Java EE – Servlets

Aplicaciones Web / Sistemas Web

Juan Pavón Mestras
Dep. Ingeniería del Software e Inteligencia Artificial
Facultad de Informática
Universidad Complutense Madrid

Java EE Servlets y JSPs

- **Servlet**
  - Clase Java que implementa un modelo de programación petición-respuesta
    - Puede usarse para procesar cualquier tipo de petición
      - Clase GenericServlet
    - Definido en el paquete **javax.servlet**
    - Hay subclases específicas para HTTP: paquete **javax.servlet.http**
      - Clase HttpServlet
  - Tiene un ciclo de vida concreto controlado por el contenedor en el que se despliega
  - Cada servlet se ejecuta como un thread independiente

- **JSP (JavaServer Page)**
  - Fichero con código (X)HTML que incluye scripts codificados en Java
  - Se compila y se convierte en un servlet
  - Utiliza etiquetas especializadas (*Custom Tags*) que amplían la sintaxis de HTML

Ciclo de vida de un servlet

- Lo controla el contenedor en el que se ha desplegado
- Al llegar una petición correspondiente a un servlet, el contenedor
  1. Comprueba si existe una instancia del servlet
     - Si no existe
       - Carga la clase del servlet
       - Crea una instancia del servlet
       - Inicializa la instancia del servlet llamando al método *init()*
  2. Se invoca al **método de servicio**, pasándole objetos de tipo *request* y *response*
     - El servlet usa estos objetos para inspeccionar la petición y generar la respuesta
  3. Si hay que eliminar el servlet, el contenedor llama al método destroy() del servlet

Servlet

- Cualquier clase que implemente la interfaz **javax.servlet.Servlet**
  - Métodos para gestionar el ciclo de vida del servlet
- También suele ser necesaria la interfaz **javax.servlet.ServletConfig**
  - Tiene los parámetros de arranque e inicialización para el servlet que le pasa el contenedor
- Normalmente los servlets se implementan a partir de una de las dos subclases siguientes que implementan ambas interfaces:
  - **javax.servlet.GenericServlet**
    - Clase que define un servlet independiente del protocolo
  - **javax.servlet.http.HttpServlet**
    - Para servlets en aplicaciones web, que procesan las peticiones con el protocolo HTTP

Métodos de la interfaz javax.servlet.Servlet

- **init(ServletConfig config)**
  - Al arrancar un servlet, solo se ejecuta una vez
  - Inicializa atributos y recursos del servlet
- **getServletConfig()**
  - Devuelve el objeto *ServletConfig* que contiene parámetros de arranque e inicialización para el servlet que le pasa el contenedor
- **service(ServletRequest request, ServletResponse response)**
  - Cada vez que se invoca un servicio al servlet
  - Dependiendo del tipo de servicio, invoca el método correspondiente al servicio solicitado
  - Normalmente no se reescribe este método, solo los correspondientes a los servicios específicos
  - Pueden invocarse concurrentemente los métodos de servicio, por ello deben ser thread safe
- **destroy()**
  - Al eliminar un servlet, solo se ejecuta una vez
  - Tiene que ser thread-safe porque puede haber otros threads en ejecución

Métodos de la clase javax.servlet.HTTPServlet

- El método **service** trata las siguientes peticiones:
  - DELETE, GET, HEAD, OPTIONS, POST, PUT, y TRACE
  - Invoca el método correspondiente **doXXX**
- Los más usados:
  - **doGet**(HttpServletRequest request, HttpServletResponse response)
  - **doPost**(HttpServletRequest request, HttpServletResponse response)
- Estos dos métodos son los que normalmente se tienen que reescribir
  - El objeto request tiene información enviada en la petición
  - El objeto response permite crear la respuesta
    - setContentType() – permite indicar el tipo MIME de lo que se va a generar (por ejemplo, "text/html")
    - getWriter() – devuelve el PrintWriter donde escribir lo que se genere
  - Pueden producir dos excepciones:
    - IOException, porque usan operaciones de E/S
    - ServletException, cualquier otra excepción que se quiera

Creación de servlets con eclipse

- Crear un proyecto Web dinámica
  - File → New → Project → Web → Dynamic Web Project
    - Nombre, p.ej. *PrimerosServlets*
    - Target runtime: *Apache Tomcat v7.0*
- En src, crear un paquete
  - Nombre, p.ej. *es.ucm.prueba*
- Dentro del paquete, crear un servlet con el asistente de creación
  - File → New → Other y seleccionar servlet en Web
    - Nombre, p.ej. *Saludo*
  - En el método *doGet*() incluir el código que aparece a continuación
    - Habrá que importar la clase java.io.PrintWriter

