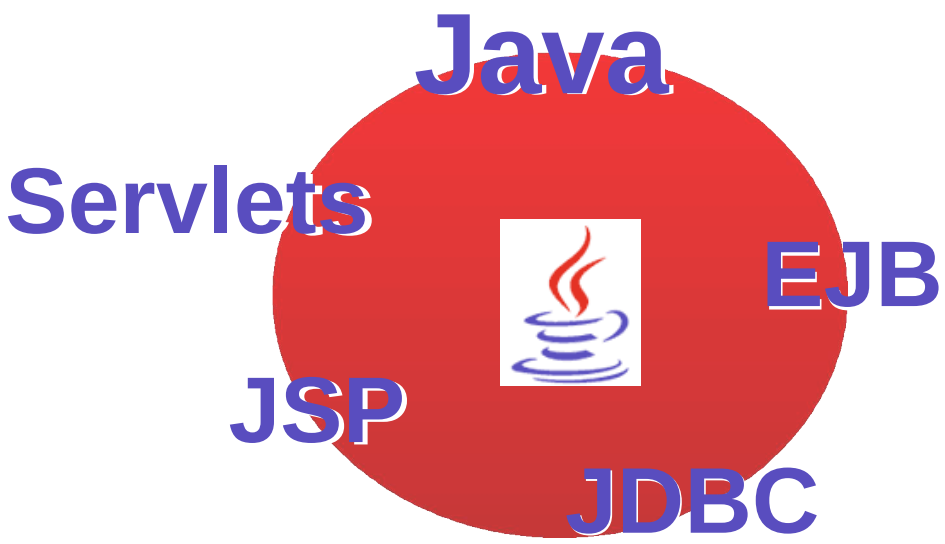
Diseño de Aplicaciones Web

2004 - 2005

http://laurel.datsi.fi.upm.es/~ssoo/DAW/



© SALVADORES OLAIZOLA, Manuel

msalvadores@inicia.es

Licencia

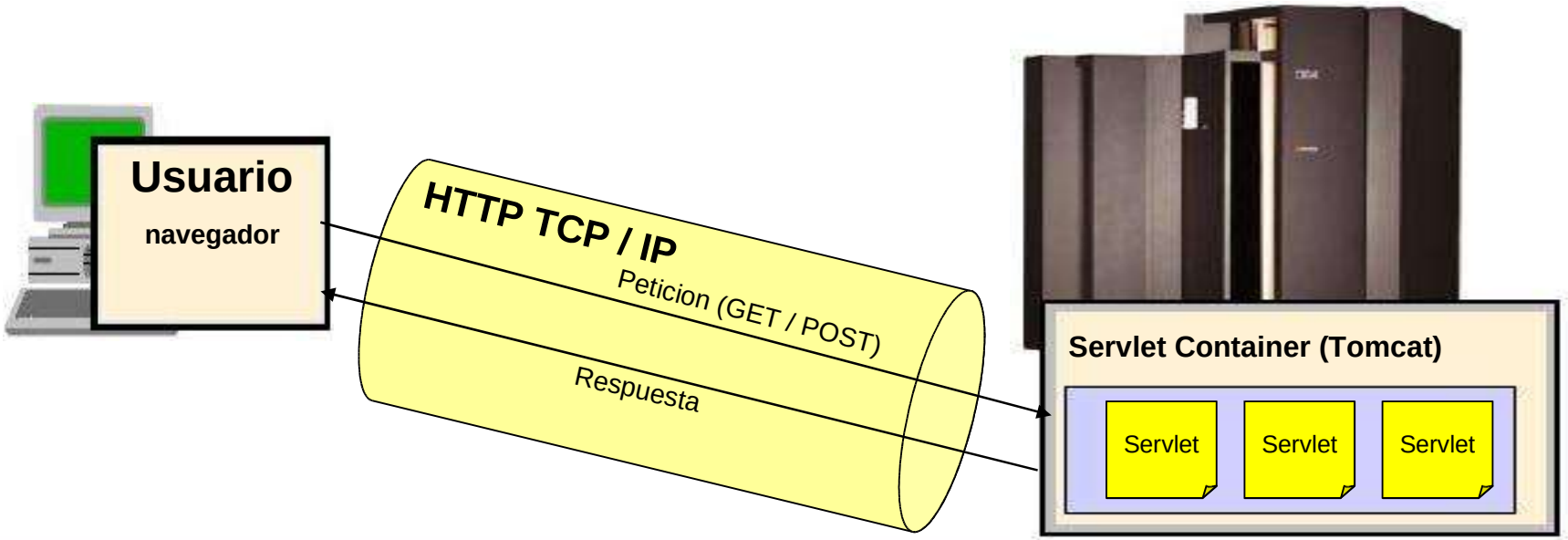
Objetivos & Indice

Adquirir conceptos básicos sobre:

- **Servlet's & JSP**
- **Tomcat & JBOSS**
- **JDBC**
- **MYSQL**
- **Eclipse y ANT**
- **EJB**

Concepto de Servlet

- Un **Servlet** es una clase Java usada para extender la capacidad de las aplicaciones basadas en el modelo cliente servidor y ejecución petición respuesta.
- Los Servlets son una potente herramienta para la generación de contenido dinámico en la Web.
- El **Servlet Container** es el componente encargado de la creación, acceso y destrucción de los Servlets, controla su ciclo de vida.



Concepto de Servlet

- Un **Servlet** es una clase Java que extiende de javax.servlet.http.HttpServlet y que sobreescribe los métodos doPost o doGet o ambos.
- La petición se representa por la clase HttpServletRequest.
- La respuesta se representa por la clase HttpServletResponse.

```
package ejemplos;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloWorldServlet 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("<body><h1>Hola Mundo</h1></body>");
     out.println("</html>");
  }
}
```

Concepto de JSP

- Un **JSP,** es una página dinamica de servidor Java.
- Es un archivo de texto compuesto de :
  1. Cabecera con importaciones y parametros.
  2. Código cliente, normalmente HTML, XML y Javascript.
  3. Código servidor Java, denominado Scriptlet y escrito entre los caracteres <% y %>.
  4. Tags: instrucciones en formato XML, asociadas a clases Java.
- Un JSP se transforma en un Servlet en tiempo de ejecución.

