

Implementación

- Implementación en el PUD
- Arquitectura de tres capas
- Capa de presentación
- Capa de gestión de datos
- SI OO distribuidos

Iteración en PUD

- **Planificación de la Iteración**
 - **Captura de requisitos:**
 - Modelo de casos de uso, Modelo de Dominio, ...
 - **Análisis:**
 - Diagrama de secuencia del sistema, Contratos, Modelo Conceptual...
 - **Diseño:**
 - Diagramas de interacción, Diagrama de Clases
 - **Implementación:**
 - codificación (Clases y métodos)
 - **Pruebas:**
 - verificación de la implementación
- **Evaluación de la iteración**

Fases y entregas del Proceso Unificado de Desarrollo

- captura de requerimientos: qué SI debemos construir?
 - Modelo de casos de uso, Modelo de Dominio, ...
- análisis: qué debe hacer el SI?
 - Diagramas de secuencia del sistema, Contratos, ...
- diseño: cómo lo debe hacer el SI?
 - Diagramas de interacción, Diagrama de Clases, ...
- codificación:
 - Código Fuente (clases y métodos)
- pruebas:
 - Especificación de las pruebas de funcionamiento
- mantenimiento:
 - Documentación y revisión de todo lo anterior

Dependiente de
la tecnología

Construcción incremental e iterativa del SI

- **Modelo dinámico del sistema (comportamiento):**
 - Captura de requerimientos: Modelo de Casos de Uso
 - Análisis: Diagramas de secuencia del sistema, contratos
 - Diseño: Diagramas de interacción
- **Modelo estático del sistema (propiedades):**
 - Captura de requerimientos: Modelo de Dominio
 - Análisis: Modelo Conceptual
 - Diseño: Diagrama de clases
- **Implementación: codificación (clases y métodos)**

Implementación

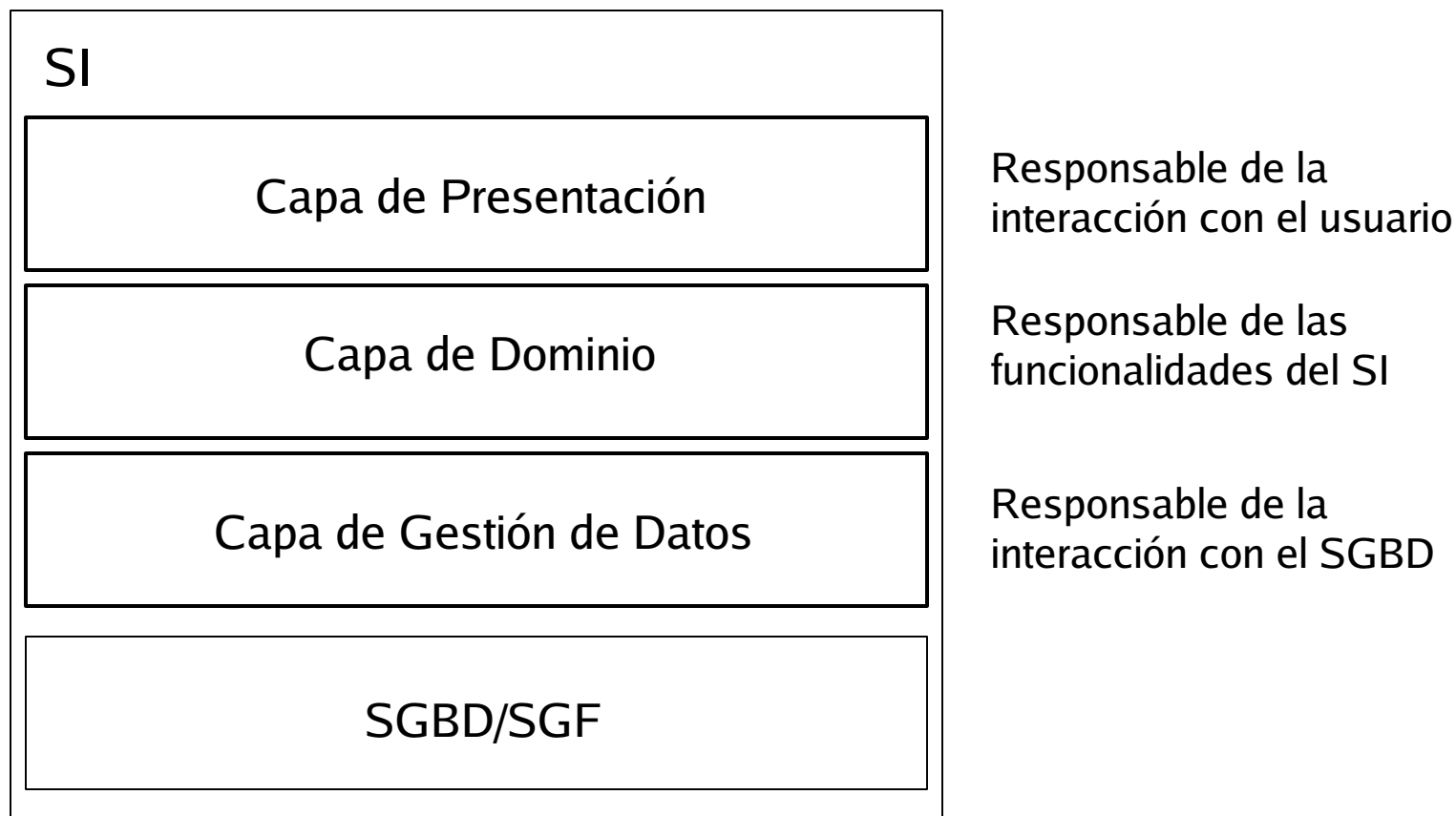
- Codificar el diseño del SI.
- Situación de partida (diseño del SI)
 - Resultado del diseño (cómo lo debe hacer el SI?)
 - Diseño de los datos (Modelo de datos)
 - Diseño de procesos (Modelo de comportamiento)
 - Diseño de la interacción con el usuario (Modelo de la interfaz)
 - Recursos tecnológicos (hardware y software disponible)

Implementación

- Situación final (solución)
 - Resultado de la implementación (cómo lo hace el SI?)
 - Estructura interna del SI (Arquitectura del SI)
 - Implementación de los datos: clases, BDs
 - Implementación de los programas: métodos
 - Implementación de la Interfaz: modelo de casos de uso reales
- Proceso de implementación
 - Codificación
 - Pruebas

 - y herramientas RAD!

Arquitectura de un SI en tres capas (1)



- Arquitectura cambiabile, reusable, portable

Arquitectura de un SI en tres capas (2)

- La capa de presentación
 - AWT y SWING
 - SI basados en la web (CGI, Java applets, servlets, JSPs, AJAX)
- La capa de dominio
 - Java
- La capa de gestión de los datos
 - JDBC
 - Serialización
- SI OO distribuidos
 - RMI
 - CORBA, SOAP
 - OLE, DCOM/COM+ y la plataforma .NET

Capa de Presentación

- Java 2 con JFC/SWING
- Componentes visuales
- Construcción de la interfaz
- Gestión de la interfaz

Java 2 JFC/Swing

- JFC (Java Foundation Class) nos proporciona una serie de herramientas que nos ayudan a construir interfaces de usuario gráficas (GUI)

- Java proporciona clases para conseguir:
 - Programación de interfaces fáciles y rápidas
 - Programación de applets para la web

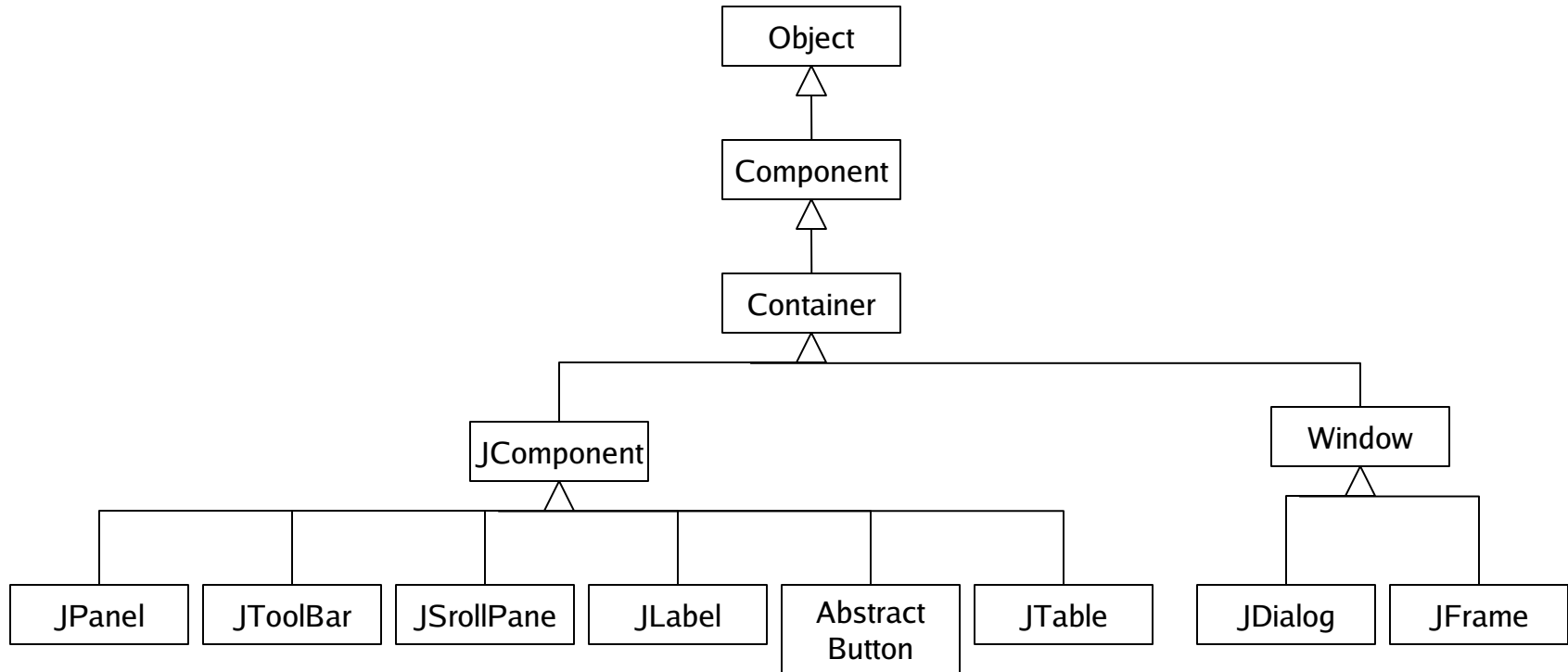
- Swing es el nombre del proyecto que desarrolló los últimos componentes que se añadieron a JFC

- Clases AWT (Abstract Windowing Toolkit)
- Clases SWING: posteriores a AWT, más portables y funcionales

Objetivos

- Entender la jerarquía de clases diseñada en Java que permiten construir interfaces de usuario
- Entender cómo se realiza la gestión de eventos
- No es objetivo aprender los nombres de todas las clases, etc. ya que pueden construirse GUIs usando herramientas visuales (p.e. Jdeveloper, Eclipse Visual Editor)

Componentes visuales (1)



Componentes visuales (2)

- Un contenedor se compone de varios componentes, los cuales pueden ser componentes concretos o pueden ser contenedores.

- Contenedores de alto nivel (Window):
 - JFrame: ventana con marco, título, botones, etc.
 - Único contenedor al que se le pueden insertar menús
 - Incluye automáticamente un JPanel
 - JDialog: ventana más limitada que la anterior con un pequeño texto.