Ejemplo de servlet elemental

```
package es.ucm.prueba;

import java.io.IOException;
import java.io.PrintWriter;

import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@WebServlet("/Saludo")
public class Saludo extends HttpServlet {
private static final long serialVersionUID = 1L;

 /**
 * @see HttpServlet#doGet(HttpServletRequest request, HttpServletResponse response)
 */
protected void doGet(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException {
          PrintWriter out = response.getWriter();
          out.println("¡Bienvenido al mundo de los servlets!");
    }

}
```

URL relativo a la webapp

Método para tratar la recepción de un HTTP GET

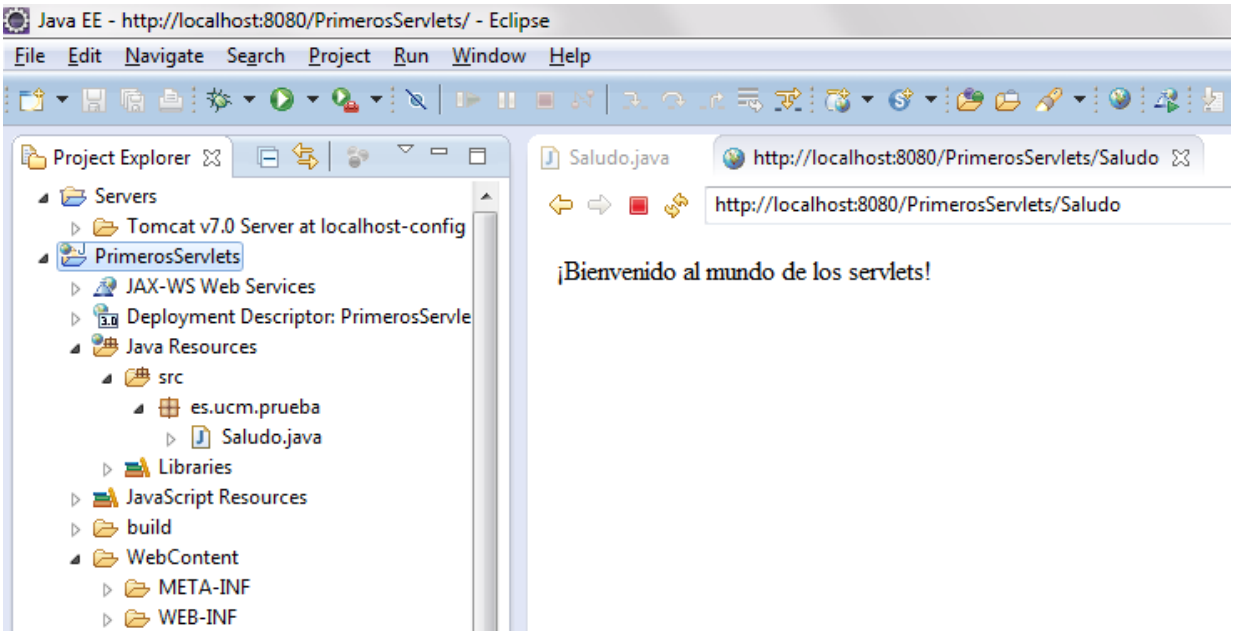
Salida generada para enviar al navegador

Generación de texto

Creación de servlets con eclipse

- Ejecutar
  - Run As...→Run On Server
    - La primera vez aparece una ventana para indicar en qué servidor se quiere ejecutar
      - Elegir Tomcat v7.0 Server at localhost
      - En Add and remove, si no está en la lista de Configured, seleccionar el proyecto en Available y pulsar el botón Add>
  - Solicitar restart server
    - Permite que el servidor arranque con la nueva configuración
  - Aparece una pestaña en el área de trabajo con el resultado de ejecutar el servlet en un navegador
    - Si el URL que aparece es http://localhost:8080/PrimerosServlets/ saldrá el error 404 (recurso no disponible)
    - Para que se vea la página, incluir en la URL el nombre del servlet
      - http://localhost:8080/PrimerosServlets/Saludo