Concepto de JSP

```
<%@ page import="ejemplo.Usuario" %>

<body>
<html>
<jsp:getProperty name="usuario" property="username" />
<br/>

<%
String sEdad = request.getParameter("edad");
if (sEdad != null) {
        int iEdad = Integer.parseInt(sEdad);
        if (iEdad > 17)
                out.print("eres mayor de edad");
        else
                 out.print("NO eres mayor de edad");
}
%>

</html>
</body>
```

Cabecera

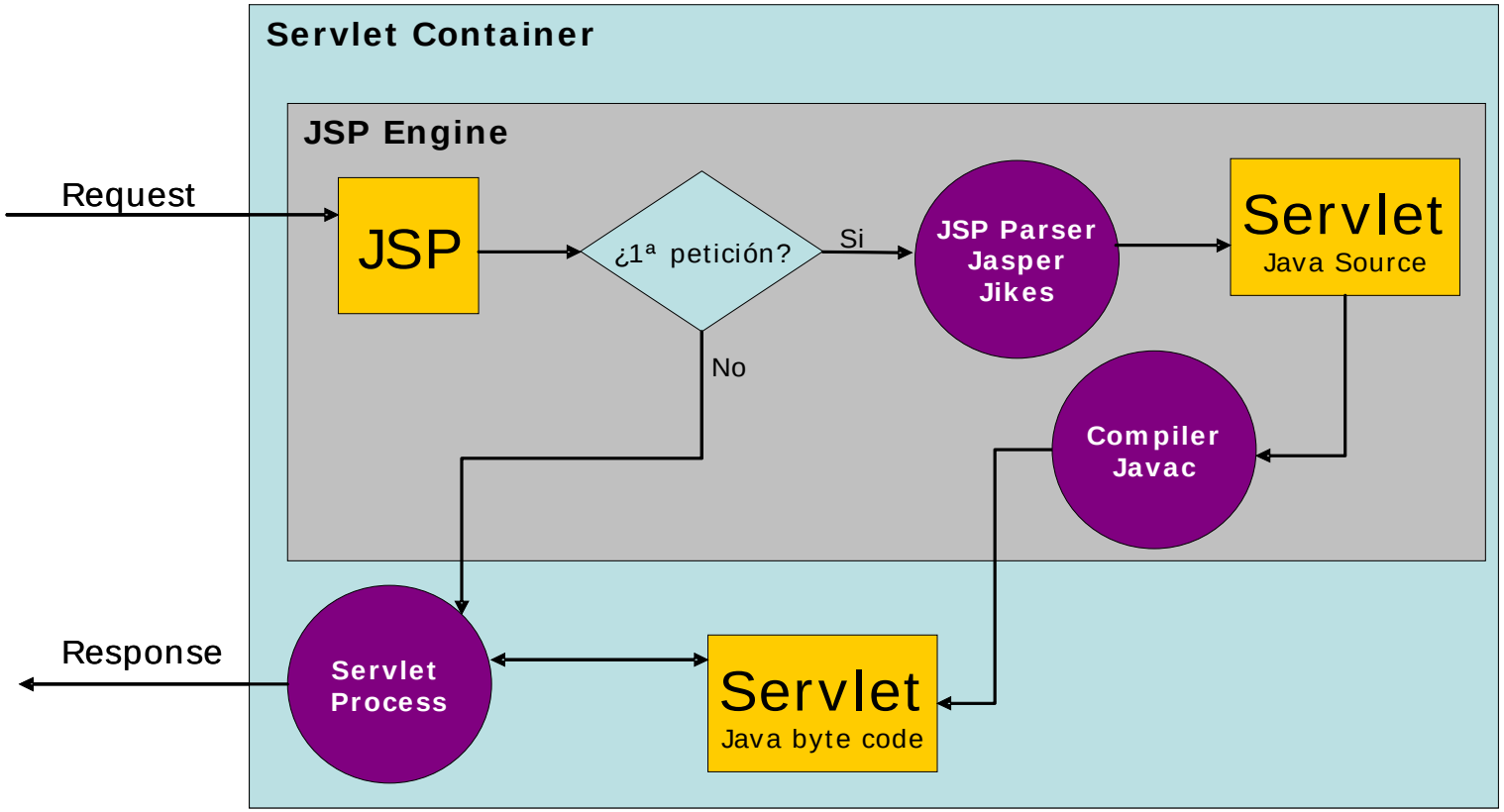
Contenido HTML

Tag

Scriptlet

Procesamiento JSP

La primera vez que se realiza una petición sobre un JSP el JSP Engine lo traduce en un Servlet, lo compila y lo procesa. En sucesivas peticiones simplemente se procesa a través del Servlet Container.



Servlet Container: Tomcat

- Tomcat es la implementación de referencia de la especificación de Servlet y de JSP.
- Es totalmente gratuito y se puede descargar de **http://jakarta.apache.org/tomcat/** con licencia Apache Software License.
- Existen distribuciones para todas las plataformas existentes.
- Cada versión de Tomcat lleva asociada una compatibilidad de las especificaciónes de Servlet y JSP.

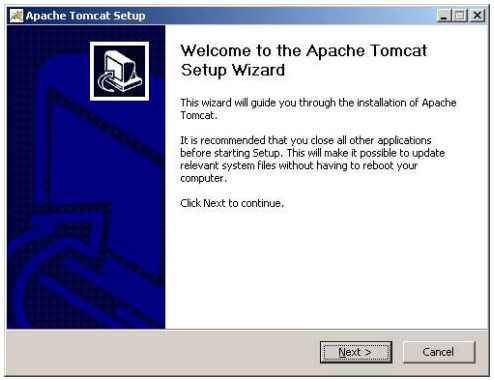
Servlet / JSP	Tomcat Version
2.4 / 2.0	5.0.28
2.3 / 1.2	4.1.31
2.2 / 1.1	3.3.2

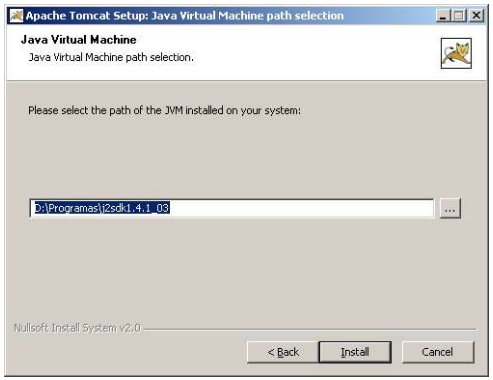
Tomcat: Instalación en windows

Paso 1: Instalación del JDK “Java Development Kit” (recomendada versión 1.4.2)

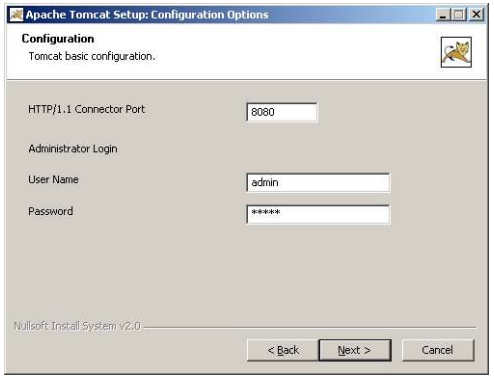
Único parametro que debemos indicar es la carpeta de instalación.

Paso 2: Instalación de Apache Tomcat versión 5.0.28

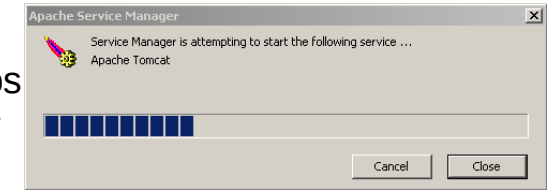




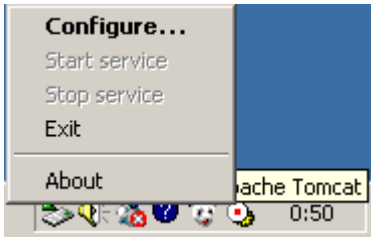
En la instalación detecta automaticamente el JDK instalado anteriormente.



