2010 - 2011

Tehnologii de Programare in Internet (TPI / RST)

Titulari curs: Mihnea Magheti, Eduard-Cristian Popovici

Suport curs: http://discipline.elcom.pub.ro/tpi/

Structura cursului

Continut curs TPI

1. Introducere in tehnologiile Internet

2. Introducere in tehnologiile desktop (SE) Java

- 2.1. Elemente de baza. Tipuri de date referinta. Clase de biblioteca
- 2.2. Clase pentru fluxuri de intrare-iesire (IO)

3. Programarea la nivel socket in Java

- 3.1. Introducere in Protocolul Internet (IP) si stiva de protocoale IP
- 3.2. Socketuri flux (TCP) Java si programe multifilare (threads)
- 3.3. Socketuri datagrama (UDP) Java

4. Tehnologii Java de programare a aplicatiilor Web (EE) Java

- 4.1. Tehnologii client. Miniaplicatii Java (applet-uri)
- 4.2. Clase pentru interfete grafice cu utilizatorul (AWT, Swing)
- 4.3. Platforma Java EE. Arhitectura si tehnologiile implicate
- 4.4. Tehnologii server. Tehnologia Java Servlet
- 4.5. Tehnologia Java ServerPages (JSP)
- 4.6. Accesul la baze de date prin tehnologii Java (JDBC, Hibernate)
- 4.7. Tehnologii avansate (frameworks, componente EJB, Servicii Web)

4. Tehnologii Java de programare a aplicatiilor Web (EE) Java

4.3. Platforma Java EE. Arhitectura si tehnologiile implicate

Arhitectura Java EE 5 include

un subsistem client (***client tier***), care poate contine **aplicatii de sine statatoare** (Java SE) sau **browsere si continut Web** (incluzand eventual applet-uri Java), si optional componente JavaBeans

un subsistem Web (*Web tier*), care contine componente Web: servlet-uri Java si pagini JSP (*Java ServerPages*) si optional componente JavaBeans

un subsistem (care poate fi distribuit) care indeplineste sarcinile aplicatiei / serviciile (***business tier***), care contine **componente *business*** (EJB – *Enterprise JavaBeans*):

- **bean-uri entitate** (in cazul EJB 2.x) **sau entitati persistente** (in cazul EJB 3.0),
- **bean-uri sesiune** si
- **bean-uri pentru comunicatii asincrone** (MDB – *Message-Driven Beans*)

un subsistem de integrare si management al datelor aplicatiei (***EIS tier***), care contine elemente de integrare si management pentru subsisteme care furnizeaza sau stocheaza datele aplicatiei (inclusiv baze de date)

Arhitectura Java EE 5

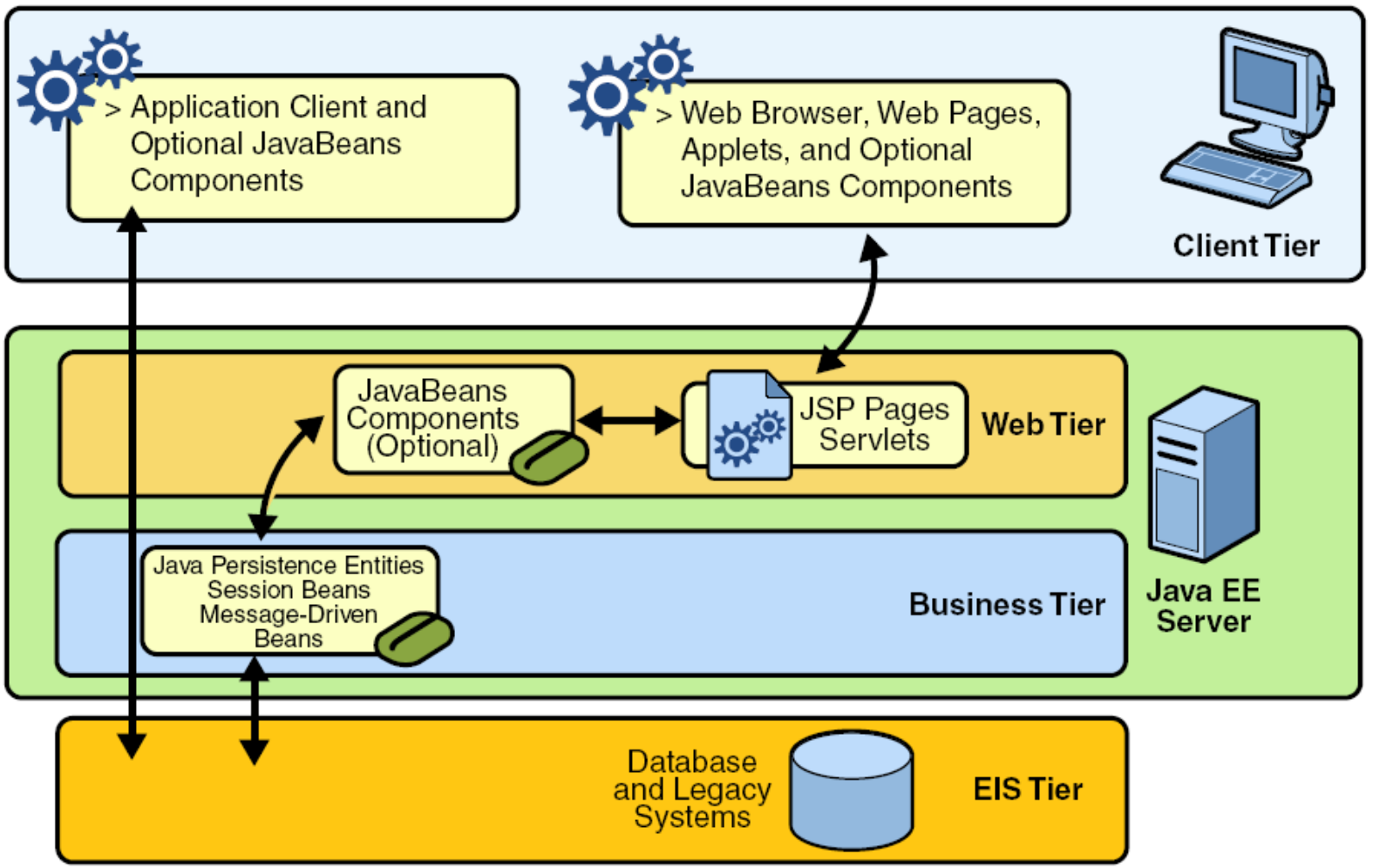
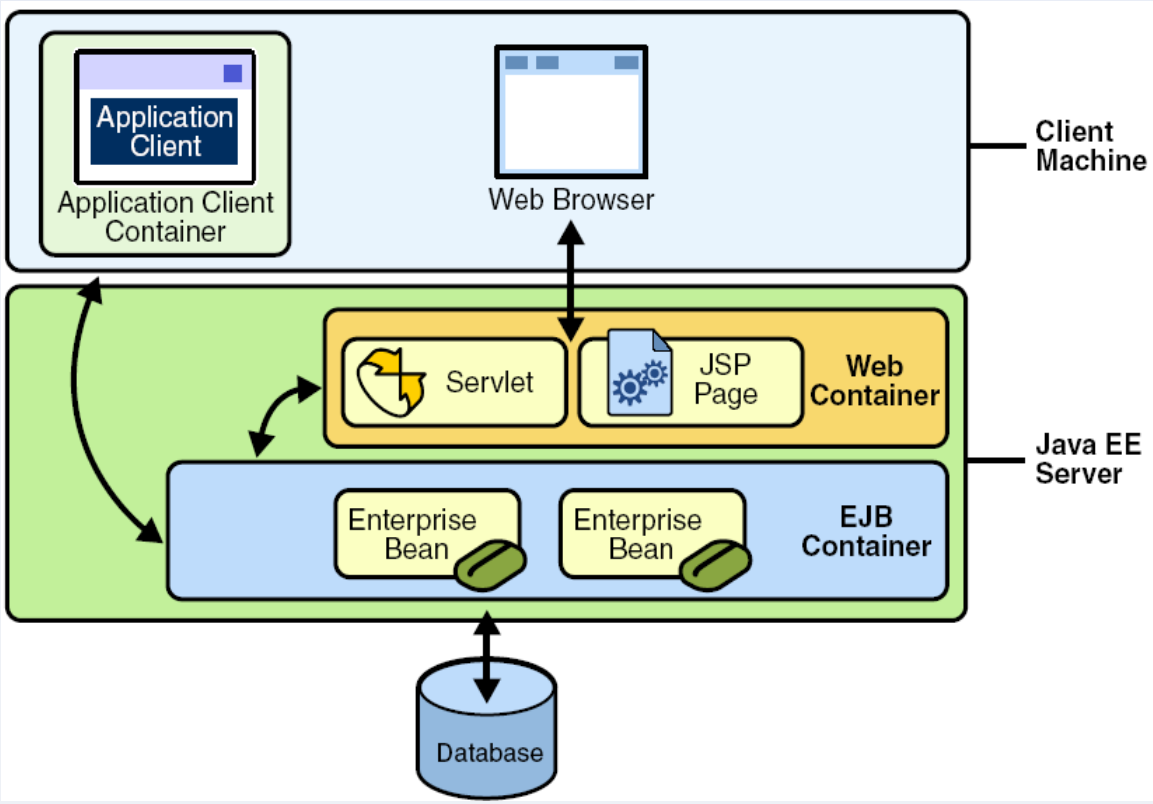