Primeros servlets con eclipse y tomcat



Ejercicios

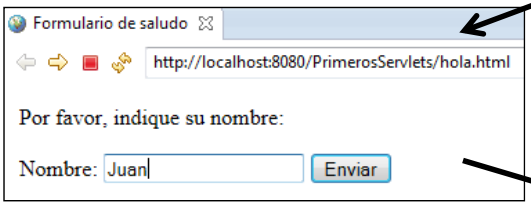
- Añadir un fichero index.html en el proyecto en WebContent
  - New → HTML file
  - Ahora sí que se podrá ver la página http://localhost:8080/PrimerosServlets/
- Crea otro servlet SaludoHTML, que genere código HTML correcto
  - Para indicar el tipo de documento
    - response.setContentType("text/html");
  - Generar el texto HTML correspondiente
- En el fichero index.html añade enlaces para poder invocar a los dos servlets
- Añadir un atributo int contador que permite al servlet llevar una cuenta de las veces que se invoca
  - Se define como un atributo normal en la clase
  - Se puede visualizar en la página

Interacción con formularios

http://localhost:8080/PrimerosServlets/hola.html

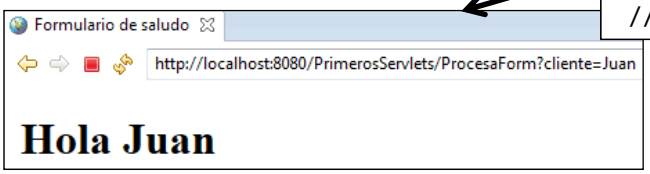
hola.html

```
<p>Por favor, indique su nombre:</p>
<form method="get"
      action="/PrimerosServlets/ProcesaForm">
<p>Nombre:
<input type="text" name="cliente" />
<input type="submit" value="Enviar"/></p>
</form>
```



ProcesaForm.java

```
...
out.println("<!DOCTYPE html>\n" +
//...
"<h1>Hola
"+request.getParameter("cliente")+"</h1>\n" +
 //...
```



Formularios

- La página HTML con los formularios va en la carpeta WebContent
  - En el ejemplo, el fichero hola.html irá en WebContent del proyecto *PrimerosServlets*
    - El URL para acceder será *http://localhost:8080/PrimerosServlets/hola.html*
- El fuente del servlet se creará en src, dentro del paquete correspondiente
  - El servlet (compilado) irá en WEB_INF/classes

Ejemplo de proceso de formulario

```
package es.ucm.prueba;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.*;

@WebServlet("/ProcesaForm")
public class ProcesaForm extends HttpServlet {
   private static final long serialVersionUID = 1L;

   protected void doGet(HttpServletRequest request, HttpServletResponse response)
                           throws ServletException, IOException {
         response.setContentType("text/html");
         PrintWriter out = response.getWriter();
         out.println ("<!DOCTYPE html>\n" +   "<html>\n" +
                  "<head><title>Formulario de saludo</title></head>\n" +
                  "<body>\n" +
                  "<h1>Hola "+request.getParameter("cliente")+"</h1>\n" +
                  "</body></html>");
         }
}
```

Parámetros de la petición

- Los parámetros que llegan con GET y POST los tiene el objeto request y se pueden acceder con varios métodos:
  - public Enumeration **getParameterNames**()
    - Devuelve un Enumeration con los nombres de los parámetros
    - Si no los hubiera, el Enumeration estará vacío
  - public String **getParameter**(String name)
    - Valor del parámetro como un String
    - null si no existiera ese parámetro
  - public String[] **getParameterValues**(name)
    - Si un parámetro puede tener varios valores mejor usar este método

- Un objeto Enumeration se usa como un Iterador:

```
Enumeration e = request.getParameterNames();
while (e.hasMoreElements()) {
    out.println(e.nextElement());
}
```


Parámetros de la petición

- Para pasar los valores de los parámetros de String al valor del tipo correspondiente
  - int n = new Integer(parametro).intValue();
  - double d = new Double(parametro).doubleValue();
  - byte b = new Byte(parametro).byteValue();
    - Estas operaciones se tienen que hacer en un try para poder capturar la excepción NumberFormatException
  - boolean p = new Boolean(param).booleanValue();