Debemos configurar en que puerto queremos que escuche el servidor y poner el usuario y el password del administrador.

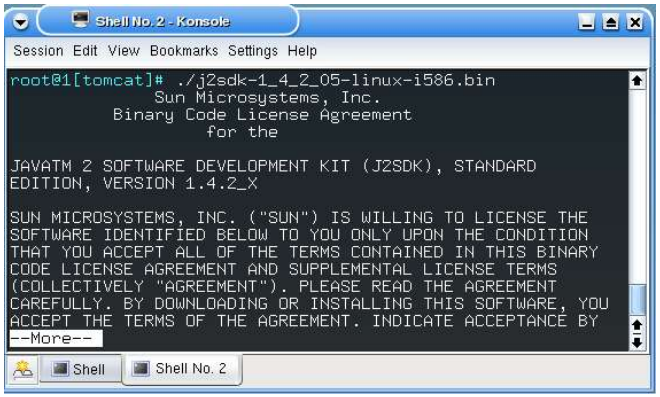


Paso 3:Arranque del servidor:



Tomcat: Instalación en linux

Paso 1: Instalación del JDK “Java Development Kit” (recomendada versión 1.4.2)

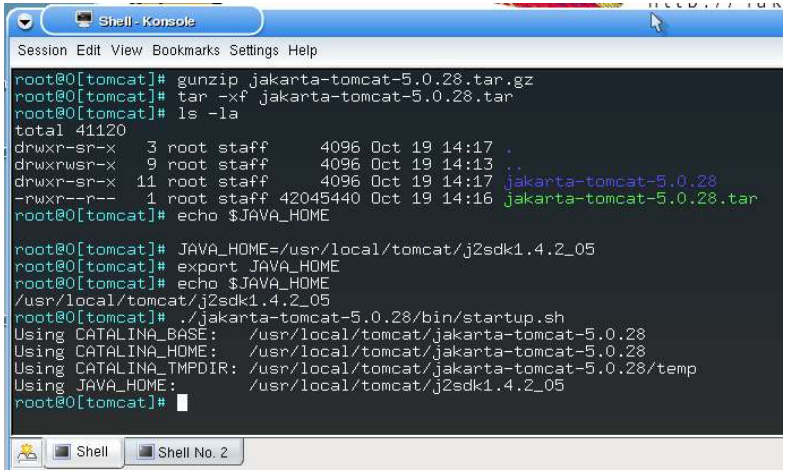


./j2sdk-1_4_2_05-linux-i586.bin instalará el jdk en el directorio que lo contiene.

Paso 2: Instalación de Apache Tomcat :

Una vez descargado jakarta-tomcat-5.0.28.tar.gz lo situamos en el directorio que deseamos instalarlo y realizamos la siguiente secuencia de comandos :

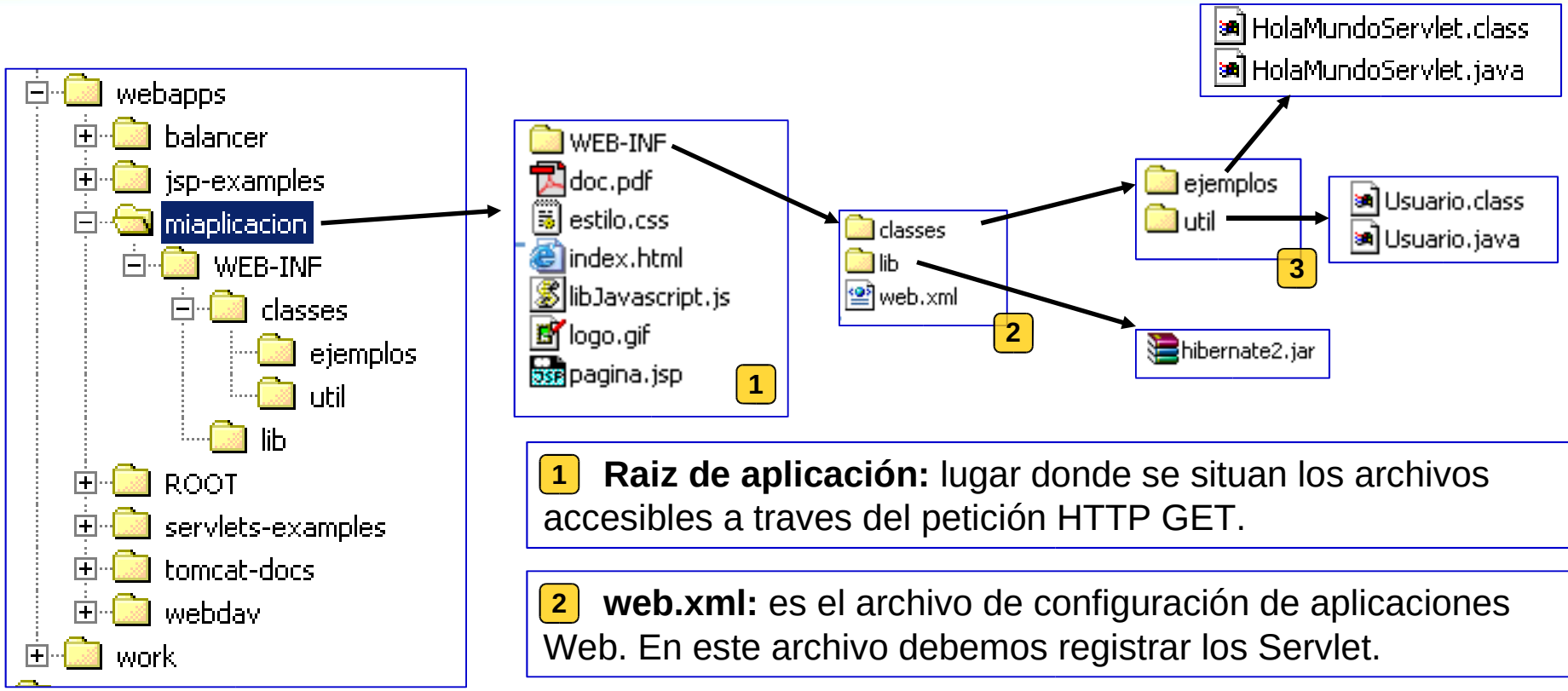
1. gunzip jakarta-tomcat-5.0.28.tar.gz
2. tar –xf jakarta-tomcat-5.0.28.tar
3. JAVA_HOME=<dir_install_jdk>
4. export JAVA_HOME



Paso 3:Arranque del servidor

En el directorio de instalación de tomcat encontramos los Shell Script de arranque y parada del servidor. Para arrancar el servidor deberemos ejecutar startup.sh

Estructura aplicación



1 **Raiz de aplicación:** lugar donde se situan los archivos accesibles a traves del petición HTTP GET.

2 **web.xml:** es el archivo de configuración de aplicaciones Web. En este archivo debemos registrar los Servlet.

3 Los subdirectorios de **classes** son los paquetes de aplicación. En este caso hay dos paquetes:

- **ejemplos:** que contiene el Servlet **HolaMundoServlet**
- **util:** que contiene la clase **Usuario**, que es utilizada por el Servlet.