FIGURE 1–4 Business and EIS Tiers

Containere J2EE

- **Arhitectura Java EE 5 se bazeaza pe mai multe containere** (elemente ale arhitecturii care ofera servicii de nivel jos specifice platformei pe care se lucreaza, si gestioneaza dezvoltarea si instalarea aplicatiilor Web prin asamblarea unor componente specifice) **aflate pe serverul aplicatiei Web**:

Containere J2EE



containerul de componente Web, care ofera servicii sistem de integrare cu serverul HTTP si gestioneaza ciclul de viata al servlet-urilor, translatia paginilor JSP in servlet-uri si compilarea acestora, asambland componentele din cadrul subsistemului Web al aplicatiei

containerul de componente business, care care ofera servicii sistem de distributie si gestioneaza ciclul de viata al componentelor EJB de tip entitate, sesiune si mesagerie asincrona, asambland aceste componente in cadrul subsistemului care indeplineste sarcinile aplicatiei

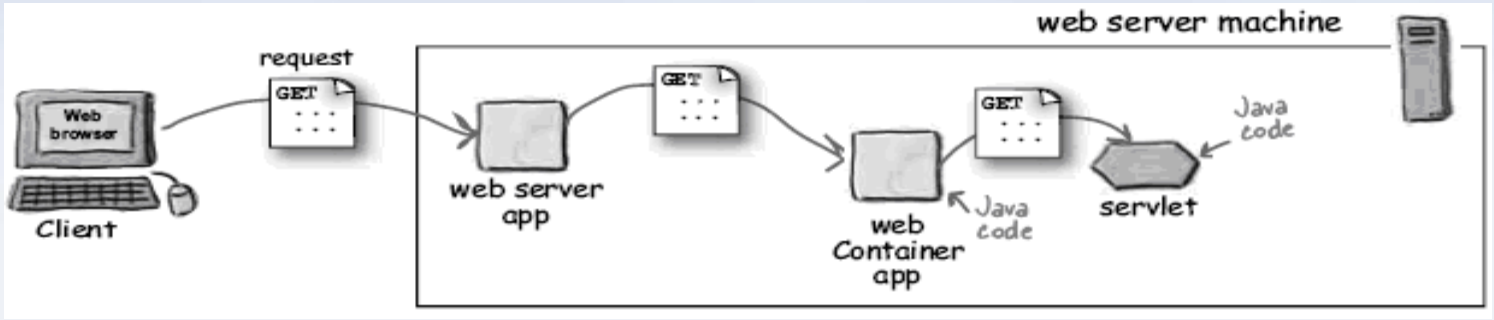
4. Tehnologii Java de programare a aplicatiilor Web (EE) Java

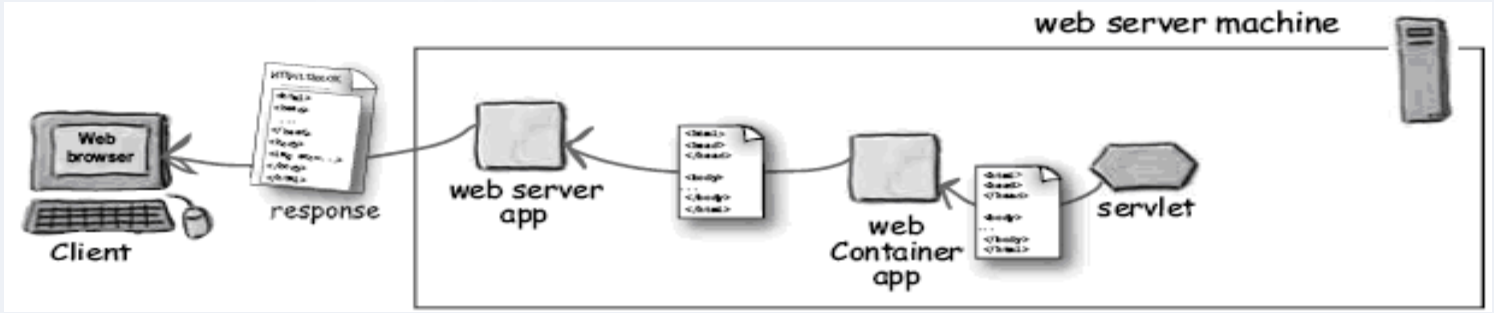
4.4. Tehnologii server. Tehnologia Java Servlet

