

Sistemas Informáticos II

Introducción al Desarrollo de Aplicaciones Web Mediante J2EE

Antonio E. Martínez (Antonio.Martinez@ii.uam.es)

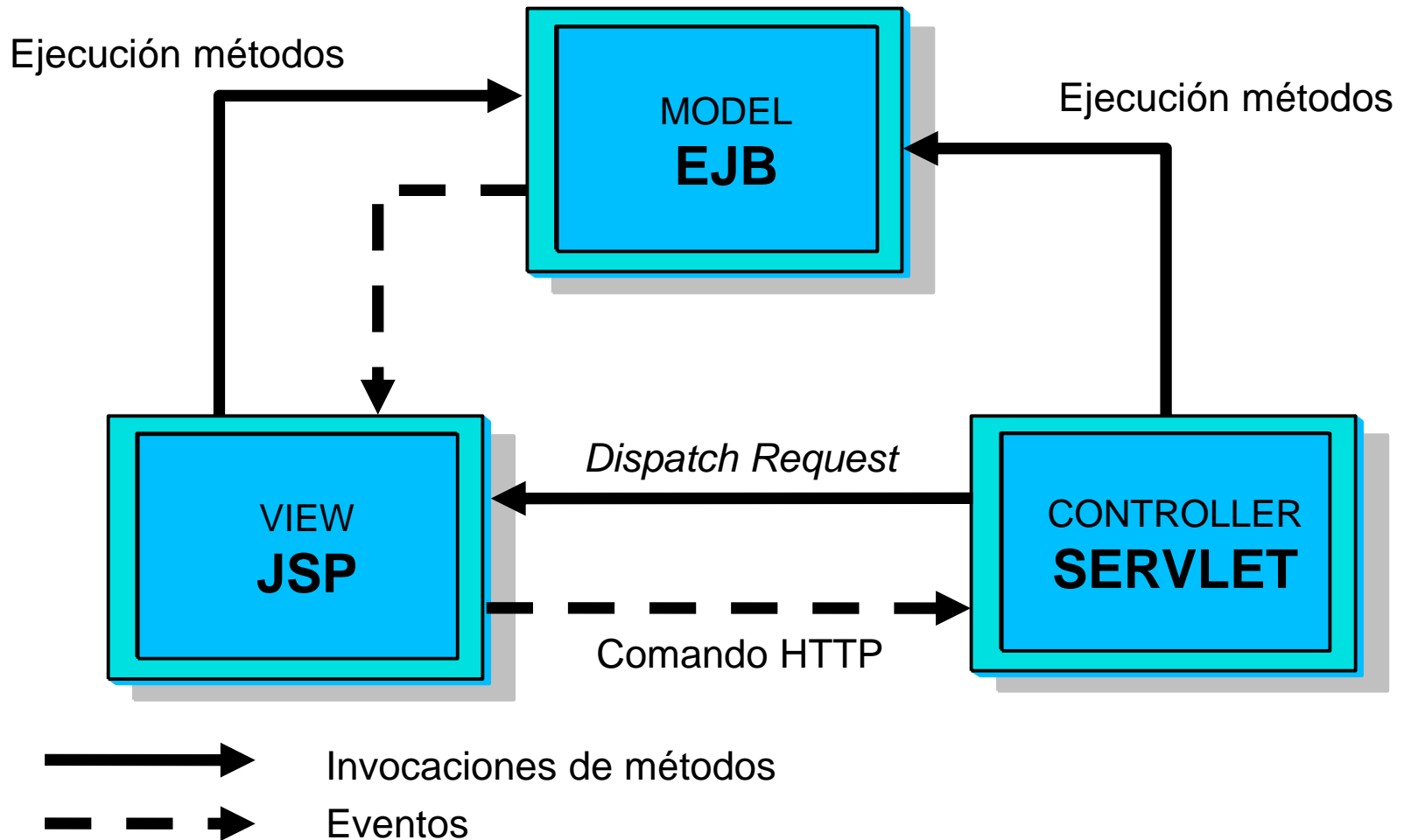
Desarrolladas según el Model-View-Controller.