- Contenedores de propósito general (JComponent): incluidos en los anteriores
 - JPanel: contenedor para añadir más componentes
 - JScrollPane: realiza una vista (scroll) sobre un componente
 - JToolBar: agrupa diversos componentes en una fila o columna

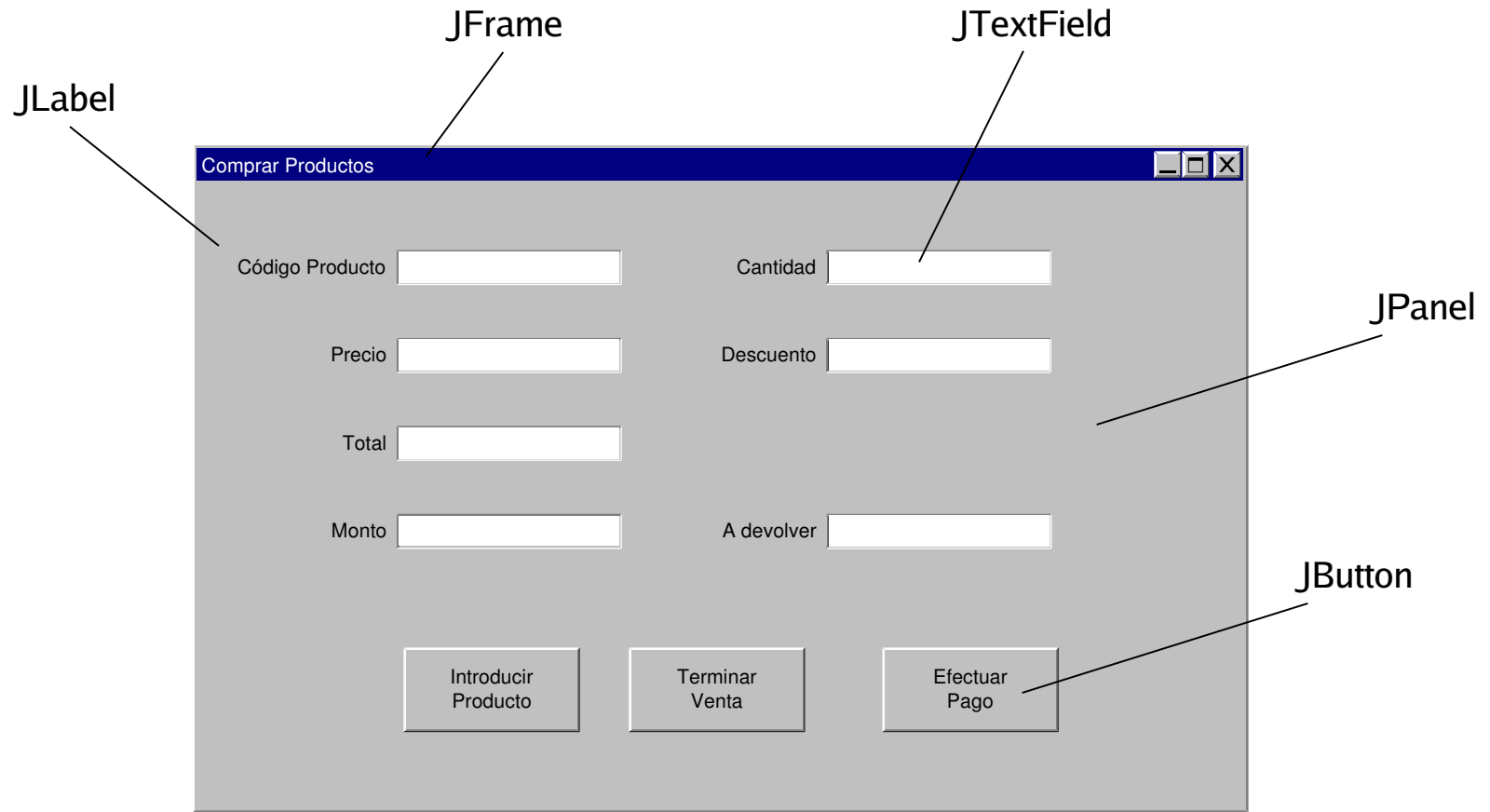
Componentes visuales (3)

- Componentes básicos (JComponent): permiten obtener información del usuario
 - JButton: botón
 - JComboBox, JList, JMenu: menús desplegable de elementos
 - JCheckBox, JRadioButton: activar/desactivar opciones
 - JTextField: permite al usuario introducir un texto
- Componentes con información no editable (JComponent):
 - JLabel: muestra un texto o una imagen no seleccionable
- Componentes con información editable (JComponent):
 - JTable: muestra tablas de datos, opcionalmente editables
 - JTextComponent: muestra texto, opcionalmente editable

Construcción de la GUI

- Escoger los contenedores de alto nivel (el más usual es JFrame)
- Escoger los contenedores de propósito general, si son necesarios
- Escoger el resto de componentes de la interfaz (JButton, JLabel,...)
- Añadir los componentes a los contenedores y mostrar las interfaces, mediante operadores de añadir y mostrar que proporcionan las clases JFrame, ...

Ejemplo GUI



Gestión de la Interfaz

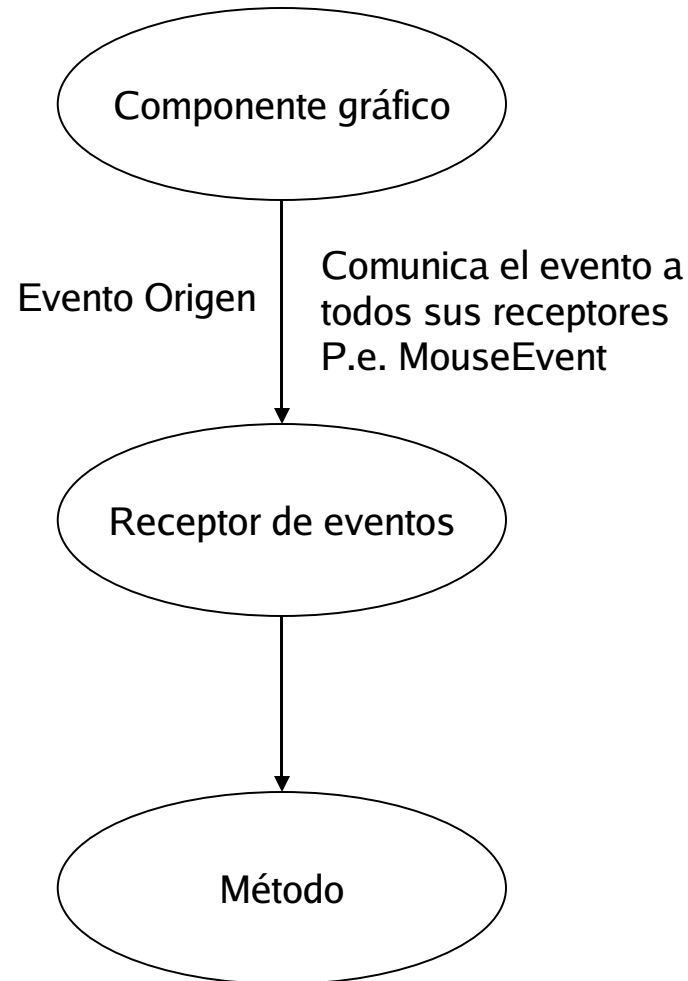
- Al diseñar una interfaz gráfica hay que tener en cuenta que se producirán ciertos eventos del usuario
 - Un evento es un suceso generado por una acción del usuario que afecta a algún componente de la interfaz
 - Por ejemplo: pulsar una tecla, mover el ratón, cambiar el formato de la ventana, cerrar una ventana, pulsar un botón, etc.
- La implementación deberá contemplar una serie de acciones de respuesta para procesar los eventos del usuario

Modelo de eventos

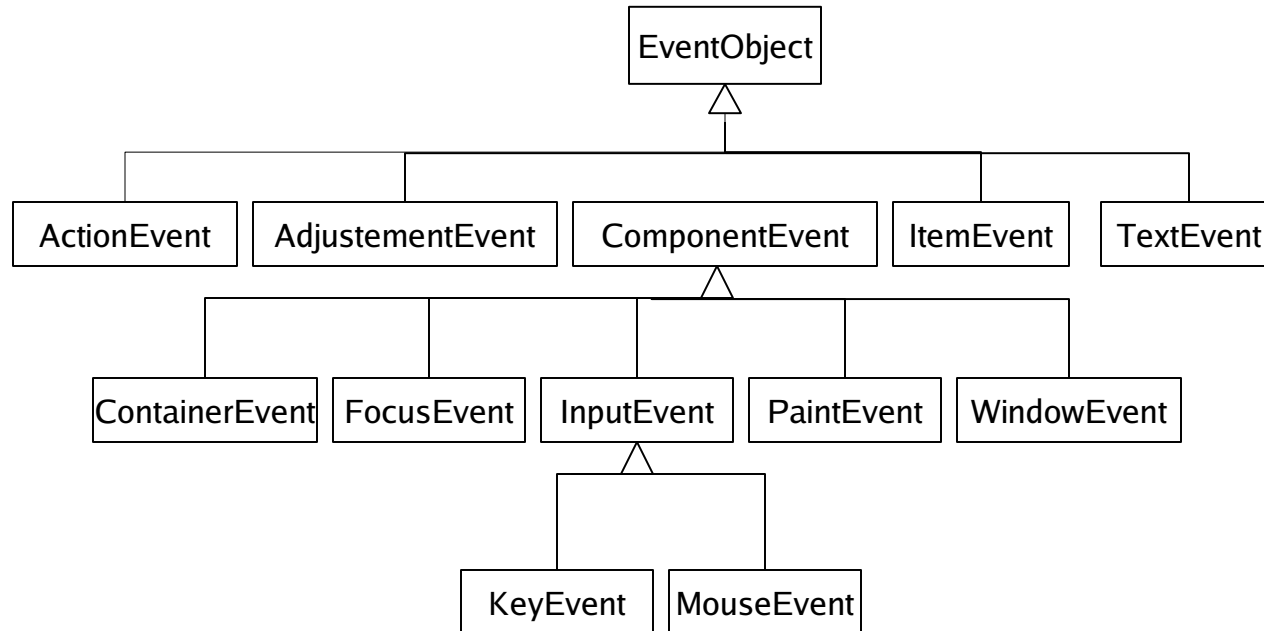
Sucede un evento que afecta a objetos
P.e. JButton, JFrame

Se crea un objeto evento y se pasa el control al objeto oyente
P.e. ActionListener
Para ser receptor de un evento origen debe registrarse al componente gráfico que lo genera con:
Add<EventType>Listener(receptor)

Método para responder al evento
P.e. mouseClicked(MouseEvent)



Modelo de eventos



Modelo de eventos

- Eventos de bajo nivel
 - Relacionados con los aspectos físicos de la GUI.
 - Por ejemplo: pulsar una tecla, mover un ratón, ...

 - `ComponentEvent`: mover, modificar la medida de un componente
 - `ContainerEvent`: añadir o borrar componentes de un contenedor
 - `FocusEvent`: un componente recibe/pierde el foco
 - `KeyEvent`: pulsar/soltar una tecla
 - `MouseEvent`: pulsar/solar botones ratón, desplazar el ratón
 - `WindowEvent`: cerrar, activar, minimizar, ... ventanas

Modelo de eventos

- Eventos de alto nivel
 - Relacionados con la semántica del componente y generalmente combinaciones de eventos de bajo nivel
 - Por ejemplo: pulsar un botón, cambiar el texto de un campo, seleccionar un elemento de un menú desplegable, ...