Parámetros de la petición

- Hay que comprobar siempre que los parámetros recibidos son correctos
  - Si falta un campo en el formulario, getParameter() devuelve null
  - Hay que comprobar que el string tiene el formato correcto
    - Por ejemplo, que al convertir a un número no se genera la excepción *NumberFormatException*
  - Cuando no se ha rellenado el campo correspondiente se recibe el string vacío
  - Conviene eliminar los espacios en blanco con el método **trim**()

```
String parametro = request.getParameter("nombre");
if ((parametro == null) || (parametro.trim().equals(""))) {
        // Tratamiento de parámetro erróneo
}
else {
        // Tratamiento normal con el parámetro
}
```


Parámetros de la petición

- Lo habitual cuando hay errores en los formularios es volver a mostrarlos de nuevo
  - Mostrando los valores correctos en sus campos (el usuario no tiene que volver a introducirlos)
  - Marcar los campos que no se han rellenado
  - Si se han rellenado mal, añadir un comentario para indicar al usuario qué tiene que hacer
- Con JSF, o frameworks como Struts, es más fácil la gestión de formularios
- También es conveniente usar JavaScript para hacer una primera comprobación de los campos del formulario

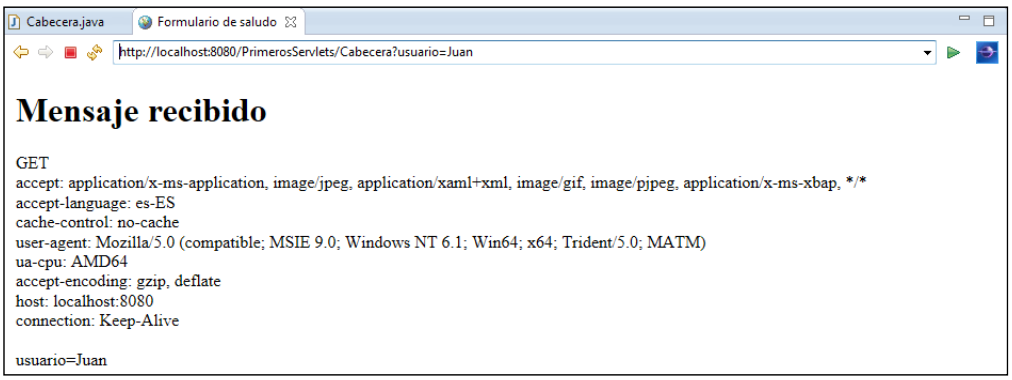
Cabecera de la petición

- Los parámetros de la cabecera de un mensaje HTTP se pueden obtener con los siguientes métodos
  - String **getHeader**(String nombre)
    - Devuelve el valor de la cabecera indicada, null si no hay una cabecera con ese nombre
  - Enumeration<String> **getHeaders**(String nombre)
    - Algunas cabeceras, como Accept-Language, pueden tener una lista de valores
  - Enumeration<String> **getHeaderNames**()
    - Los nombres de todas las cabeceras presentes
  - String **getMethod**()
    - Indica el método HTTP: GET, POST, o PUT
  - String **getQueryString**()
    - La query string que hay en el URL, o null si no hay

Ejercicios

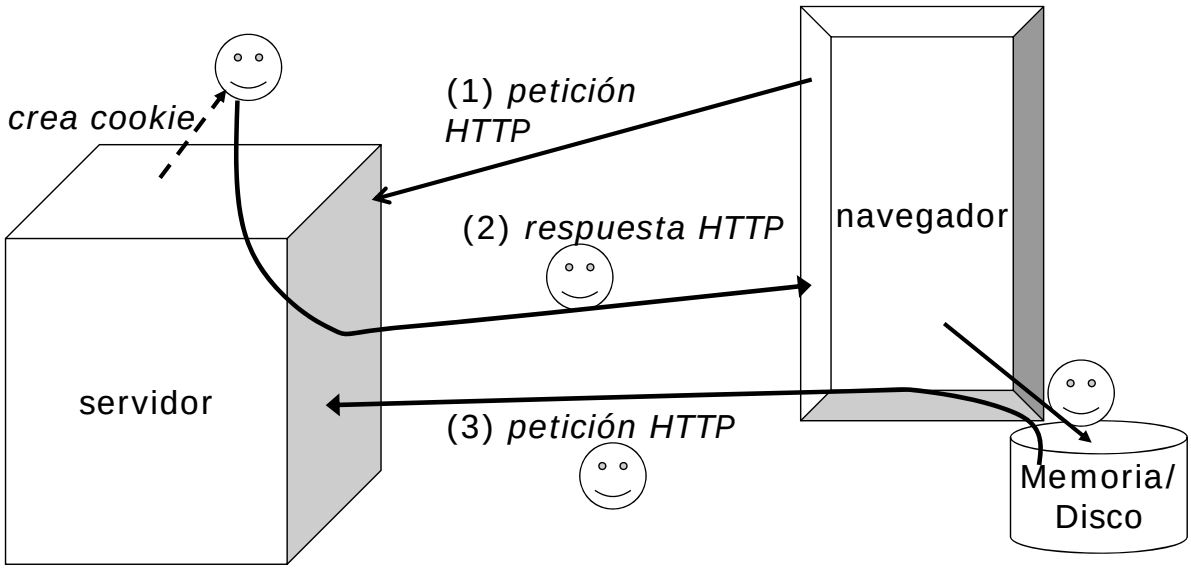
- Crear un servlet que muestre en una página el mensaje GET o POST que haya recibido, con todas las cabeceras y los parámetros
  - El código del doGet y doPost será igual, por lo cual se podría poner el código en el doGet y en el doPost invocar al doGet:

```
public void doPost(HttpServletRequest request,
                   HttpServletResponse response) {
    doGet(request, response);
}
```