Un servlet este un obiect al unei clase Java ce extinde functionalitatea unui server care lucreaza dupa modelul de acces cerere-raspuns (cum este cel utilizat de protocolul HTTP, pe care se bazeaza aplicatiile Web) prin crearea unui continut dinamic

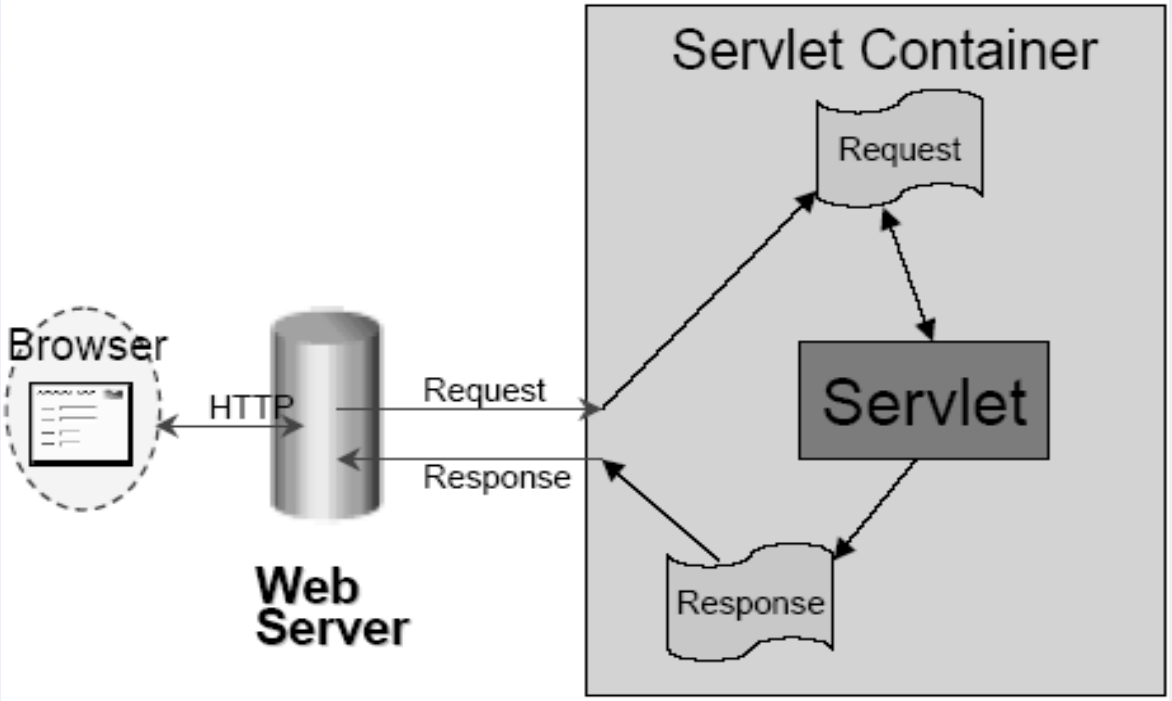
Un **servlet Web** (care adauga functionalitate unui server **HTTP**) **trebuie sa extinda** (prin mostenire) clasa **HttpServlet** din pachetul **javax.servlet.http**

Servlet-urile Web sunt **componente** care **se executa intr-un container Web** (*Web container* sau *Web engine*), tot asa cum applet-urile sunt executate intr-un browser Web

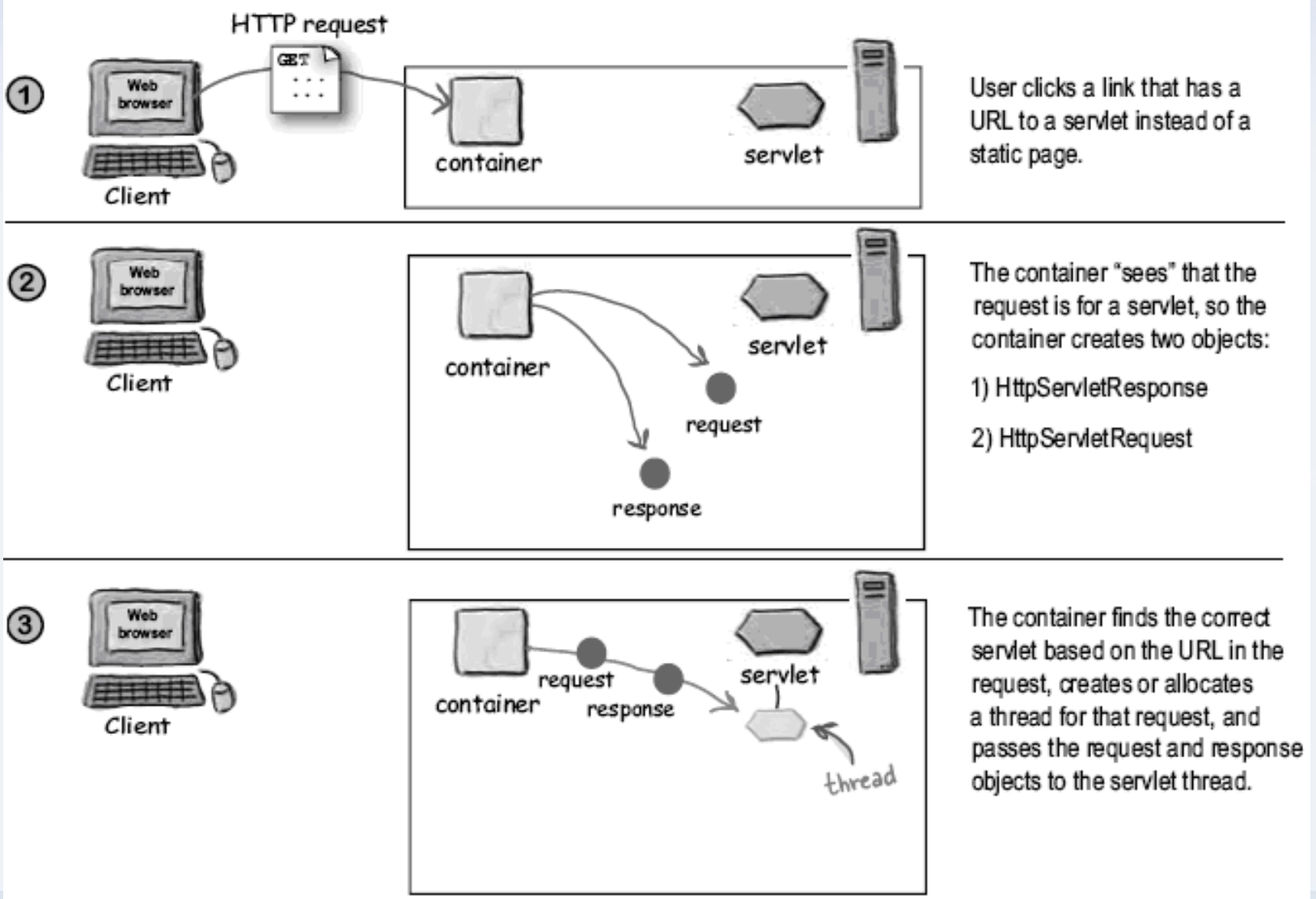

web server machine
request
GET
Web browser
Client
web server app
GET
GET
Java code
web Container app
Java code
servlet