 - ActionEvent: pulsar botón, seleccionar menú, pulsar ENTER
 - AdjustmentEvent: mover la barra de desplazamiento
 - ItemEvent: seleccionar entre una lista de opciones
 - TextEvent: introducir texto

Modelo de eventos

Componente Gráfico	Evento Origen	Receptor de eventos	Métodos
Jbutton, JTextField...	ActionEvent	ActionListener	actionPerformed(ActionEvent)
Componentes	ComponentEvent	ComponentListener	componentHidden(ComponentEvent) componentMoved(ComponentEvent) componentResized(ComponentEvent) componentShown(ComponentEvent)
Componentes	FocusEvent	FocusListener	focusGained(FocusEvent) focusLost(FocusEvent)
Componentes	KeyEvent	KeyListener	keyPressed(KeyEvent) keyReleased(KeyEvent) keyTyped(KeyEvent)

Modelo de eventos

Componente Gráfico	Evento Origen	Receptor de eventos	Métodos
Componentes	MouseEvent	MouseListener	MouseClicked(MouseEvent) MouseEntered(MouseEvent) MouseExited(MouseEvent) MousePressed(MouseEvent) MouseReleased(MouseEvent) MouseDragged(MouseEvent) MouseMoved(MouseEvent)
		MouseListener	MouseClicked(MouseEvent) MouseEntered(MouseEvent) MouseExited(MouseEvent) MousePressed(MouseEvent) MouseReleased(MouseEvent)
		MouseMotionListener	MouseDragged(MouseEvent) MouseMoved(MouseEvent)

Modelo de eventos

Componente Gráfico	Evento Origen	Receptor de eventos	Métodos
Contenedores	ContainerEvent	ContainerListener	ComponentAdded(ContainerEvent) ComponentRemoved(ContainerEvent)
Windows	WindowEvent	WindowListener	windowActivated(WindowEvent) windowClosed(WindowEvent) windowClosing(WindowEvent) windowDeactivated(WindowEvent) windowDeiconified(WindowEvent) windowIconified(WindowEvent) windowOpened(WindowEvent)

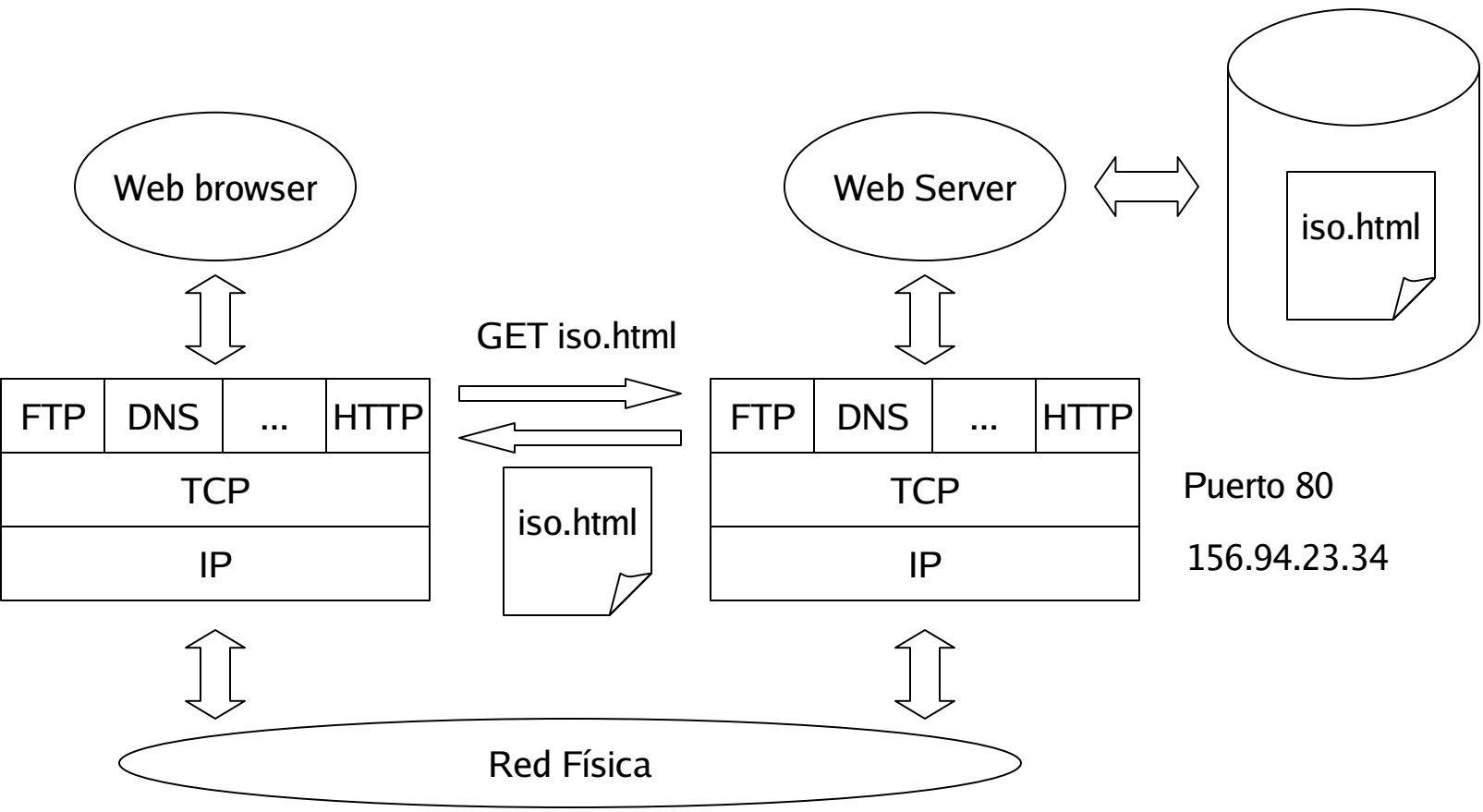
SI Cliente/servidor sobre la web

- World Wide Web
- Aplicaciones estáticas
 - HTML
 - CSS
- Aplicaciones dinámicas web server
 - Formularios HTML
 - CGI – Servlets – jsp
- Aplicaciones dinámicas web browser
 - Java Applets – Ajax

World Wide Web

- El proyecto WWW (web, w3) comenzó el año 1989 en el CERN de la mano de Tim Berners-Lee. El objetivo inicial era desarrollar un “sistema de hipertexto” que permitiera el intercambio eficiente y fácil de información entre los equipos de investigación, dispersos geográficamente.
- WWW opera sobre el protocolo TCP/IP y comprende las siguientes tecnologías:
 - Web servers
 - Web browsers
 - URL (Uniform Resource Locator)
 - HTTP (Hypertext Transfer Protocol)
 - HTML (Hypertext Markup Language)
 - Etc.

World Wide Web



El cliente universal: web browser (WB)

- El WB proporciona una interfaz, normalmente gráfica, para obtener, interpretar, formatear y finalmente presentar documentos –páginas- HTML, con enlaces a otras páginas o documentos, que encapsulan URLs
- Un URL proporciona al WB la información necesaria para obtener un documento de la web
- Otras funcionalidades:
 - Otros protocolos: POP3 y/o IMAP, SMTP, FTP, NNTP
 - Soporte para extensiones no estándar de HTML: Javascript, ...
 - Plugins, ...
 - Java applets, Contoles ActiveX, ...

El servidor: web server (WS)

- El WS es un proceso servidor que “escucha” un puerto TCP, generalmente el 80, esperando a un cliente WB
- Una vez establecida la conexión, el WB envía una petición WS y le devuelve una respuesta, liberando la conexión
- El protocolo que define las peticiones y respuestas legales es HTTP
- Normalmente, los recursos que se solicitan son ficheros accesibles por el WS
- Otras funcionalidades:
 - Control y registro, diario de accesos
 - Protocolos y servicios Internet: FTP, News, etc.
 - Paso de peticiones a otros procesos usando CGI
 - Active Server Pages (ASP) / Servlets

Uniform Resource Locator (URL)

- El URL informa al WB de:
 - En qué máquina está el recurso
 - Qué protocolo se usará para obtener el recurso
 - Cómo localizar el recurso en la máquina servidora
- Protocolo://host[:puerto]/[path]
- Protocolo: http, ftp, file, news, gopher, mailto, telnet, ...

HyperText Transfer Protocol (HTTP)

- HTTP es un protocolo sobre TCP/IP
- Versiones: 0.9, 1.0, 1.1, NG (en proceso)
- Esquema de una sesión HTTP:
 - Se establece una conexión TCP/IP entre el WB y el WS
 - WB realiza una petición al WS
 - WS revuelve una respuesta al WB
 - WB y WS cierran la conexión
- HTTP es un protocolo sin estado: el WS no “recuerda” nada de las sesiones HTTP anteriores!

La petición HTTP

```
<Método><Recurso><VersiónHTTP><CR> // Petición  
(<campo>:<valor><CR>)* // Parámetros  
<CR> //  
[CuerpoMensaje] //
```

Método: GET, POST, HEAD

Ejemplo:

```
GET jiwriclg/iso.html HTTP/1.1  
host: www.sc.ehu.es  
accept: text/html
```


La respuesta HTTP

<VersiónHTTP><CódigoResultado>[explicación]<CR>
(<campo>:<valor><CR>)*
<CR>
[CuerpoMensaje]

CódigoResultado: 200 [OK], 400 [Bad Request], 404 [Not Found]

Ejemplo:

```
HTTP/1.1 200 OK
server: Netscape-enterprise/3.5.1
Date: Mon, 19 Jul 1999, 13:52:18 GMT
Content-Type: text/html
... <HTML> ... </HTML>
```

HyperText Markup Language (HTML)

- HTML lenguaje para la publicación de contenidos hypermedia (texto, multimedia, hyperlinks) en la web
- HTML está basado en SGML (Standard Generalized Markup Language, ISO 8879)
- Versión actual HTML 4.0
- Página HTML: contenido + marcaje
- El marcaje permite definir:
 - Cómo se estructura el contenido
 - Cómo se debe presentar el contenido+estructura
 - Referencias a URLs
 - Objetos “incrustados”: imágenes, audio, video, scripts, applets, ...
 - Las marcas son etiquetas o TAGs:
 - `<TAG (atributo=valor)*> Contenido+marcaje </TAG>`

SI Cliente/Servidor sobre la web

- Aplicaciones estáticas: muestran documentos HTML disponibles en el WS
- Aplicaciones basadas en WS: usan WB para obtener y mostrar información HTML y/o rellenar formularios generados por procesos servidores que interactúan con los WB via los WS: CGI
- Aplicaciones basadas en WB: Java applets que se ejecutan en la MVJ del WB
- Aplicaciones Cliente/Servidor OO: Java applets que son clientes de objetos CORBA