- **Modelo:**
 - Representa los datos de la aplicación y las funciones de negocio.
 - Los encapsula para hacer transparente su manejo al resto de la aplicación.
 - Permite tener la funcionalidad de la aplicación independiente, y en cualquier tipo de soporte, aunque sean sistemas heredados.
- **Vista:**
 - Realiza la presentación de los datos del modelo.
 - Accede al modelo y adapta los datos presentados al cliente final.
 - Permite múltiples vistas dependiendo del medio de salida de los datos, sin necesidad de cambiar la lógica de la aplicación ni la navegación de la misma.
- **Controlador:**
 - Recibe las interacciones del usuario y las traslada en acciones sobre el modelo.
 - Decide la salida que es necesario presentar al usuario, solicitándolo a la Vista.
 - Permite realizar la navegación que se necesite para el usuario sin necesidad de alterar la vista ni el modelo.

Componentes Java para servidor

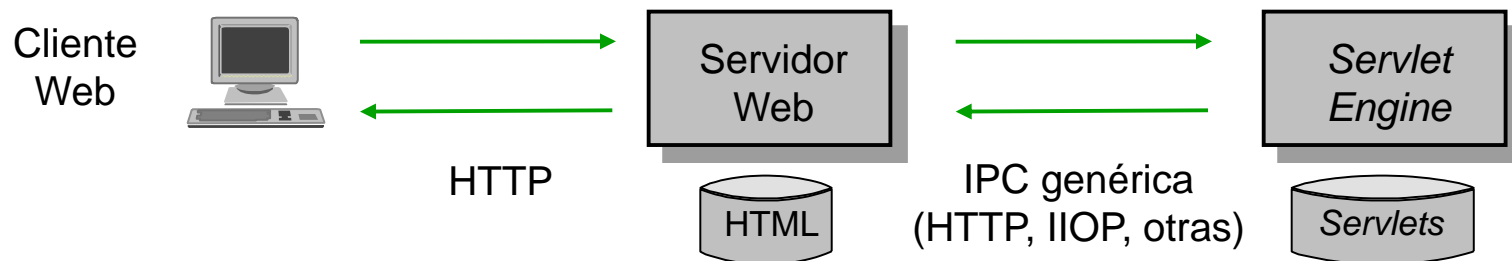
- Simplifican la programación de aplicaciones en el modelo web de Java.
- Definen una interfaz con los elementos de comunicación establecidos en el protocolo HTTP y formularios HTML.
- Tres tipos:
 - Programas en el servidor: *Servlets*. (*controlador*).
 - Páginas dinámicas: *Java Server Pages, JSP* (*vista*).
 - *Enterprise Java Beans, EJB* (*modelo, se usarán en la p1b*).
- Definidos en la *Java 2 Enterprise Edition, J2EE*.

Model-View-Controller. Implementación en J2EE



Invocación básica de un *Servlet*

- Un *Servlet* es una aplicación Java que se ejecuta en el servidor Web para extender sus capacidades.
- El cliente realiza una petición al servidor web, especificando el nombre del *servlet* como parte de la URL.
- El servidor web pasa la petición a la *servlet engine* o *servlet container*, que localiza una instancia de la clase del *servlet*, y crea una nueva tarea.
- La *servlet engine* ejecuta el método `service` del *servlet*.



- La interacción del *servlet* con el sistema se realiza a través de servicios estándar de la *servlet engine*.

Http Servlet

- *Servlet* específico para llamadas HTTP.
- Define dos nuevos métodos:
 - `doGet()` : Atiende a las peticiones HTTP GET.
 - `doPost()` : Atiende a las peticiones HTTP POST.
- Tienen dos parámetros:
- `HttpServletRequest`: Recibe los parámetros de la petición.
 - `Content-type`, `length`, `method`, `URL`, `path`...
- `HttpServletResponse`: Permite enviar la respuesta al cliente.
 - `Set Content-type`, `redirect`.
- Sus subclases deben redefinir estos métodos.
 - También pueden redefinir los métodos `init()` y `destroy()`.