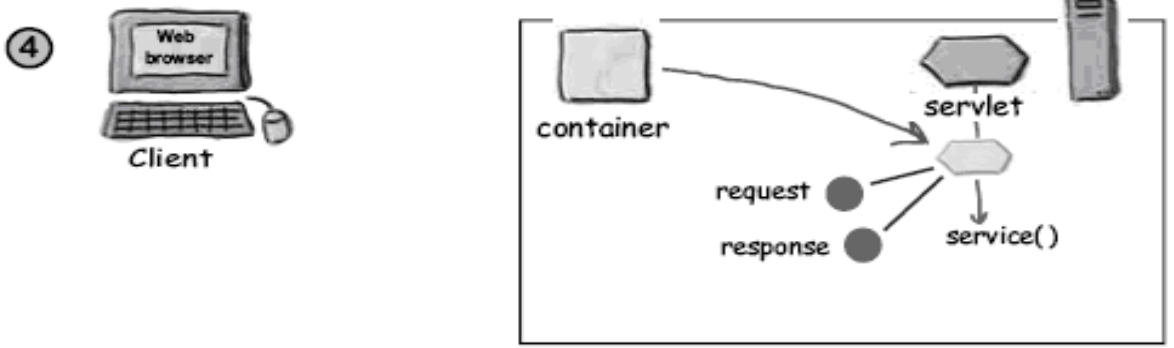

web server machine
Web browser
Client
response
web server app
web Container app
servlet

operatiile care tin de ciclul de viata al servlet-ului (apelul metodelor **`init()`**, **`destroy()`**, **`service()`**) sunt realizate de catre container in momentele in care acestea sunt necesare (initializare, incarcare, etc.).



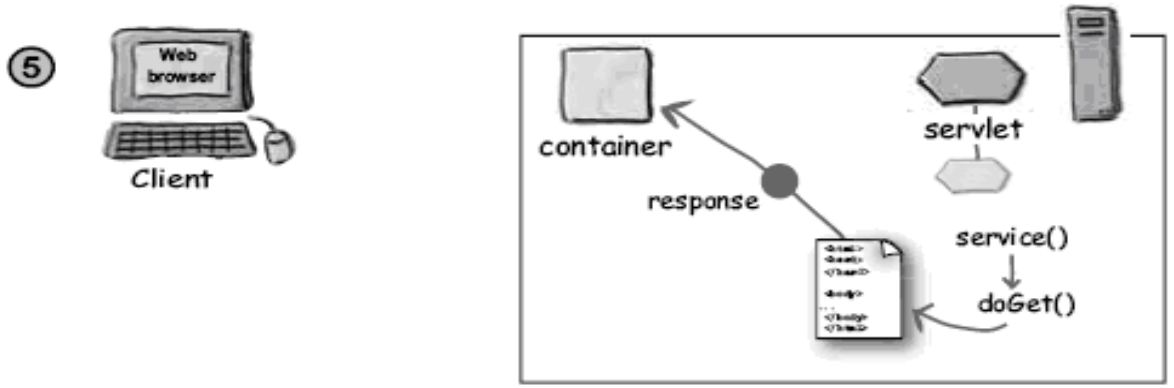
De asemenea, crearea obiectelor care incapsuleaza cererea si raspunsul HTTP, pasarea acestora catre metoda service(), gestionarea variabilelor CGI precum si multe alte servicii sunt realizate de catre container la momentul potrivit


HTTP request
GET
Web browser
Client
container
servlet
User clicks a link that has a URL to a servlet instead of a static page.
Web browser
Client
container
servlet
request
response
The container "sees" that the request is for a servlet, so the container creates two objects:
1) HttpServletResponse
2) HttpServletRequest
Web browser
Client
container
request
response
servlet
thread
The container finds the correct servlet based on the URL in the request, creates or allocates a thread for that request, and passes the request and response objects to the servlet thread.

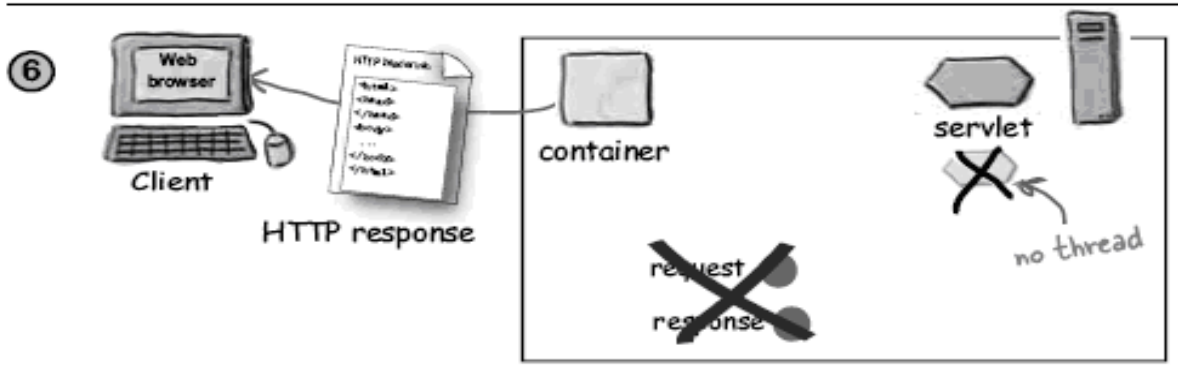


The container calls the servlet's service() method. Depending on the type of request, the service() method calls either the doGet() or doPost() method.

For this example, we'll assume the request was an HTTP GET.