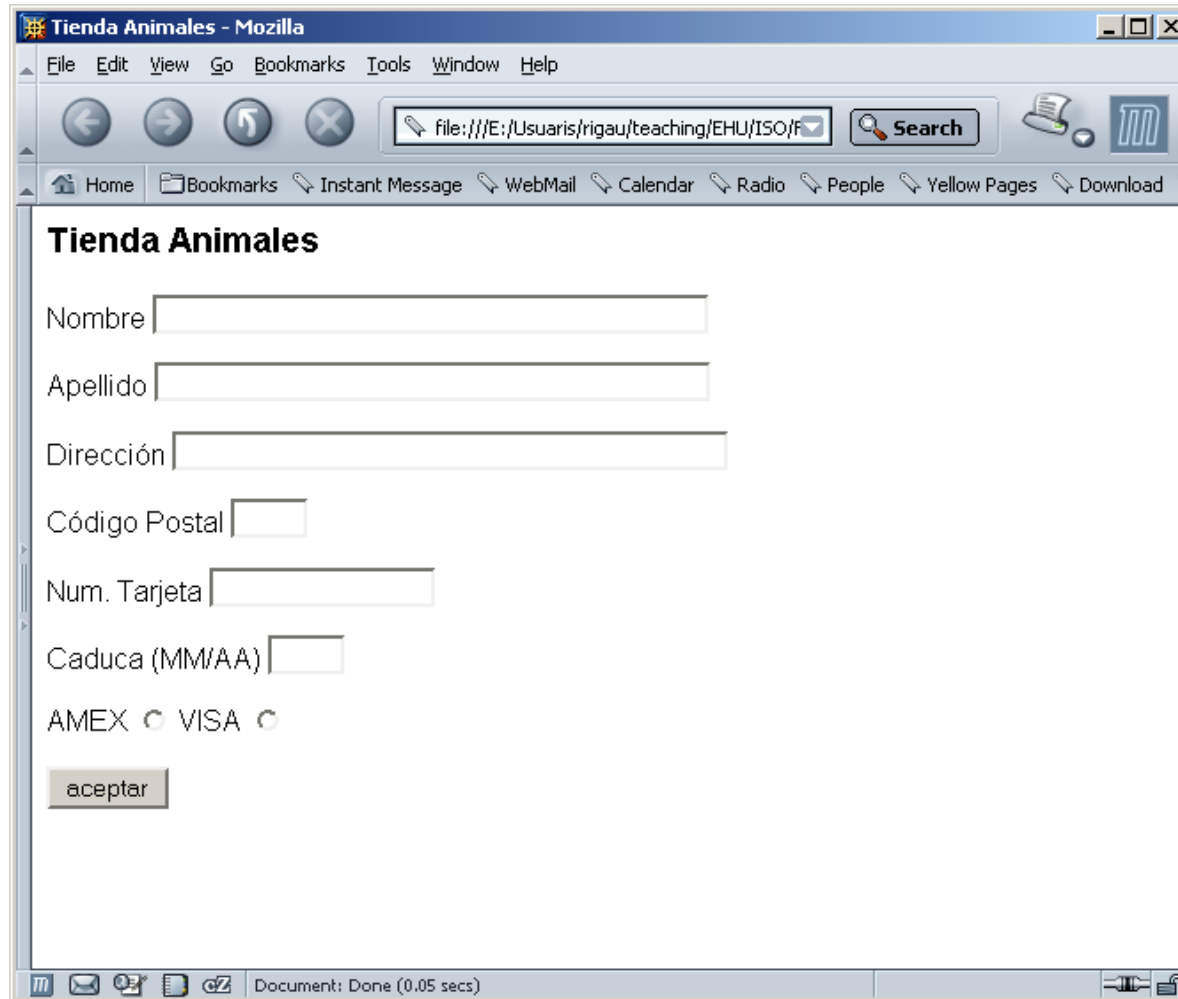
Common Gateway Interface (CGI)

- CGI es un protocolo que define como el WB puede interaccionar mediante el mecanismo HTTP, con programas situados en el WS –llamados programas CGI-
- Los programas CGI pueden ser:
 - Scripts: UNIX, shell, Perl, phyton, ...
 - Ejecutables: C, C++, Ada, ...
- Suelen estar accesibles en un directorio especial en el WS:
 - <http://www.sc.ehu.es/cgi-bin/alumnos.pl>
- CGI, al igual que HTTP, es un protocolo sin estado

Formularios HTML

```
<html><head><title>Tienda Animales</title></head>
<body><h3>Tienda Animales</h3>
<form action='http://www.tienda.es/cgi-bin/tienda.pl' method=post>
Nombre <input name='nombre' size=50><p>
Apellido <input name='apellido' size=50><p>
Dirección <input name='direccion' size=50><p>
Código Postal <input name='cp' size=5><p>
Num. Tarjeta <input name='nt' size=19><p>
Caduca (MM/AA) <input name='ft' size=5><p>
AMEX <input name='cc' type=radio value='amex'>
VISA <input name='cc' type=radio value='visa'><p>
<input type=submit value='aceptar'>
</form>
</body>
</html>
```

Formularios HTML



The image shows a screenshot of a Mozilla browser window. The title bar reads "Tienda Animales - Mozilla". The address bar contains the file path "file:///E:/Usuarios/rigau/teaching/EHU/ISO/F...". The browser's menu bar includes "File", "Edit", "View", "Go", "Bookmarks", "Tools", "Window", and "Help". The toolbar contains navigation buttons (back, forward, home, stop), a search box, and icons for printing and home. The browser's status bar at the bottom shows "Document: Done (0.05 secs)".

The main content area displays a form titled "Tienda Animales" with the following fields and controls:

- Nombre
- Apellido
- Dirección
- Código Postal
- Num. Tarjeta
- Caduca (MM/AA)
- Payment options: AMEX VISA
- Submit button: aceptar

Paso de parámetros

- METHOD=GET
 - Los parámetros aparecen encadenados en la URL que invoca el WB
`http://www.tienda.es/cgi-bin/tienda.pl?nombre=German&...`
 - Cuando recibe la respuesta, el WS almacena la cadena de parámetros en la variable de entorno QUERY_STRING
 - La mayoría de los SO limitan las variables de entorno a 256/1024 bytes

- METHOD=POST
 - La cadena de parámetros tiene el mismo formato que el anterior, pero aparece en el cuerpo de la petición HTTP
 - El WS prepara los parámetros como entrada estándar al programa CGI

Servlets

- Los servlets son programas Java en el WS
- Permiten implementar todo tipo de servicios a partir de un protocolo sin estado basado en el paradigma HTTP
- Eficiencia y escalabilidad
 - Un servlet se inicializa una sola vez, en la primera petición
 - Cada nueva petición es servida por un nuevo “thread” (con CGI, cada petición es servida por un proceso distinto)
- Facilidad para crear y gestionar sesiones (mantener el estado a lo largo de las distintas conexiones HTTP de un mismo caso de uso de un SI web)
- Ventajas: multiplataforma, OO, Multithreading, utilización de otras Java APIs: RMI, JDBC, etc.

Servlets

```
public class TiendaAnimales extends javax.servlet.http.HttpServlet {
    public void doPost(javax.servlet.http.HttpServletRequest peticion,
        javax.servlet.http.HttpServletResponse resultado)
        throws javax.servlet.ServletException java.io.IOException {
        res.setContentType("text/html");
        java.io.PrintWriter salida = resultado.getWriter();
        salida.println("<html>");
        salida.println("<title>Datos introducidos Tienda Animales</title>");
        salida.println("<h3>Los datos introducidos son:</h3>");
        salida.println("<table border=0 width=50%>");
        java.util.Enumeration ListaCampos = peticion.getParameterNames();
        while (ListaCampos.hasMoreElements()) {
            String nombre = (String) ListaCampos.nextElement();
            String valor = peticion.getParameter(nombre);
            salida.println("<tr><td><h4>" + nombre + "</h4></td>
                <td><h4>" + valor + "</h4></td></tr>");
        }
        salida.println("</table></body></html>");
        salida.close();
    }
}
```

JavaServer Pages (JSP)

- Las JSP extienden la tecnología Java Servlet para facilitar la construcción y mantenimiento de los procesos del WS
- Una JavaServer Page es un documento que describe cómo debe procesarse una petición de un WB para crear una respuesta. Esta descripción combina contenido estático (HTML o XML) con elementos de un lenguaje script basado en Java que proporciona acceso a cualquiera de los servicios y APIs Java
- El documento se “ejecuta” en el WS cada vez que lo pide un WB
- Lo que recibe un WB es un documento HTML o XML con los componentes estáticos del JSP y los generados ad-hoc por el WS

JavaServer Pages (JSP)

- Cuando un WB pide una JSP a un WS:
 - Si es la primera vez que alguien pide la JSP:
 - la JSP se compila para generar un servlet
 - se ejecuta el servlet para construir la respuesta
 - se proporciona la respuesta al WB
 - Si no es la primera vez:
 - se ejecuta el servlet para construir la respuesta
 - se proporciona la respuesta al WB

- Tienen todas las ventajas de los servlets y permiten separar muy claramente:
 - El diseño de la página y su contenido estático
 - El código usado para generar el contenido dinámico

Elementos de una JSP

- Contenido estático: cualquier contenido HTML, que el compilador JSP ignora y que se incluye en la respuesta al cliente
- JSP Directives: Información de carácter general sobre la página (qué clases o packages se deben importar)
<%@ page [atributo="valor"]+ %>
- JSP Declaratives: Definiciones de variables y métodos globales
<%! [declaración;]+ %>
- JSP Scriptlets: Fragmentos de código Java. Permiten el acceso a objetos implícitos, como los objetos *request* (equivalentes a los `HttpServletRequest`) y *response* (`HttpServletResponse`)
- JSP Expressions: expresiones que son evaluadas y cuya respuesta es incluida en la página de retorno al WB
<%=expresión%>
- JSP Tags: permiten llamar a otras páginas o servlets (<jsp:forward>)

JSP

```
<%@ page language=java import="java.util.*" %>
<html>
<title>Datos introducidos Tienda Animales</title>
<h3>Los datos introducidos son:</h3>
<table border=0 width=50%>
<%   Enumeration ListaCampos = petition.getParameterNames();
      while (ListaCampos.hasMoreElements()) {
          String nombre = (String) ListaCampos.nextElement();
          String valor = petition.getParameter(nombre); %>
          <tr><td><h4><%=nombre%></h4></td>
              <td><h4><%=valor></h4></td></tr>
<%   } %>
</table></body></html>
```

Aplicaciones basadas en WB: Java applets

- No se corresponde al paradigma clásico de Cliente/Servidor
- El WS es literalmente un servidor de aplicaciones
- Cualquier WB que acepte applets es un cliente potencial
- No hay problemas de portabilidad de código, ni de actualización del software del cliente (siempre tiene la última versión)
- Pero ... sólo es válido para aplicaciones pequeñas debido a las múltiples restricciones de seguridad (se ejecutan completamente en la máquina del cliente)

Creación de un applet

- Compilamos un programa.java
- El correspondiente programa.class es un conjunto de bytecodes: código de la máquina virtual java JVM
- Se “incrusta” el código en la página html

```
<html>
```

```
<head> ... </head>
```

```
<body>
```

```
<applet codebase="http://www.sc.ehu.es/"  
code="programa.class" height=1000 width=400>
```

```
<param name=param1 value=100>
```

```
Su navegador no soporta Java
```

```
</applet>
```

```
</body>
```

Restricciones de seguridad

- A un applet no le está permitido:
 - Comunicarse con ninguna otra máquina que no sea el WS
 - Leer, crear, modificar o ejecutar ningún fichero del cliente
 - Obener información del cliente: nombre, versión del SO, direcciones e-mail, etc.

- La seguridad la garantiza el *applet security manager*
 - Los applets no se ejecutan directamente en el cliente
 - Son interpretados por una JVM
 - Control de seguridad antes y durante su ejecución

- Hay situaciones en los que la seguridad se puede relajar:
 - Applets con “certificado”
 - Configuramos el WB para autorizar a determinados applets certificados su ejecución sin algunos niveles de seguridad

AJAX

- AJAX (Asynchronous JavaScript and XML)
- Conjunto de tecnologías para el desarrollo de aplicaciones web interactivas que permite:
 - Hacer peticiones al web server sin tener que recargar la página cada vez que el usuario realiza una solicitud
 - + Menor volúmen de datos entre cliente y servidor
 - + Mayor velocidad
 - + Mayor interactividad
 - + Mayor usabilidad

AJAX

Conjunto de tecnologías para invocar y resolver de forma asíncrona un evento en el servidor web:

- XMLHttpRequest: objeto incluido en los exploradores y accesible desde JavaScript que permite hacer llamadas asíncronas desde un navegador al servidor sin refrescar la página.
- Formato de transporte normalmente:
 - XML
 - JSON (JavaScript Object Notation)
 - HTML, texto plano, JavaScript, ...
- DOM (Document Object Model): Objeto que representa la página web visualizada por el navegador. Puede ser modificado por el navegador (JavaScript) sin necesidad de hacer nuevas solicitudes al servidor.

