiones
  - Autentificación de usuarios

HttpServletRequest

- Cabeceras HTTP de Petición

String **getHeader**(nombCab) : Devuelve el valor de la cabecera nombCab.

Enumeration **getHeaders**(nombCab) : Devuelve un objeto Enumeration con los valores de la cabecera nombCab.

Enumeration **getHeadersNames**() : Devuelve un objeto Enumeration con los nombres de las cabeceras.

int **getIntHeader**(nombCab) : Devuelve el valor de la cabecera nombCab como un valor entero.

long **getDateHeader**(nombCab) : Devuelve el valor de la cabecera nombCab como un objeto tipo fecha (Date).

HttpServletRequest

- Cabeceras HTTP de Petición

  Ejemplos

  1) Contenido de la cabecera user-Agent (ejercicio identificar el navegador)

```
String userAgent = request.getHeader("user-Agent");
```

  2) Leer todas las cabeceras de la petición

```
Enumeration nombresEncabezados = request.getHeaderNames();
while(nombresEncabezados.hasMoreElements()) {
  String nombreEncabezado = (String) nombresEncabezados.nextElement();
  out.println("<p>" + nombreEncabezado + ": ");
  out.println(request.getHeader(nombreEncabezado) + "</p>" );
}
```

Servlets: Petición. Objeto HttpServletRequest

- Lectura de parámetros enviados por el cliente (peticiones GET/POST)
  - Query Parameters

    String **getParameter**(*nombre*) : Devuelve el valor del parámetro *nombre* enviado en la petición.

    String[] **getParameterValues**(*nombre*) : Devuelve un array con los valores del parámetro *nombre* enviado en la petición.

    Enumeration **getParameterNames**() : Devuelve un objeto Enumeration con los nombres de los parámetros enviados en la petición

  - Path Parameters

    String **getPathInfo()** : Devuelve la ruta que sigue a la ruta del Servlet

Glassfish utiliza encoding ISO-8859-1 para usar UTF-8 incluir en glassfish-web.xml
<parameter-encoding default-charset="UTF-8"/>

Servlets: Petición. Objeto HttpServletRequest

- Lectura de parámetros

  GET: http://www.loquesea.es:8080/miServlet/accion?nombre=José&edad=25

  POST:

```
<form action="miServlet"  method="POST">
    Nombre: <input type="text" name="nombre" />
    Edad: <input type="text" name="edad" />
    <input type="submit" value="Enviar" />
</form>
```

  En el Servlet

```
…
String user = request.getParameter("nombre");
String year = request.getParameter("edad");
…
```

  NOTA: Por cuestiones de seguridad, es muy importante validar la información enviada al Servlet

Servlets: Petición. Objeto HttpServletRequest

- Lectura de todos los parámetros enviados por el cliente

```
...
 Enumeration nombresParam = request.getParameterNames();
 while(nombresParam.hasMoreElements()) {
      String nombreParam = (String) nombresParam.nextElement();
      out.print("<p>" + nombreParam + ": ");
      String[] valoresParam = request.getParameterValues(nombreParam);
      if (valoresParam.length == 1) {
           String valorParam = valoresParam[0];
           if (valorParam.length() == 0) out.println("<i>Sin valor</i>");
           else out.println(valorParam);
      } else {
        out.println("<ul>");
        for(int i=0; i<valoresParam.length; i++) {
          out.println("<li>" + valoresParam[i] + "</li>");
        }
        out.println("</ul></p>");
      }
}
...
```

Ver javadoc API

HttpServletResponse (hereda ServletResponse)

- Objeto generado como respuesta para el cliente
- Métodos
  - Cabeceras de la respuesta
  - Cookies

HttpServletResponse

getWriter(): devuelve un objeto tipo PrintWriter para el envío de texto plano.
Un objeto PrintWriter dispone de los métodos: print(), println() o write().
getOutputStream(): devuelve un objeto tipo ServletOutputStream que permite enviar un flujo de bytes.