HttpServletRequest - Ejemplo

```
public class logon extends HttpServlet {
    public void doPost(HttpServletRequest req,
                       HttpServletResponse res)
        throws ServletException, IOException
    {
        // Código previo, iniciaciones, etc...

        Enumeration enum = req.getParameterNames();
        while (enum.hasMoreElements()) {
            String name = (String) enum.nextElement();
            String value = req.getParameter(name);
            // Operar con el par parámetro-valor obtenido
        }
        // Resto código de ejecución de la solicitud.
    }
}
```

Gestión de la sesión

Los *servlets* gestionan el estado de una conexión de un usuario a través del **objeto *HttpSession***:

- Representa una conexión cliente / servidor.
- La vida de la sesión se alarga a través de múltiples invocaciones del cliente, **incluso a distintos *servlets***.
- Se identifican a través de la consulta mediante un identificador de sesión.
- El *servlet* accede al objeto *HttpSession* mediante el método `getSession(boolean create)` de la clase *HttpServletRequest*.
 - Si el parámetro `create` es *true* y no existe una sesión, se crea una nueva.
- Las sesiones almacenan información específica de la aplicación como parejas <"clave", objeto>, a través de los métodos:
 - `void setAttribute(String nombre, Object valor)`
 - `Object getAttribute(String nombre)`
- Los objetos que se almacenen en la sesión **deben ser serializables**.
- La *servlet engine* se encarga de establecer el método de identificarlas.

Ejemplo de uso de la sesión

Recordando la URL del ejemplo del formulario HTML

`http://host/cgi-bin/logon?usuario=Superman&clave=loislane&nuevo=si`

El método `doGet` del servlet `logon` podría hacer:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
{
    HttpSession session = req.getSession(true);
    String nombre = req.getParameter("usuario");
    session.setAttribute("usuario", nombre);
    ...
}
```

Otro servlet posterior podría recuperar esta información:

```
public void doGet(HttpServletRequest req, HttpServletResponse res)
{
    HttpSession session = req.getSession(false);
    String nombre = (String) session.getAttribute("usuario");
    ...
}
```

Servlet Context

Dentro de una *servlet engine* se almacena información asociada al servidor y a los *servlets* o a determinados grupos de *servlets* que ejecuta.

Los *servlets* acceden a ella a través de la clase `ServletContext` que contiene:

- Parámetros de iniciación.
- Atributos.
- Se accede a él a través del método `getServletContext` de la clase `GenericServlet`:

```
ServletContext contexto = getServletContext();
```
- El *servlet* puede almacenar y recuperar información en él a través de los métodos:
 - `setAttribute(String name, Object value)`
 - `Object getAttribute(String name)`

Request Dispatcher

- Interfaz que permite a un *servlet* realizar consultas HTTP.
 - Redirigiendo la consulta a una nueva URL (*forwarding*)
 - Ejecutando la consulta y recuperando el control tras su ejecución (*include*).
- Se adquiere a través del contexto, mediante el método `getRequestDispatcher("path");`
- Tiene dos métodos básicos, para realizar los dos tipos de llamadas descritos, que reciben como parámetros los mismos objetos que el *servlet* origen:
 - `forward(HttpServletRequest req, HttpServletResponse res)`
 - `include(HttpServletRequest req, HttpServletResponse res)`
- El método `forward` modifica el objeto *HttpServletRequest* para reflejar el nuevo *servlet* como destino de la petición. En el método `include` no se alteran los objetos.

Java Server Pages, JSP

- Tecnología Java de *server side scripts*.
- Un archivo JSP contiene:
 - Código HTML nativo (*template data*).
 - Elementos de JSP
 - Directivas estándar.
 - Acciones estándar.
 - Elementos de lenguaje (*scripting elements*).
 - Mecanismos de extensión de etiquetas.
- Se ejecutan en una *servlet engine* (*jsp engine*):
 - **Convertidas en un *servlet* dinámicamente** la primera vez que se ejecutan.
 - Se crea un objeto que implementa la interfaz *HttpJspPage*.

Ejemplo de JSP