AJAX

XMLHttpRequest es un API que puede ser usado por varios lenguajes de scripting para transferir datos usando HTTP, estableciendo un canal de comunicación independiente entre el cliente y el servidor

Introducido por IE como un objeto ActiveX, llamado XMLHttpRequest

```
if (window.XMLHttpRequest) { // Firefox, Safari, ...
    http_request = new XMLHttpRequest();
} else if (window.ActiveXObject) { // IE
    http_request = new ActiveXObject("Microsoft.XMLHTTP");
}
```

AJAX: example

```
var the_object;
var http_request = new XMLHttpRequest();
http_request.open("GET", url, true);
http_request.onreadystatechange = function () {
    if (http_request.readyState == 4) {
        if (http_request.status == 200) {
            the_object = eval("(" + http_request.responseText + ")");
        } else {
            alert("There was a problem with the URL.");
        }
        http_request = null;
    }
};
http_request.send(null);
```

AJAX

- JSON (JavaScript Object Notation): Formato alternativo de intercambio de información más ligero que XML
- Subconjunto de JavaScript y Python
- Tipos básicos: número, string, booleano, array, objeto, null.

```
{ "firstName": "John",  
  "lastName": "Smith",  
  "address": { "city": "New York, NY", "zipCode": 10021, "streetAddress": "21  
  2nd Street"  
  },  
  "phoneNumbers": [ "212 732-1234", "646 123-4567" ]  
}
```

```
var p = eval( "(" + Texto_JSON + ")" );  
p.firstname, p.lastName, ... p.phoneNumbers[0]
```

AJAX

- Librerías AJAX:
 - Dojo
 - Open Rico
 - ...
- Compiladores Java a JavaScript:
 - Google Web Toolkit
 - J2S (Java to JavaScript)
 - ...
- Reverse AJAX: el servidor envía código JavaScript
- Atención con la seguridad!

Capa de Gestión de Datos

- JDBC
- Serialización

Persistencia

- Las instancias y objetos de las clases sólo existen mientras se ejecuta el programa Java
- Persistencia: almacenamiento de los objetos en memoria secundaria (disco)
- Ficheros
 - No es conveniente si existen actualizaciones concurrentes
 - Java ofrece dos posibilidades:
 - Guardar los valores de los objetos en ficheros
 - Mecanismos de serialización
- SGBD
 - El SGBD y los programadores gestionan la concurrencia
 - JDBC
 - SGBD OO

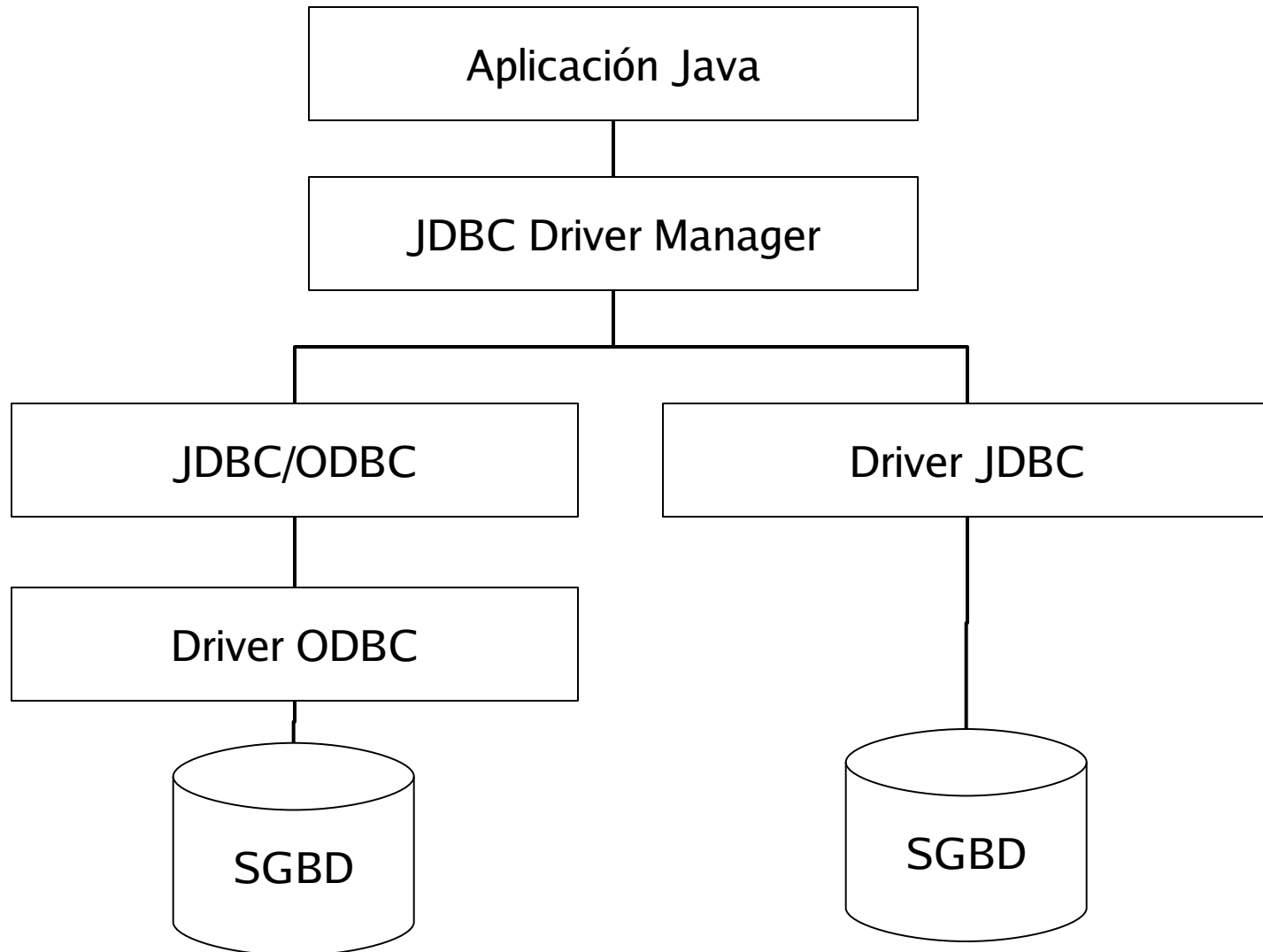
JDBC

- JDBC (Java DataBase Connectivity) es un API de java
 - Cada SGBD debe proporcionar controladores JDBC ...
 - ... o la forma de “interpretar” llamadas a JDBC
- Conjunto de clases y métodos que se encuentran en java.sql
- Sirven para:
 - Establecer conexiones con BD
 - Enviar sentencias SQL a dichas BDs
 - Procesar los resultados

JDBC y ODBC

- JDBC ofrece igual funcionalidad que ODBC (Open DataBase Connectivity) de Microsoft
- ODBC está escrito en C
- La gran mayoría de SGBD disponen de controladores ODBC
- JDK proporciona un puente JDBC-ODBC que permite convertir llamadas JDBC a ODBC y poder acceder así a BDs que ya tienen un controlador ODBC

JDBC y ODBC



JDBC: Driver Manager

- Cargar el controlador “driver” del SGBD
- Se puede usar el método `forName()` de `Class` (carga clases Java)
- Cargar el puente JDBC/ODBC
`Class.forName(“sun.jdbc.odbc.JdbcOdbcDriver”);`
- Cargar el controlador JDBC de MySQL
`Class.forName(“org.gjt.mm.mysql.Driver”);`
`Class.forName(“com.mysql.jdbc.Driver”);`
- En tiempo de ejecución, crear una instancia de la clase “`org.gjt.mm.mysql.Driver`” y asignarla a `o`

```
Object o =  
Class.forName(“org.gjt.mm.mysql.Driver”).newInstance();
```

JDBC: Connection

- Registrar los controladores (no es necesario si se ha cargado con `Class.forName`)

```
DriverManager.registerDriver(new sun.jdbc.odbc.JdbcOdbcDriver());
```

- Establecer conexiones con BDs

```
Connection c = DriverManager.getConnection(String d, String u, String c);  
// d = identifica la BD, jdbc:subprotocolo//servidor:puerto/BDs  
// u = identifica el usuario  
// c = identifica la clave
```

- Las conexiones hay que cerrarlas cuando ya no se necesitan:

```
c.close();
```

JDBC

- Por defecto, toda sentencia SQL enviada a un Statement termina con un “commit” si tiene éxito
- Si se quiere que varias sentencias SQL formen una transacción:

```
c.setAutoCommit(false);
```

```
// Lanzar las sentencias SQL que forman parte de la transacción
```

```
c.commit(); // si todo ha ido bien y confirmamos la transacción  
c.rollback(); // si queremos deshacer la transacción (y la BDs lo  
permite)
```

JDBC: Statement

- Al tener la conexión abierta, se pueden lanzar sentencias SQL desde el programa Java
- Para ello, hay que crear los objetos statement:

```
Statement s = c.createStatement(); // c es un objeto Connection
```

```
int i = s.executeUpdate(String sql);
```

```
// Ejecuta una sentencia SQL INSERT, UPDATE o DELETE
```

```
// Devuelve el número de registros afectados
```

```
ResultSet r = s.executeQuery(String sql);
```

```
// Ejecuta una sentencia SQL SELECT y devuelve el resultado en
```

```
// un objeto ResultSet
```

JDBC: Statement

- Sólo puede haber un ResultSet “abierto” sobre un objeto Statement
- Por tanto, si queremos trabajar con varias preguntas a la vez, debemos tener varios Statement distintos.
- Para cerrar Statements (y liberar recursos)
s.close()
- Se puede limitar el número máximo de registros de retorno
s.setMaxRows(maxTuplas);
- Se puede limitar el tiempo (en segundos) que queremos que espere
s.setQueryTimeout(maxSegundos);

JDBC: ResultSet

- Para trabajar con las respuestas a las preguntas SQL realizadas

```
boolean b = r.next();
```

```
// r es un objeto de tipo ResultSet
```

```
// se posiciona en el siguiente registro del resultado
```

```
// devuelve true si no ha llegado al final (false en caso contrario)
```

- Existen métodos get que devuelven el valor de un atributo del registro actual
- El atributo se puede identificar por el nombre dado en el SELECT o por su posición en la misma
- El tipo del atributo puede ser desconocido

JDBC: ResultSet

- Si conocemos el tipo del atributo

```
string s = r.getString(numAtributo);
```

```
string s = r.getString(nombreAtributo);
```

```
int i = r.getInt(numAtributo);
```

```
int i = r.getInt(nombreAtributo);
```

```
boolean b = r.getBoolean(numAtributo);
```

```
boolean b = r.getBoolean(nombreAtributo);
```

- Si no lo conocemos

```
Object o = r.getObject(numAtributo);
```

```
Object o = r.getObject(nombreAtributo);
```