sendRedirect(*url*):La respuesta es la url indicada.
sendError(error, mensaje) : La respuesta es creada con el código error y el mensaje

setContentType(*tipo*): Establece el tipo de respuesta. El más común text/html

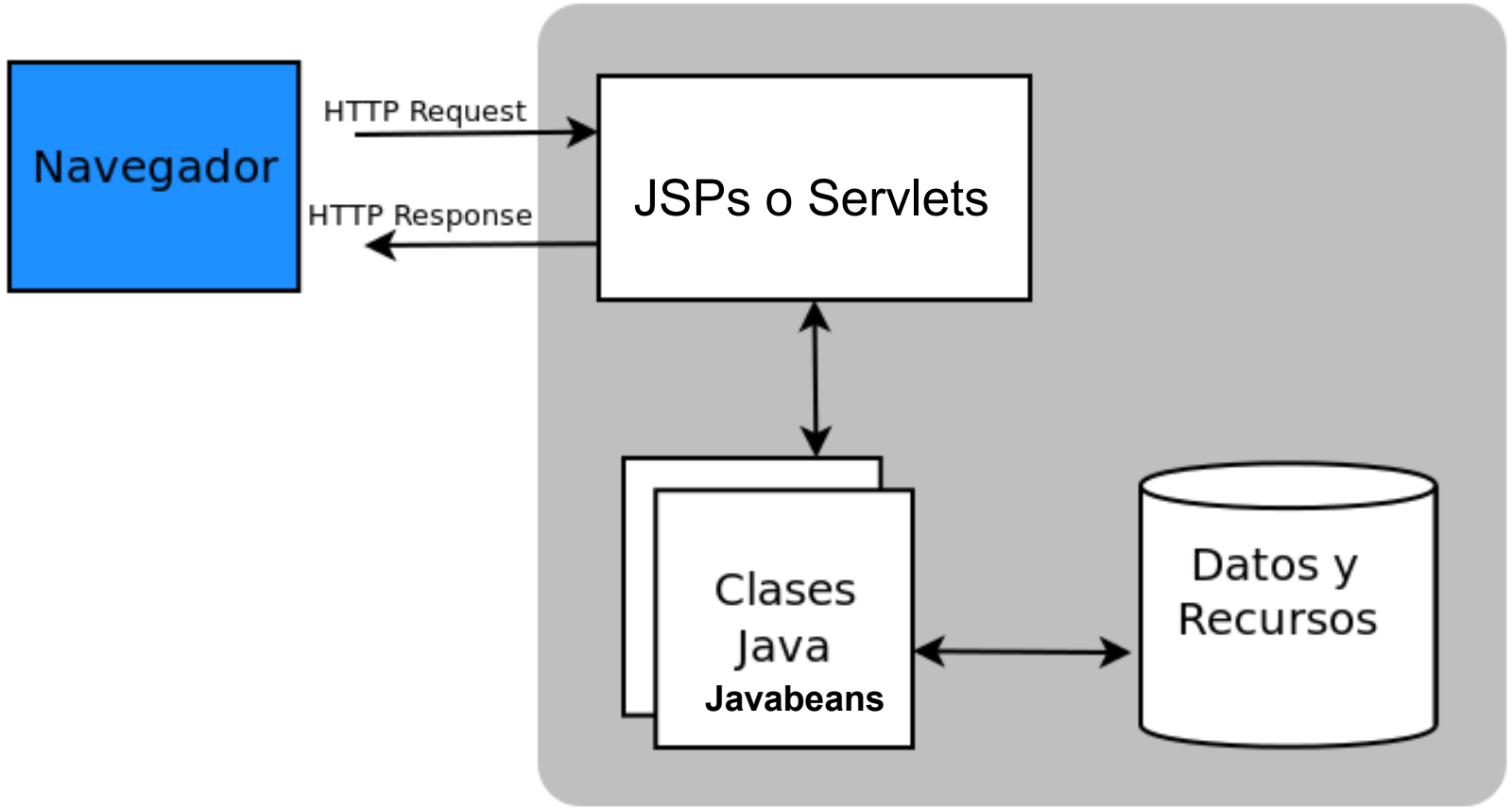
containsHeader(nomb) : Devuelve verdadero su la cabecera nomb está definida
setHeader(nomb, valor) : Asigna el valor indicado a la cabecera nomb.
setIntHeader(nomb, valorInt) . Asigna el valor entero a la cabecera nomb.
setDateHeader(nomb, fecha) : Asigna el valor (fecha) a la cabecera nomb.
addHeader(nomb, valor) : Añade la cabecera nomb con el valor indicado
addIntHeader(nomb, valorInt) : Añade la cabecera nomb.
addDateHeader(nomb, fecha) : Añade la cabecera nomb.

addCookie(*cookie*): Envía una cookie al cliente, Donde *cookie* es un objeto javax.servlet.http.Cookie que debe ser creado,