```
<%@ page import="java.util.*" %>
<%@ page import="es.uam.eps.misclases.*" %>
<html><body>
<table>
<% Vector listaProductos = empresa.getProductos();    for(int i
    = 0; i<productos.getNumProductos(); i++) {
        Producto item = (Producto)listaProductos.elementAt(i);
    %>
<tr>
<td>Producto:
<%= item.nombreProducto() %>
</td>
<td>Precio:
<%= item.precioProducto() %>
</td>
</tr>
<% } // Fin del bucle de productos %>
</table></body></html>
```

Enterprise Java Beans, EJB

- Componentes reutilizables en el servidor.
 - Extensibles y reutilizables. Se pueden crear nuevos EJBs a partir de los existentes.
- Se ejecutan en un contenedor.
 - El contenedor proporciona servicios del sistema a los EJBs, como seguridad, transacción...
- Accesibles de modo local o remoto.
 - Localización transparente para el programa cliente.
- Dos tipos:
 - *Session Beans*.
 - *Message-Driven Beans*.

Session EJB

- Representa una operación o conjunto de operaciones de un cliente dentro del servidor.
 - Oculta la complejidad de la operación al cliente.
 - No compartido por varios clientes. Único por sesión.
 - No persistente.
- Dos tipos:
 - *Stateless Session*: Para servicios iterativos. Múltiples clientes pueden compartir **una misma instancia secuencialmente**.