Especificación Servlet

- El API que proporciona Sun para la especificación de Servlets se encuentra publicada en **http://java.sun.com/products/servlet/2.X/javadoc/**
- La especificación se divide en dos paquetes:
  - javax.servlet
  - javax.servlet.http
- Clases más importantes, para el protocolo HTTP.

HttpServlet	**<<HttpServletResquest>>**	**<<HttpServletResponse>>**
•Clase que extenderemos para implementar nuestro servlets. •Sobrescribiremos los métodos **doPost y doGet** para implementar la lógica de nuestras aplicaciones.	•Interfaz que representa la petición que se realiza sobre un servlet. •Mediante esta interfaz podemos acceder la sesión del usuario, parametros enviados por POST/GET o parametros de configuración.	•Interfaz que representa la respuesta generada por un Servlet •Proporciona métodos para generar la salida dinámica.

Implementación Servlet

- Pasos para la implementación de un Servlet (Ejemplo):
  1. Crear una clase Java que extienda de HttpServlet
  2. Sobreescribir el método doPost o doGet o ambos.
  3. Obtener la salida
  4. Enviar el contenido dinámico
  5. Compilar el Servlet
  6. Mapear la clase a una URL en el fichero **web.xml**
  7. Ejecutarlo / Probarlo

Implementación Servlet

HolaMundoServlet.java

```
package ejemplos;

import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaMundoServlet 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><title>Titulo</title></head>");
    out.println("<body><h1>Hola Mundo</h1></body>");
    out.println("</html>");
  }
}
```

Implementación Servlet

HolaMundoServlet.java

```
package ejemplos;
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HolaMundoServlet 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><title>Titulo ejemplo Servlet 1</title></head>");
    out.println("<body><h1>Hola Mundo</h1></body>");
    out.println("</html>");
  }
}
```

Paquete donde se encuentra el Servlet

Importaciones, librerias necesarias.

Declaración de clase que extiende de HttpServlet

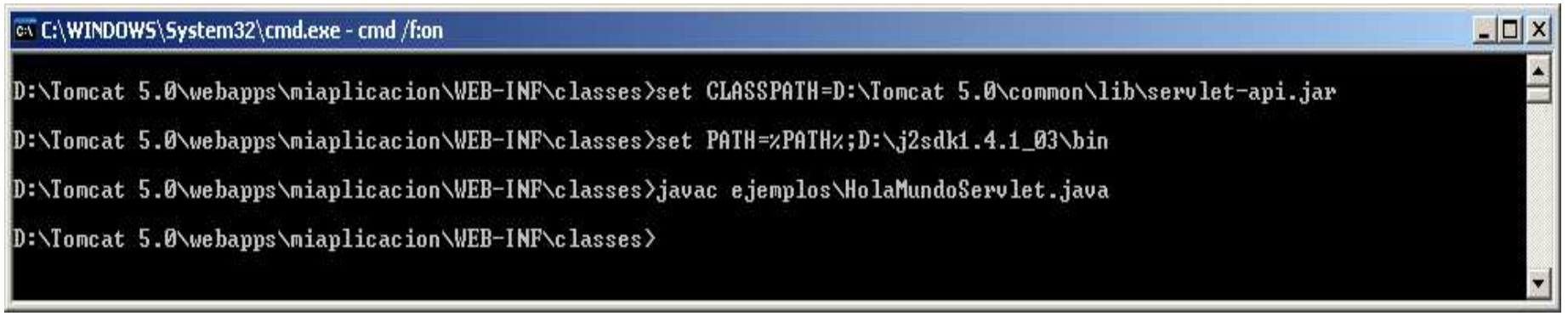
Sobreescritura método doGet

Respuesta texto en formato HTML

Obtención de la salida.

Generación de contenido dinámico.

Compilación Servlet



Secuencia de instrucciones para la compilación de un Servlet:

2. set CLASSPATH=D:\Tomcat 5.0\common\lib\servlet-api.jar añadimos la librería que contiene el API de Servlet
3. set PATH=%PATH%;D:\j2sdk1.4.1_03\bin añadimos el directorio donde se encuentra el comando JAVAC para poder compilar desde el directorio CLASSES de nuestra aplicación
4. javac ejemplos\HolaMundoServlet.java llamada al compilador de Java pasando como parametro el fichero que queremos compilar.

Configuración: web.xml

Web.xml :

- Es el fichero de configuración de aplicaciones Web.
- Registra los Servlets mediante el Tag XML <servlet>.
- Asocia URL's con los Servlets. <servlet-mapping>
- Configura parametros de seguridad.
- Debe estar siempre situado en el directorio WEB-INF de cada aplicación contenida en el Servlet Container

miaplicacion/WEB-INF/web.xml

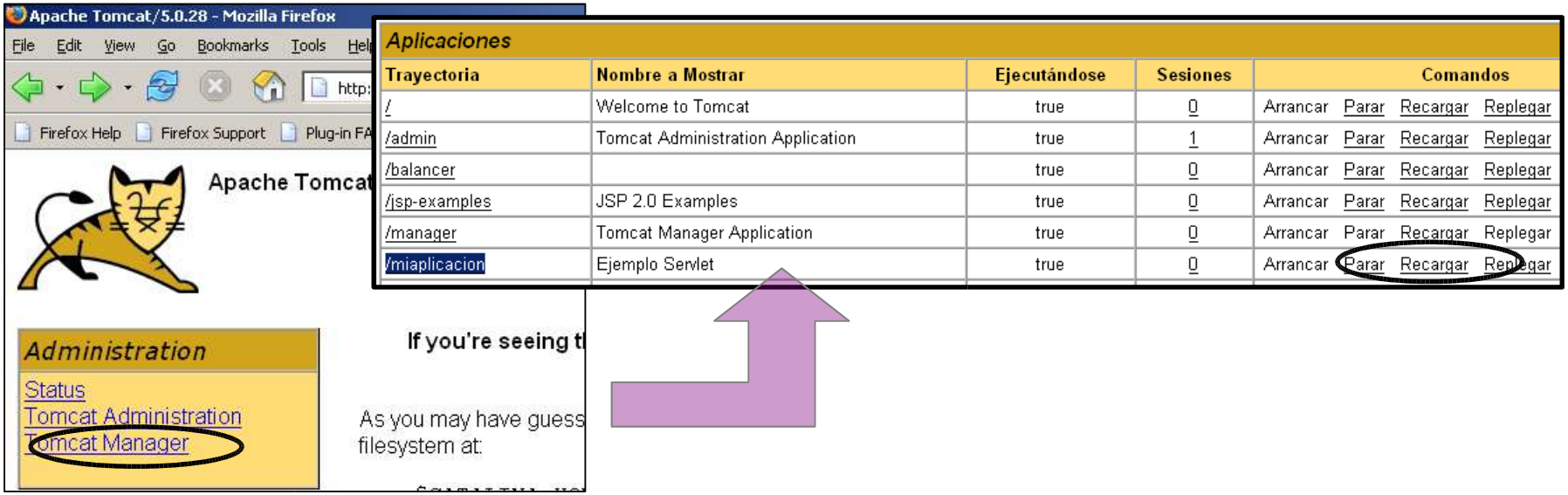
```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!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>

  <display-name>Ejemplo Servlet</display-name>
  <description>
     Ejemplo de desarrollo de Servlets.
  </description>

  <servlet>
    <servlet-name>HolaMundo</servlet-name>
    <servlet-class>ejemplos.HolaMundoServlet</servlet-class>

  </servlet>

  <servlet-mapping>
    <servlet-name>HolaMundo</servlet-name>
    <url-pattern>/ejemplo1</url-pattern>
  </servlet-mapping>

  <welcome-file-list>
    <welcome-file/>
  </welcome-file-list>

</web-app>
```

Despliegue

Una vez compilado y registrado el Servlet en el web.xml procedemos al despliegue y prueba de ejecución.