Cookies

- Funcionamiento de los Cookies



Programación de cookies con Java

- Creación y envío de un cookie
  - Por un *servlet* como respuesta a una petición HTTP
  - clase *javax.servlet.http.Cookie*

```
// 1. crea el cookie
Cookie cookie = new Cookie("nombre", "valor");
```

  - El cookie se envía como parte de una respuesta HTTP

```
// 2. envía el Cookie en un HTTPServletResponse
public void doGet (HttpServletRequest, request,
        HttpServletResponse response) throws IOException
{
        response.addCookie(cookie);
}
```

Programación de cookies con Java

- Envío de un identificador único
  - La clase *java.rmi.server.UID* permite crear identificadores que son únicos dentro de la máquina en que se generan

```
String uid = new java.rmi.server.UID().toString();
```

  - Para enviarlo por HTTP hace falta codificarlo con el método *enconde()* de *java.net.URLEncoder*, que convierte el string al formato MIME denominado *x-www-form-urlencoded*
    - 'a' - 'z', 'A' - 'Z', y '0' - '9' no se modifican.
    - El carácter de espacio se transforma en '+'.
    - Los demás caracteres se convierten en string de 3-caracteres "%xy", donde xy es la representación hexadecimal con dos dígitos de los 8-bits del carácter

```
Cookie cookie = new Cookie("uid",
java.net.URLEncoder.encode(uid));
// ...
response.addCookie (cookie));
```


Programación de cookies con Java

- Recepción de cookies
  - Con una respuesta HTTP se pueden recibir varios cookies asociados

```
public void doGet (HttpServletRequest request,
    HttpServletResponse response) throws IOException
{
    Cookie[] cookies = request.getCookies();
    if ( cookies != null )
        for (int i=0; i<cookies.length; i++) {
                Cookie unCookie = cookies[i];
                System.out.println(" nombre: " + unCookie.getName()
                        + ", valor: " + unCookie.getValue());
        }
}
```

Programación de cookies con Java

- Configuración de cookies
  - La clase Cookie ofrece operaciones para configurar los atributos de cada cookie
    - Tiempo de expiración

```
cookie.setMaxAge (numero_segundos);
  // si 0, entonces el cookie expira inmediatamente
  // si negativo, el cookie expira al apagar el navegador
```

    - Comentario

```
cookie.setComment ("comentario");
Sring txt=cookie.getComment();
```

    - Dominio

```
cookie.setDomain ("patron_de_dominio");
```

    - Valor (para modificar el que se le dio al crearlo)

```
cookie.setValue ("nuevovalor");
```


Gestión de sesiones con servlets

- Las sesiones se representan con objetos de la clase **javax.servlet.http.HttpSession**
  - Basado en el mecanismo de cookies
  - Permite mantener el estado del cliente
  - La información sobre la sesión se mantiene automáticamente de una petición a otra, incluso en el caso de rearranque del servidor
- Crear una sesión:

```
HttpSession sesion = request.getSession();
```