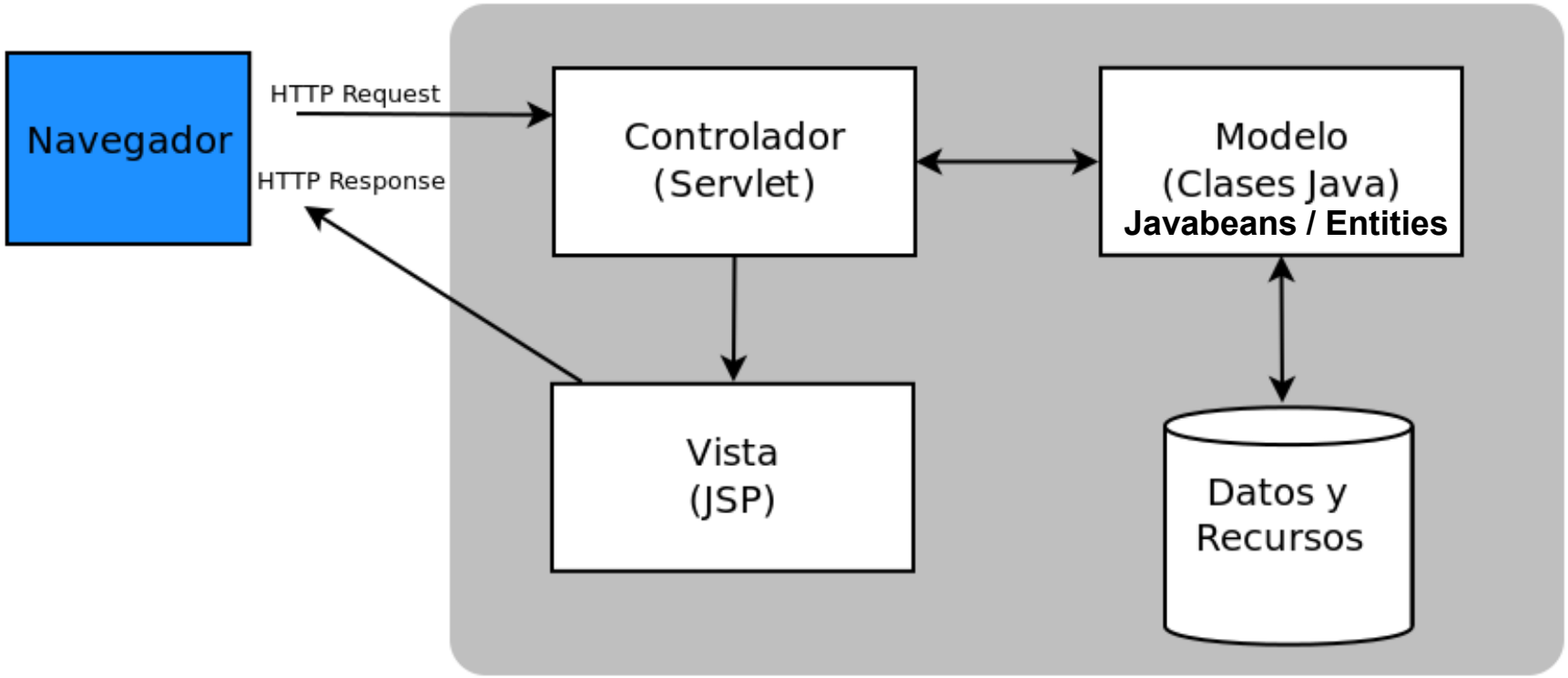
Contenido

- Tecnologías Java
- Introducción JEE
- Servlets
- MVC: Controlador

MODEL 1


Navegador
HTTP Request
HTTP Response
JSPs o Servlets
Clases
Java
Javabeans
Datos y
Recursos

MODEL 2 – MVC: Model View Controller



MVC: Model View Controller (Model 2)

- Delegar peticiones de un Servlet
  - Obtener un objeto RequestDispacher desde el objeto petición:

    RequestDispacher rd = request.getRequestDispacher("vista");

  - Delegar la petición a partir de ese objeto

    rd.forward (request, response).

  - Las vistas no deberían ser accesibles desde el navegador, por lo que se incluirán en la carpeta WEB-INF

MVC: Model View Controller (Model 2)

- Pasar datos desde el controlador a la vista

  - Ámbito Petición

    Los atributos creados en el ámbito de la petición son pasados a la vista.