 - *Stateful Session*: Mantienen una instancia por cliente, para almacenar los valores asociados a la sesión en curso.
 - *Singleton*: Una única instancia compartida por todos los clientes. Se instancia una clase por aplicación.

Message-Driven EJB

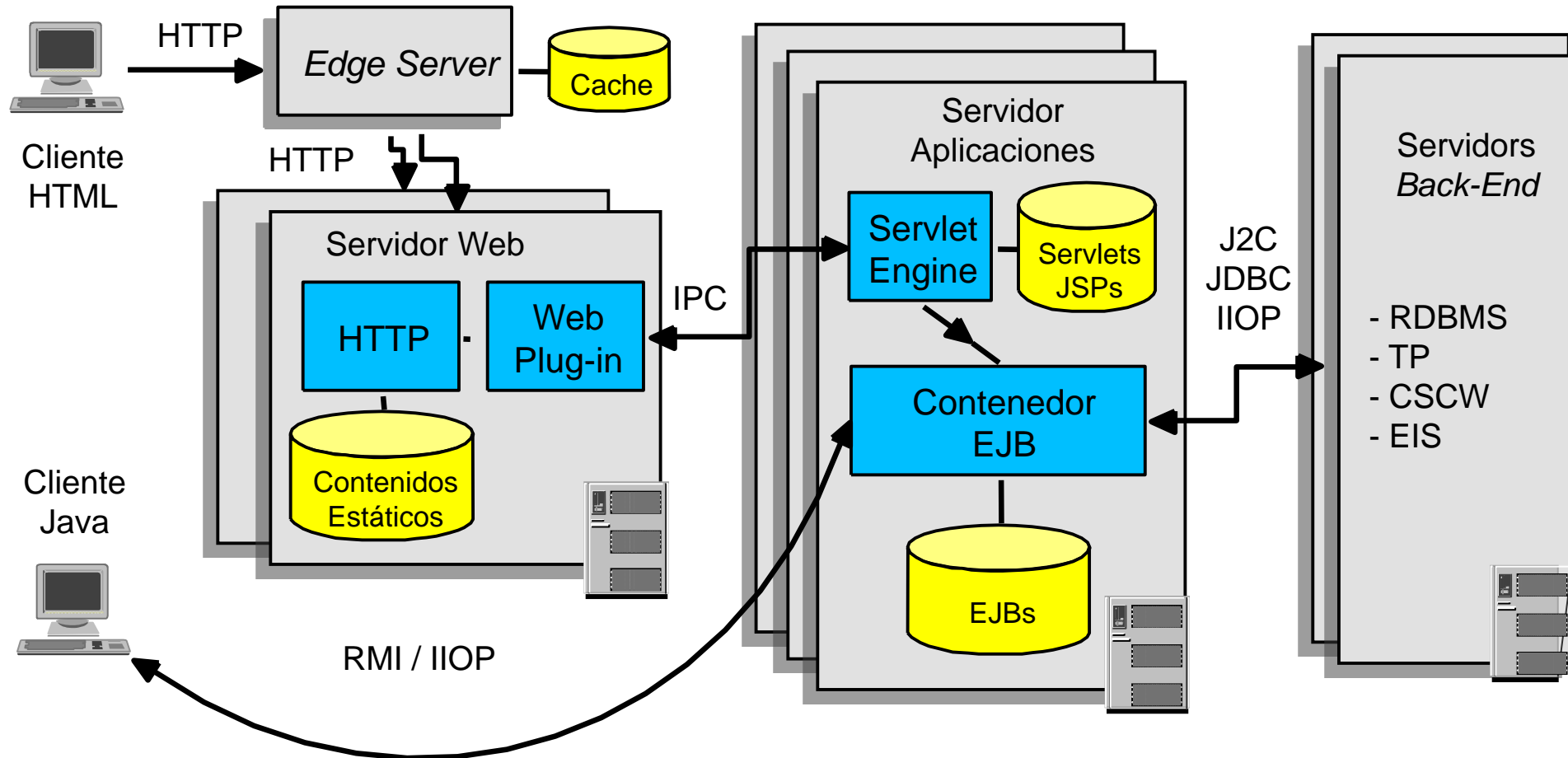
- Permiten procesar mensajes de forma asíncrona.
 - Actúa como un *Message Listener* conforme a las especificaciones del *Java Message Service, JMS*. Comunicación mediante colas de mensajes (MOM).
 - Acepta mensajes JMS enviados por cualquier cliente del MOM.
- Características:
 - Los clientes no acceden al EJB mediante una interfaz, sino por mensajes.
 - Sistema débilmente acoplado, como todo MOM.
 - Similar a un *Stateless Session EJB*:
 - No retiene los datos ni mantiene estado de la conversación con los clientes.
 - Todas las instancias de un *Message-Driven EJB* son idénticas.
 - Un *Message-Driven EJB* puede procesar mensajes de diversos clientes.
- El contenedor, al recibir un mensaje, ejecuta el método `onMessage` de la instancia del EJB elegida.
- Los mensajes se pueden enviar en el contexto de una transacción.

Servidores de aplicaciones J2EE

Aplicación que contiene:

- Entornos de ejecución para *servlets*, JSPs y EJBs.
- Servicios generales de seguridad para todos los elementos.