Dos opciones:

1.- Servidor arrancado: debemos recargar la aplicación desde el planel de administración del Tomcat.



2.- Servidor caido: simplemente iniciandolo se desplegará nuestra aplicación.

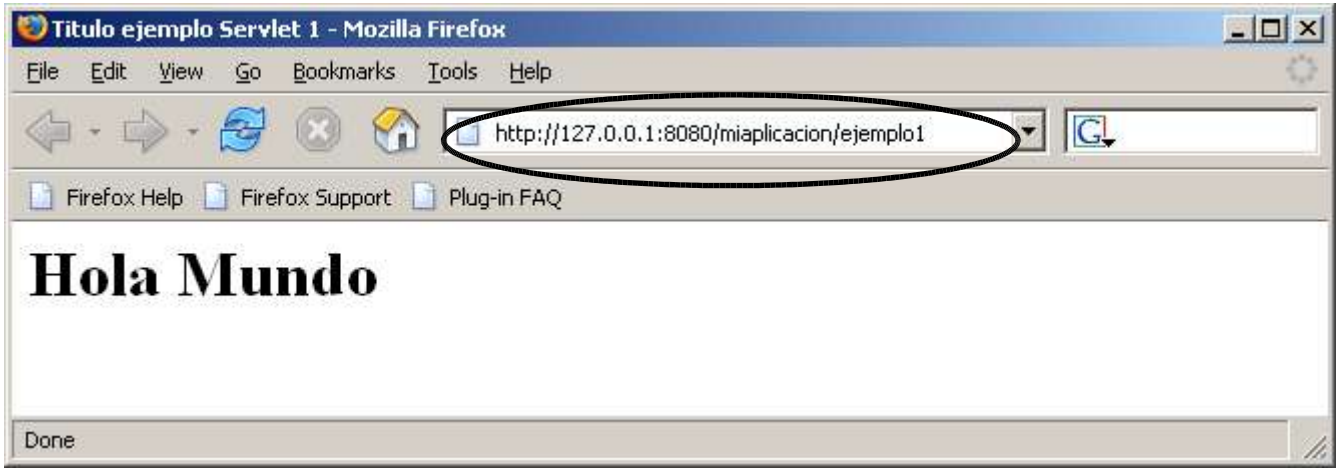
Prueba

Abriendo un navegador podemos probar el Servlet desarrollado.

http://127.0.0.1/miaplicacion/ejemplo1

Directorio donde se situa la aplicación dentro de webapps

URL mapping introducido en el web.xml. Parametro que asocia la ejecución de classes Servlet con URL's



Implementación JSP

- Pasos para la implementación de un JSP (Ejemplo):
  1. Escribir el JSP
  2. Ejecutarlo / Probarlo.

JSP VS Servlet

Generalmente el desarrollo de un JSP es mucho más rapido y cómodo que el de un Servlet.

Pero hay casos en los cuales se deben implmentar Servlets.

- **Pasos para la implementación de un Servlet (Ejemplo):**
  1. **Crear una clase Java que extienda de HttpServlet**
  2. **[illegible]étodo doPost o doGet o**
  3. **[illegible] dinámico**
  5. **Compilar el Servlet**
  6. **Mapear la clase a una URL en el fichero web.xml**
  7. **Ejecutarlo / Probarlo**

Implementación JSP

miaplicacion/ejemplo2.jsp

```
<%@page language="java"%>
<HTML>
<HEAD>
<TITLE>Ejemplo de JSP</TITLE>
</HEAD>

<BODY>

<% if (request.getParameter("nombre")==null) { %>

    <h1>Hola Mundo</h1>

  <% } else { %>

    <h1> Hola <%=request.getParameter("nombre")%> </h1>

  <% } %>

</BODY>

</HTML>
```

ejemplo2.jsp:

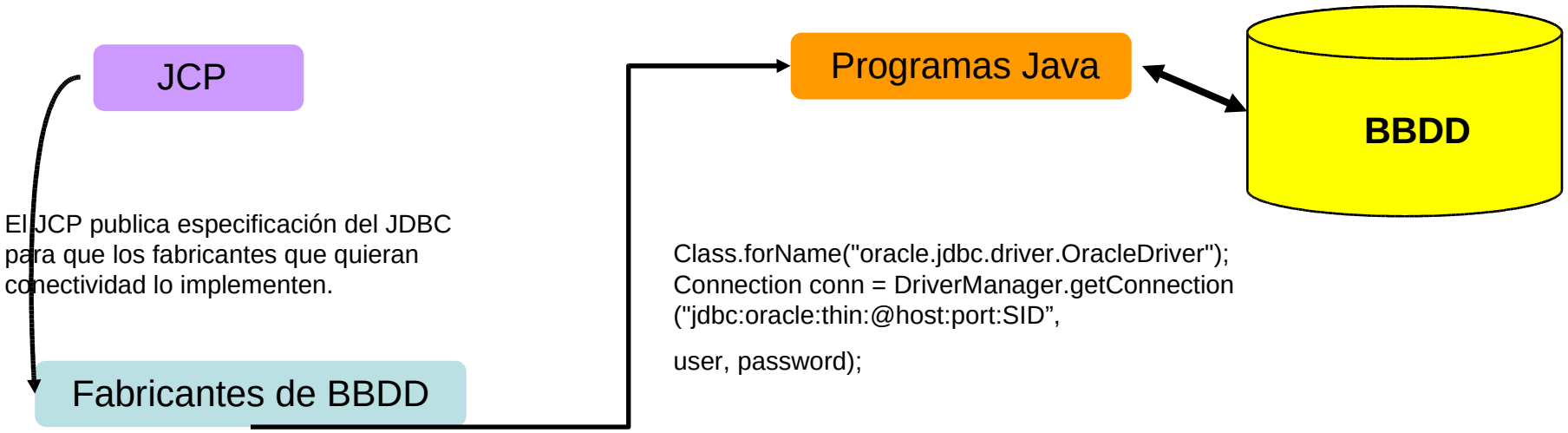
es un JSP que en caso de no recibir el parametro “nombre” en el request imprimirá “Hola Mundo”, en otro caso “Hola <nombre>”

Probando el JSP

Con situar el fichero JSP, en el directorio de la aplicación ya estará accesible a través del servidor. Desarrollando JSP no es necesario reacargar las aplicaciones ni reiniciar el servidor para comprobar los cambios.