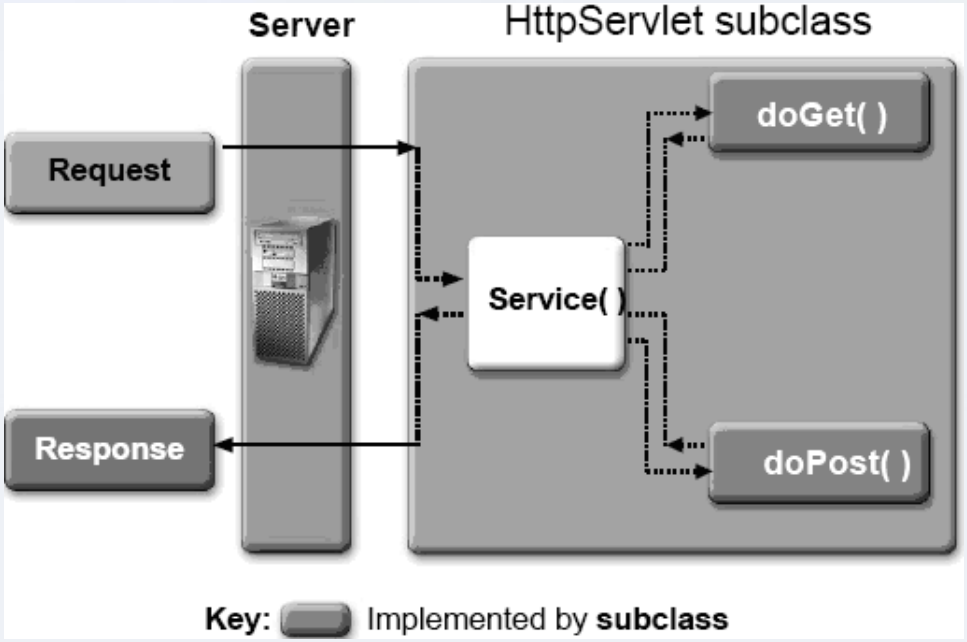


The doGet() method generates the dynamic page and stuffs the page into the response object. Remember, the container still has a reference to the response object!



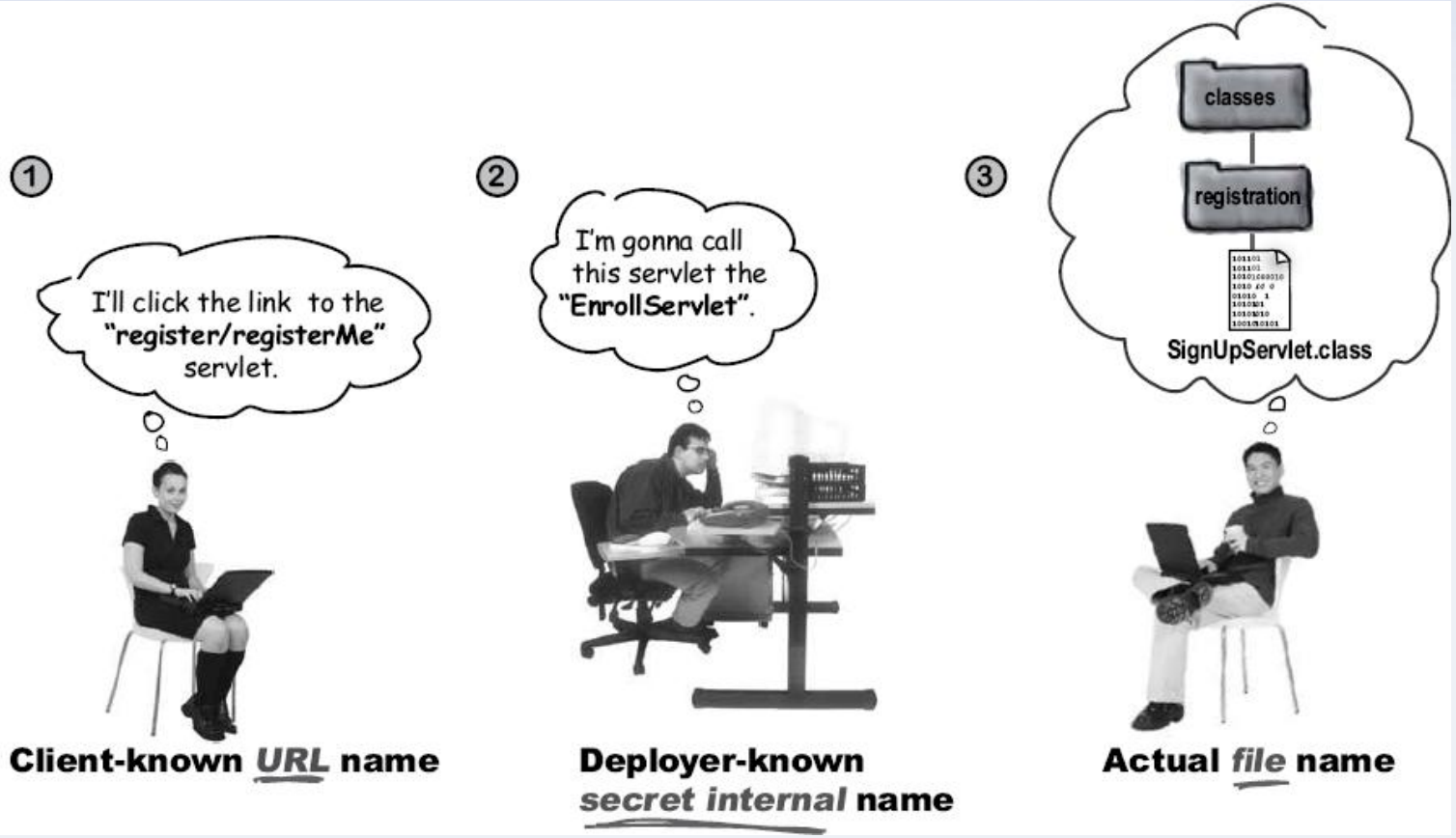
The thread completes, the container converts the response object into an HTTP response, sends it back to the client, then deletes the request and response objects.

Metoda service() mostenita de la clasa **HttpServlet are o implementare generica** dar care **se recomanda sa fie pastrata, deoarece ea identifica tipul de metoda a cererii HTTP si apeleaza metoda potrivita** (**doPost()** in cazul metodei POST, **doGet()** in cazul metodei GET, etc.).