- Capacidad de replicación y distribución de aplicaciones entre múltiples servidores (*Application Server Clustering*).
- Herramientas centralizadas de configuración, administración y control.

Estructura general (I)



JDBC - Introducción

- La clave del éxito de las aplicaciones Web, reside en la consulta de BBDD por parte del aplicativo software, que hace de intermediario entre el cliente y el sistema que almacena la información.



JDBC - Introducción

- Conjunto de clases e interfaces Java que permiten la manipulación de sentencias SQL de una BBDD.
- Cada fabricante de BBDD se encargará de proporcionar un driver JDBC específico para su BBDD.
- Las operaciones básicas que vamos a realizar en un JDBC son:
 - Conectarnos a una BBDD.
 - Enviar Querys y Updates a la BBDD.
 - Recuperar y procesar los resultados obtenidos de la BBDD.
- El producto JDBC incluye 4 componentes:

1

El API de
JDBC

3

JDBC Test
Suite

2

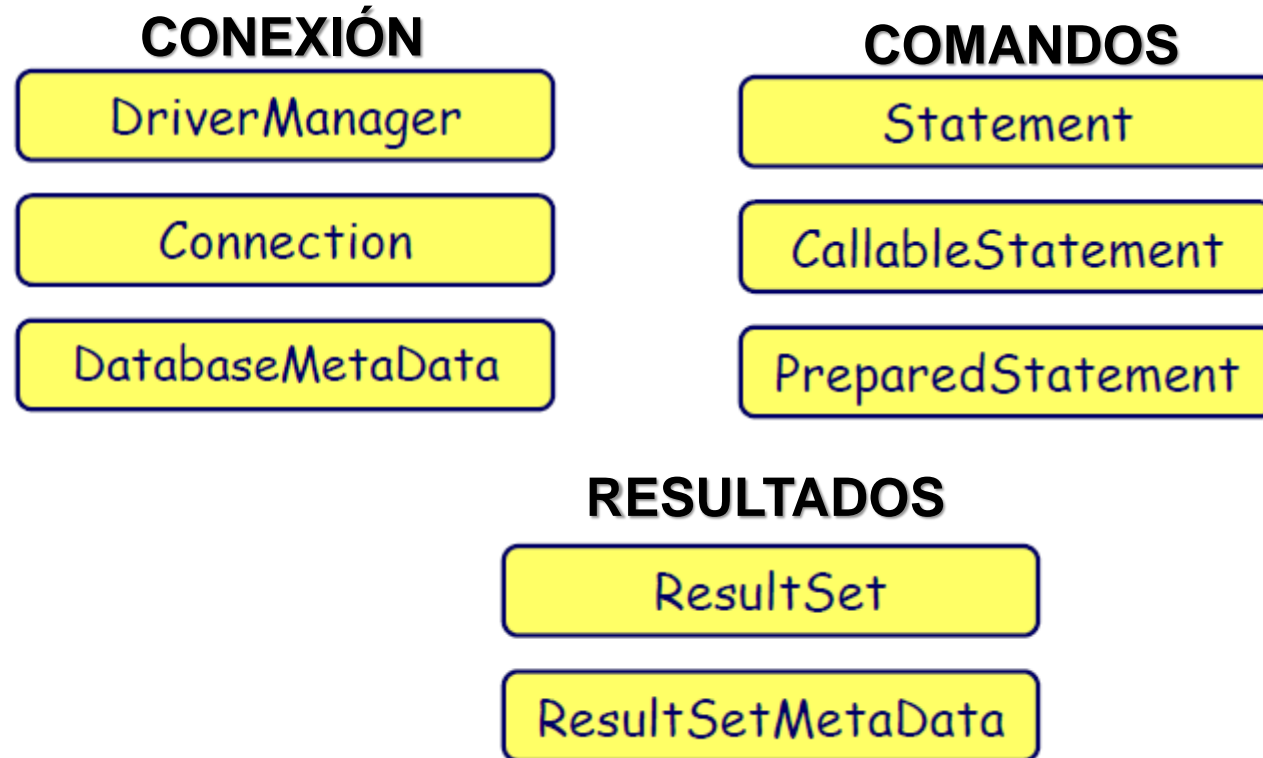
JDBC Driver
Manager

4

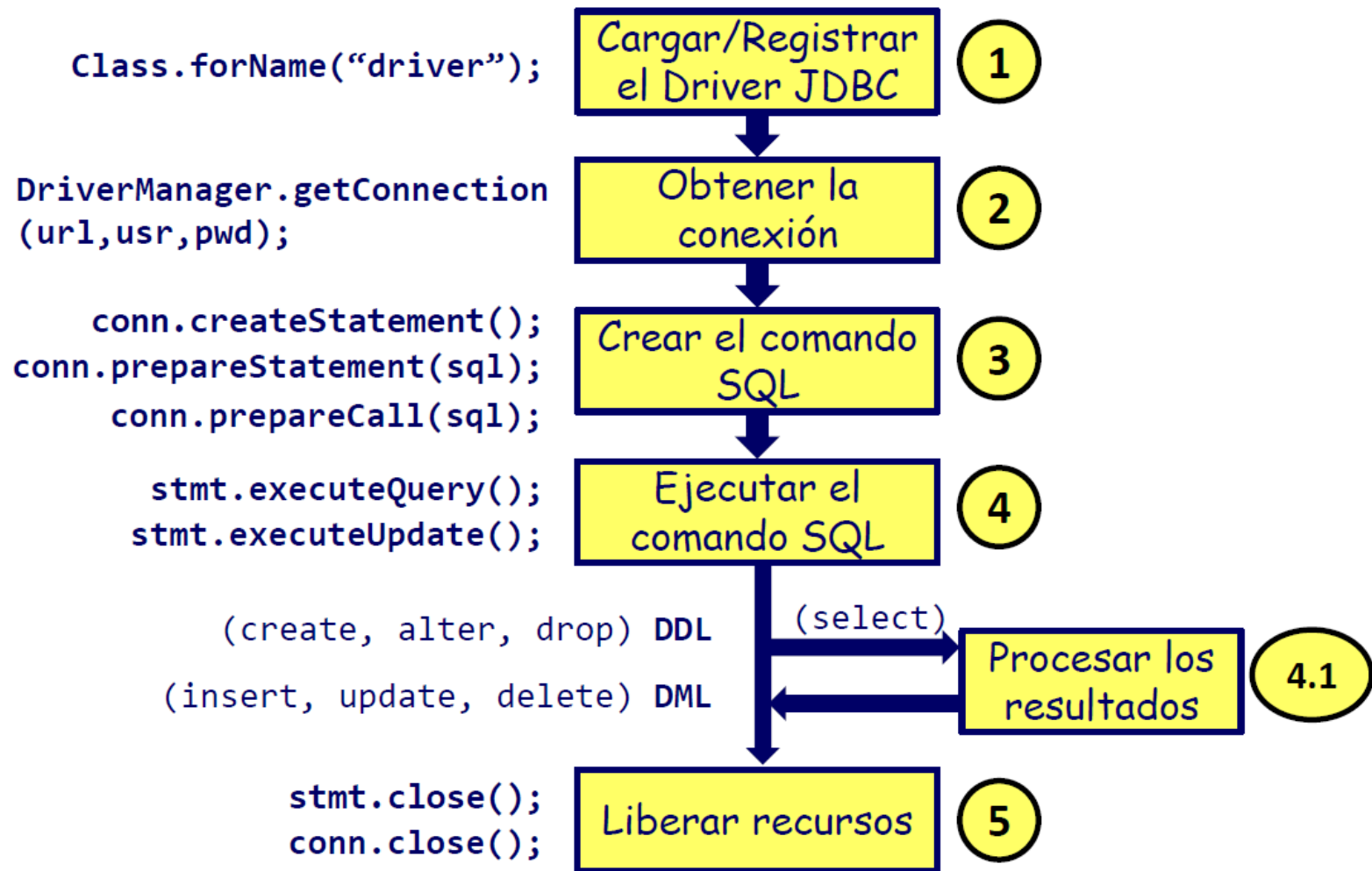
Puente
JDBC-ODBC

JDBC - Introducción

- En la librería java.sql se encuentran las interfaces principales:



JDBC – Pasos a utilizar JDBC en apps Java



Servidor de aplicaciones Glassfish - Mandatos