JDBC

- Parametrización de sentencias SQL

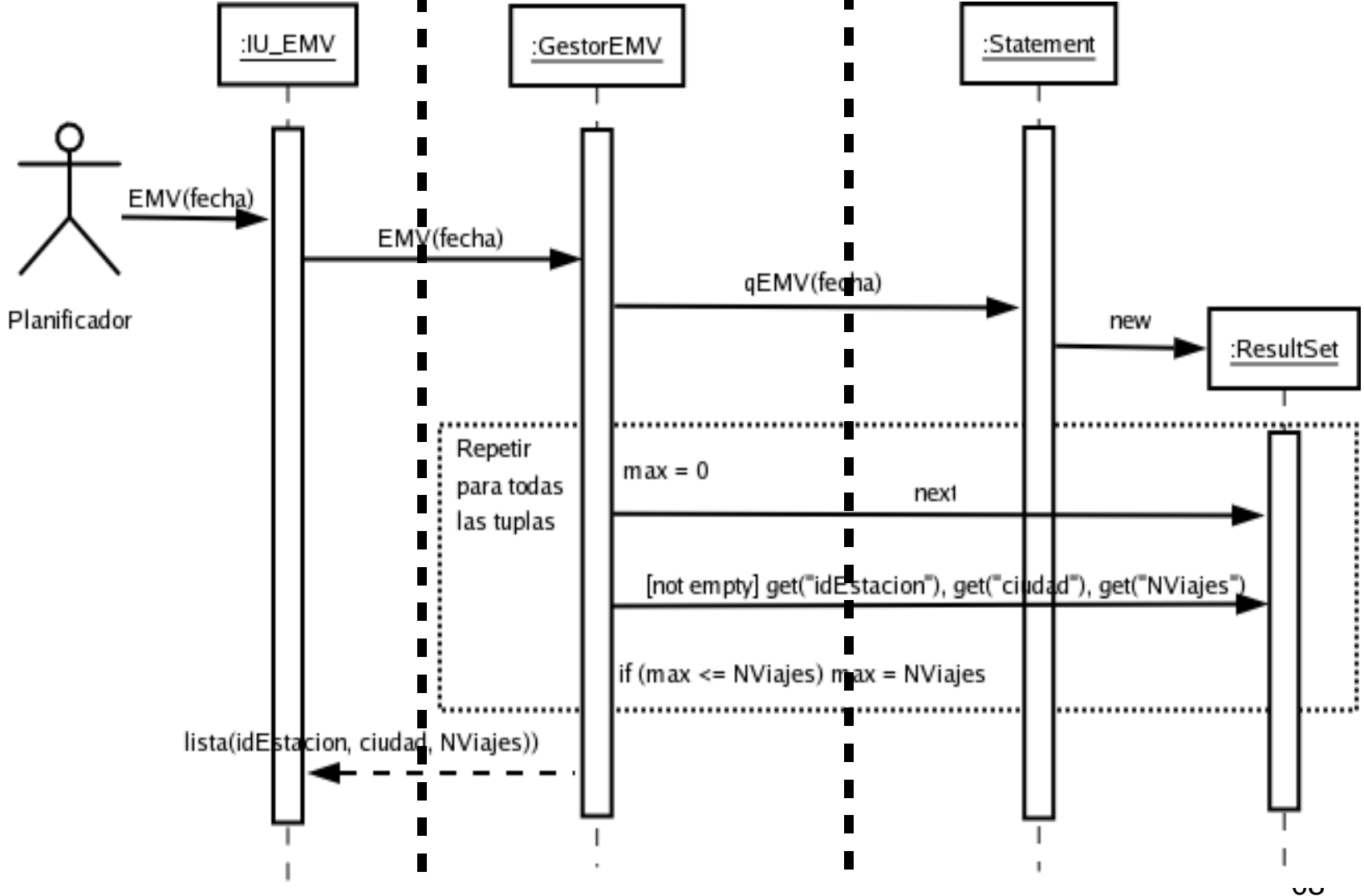
```
PreparedStatement s =  
c.prepareStatement("SELECT NOMBRE FROM PERSONAS  
WHERE CIUDAD=? AND EDAD=?");  
s.setString(1, "BCN"); // pone BCN en el primer parámetro  
s.setInt(2, 25); // pone 25 en el segundo parámetro  
ResultSet r = s.executeQuery();  
s.setString(1, "SS");  
s.setInt(2, 30);  
ResultSet r = s.executeQuery();
```

Ingeniería del Software

Capa Presentación

Capa Lógica de Negocio

Capa Gestión de datos

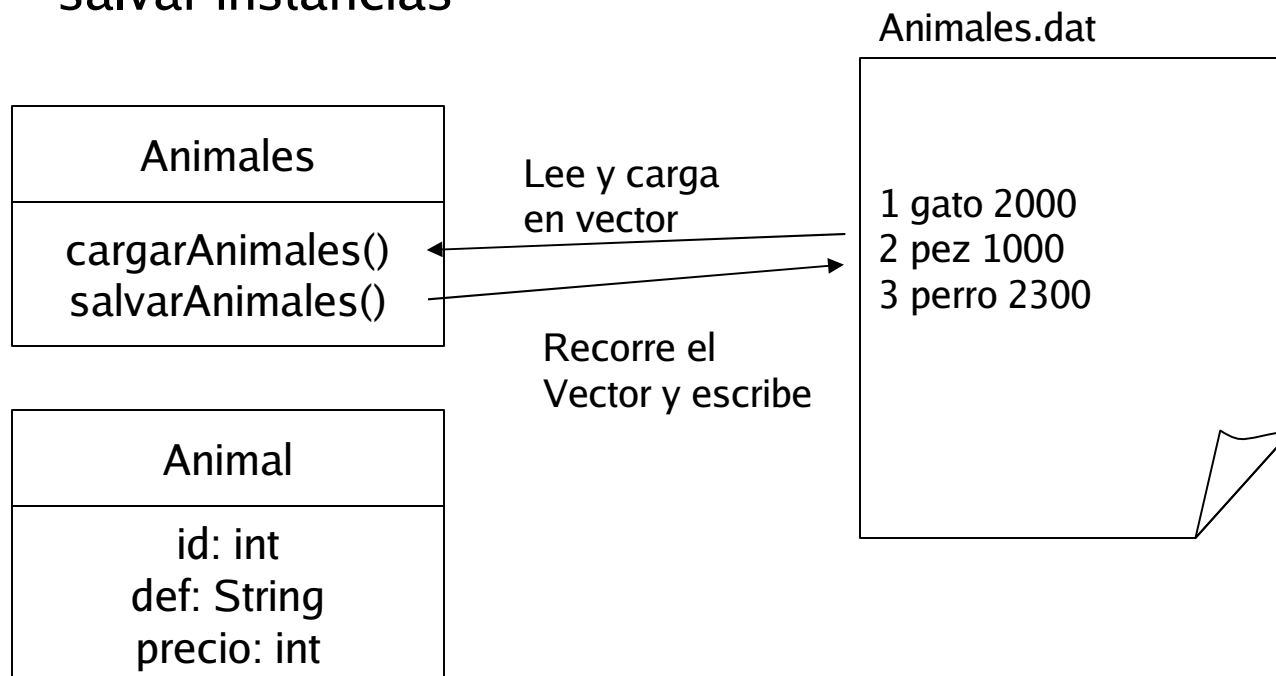


Ingeniería del Software

```
public list EMV (String fecha) throws RemoteException {
    String SQL = "SELECT E.idEstacion, E.ciudad, COUNT(DISTINCT(V.idViaje)) AS
Nviajes FROM Viaje AS V INNER JOIN Convoy as C ON V.idViaje=C.idViaje INNER JOIN Via AS
VIA ON C.idVia=VIA=idVia INNER JOIN Estacion AS E ON VIA.Origen=E.idEstacion WHERE
V.fechaSalida = "+fecha+" GROUP BY E.idEstacion ORDER BY NViajes DESC";
    list l = new LinkedList();
    int max = 0;
    int NViajes = 0;
    boolean continue = true;
    try {
        Statement s = c.createStatement(); // c anteriormente definida
        ResultSet r = s.executeQuery(SQL);
        while (r.next() && continue) {
            NViajes = r.getInt("NViajes");
            if (max <= Nviajes) {
                list e = new linkedList();
                e.add(r.getString("idEstacion"));
                e.add(r.getString("ciudad"));
                e.add(r.getInt("NViajes"));
                l.add(e);
                max = Nviajes;
                continue = true
            } else {
                continue = false
            }
        }
    }
    catch (Exception ex) {
        System.out.println(ex.getMessage());
        list e = new linkedList(); e.add("error"); e.add("null"); e.add(0);
        l.add(e);
    }
    return l;
}
```

Persistencia en ficheros

- Se guardan todas las instancias de una clase en un fichero. Hay que añadir los métodos para:
 - cargar instancias
 - salvar instancias



Persistencia en ficheros

```
package tienda;
```

```
class animal {
```

```
    private int id;  
    private String def;  
    private int precio;
```

```
    public getId() {return id;}  
    public getDef() {return def;}  
    public getPrecio() {return precio;}  
    public Animal (int id, String def, int precio) {  
        this.id = id;  
        this.def = def;  
        this.precio = def;  
    }
```

```
    public String toString() {return id+" "+def+" "+precio;}  
}
```

Persistencia en ficheros

```
Import java.util.*; import java.io.*;
```

```
public class animales {  
    private static Vector losAnimales = new Vector();  
    public static final String EOF=null;  
    public static void cargarAnimales() throws FileNotFoundException,IOException {  
        BufferedReader entrada = new BufferedReader(new FileReader("animales.dat"));  
        String linea;  
        while ((linea=entrada.readLine())!=EOF) {  
            StringTokenizer st = new StringTokenizer(linea, " ");  
            Animal a = new Animal(Integer.parseInt(st.nextToken()),  
                                   st.nextToken(),  
                                   Integer.parseInt(st.nextToken()));  
            losAnimales.addElement(Animal a);  
        }  
        entrada.close();  
    }  
}
```


Persistencia en ficheros

```
Import java.util.*; import java.io.*;
```

```
public class animales {
```

```
    ...
```

```
    public static void guardarAnimales() throws FileNotFoundException,IOException {
```

```
        PrintWriter salida = new PrintWriter(new FileWriter("animales.dat"));
```

```
        for(int i=0;i<losAnimales.size();i++) {
```

```
            salida.println(losAnimales.elementAt(i).toString());
```

```
        }
```

```
        salida.close();
```

```
    }
```

Persistencia en ficheros

```
public static void main(String[] argv) {  
    try {  
        Animales.cargarAnimales();  
        Animal a = new Anima(123, "siamés", 2400);  
        Animales.guardarAnimales();  
    }  
    catch (Exception e) {  
        System.out.println("Error: "+e.toString());  
    }  
}
```

Serialización

- Conversión de objetos Java en series de bytes
- Son útiles para:
 - Proporcionar persistencia de objetos:
 - Convertir objetos en bytes y guardar en streams de bytes (ficheros)
 - Enviar mensajes entre objetos de distintas máquinas
 - Usando sockets, RMI
- Para serializar objetos de una clase, en la definición de dicha clase hay que indicar que implementa la interfaz `java.io.Serializable`
- Entonces se pueden leer/escribir objetos en `ObjectInputStream/ObjectOutputStream` con los métodos `readObject/writeObject`
- `writeObject` serializa todos los objetos contenidos en él (recursivo)
- No se serializan los atributos static

Serialización

- Los métodos `writeObject` y `readObject` funcionan bien con los tipos básicos `string`, `arrays`, `vector`, etc.
- `writeObject` serializa todos los objetos contenidos en él (recursivo)
- Para no serializar un atributo debe declararse `transient`
- No se serializan los atributos `static`

```
ObjectOutputStream s =  
    new ObjectOutputStream(new FileOutputStream("animales.dat"));  
s.writeObject(losAnimales);
```

```
ObjectInputStream e =  
    new ObjectInputStream(new FileInputStream("animales.dat"));  
losAnimales = (Vector) e.readObject();
```

Persistencia en ficheros

```
package tienda;
```

```
class Animal implements serializable {
```

```
    private int id;  
    private String def;  
    private int precio;
```

```
    public getId() {return id;}  
    public getDef() {return def;}  
    public getPrecio() {return precio;}  
    public Animal (int id, String def, int precio) {  
        this.id = id;  
        this.def = def;  
        this.precio = def;  
    }
```

```
    public String toString() {return id+" "+def+" "+precio;}  
}
```

Persistencia en ficheros

```
Import java.util.*; import java.io.*;
```

```
public class animales {  
    private static Vector losAnimales;  
    private static final String nombreFichero = "animales.dat";  
    public static final String EOF=null;  
  
    synchronized public static void cargarAnimales() throws  
        ClassNotFoundException, IOException {  
        ObjectInputStream e =  
            new ObjectInputStream(new FileInputStream(nombreFichero));  
        losAnimales = (Vector) e.readObject();  
    }  
    synchronized public static void guardarAnimales() throws IOException {  
        ObjectOutputStream s =  
            new ObjectOutputStream(new FileOutputStream(nombreFichero));  
        s.writeObject(losAnimales)  
    }  
}
```

SI OO distribuidos

- RMI
- CORBA, SOAP
- OLE, DCOM/COM+ y la plataforma .NET

Java Remote Method Invocation (RMI)