  - Devuelve la sesión asociada con la petición
  - Si no existiera, la crea
- Crear o modificar atributos de la sesión:

```
sesion.setAttribute(nombre, valor);
```

- Consulta de valores de atributos de la sesión:

```
Object obj = sesion.getAttribute(nombre);
```

- Consulta de todos los nombres de atributos de la sesión:

```
Enumeration<String> e = sesion.getAttributeNames()
```

Gestión de sesiones con servlets

- Tras un periodo de inactividad las sesiones se cierran
  - Puede configurarse el tiempo, en segundos, que el contenedor del servlet mantendrá la sesión abierta entre accesos del cliente:
    - void **setMaxInactiveInterval**(int interval)
      - Si interval es 0 o negativo, el contenedor no cerrará la sesión
    - int **getMaxInactiveInterval**()
- Otros métodos para trabajar con sesiones:
  - long **getCreationTime**()
  - long **getLastAccessedTime**()
    - Ambos en número de milisegundos desde 1/1/1970 GMT

Ejercicios de sesión con servlets

- Prueba a contar el número de accesos de un cliente a una página durante una sesión
  - La página tendrá un botón para iniciar la sesión y otro para cerrarla
  - También visualizará en cada momento el número de accesos que se han producido a la página durante la sesión
  - Adicionalmente podrá mostrar información de cuánto tiempo ha transcurrido desde el último acceso
- Crea una secuencia de páginas que soliciten información sobre un usuario
  - En la primera página su nombre, en la segunda su número de teléfono y en la tercera su email
  - En la cuarta se mostrarán los datos recibidos
- En ambos ejercicios, prueba a acceder a la vez desde dos navegadores distintos para comprobar que se pueden gestionar dos sesiones diferentes a la vez

ServletContext

- Cuando se despliega un proyecto, el contenedor web creará un objeto ServletContext
  - Hay uno por cada aplicación web
- Tiene la información de configuración del fichero web.xml
  - Permite compartir información entre servlets de la aplicación con el elemento **<context-param>**
  - Puede servir para intercambiar información entre servlets y entre aplicaciones
- Métodos
  - Para tratar los parámetros de inicialización
    - String **getInitParameter**(String name)
    - Enumeration<String> **getInitParameterNames**()
    - void **setInitParameter**(String name, String value)
  - Para crear, modificar y consultar atributos
    - Object **getAttribute**(String name)
    - Enumeration<String> **getAttributeNames**()
    - void **setAttribute**(String name, Object object)
      - Si object==null se elimina el atributo

Ejercicios ServletContext

- Listar todos los parámetros de inicialización
  - Tiene que haber un fichero web.xml en WEB_INF
    - Si no lo hubiera se puede crear en eclipse con Java EE Tools → Generate Deployment Descriptor Stub
    - Se pueden añadir parámetros de inicialización con

```
<context-param>
    <param-name>Autor</param-name>
    <param-value>Juan Pavón</param-value>
</context-param>
```

  - Se leerán desde el servlet como sigue:

```
ServletContext aplicacion=getServletContext();
Enumeration<String> e = aplicacion.getInitParameterNames();
String s="";
while (e.hasMoreElements()) {
        s= e.nextElement();
        out.print("</p>");
        out.print(s+": "+aplicacion.getInitParameter(s));
        out.println("</p>");
}
```

Ventajas de los servlets respecto a CGIs

- Los servlets se ejecutan como threads
  - No requieren crear un proceso cada vez
  - Se quedan en memoria, por lo cual no tienen que recargarse cada vez
  - Una instancia puede gestionar múltiples peticiones
  - Por seguridad, se pueden ejecutar servlets en un sandbox
- Arquitectura de aplicaciones web
  - Java EE facilita la aplicación de patrones de diseño
  - Varios frameworks
- Acceso a todas las librerías Java existentes
- Portabilidad
  - Código Java
- Seguridad
  - El código Java se ejecuta en un contenedor

Bibliografía

- Eric Jendrock et al. The Java EE 6 Tutorial (2013). http://docs.oracle.com/javaee/6/tutorial/doc/
- Building Web Apps in Java: Beginning & Intermediate Servlet & JSP Tutorials. http://courses.coreservlets.com/Course-Materials/csajsp2.html
- Documentación oficial
  - Java EE Specifications http://www.oracle.com/technetwork/java/javaee/tech/index.html
  - API specification for version 6 of Java EE http://docs.oracle.com/javaee/6/api/