    Métodos: request.getAttribute()
    request.setAttribute()
    request.removeAttribute()

    Objeto EL: requestScope

MVC: Model View Controller (Model 2)

- Ejemplo 1: MVC

```
@WebServlet(name = "Controller", urlPatterns = {"/do/*"})
public class Controller extends HttpServlet {

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

        String accion = request.getPathInfo();
        String vista;

        switch(accion) {
            case "/accion1" -> {
                // Código para la acción1
                vista = "vista1";
            }
            case "/accion2" -> {
                // Código para la acción2
                vista = "vista2";
            }
            default -> vista = "index";
        }

        RequestDispatcher rd = request.getRequestDispatcher("/WEB-INF/views/" + vista + ".jsp");
        rd.forward(request, response);
    }
}
```

MVC: Comunicación Model View Controller

- Atributos de la petición (ámbito request)

  request.setAttribute(“att”, objeto) : Crean un atributo de ámbito request
  request.getAtibute(“att”) : Obtiene el objeto “att” de la petición
  request.removeAttribute(“att”) : Elimina el atributo de la petición

- Atributos Globales (ámbito application)

```
ServletContext sc = getServletContext();
sc.setAttribute("att", objeto);
sc.getAttribute("att");
sc.removeAttribute("att", objeto);
```

- Atributos de sesión (ámbito session) (lo veremos más adelante)

MVC: Comunicación Model View Controller

- Ejemplo 2: MVC – Pasando datos a las vistas (Ámbitos)

```java
@WebServlet(name = "controller", urlPatterns = {"/do/*"})
public class controller extends HttpServlet {

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

        request.setAttribute("att1", "hola");
        ServletContext sc = getServletContext();
        sc.setAttribute("att1", new Integer(10));

        String accion = request.getPathInfo();
        String vista = "/index.jsp";

        /* ... */

        RequestDispatcher rd = request.getRequestDispatc
        rd.forward(request, response);
```

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>

        <p> Att1: <%= request.getAttribute("att1") %> </p>
        <p> Att1: ${requestScope.att1} </p>


        <p> Att1: <%= application.getAttribute("att1") %> </p>
        <p> Att1: ${applicationScope.att1} </p>

    </body>
</html>
```

MVC: Comunicación Model View Controller

- Ejemplo 3: MVC – Usar un Beans

```java
@WebServlet(name = "controller", urlPatterns = {"/do/*"})
public class controller extends HttpServlet {

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

        Usuarios u=new Usuarios(); // Completamos sus datos
        u.setNombre("Pepe");
        request.setAttribute("user", u);
        String vista = "/index.jsp";
        /* ... */
        RequestDispatcher rd = request.getRequestDispatcher(vista);
        rd.forward(request, response);
```

```java
package edu.daw.ejMVC;

public class Usuarios {

    private String nombre;

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

```jsp
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Hello World!</h1>

        <jsp:useBean id="user" scope="request" class="edu.daw.ejMVC.Usuarios" />

        <p>Hola <jsp:getProperty name="user" property="nombre"/></p>

        <jsp:setProperty name="user" property="nombre" value="Pedro"/>
        <p>Te he bautizado ahora como ${user.nombre}</p>

    </body>
</html>
```

MVC: Comunicación Model View Controller

- Ejemplo 4: MVC – Pasar una lista de objetos

Controlador

```
case "/accion3":
    Usuario u1 = new Usuario();
    u1.setNombre("Pepe");
    Usuario u2 = new Usuario();
    u2.setNombre("Juan");
    ArrayList<Usuario> lu = new ArrayList();
    lu.add(u1);
    lu.add(u2);
    request.setAttribute("users", lu);

    vista = "/vista3.jsp";
    break;
```

Vista

```
<h1>Usuarios</h1>
<%
    List<Usuario> lu = (List<Usuario>) request.getAttribute("users");
    if (lu != null) {
        out.println("<ul>");
        for (Usuario u : lu) {
            out.println("<li>" + u.getNombre() + "</li");
        }
        out.println("</ul>");
    } else {
        out.println("<p>No hay Usuarios</p>");
    }

%>
```

MVC: Comunicación Model View Controller

- Ejemplo 4: MVC – Pasar una lista de objetos (Alternativa JSTL)

Controlador

```
case "/accion3":
    Usuario u1 = new Usuario();
    u1.setNombre("Pepe");
    Usuario u2 = new Usuario();
    u2.setNombre("Juan");
    ArrayList<Usuario> lu = new ArrayList();
    lu.add(u1);
    lu.add(u2);
    request.setAttribute("users", lu);

    vista = "/vista3.jsp";
    break;
```

Vista

```
<%@ taglib prefix="c" uri="http://java.sun.com/jstl/core" %>
<h1>Usuarios</h1>
<c:choose>
    <c:when test="${!empty requestScope.users}">
        <ul>
            <c:forEach var="user" items="${requestScope.users}">
                <li>${user.nombre}</li>
            </c:forEach>
        </ul>
    </c:when>
    <c:otherwise>
        <p>No hay Usuarios</p>
    </c:otherwise>
</c:choose>
```