- API que proporciona Java
- Conjunto de clases y métodos que se encuentran en *java.rmi*
- Propone el entorno homogéneo de Java Virtual Machine (JVM)
- Conserva la sintaxis y la semántica del modelo de objetos Java no distribuidos
- Permite la conexión con otras aplicaciones o librerías escritas en otros lenguajes de programación utilizando Java Native Method Interface (JNI)
- Permite la conexión con BD relacionales utilizando JDBC
- <http://java.sun.com/docs/books/tutorial/rmi/overview.html>

Java Remote Method Invocation (RMI)

- Las aplicaciones RMI normalmente comprenden dos programas separados: un servidor y un cliente.
- Una servidor típico crea objetos remotos, hace accesibles las referencias a esos objetos y espera a que los clientes invoquen métodos a los objetos remotos
- Un cliente típico obtiene las referencias remotas a uno o más objetos remotos en el servidor e invoca métodos sobre ellos
- RMI proporciona el mecanismo por el que se comunican el cliente y el servidor para pasarse información.
- Este tipo de aplicaciones se denominan aplicaciones de objetos distribuidos

Ventajas de RMI

- Orientación a objetos: RMI permite pasar objetos como parámetros y tipos de retorno (no solamente tipos predefinidos)
- Seguridad: RMI usa los mecanismos de seguridad de Java
- Portabilidad: a otras JVM...
- Garbage Collection distribuida
- Paralelismo: el servidor RMI es multi-thread

Modelo de objetos distribuidos

- Los métodos de un objeto remoto pueden ser invocados por una máquina distinta a aquella que contiene el objeto
- Un objeto remoto se describe con una o más interfaces remotas. Estas interfaces se escriben en Java y declaran los métodos del Objeto Remoto
- Remote Method Invocation (RMI) es la acción de invocar un método de una interfaz remota en un objeto remoto
- La sintaxi de llamada a un objeto remoto es exactamente la misma que a un objeto local

Modelo de objetos distribuidos

- Para construir una aplicación cliente/servidor donde un cliente accede a un servicio remoto (proporcionada por una clase remota) usando RMI debemos:
 - Construir una interfaz remota
 - Implementar dicha interfaz remota (servidor RMI)
 - Implementar el cliente RMI que accede al servicio remoto

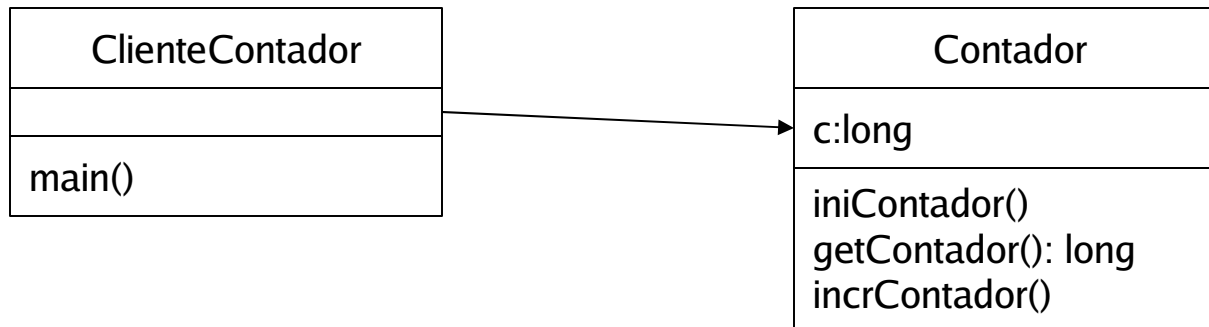
Modelo de objetos distribuidos

- Los clientes de un Objeto Remoto interactúan con interfaces remotas (nunca con las clases de implementación de dichas interfaces)
- Los parámetros no-remotos y los resultados que devuelven las invocaciones a métodos remotos se pasan por copia (en el modelo no distribuido por referencia) utilizando la serialización de objetos
- Un objeto remoto se pasa por referencia
- Los clientes de Objetos remotos deben tratar excepciones adicionales que pueden producirse debido a RMI

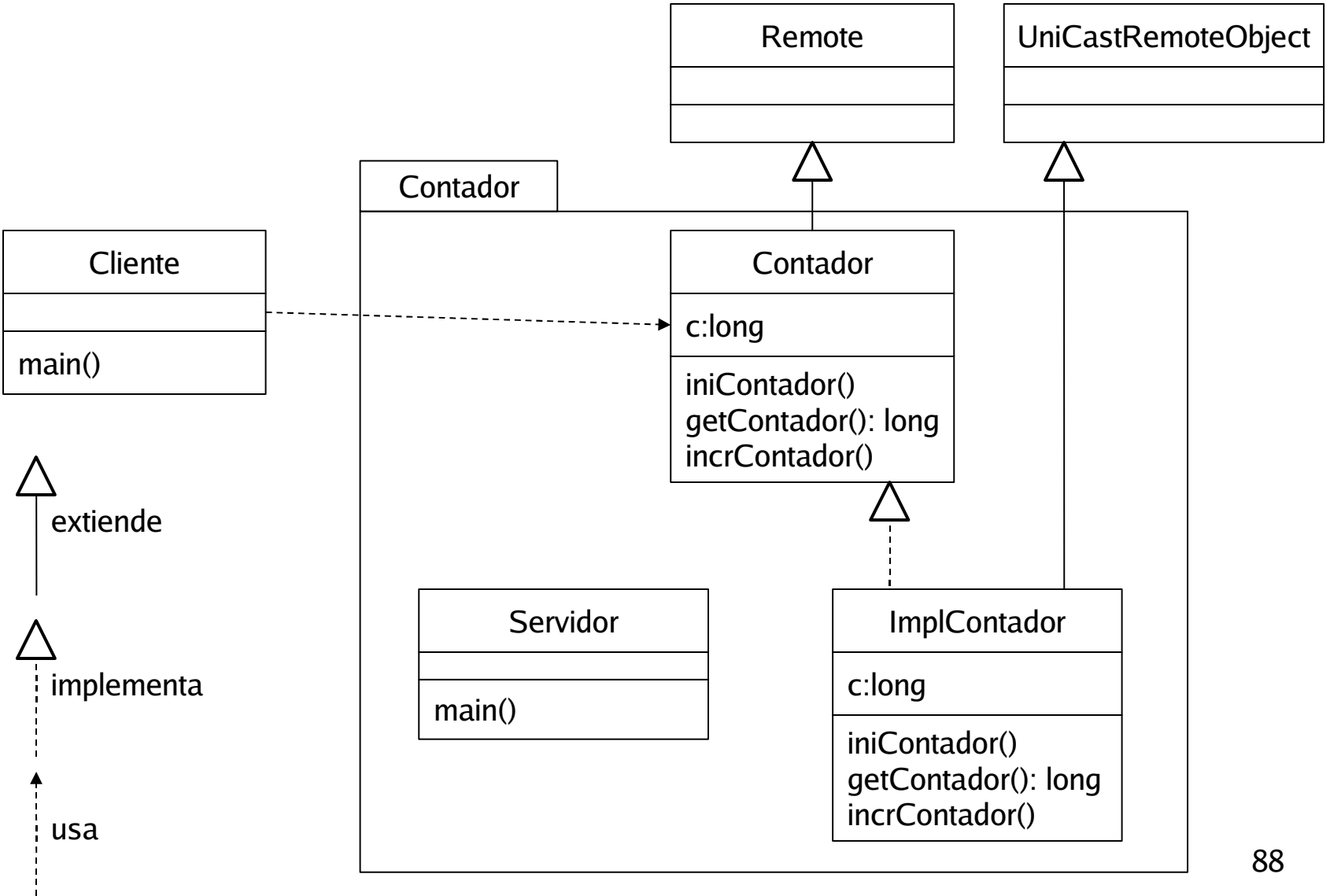
Localización de objetos remotos

- El sistema RMI ofrece un registro de nombres simple (rmiregistry) con el que pueden obtenerse las referencias a objetos remotos
- El nombre de un objeto remoto es parecido a un URL (Uniform Resource Locator)
 - //host[:port]/nombre
 - El host es del registro de nombres
 - port donde el registro espera recibir las invocaciones
 - Nombre es el string que identifica el objeto dentro del registro
- El acceso al registro se realiza con los métodos de la clase `java.rmi.Naming`

Ejemplo: Contador no distribuido



Ejemplo: Contador distribuido



Interfaz Remota

- Una interfaz remota declara un conjunto de métodos que pueden ser invocados desde una JVM remota
- Una interfaz remota debe extender (directa o indirectamente) la interfaz `java.rmi.Remote`
- Las definiciones de métodos en las interfaces remotas deben contemplar la excepción `java.rmi.RemoteException`

Interfaz remota

```
package Contador;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Contador extends Remote {

    void iniContador() throws RemoteException;
    long getContador() throws RemoteException;
    void incrContador() throws RemoteException;

}
```

Implementación con Objetos “transitorios”

- La clase `java.rmi.server.UnicastRemoteObject` permite crear y exportar objetos remotos cuyas referencias únicamente son válidas durante la vida del proceso servidor que crea el objeto remoto (objetos “transitorios”)

```
package Contador;
```

```
import java.rmi.RemoteException;
```

```
import java.rmi.server.UnicastRemoteObject;
```

```
public class interface ImplContador
```

```
    extends UnicastRemoteObject implements Contador {
```

```
    ...
```

```
}
```

Implementación ServidorContador

- Es necesario un proceso servidor que cree el objeto remote y publique su referencia en el registro de nombres de RMI (Naming.rebind)

```
package Contador;
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Servidor {
    public static void main(String args[]) {
        ...
        ImplContador c = new ImplContador(); // crea el objeto contador
        Naming.rebind("//157.147.20.15/MiContador", c); // Pone MiContador en el registro de nombres
        System.out.println("Contador creado y registrado.");
        ...
    }
}
```

Implementación Cliente

- El cliente invocará los métodos del objeto remoto un vez localizado su referencia en el registro de nombres del RMI (Naming.lookup)

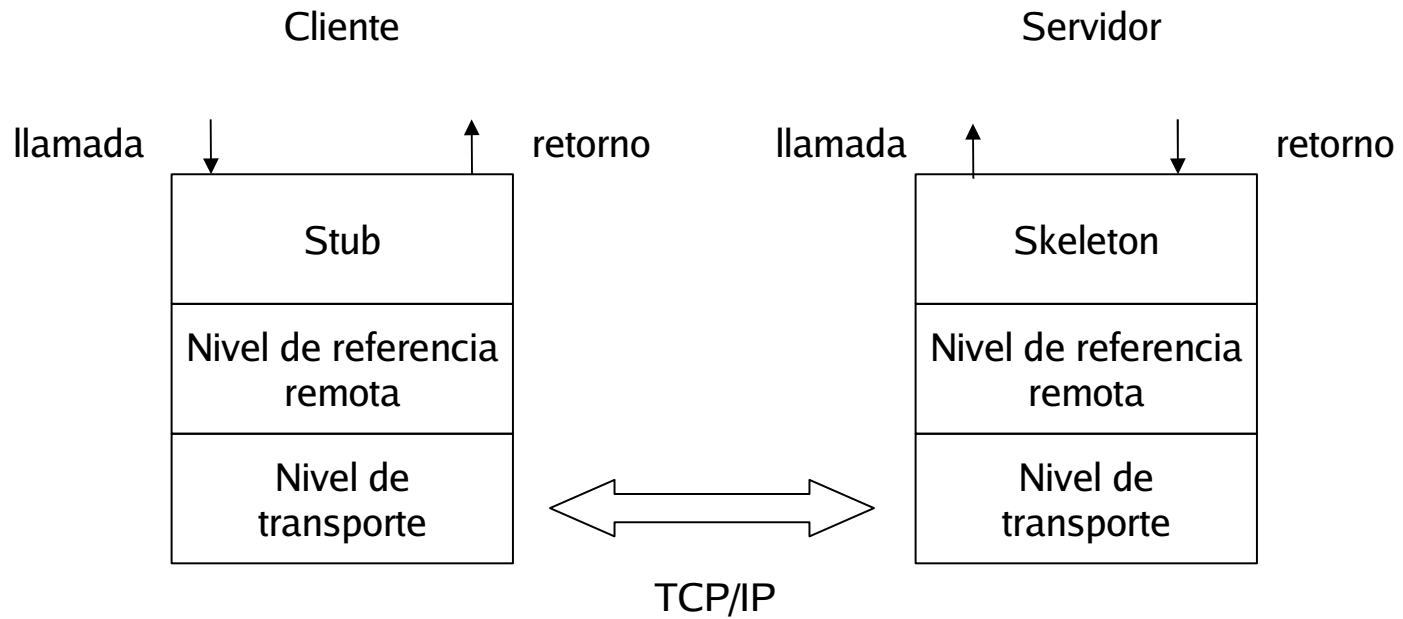
```
package Contador;
import java.rmi.Naming;
import java.rmi.RemoteException;

public class Cliente {
    public static void main(String args[]) {
        ...
        // Obtiene MiContador del registro de nombres
        Contador c = (Contador)Naming.lookup("//157.147.20.15/MiContador");
        c.iniContador();
        for(int i=0;i<1000;i++) { c.incrContador() };
        System.out.println("El valor del contador es:"+getContador());
        ...
    }
}
```