```
#  
# ARRANQUE y PARADA del servidor de aplicaciones  
#  
# Para establecer las rutas  
export J2EE_HOME=/usr/local/glassfish-4.0/glassfish  
export PATH=/usr/local/glassfish-4.0/bin:/usr/java/jdk/bin:$PATH  
# Para evitar teclear usuario y contraseña  
export AS_ADMIN_USER=admin  
export AS_ADMIN_PASSWORDFILE=passwordfile  
# Arranque de servidor de aplicaciones  
asadmin start-domain domain1  
# ... para que ant pueda ejecutar mandatos remotos (requiere restart)  
asadmin enable-secure-admin  
# ... rearranque de servidor de aplicaciones  
asadmin restart-domain domain1  
# Parada de servidor de aplicaciones  
asadmin stop-domain domain1
```

Servidor de aplicaciones Glassfish - Mandatos

```
#
# DESPLEGAR y REPLEGAR una aplicación
#
# ... desplegar
asadmin --host 10.1.1.1 --port 4848 deploy --name P1 --target server dist/P1.war
# ... replegar
asadmin --host 10.1.1.1 --port 4848 undeploy --target server P1
#
# OTROS mandatos del servidor de aplicaciones
#
# Lista de mandatos del servidor de aplicaciones
asadmin list-commands
# Ayuda de un mandato (e.g, get)
asadmin help get
# Muestra atributos del servidor de aplicaciones
asadmin get '*' | more
# Establece un atributo
asadmin set configs.config.server-config.monitoring-service.module-monitoring-
    levels.jvm=HIGH
# Revisión de logs (en VM's)
cat /opt/glassfish4/glassfish/domains/domain1/logs/server.log
```


Servidor de aplicaciones Glassfish – Consola Web

The screenshot displays the Glassfish Web Console interface. The browser address bar shows the URL `https://localhost:4848/common/index.jsf`, which is circled in red and labeled "URL de consola de administración". The page header includes "Home", "About...", "User: admin", "Role: domain1", and "Server: localhost". The main title is "GlassFish™ Server Open Source Edition".

The left sidebar contains a tree view of the server structure. The "server (Admin Server)" node is circled in red and labeled "Administración de servidor". Under it, the "P1" application is circled in red and labeled "Aplicación desplegada". In the "Resources" section, the "jdbc/VisaDB" resource is circled in red and labeled "Recurso JDBC de acceso a DB". Under "JDBC Connection Pools", the "VisaPool" is circled in red and labeled "Pool de conexiones a DB".

The main content area shows the "General Information" tab. It includes buttons for "Stop", "Restart", "View Log Files" (circled in red and labeled "Revisión de logs"), "View Raw Log", "Rotate Log", "Recover Transaction...", and "Secure Administration...". Below these buttons, a table of server properties is displayed:

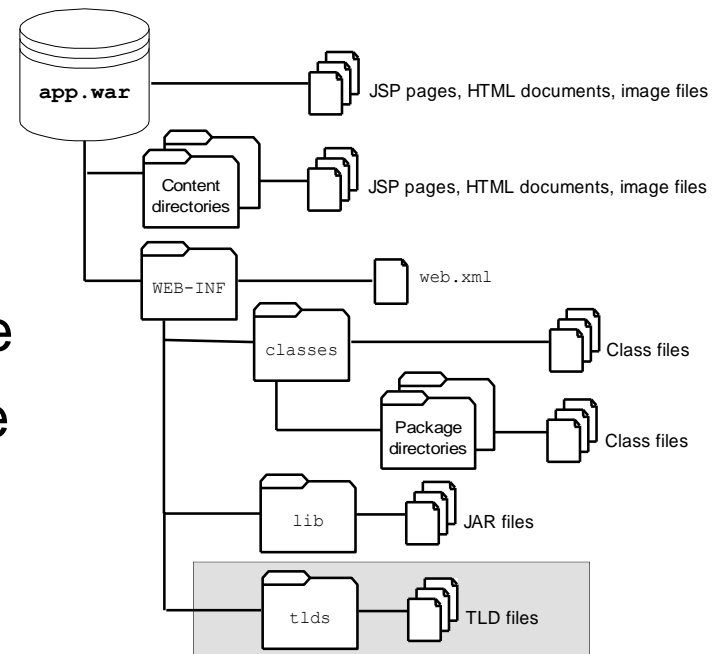
Name:	server
Status:	Running
JVM:	JVM Report
Configuration:	server-config
Installation Directory:	/usr/local/glassfish-4.1/glassfish
Installed Version:	GlassFish Server Open Source Edition 4.1 (build 13)
Secure Administration:	Enabled