Acceso a bases de datos: JDBC

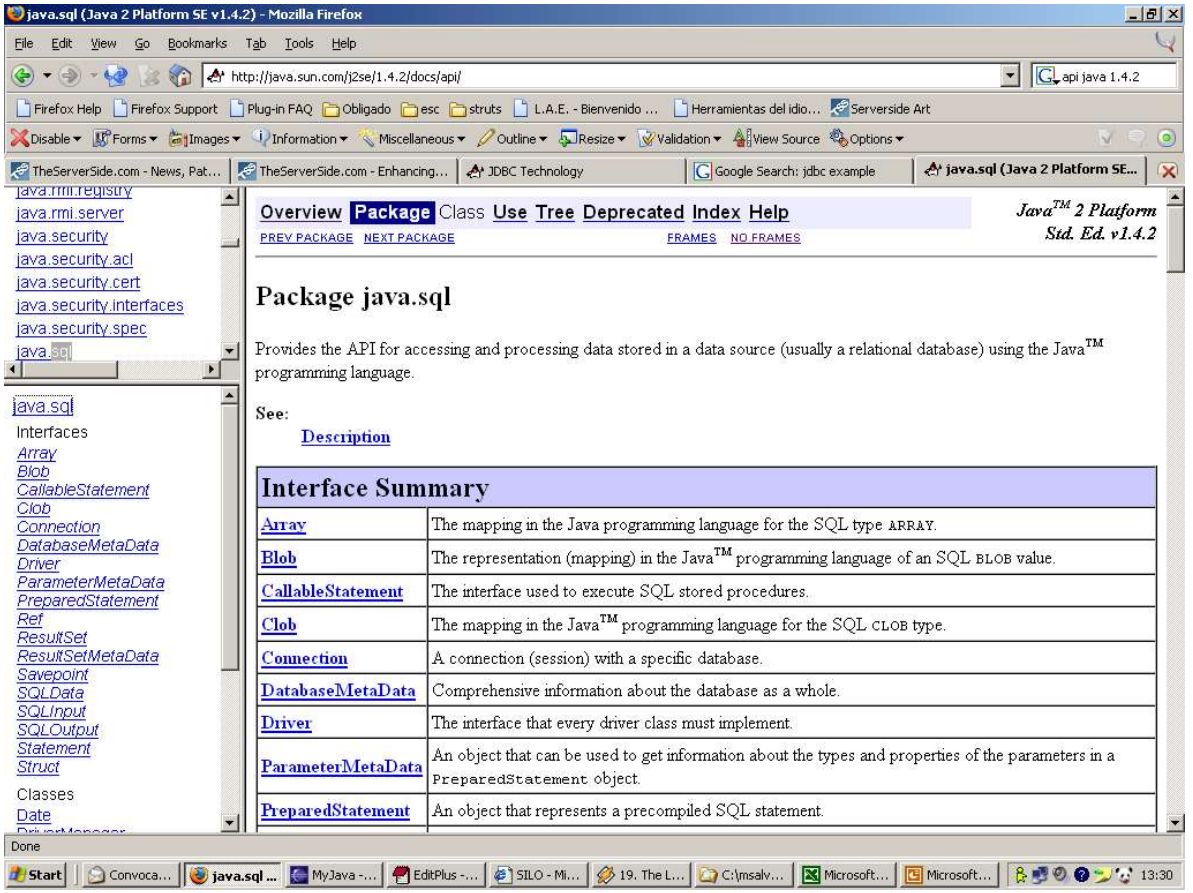
- JDBC es un API (incluida tanto en las diferentes versiones de J2SE y J2EE) que proporciona conectividad con gestores de bases de datos.
- Es una “interfaz” de acceso a bases de datos, es decir SUN no proporciona una implementación, sino que son los fabricantes los que proporcionan drivers JDBC para que los programas Java tengan conectividad con sus bases de datos.



API JDBC

El api JDBC lo podemos encontrar en los paquetes :

- **java.sql**
- **javax.sql**



JDBC: Las clases más usadas

java.sql.Connection

Interfaz del paquete java.sql que representa la conexión con la base de datos.

Pasos para obtener una conexión :

Indicamos el driver de la base de datos que vamos a utilizar.

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

Mediante la clase DriverManager (Gestor de drivers para acceso a bases de datos) obtenemos la conexión pasando como parametros la cadena de conexión especificada por el proveedor de base de datos, usuario y password. La cadena de conexión suele llevar incluido la maquina (host) y puerto (port) donde se encuentra la instancia de base de datos a la que accedemos.

```
Connection conn = DriverManager.getConnection
("jdbc:oracle:thin:@HOST:PORT:SID",“admin”,”pass”);
```

JDBC: Cadenas de conexión

Algunas cadenas de conexión para driver JDBC

ORACLE **jdbc:oracle:thin:@HOST:PORT:SID**

DB2 software **jdbc:db2://HOST:PORT/INSTANCE**

MySQL **jdbc:mysql://host:port/db&user=your_user&password=your_password**

SAP DB **jdbc:sapdb://host/dbname?autocommit=off&timeout=30**

JDBC: Las clases más usadas

java.sql.Statement

Es el objeto utilizado para ejecutar las sentencias SQL. Hay que pedirselo al objeto Connection

Pasos para ejecutar una sentencia SQL :

Obtenemos el objeto Statement a través de la conexión.

```
Stament stmt = conn.createStatement();
```

A través del objeto Statement podemos lanzar las sentencias SQL. Los métodos más utilizados para ejecutar SQL son : “executeUpdate”, “executeQuerie” o “execute”.

```
int updates = stmt.executeUpdate(“update USUARIOS set TLF = ‘917658991’ where COD = ‘1’ ”);
```

Cierre de los objetos al finalizar su uso. Es importante.

```
stmt.close();
conn.close();
```

JDBC: Las clases más usadas

java.sql.Resultset

Es el objeto en el cual se almacenan el resultado de las consultas, se obtiene a través del Statement.

Pasos para obtener los datos de una consulta :

A través del Statement obtenido lanzamos consultas con el método executeQuery, este método nos devuelve un objeto Resultset con el resultado de la consulta.

```
Resultset rs = stmt.executeQuery("SELECT * FROM USUARIOS");
```

Para obtener los datos debemos ejecutar siempre el metodo "next" del Resultset, este metodo devuelve true/false en funcion de la disponibilidad de datos.

```
while(rs.next()) {
        System.out.println(rs.getString("NOMBRE"));
        System.out.println(rs.getString("APELLIDO1"));
}
```

```
rs.close();
```

Una vez procesada la información debemos cerrar el resultset.