Arquitectura RMI

- Hay que conectar el objeto cliente con el objeto servidor para que las llamadas a métodos del primero sean ejecutadas por el segundo
- Hay que pasar los valores de los parámetros de los métodos del cliente al servidor
- Hay que pasar los resultados de los métodos del servidor al cliente
- Los objetos Stub (cliente) y Skeleton (servidor) se encargan de realizar la conexión y el paso de parámetros y resultados

Stubs y Skeletons



Stubs y Skeletons

- rmic se encarga de la generación de los stubs y skeletons a partir de la clase implementación

```
sistema> javac contador.java
```

```
sistema> javac implContador.java
```

```
sistema> javac servidor.java
```

```
sistema> rmic implContador
```

Y se obtienen las clases

implContador_Stub.class => debe quedar accesible al cliente!

implContador_Skeleton.class

rmiregistry

- Es un servidor de nombres que relaciona objetos con nombres
- Hay que lanzarlo como proceso independiente en la máquina servidor
- `rmiregistry [numPuerto]`
- También se puede lanzar desde una aplicación Java (en el servidor RMI)
- `Java.rmi.registry.LocateRegistry.createRegistry(p)`
 - Crea el proceso `rmiregistry` en el puerto `p`
 - `Rmiregistry` no termina aunque acabe el servidor RMI
 - Lanza una excepción si el puerto está ocupado

Política de seguridad

- Un programa java debe especificar un gestor de seguridad que determine su política de seguridad
- Algunas operaciones requieren que exista dicho gestor. En concreto, las RMI.
- RMI sólo cargará una clase serializable desde otra máquina si hay un gestor de seguridad que lo permita
- Para establecer un gestor de seguridad por defecto para RMI:
`System.setSecurityManager(new RMISecurityManager());`
- El gestor de seguridad por defecto de RMI sigue una política muy restrictiva (sólo permite ejecutar STUBs del CLASSPATH local)
- `java -Djava.security.policy=java.policy Clase`
- Fichero `java.policy`:
`grant { permission java.security.Allpermission };`

Simple Object Access Protocol (SOAP)

- SOAP es un simple protocolo en XML que permite a las aplicaciones intercambiar información mediante HTTP.
- SOAP es un protocolo para facilitar los Servicios Web
- <http://www.w3schools.com/soap/default.asp>
- <http://www.w3schools.com/xml/default.asp>
- SOAP se inició en 1999 por W3C.
- SOAP 1.0 estaba basado por entero en HTTP.
- La siguiente versión SOAP 1.1 (Mayo 2000) era más genérica e incluía otros protocolos de transporte.
- La versión actual de SOAP 1.2 (Junio 2003) ha sido promovida a “Recomendación”.

Qué es SOAP?

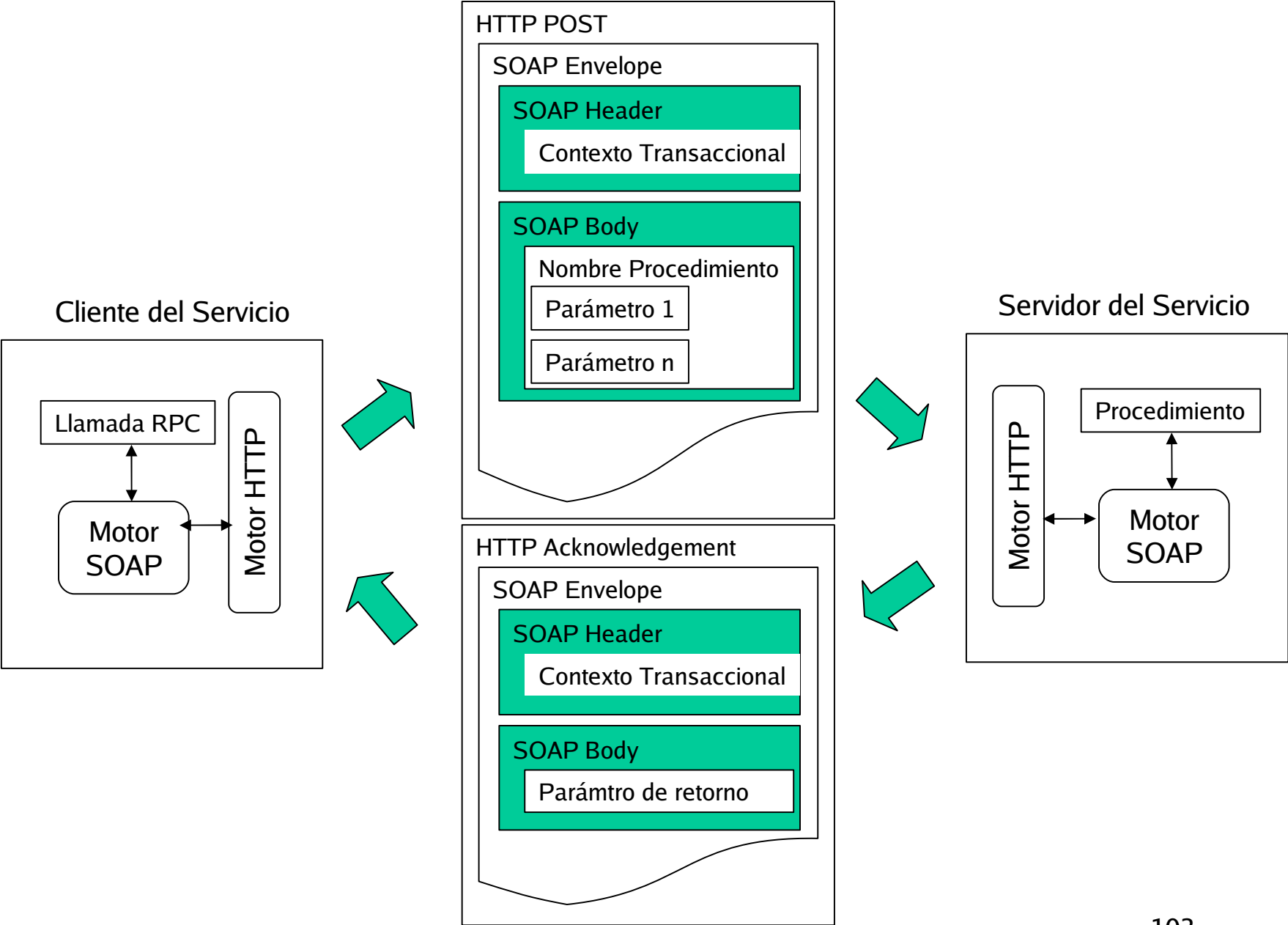
- SOAP significa **Simple Object Access Protocol**
- SOAP es un **protocolo de comunicación**
- SOAP permite la comunicación entre **aplicaciones**
- SOAP es un formato para **enviar mensajes**
- SOAP está diseñado para comunicarse via **Internet**
- SOAP es **independiente de la plataforma**
- SOAP es **independiente del lenguaje de programación**
- SOAP está **basado en XML**
- SOAP es **simple y extensible**
- SOAP evita **firewalls**
- SOAP es un **estándar W3C**

Por qué SOAP?

- Actualmente, las aplicaciones se comunican usando Remote Procedure Calls (RPC) entre objetos como DCOM y CORBA.
- Sin embargo, RPC representa un problema de compatibilidad y seguridad: los firewalls y proxies normalmente bloquearán este tipo de tráfico.
- Es importante para el desarrollo de aplicaciones permitir la comunicación entre programas usando Internet.
- Una mejor forma de comunicar aplicaciones es usando HTTP, porque HTTP es soportada por todos los navegadores y servidores.
- Usa estándares: HTTP y XML.
- SOAP proporciona una forma de comunicación entre aplicaciones que pueden estar ejecutándose en distintos SO, distintas tecnologías y distintos lenguajes.

Componentes SOAP

- Un formato de mensaje en XML para comunicaciones de una sola dirección
- Una descripción de cómo un mensaje SOAP se transporta a través de la web (usando HTTP) o e-mail (usando SMTP).
- Un conjunto de reglas que seguir cuando se procesa un mensaje SOAP y una clasificación simple de las entidades involucradas en el proceso. También especifica qué partes del mensaje deben ser leídas por quién y cómo reaccionar en caso de que el contenido no ha sido entendido.
- Un conjunto de convenciones sobre cómo convertir llamadas y retornos tipo RPC en mensajes SOAP



Mensajes SOAP

- SOAP se basa en el intercambio de mensajes
- Los mensajes pueden verse como sobres donde las aplicaciones encapsulan los datos a ser enviados
- Un mensaje tiene dos partes: cabecera (header) y cuerpo (body), que pueden ser divididos en bloques. La cabecera es opcional y el cuerpo obligatorio
- El uso de la cabecera y el cuerpo es implícito. La cabecera es para los datos del nivel de infraestructura y el cuerpo para los datos del nivel de aplicación.

Sintaxis SOAP

- Un mensaje SOAP es un documento XML que contiene los siguientes elementos:
 - Una parte obligatoria (Envelope) que identifica el documento XML como un mensaje SOAP
 - Una cabecera opcional (Header) que contiene la información de la cabecera
 - Un cuerpo obligatorio (Body) que contiene la información de la llamada y la respuesta
 - Una sección opcional (Fault) que proporciona información sobre los errores que ocurren mientras se procesa el mensaje

Ejemplo SOAP: Petición

POST /InStock HTTP/1.1

Host: www.stock.org

Content-Type: application/soap+xml; charset=utf-8

Content-Length: nnn

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2001/12/soap-envelope"

soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Body xmlns:m="http://www.stock.org/stock">

<m:GetStockPrice>

<m:StockName>IBM</m:StockName>

</m:GetStockPrice>

</soap:Body>

</soap:Envelope>

Ejemplo SOAP: Respuesta

HTTP/1.1 200 OK

Content-Type: application/soap; charset=utf-8

Content-Length: nnn

```
<?xml version="1.0"?>
```

```
  <soap:Envelope
```

```
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
```

```
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
```

```
  <soap:Body xmlns:m="http://www.stock.org/stock">
```

```
    <m:GetStockPriceResponse>
```

```
      <m:Price>34.5</m:Price>
```

```
    </m:GetStockPriceResponse>
```

```
  </soap:Body>
```

```
</soap:Envelope>
```

Sintaxis SOAP

```
<?xml version="1.0"?>
<soap:Envelope
  xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
  soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">
<soap:Header> ... .. </soap:Header>
<soap:Body> ... ..
<soap:Fault> ... .. </soap:Fault>
</soap:Body>
</soap:Envelope>
```

Abstracción vs. Eficiencia