Default Up Time:	Not Enabled
HTTP Port(s):	4848,8080,8181
IIOP Port(s):	3820,3920,3700

Aplicaciones Web J2EE

- Una aplicación web está organizada en una estructura jerárquica de directorios con dos partes
 - Un directorio privado WEB-INF que contiene los recursos que no son descargables para el cliente
 - Un directorio público que contiene los recursos públicos
 - Ejemplo: miaplicación\
 - Index.html
 - login.jsp
 - images\ logo.gif
 - doc\ tutorial.pdf
 - WEB-INF\
 - web.xml (Deployment Descriptor)
 - classes\ ServletCompras.class
 - lib\ cualquierOtraApi.jar
- Una aplicación web puede ser empaquetada en un fichero WAR.

Archivos WAR

- Web Application aRchive
- Permiten empaquetar en una sola unidad aplicaciones web java completas
- Simplifican el despliegue de Aplicaciones Web
 - Facilidad de Instalación
 - Un solo fichero para cada servidor en un cluster
 - Seguridad: No permite el acceso entre Aplicaciones Web distintas
- Contiene el descriptor de despliegue WEB-INF/web.xml donde se dan de alta:
 - Servlets
 - Parámetros de Contexto
 - Otros: TLD,s, Filtros, etc



Ant

- Herramienta para automatizar procesos similar a make pero implementada utilizando java
- Los ficheros de especificación son XML (build.xml)
- Cada fichero build.xml contiene un project, propiedades (macros en makefile) y al menos un target.
- Los targets están compuestos de tasks y pueden depender de otros targets. Normalmente se define un target por defecto.
- Una Task representa una acción que necesita ejecutarse:
 - Crear directorio
 - Compilar código fuente
 - Crear archivo war, etc.

Ant

```
<project name="Sample Project" default="compile" basedir=". ">
  <description>
    A sample build file for this project
  </description>

  <!-- global properties for this build file -->
  <property name="source.dir" location="src"/>
  <property name="build.dir" location="bin"/>
  <property name="doc.dir" location="doc"/>

  <!-- set up some directories used by this project -->
  <target name="init" description="setup project directories">
    <mkdir dir="${build.dir}"/>
    <mkdir dir="${doc.dir}"/>
  </target>

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

Ant

```
<!-- Generate javadocs for current project into ${doc.dir} -->  
<target name="doc" depends="init" description="generate documentation">  
    <javadoc sourcepath="${source.dir}" destdir="${doc.dir}"/>  
</target>
```

```
<!-- Delete the build & doc directories and Emacs backup (*~) files -->  
<target name="clean" description="tidy up the workspace">  
    <delete dir="${build.dir}"/>  
    <delete dir="${doc.dir}"/>  
</target>
```

```
</project>
```

- Para ejecutar ant, situarse en el directorio donde se encuentra el fichero build.xml y ejecutar:
 - Ant (para target por defecto)
 - Ant init (para ejecutar target init)
 - Ant init compile doc clean (para ejecuta varrios targets)