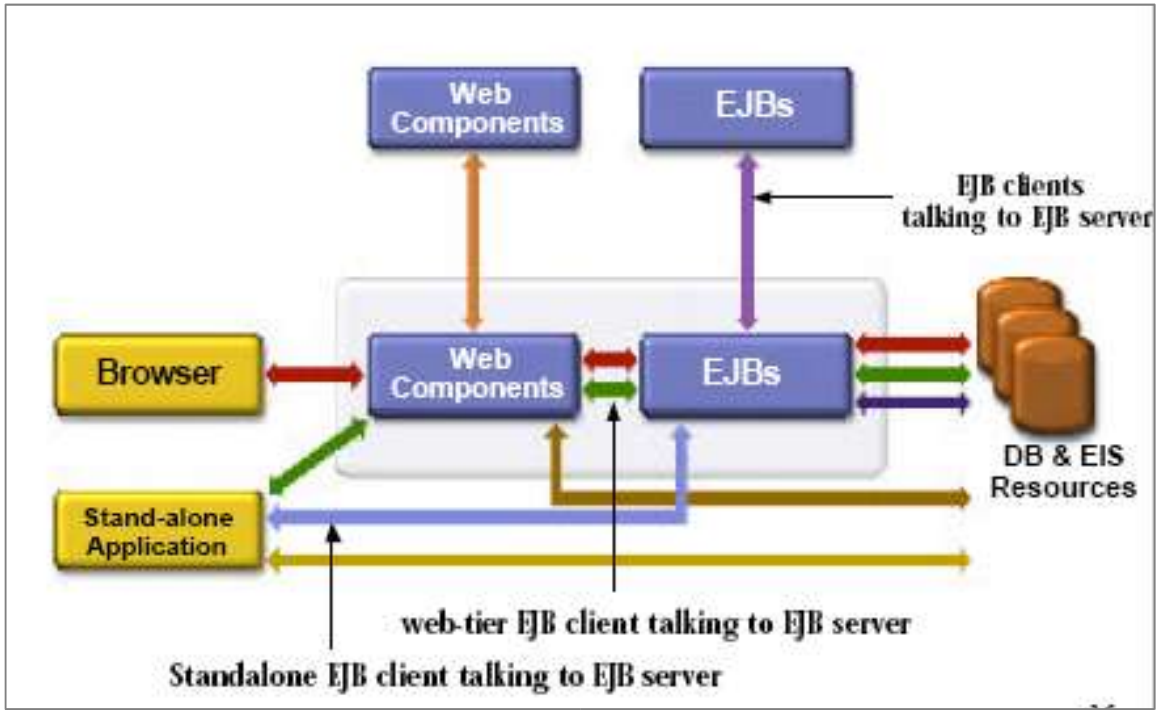
JDBC: Buenas prácticas

1. Centralizar el acceso a BBDD en algún paquete de nuestra aplicación.
2. Cerrar los objetos JDBC dentro de bloques “finally”
3. Usar PreparedStatement en vez de Statement
4. Especificar la lista de campos en las sentencias “SELECT” e “INSERT”
5. Intentar utilizar SQL estandar siempre.
6. Externalizar las sentencias SQL en ficheros, no incrustarlas en el codigo fuente.
7. Intentar no utilizar objetos propietarios (Ejemplo: OracleStatement …)

Enterprise JavaBeans “EJB”

EJB es una arquitectura para desarrollar y desplegar aplicaciones distribuidas basadas en componentes.

Las aplicaciones basadas en EJB son escalables, transaccionales y con seguridad multiusuario.



Cuando usar EJB's

Los EJB son adecuados cuando:

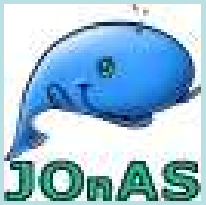
- Queremos construir una aplicación escalable, donde el posible crecimiento de usuarios puede provocar que el código que procesa nuestra aplicación esté repartido en varios servidores.
- Necesitemos asegurar la integridad de los datos. Los EJB al ser transaccionales aseguran la coherencia de los objetos compartidos y la atomicidad de los procesos complejos.
- La aplicación servidora sea accesible a través de diferentes tipos de clientes.

Tipos de EJB

- Session EJB: representación de procesos de lógica de negocio.
  - Stateless: EJB de sesión sin estado
  - Statefull: EJB de sesión con estado
- Entity EJB: representación de objetos de negocio.
  - CMP “Container Managed Persistence”: persistencia manejada por el contenedor EJB.
  - BMP “Bean Managed Persistence”: persistencia manejada por el JavaBean.
- MDB “Message Driven Beans”: EJB para procesado de mensajes asíncronos.

EJB Containers

Open source

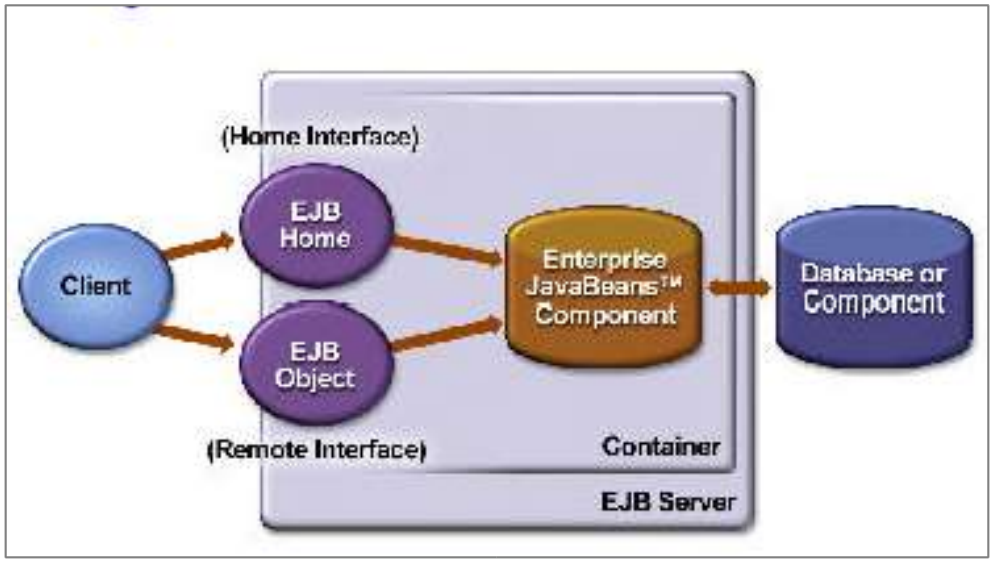


BEA WebLogic Server 8.1

Arquitectura EJB

Un EJB necesita de la implementación de tres ficheros:

1. Remote Interface
2. Home Interface
3. Enterprise Bean Class



Implementando Session EJB

Un EJB necesita de la implementación de tres ficheros:

1. Remote Interface
2. Home Interface
3. Enterprise Bean Class

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;
import java.math.*;

public interface Converter extends EJBObject {

  public BigDecimal dollarToYen(BigDecimal dollars)
    throws RemoteException;

  public BigDecimal yenToEuro(BigDecimal yen)
    throws RemoteException;

}
```

Implementando Session EJB

Un EJB necesita de la implementación de tres ficheros:

1. Remote Interface
2. Home Interface
3. Enterprise Bean Class

```
import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBHome;


public interface ConverterHome extends EJBHome {

  Converter create() throws RemoteException, CreateException;

}
```

Implementando Session EJB

Un EJB necesita de la implementación de tres ficheros:

1. Remote Interface
2. Home Interface
3. Enterprise Bean Class

```
import java.rmi.RemoteException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;


public class ConverterBean implements SessionBean {


 public BigDecimal dollarToYen(BigDecimal dollars) {
  BigDecimal result = dollars.multiply(yenRate);
  return result.setScale(2,BigDecimal.ROUND_UP);
 }

  public BigDecimal yenToEuro(BigDecimal yen) {
    BigDecimal result = yen.multiply(euroRate);
    return result.setScale(2,BigDecimal.ROUND_UP);
  }

  public ConverterBean() {}
  public void ejbCreate() {}
  public void ejbRemove() {}
  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void setSessionContext(SessionContext sc) {}
}
```

Cliente EJB

Pasos para la instanciación de un EJB:

```
InitialContext ic = new InitialContext();

Object objRef = ic.lookup(" java:comp/env/ejb/TheConverter");

ConverterHome home =  (ConverterHome)PortableRemoteObject.narrow(
                                                objRef, ConverterHome.class);
Converter converter  = home.create();
```

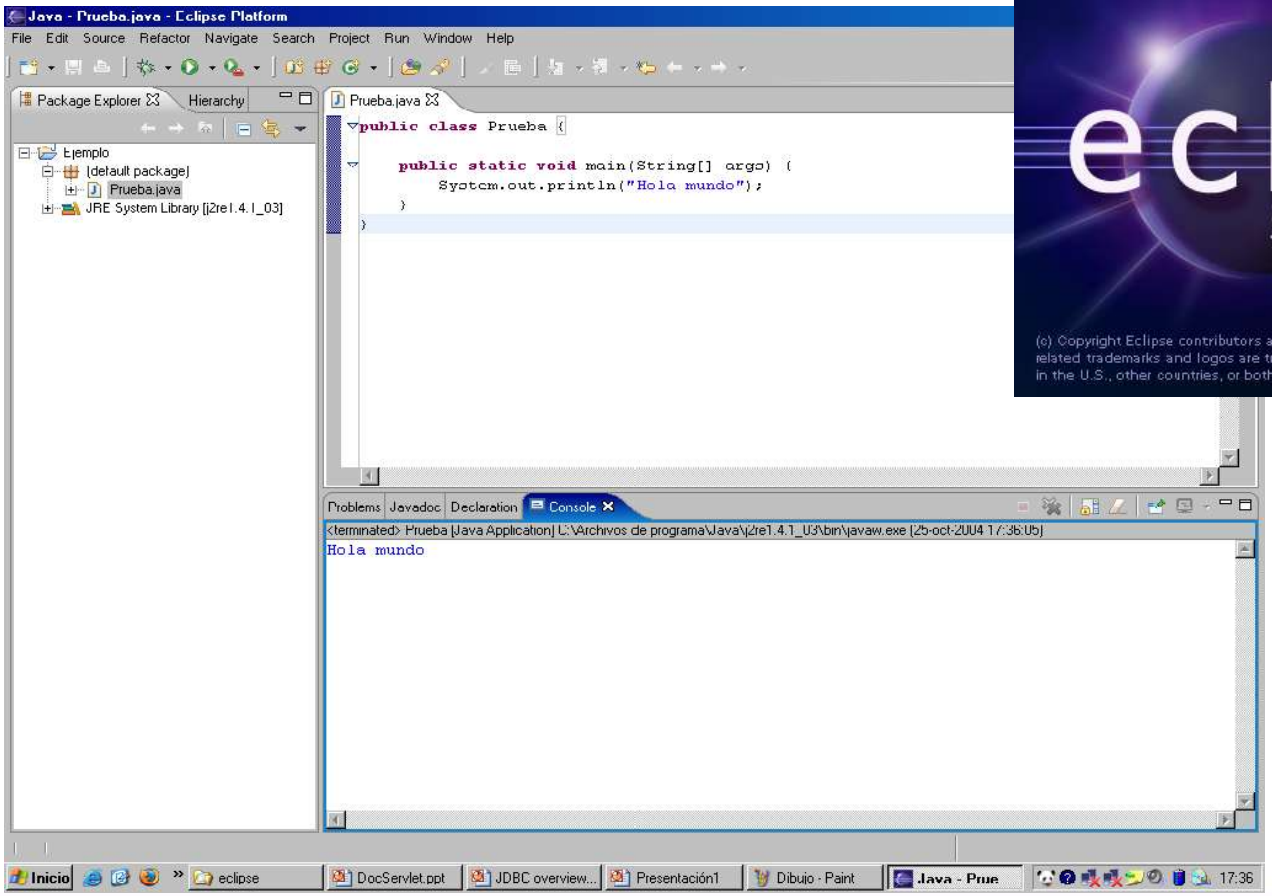
Empaquetamiento y despliegue

- jar cvf converter.jar
  org\ejemplo\ejb\session\Converter.class
  org\ejemplo\ejb\session\ConverterHome.class
  org\ejemplo\ejb\session\ConverterBean.class
  META-INF
- Copy converter.jar
  <JBOSS_HOME>/server/default/deploy

Metodología de trabajo

Entorno de desarrollo integrado (IDE) :

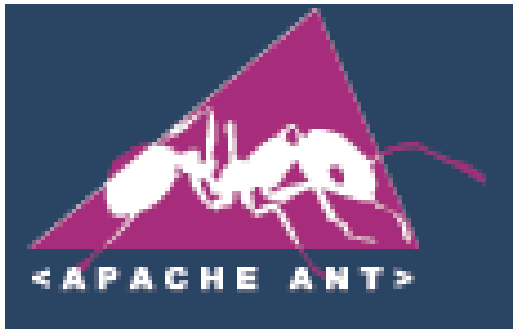
www.eclipse.org



Metodología de trabajo

ANT: Herramienta de Scripting Java basada en XML.

http://jakarta.apache.org/ant




```
<project name="MyProject" default="dist" basedir=".">
    <description>
        simple example build file
    </description>
  <!-- set global properties for this build -->
  <property name="src" location="src"/>
  <property name="build" location="build"/>
  <property name="dist"  location="dist"/>

  <target name="init">
    <!-- Create the time stamp -->
    <tstamp/>
    <!-- Create the build directory structure used by compile -->
    <mkdir dir="${build}"/>
  </target>

  <target name="compile" depends="init"
      description="compile the source " >
    <!-- Compile the java code from ${src} into ${build} -->
    <javac srcdir="${src}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile"
      description="generate the distribution" >
    <!-- Create the distribution directory -->
    <mkdir dir="${dist}/lib"/>

    <!-- Put everything in ${build} into the MyProject-${DSTAMP}.jar file -->
    <jar jarfile="${dist}/lib/MyProject-${DSTAMP}.jar" basedir="${build}"/>
  </target>

  <target name="clean"
      description="clean up" >
    <!-- Delete the ${build} and ${dist} directory trees -->
    <delete dir="${build}"/>
    <delete dir="${dist}"/>
  </target>
</project>
```

FIN

Dudas:

enviar mail a lista de correo de la asignatura con asunto “DUDA JAVA”.

daw-alumnos@laurel.datsi.fi.upm.es