Pentru a putea fi accesat servlet-ul, **clientului** trebuie sa i se furnizeze o **adresa URL** care **in general difera de adresa (calea) la care se afla cu adevarat fisierul cu co**

Servlet mapping



1. **<servlet>**
   maps internal name to fully-qualified class name
2. **<servlet-mapping>**
   maps internal name to public URL name

Adresa URL (1) este asociata prin intermediul unui alias (2) dat de programator cu calea completa necesara identificarii fisierului sursa (3) prin codul XML scris intr-un fisier (web.xml) denumit *deployment descriptor* (descriptor de desfasurare/instalare - DD

Servlet mapping

```
<web-app>
    <servlet>
        <servlet-name>numeintern</servlet-name>
        <servlet-class>ClasaServlet</servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>numeintern</servlet-name>
        <url-pattern>/ServletAccesServiciu</url-pattern>
    </servlet-mapping>
</web-app>
```

•**existenta unui servlet** cu numele **ClasaServlet** (al carui cod sursa se afla in **ClasaServlet.java** iar codul compilat in **ClasaServlet.class**) cu ajutorul tag-urilor XML **<servlet>** si **<servlet-class>**

•asocierea servlet-ului **ClasaServlet** cu **aliasul numeintern** (prin intermediul tag-ului **<servlet-name>**)

•asocierea **aliasului numeintern** cu **formatul utilizat de client pentru URL /ServletAccesServiciu** (prin intermediul tag-urilor **<servlet-mapping>** si <**url-pattern**>)

Rolurile pe care componentele Web (servlet-urile dar si paginile JSP)

1) **primirea cererilor** HTTP **de la client** (sub forma de obiecte HttpServletRequest) si eventual **utilizarea parametrilor obtinuti din formularul care a generat cererea**,

2) **executarea sarcinilor aplicatiei** (denumite ***business logic***) fie **direct** fie prin **delegarea catre o alta componenta**:
   - alte componente **Web** – servlet-uri sau pagini JSP,
   -componente **business** locale (JavaBeans) sau distribuite (Enterprise JavaBeans),

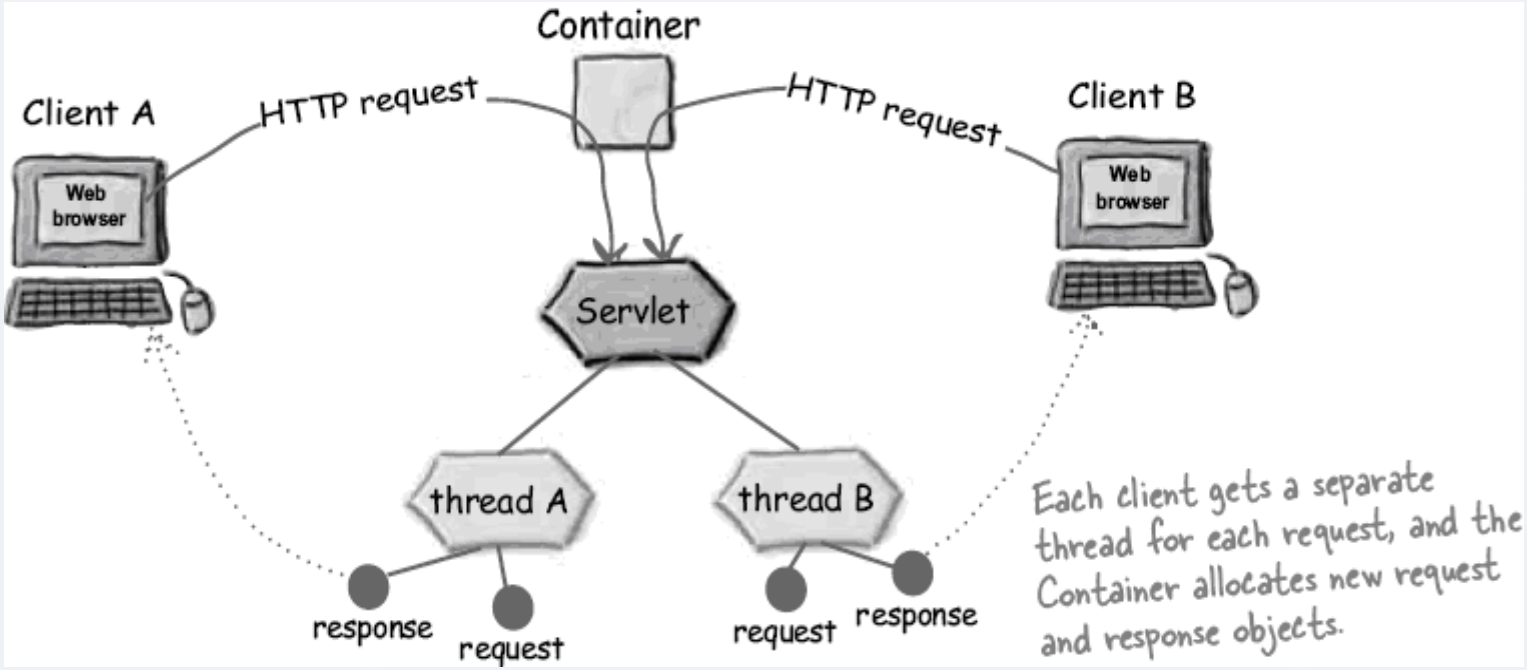
3) **generarea dinamica a continutului** si **trimiterea lui in raspunsul catre client prin intermediul raspunsurilor HTTP** (sub forma de obiecte HttpServletResponse).

Template al servlet-urilor Java

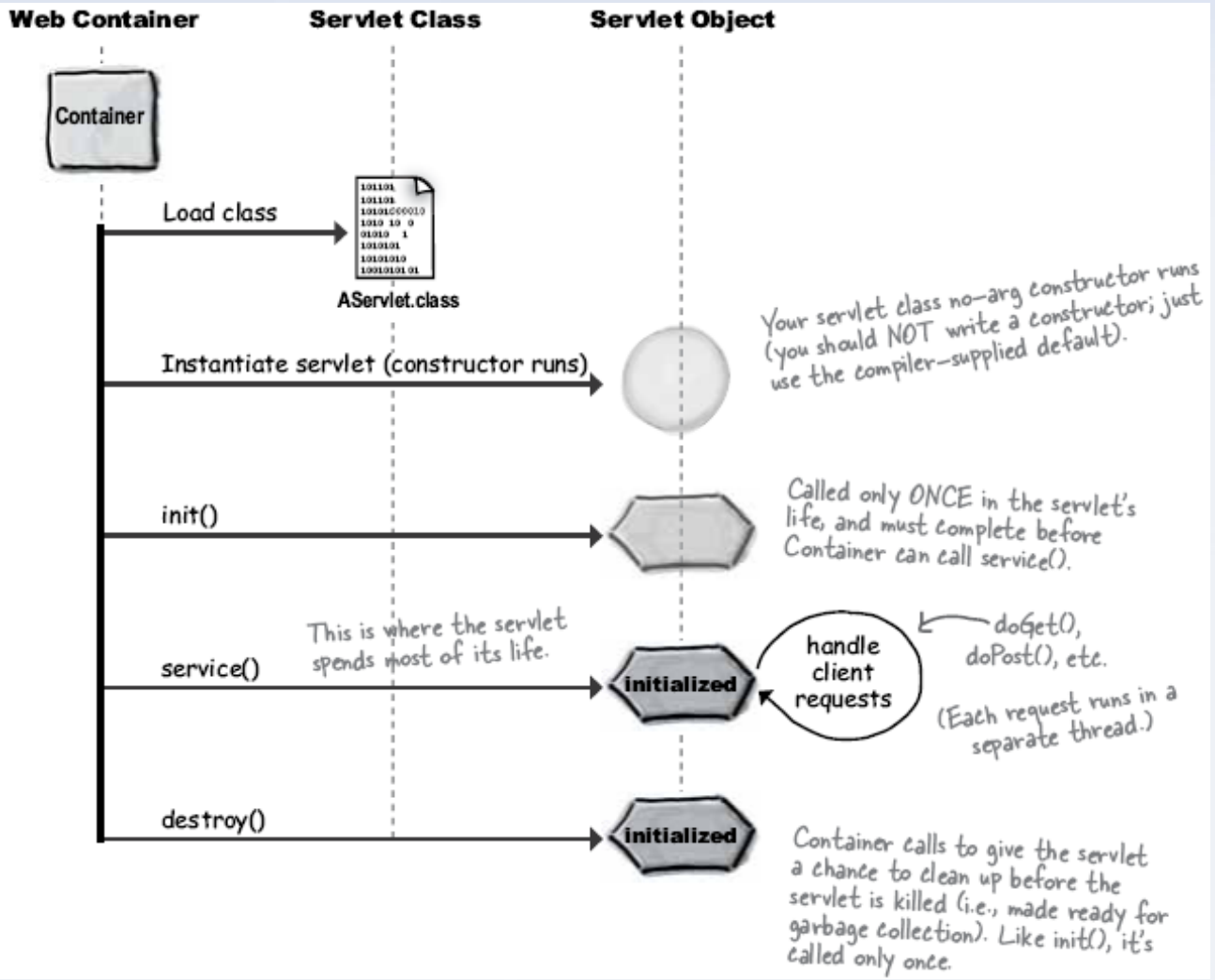
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class ClasaServlet extends HttpServlet {

  protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    processRequest(request, response);
  }
  protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    processRequest(request, response);
  }

  protected void processRequest(HttpServletRequest request,
            HttpServletResponse response) throws ServletException, IOException {
    // Stabilirea tipului de continut
    response.setContentType("text/html");
    // Utilizare "request" pentru a citi antetele HTTP primite (de ex. cookies)
    // si datele formularului HTML (pe care utilizatorul le-a introdus si trimis)

    // Utilizare "response" pentru a specifica linia si antetele raspunsului HTTP
    // (tipul de continut, cookies).

    PrintWriter out = response.getWriter();
    // Utilizare "out" pentru a trimite continut HTML catre browser
  }
}
```

Protocolul HTTP nu are stari (este *stateless*) asa incat **serverul HTTP nu retine informatii privind cererile anterioare**.

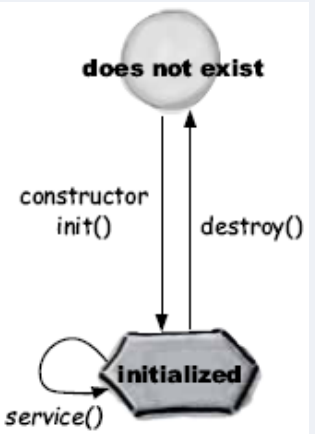
Servlet-urile sa fie accesate eficient de catre mai multi clienti in acelasi timp (concurente) containerul de servlet-uri formeaza un asa-numit ***thread pool*** cu fire de executie ale servlet-ului



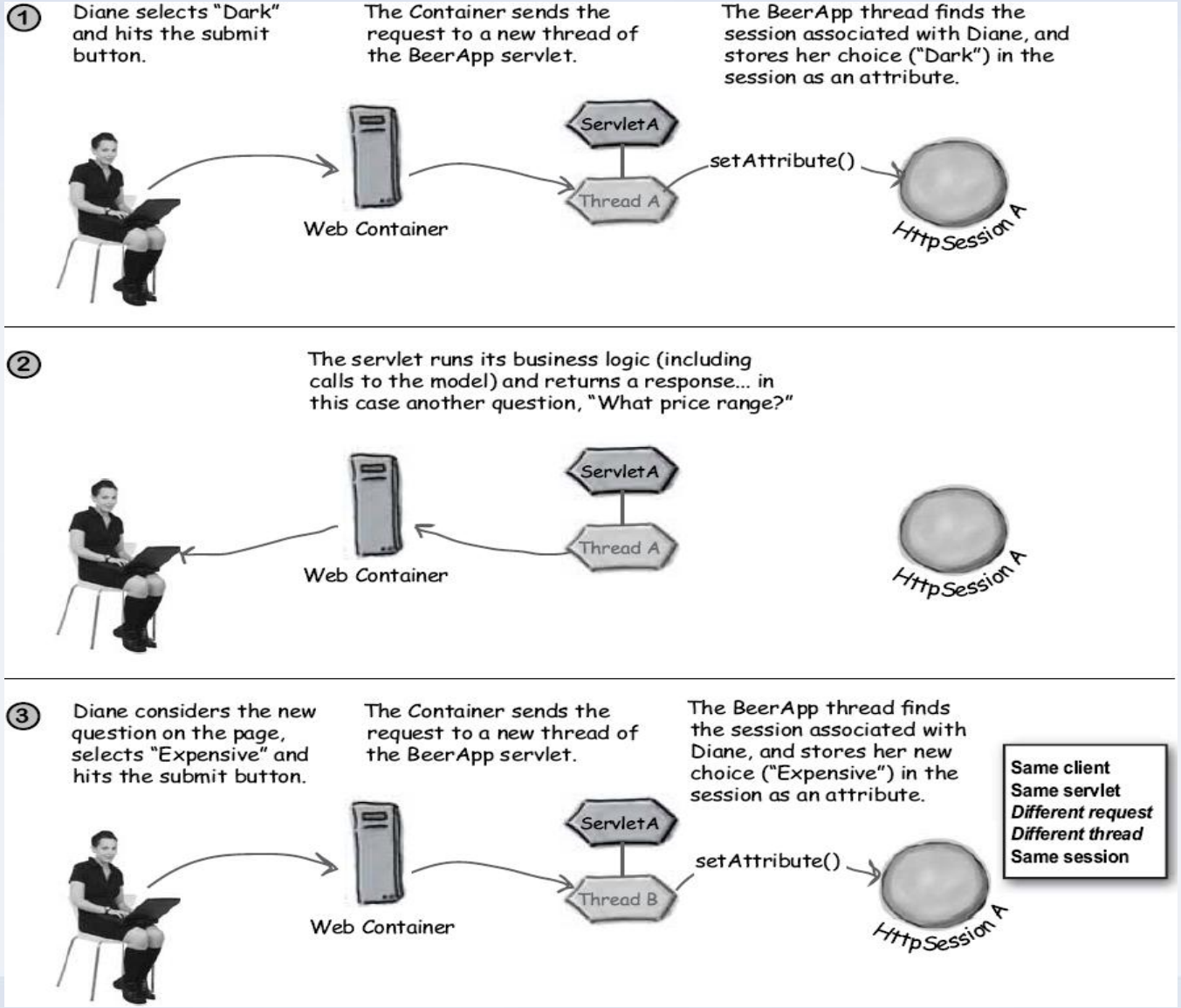
Ciclul de viata al servlet-urilor

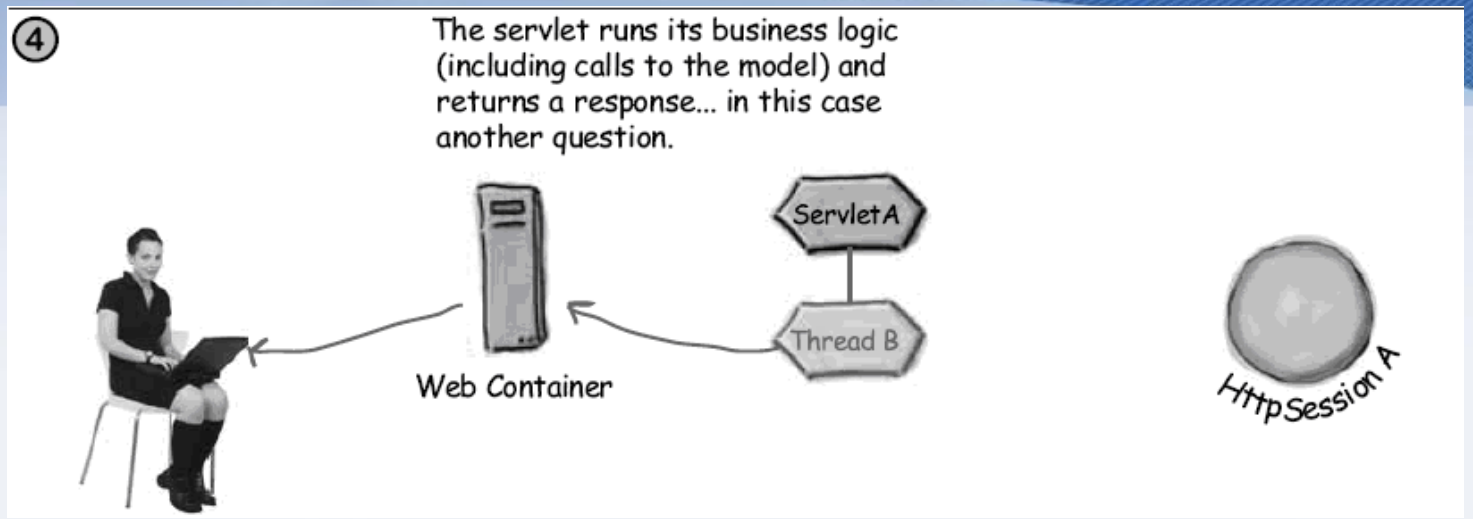


Ciclul de viata al servlet-urilor este gestionat de container si incepe cu crearea servlet-ului prin apelul constructorului implicit (fara parametri si fara cod) urmat de apelul metodei init(), ceea ce conduce servlet-ul in starea initializat, in care accepta si trateaza apelurile service() (ca fire de executie separate pentru fiecare client), stare din care iese prin apelul metodei destroy() de catre container

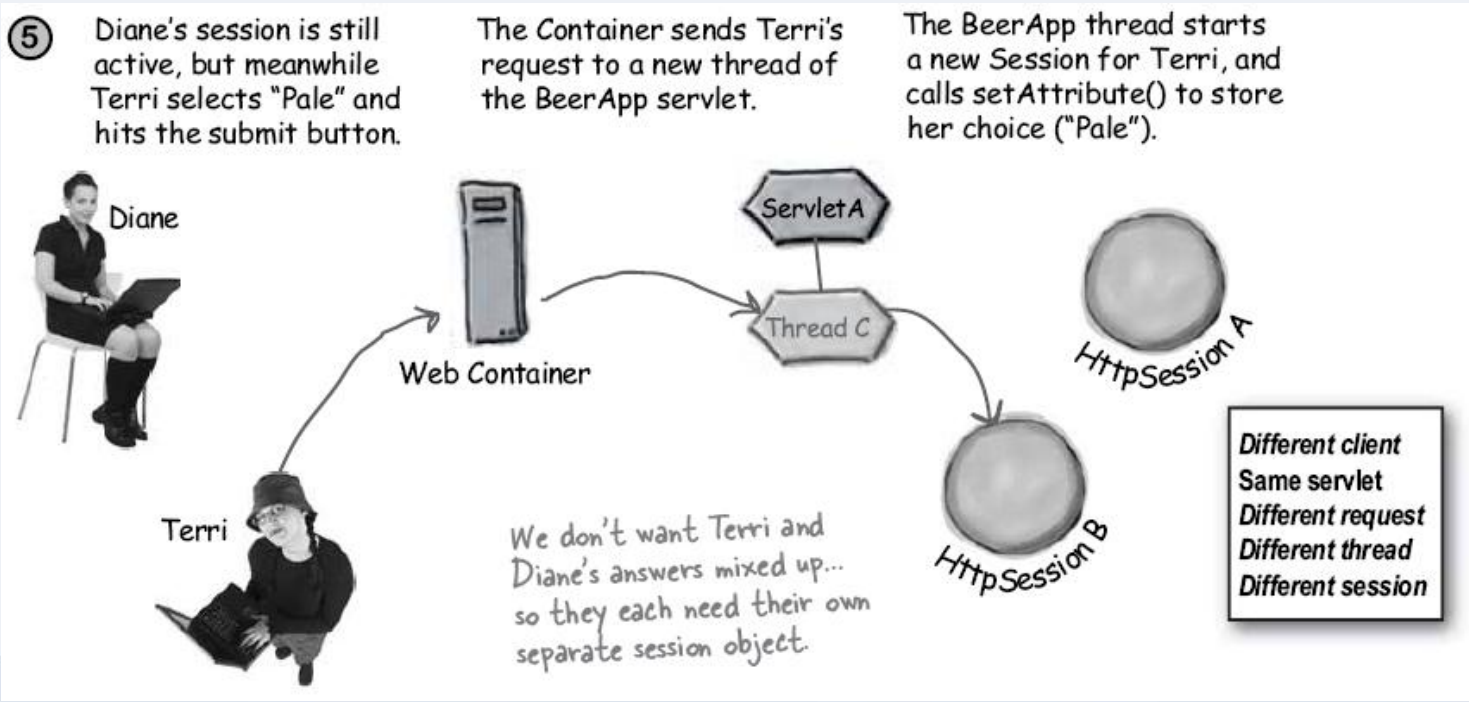


Modul de lucru cu sesiunile HTTP

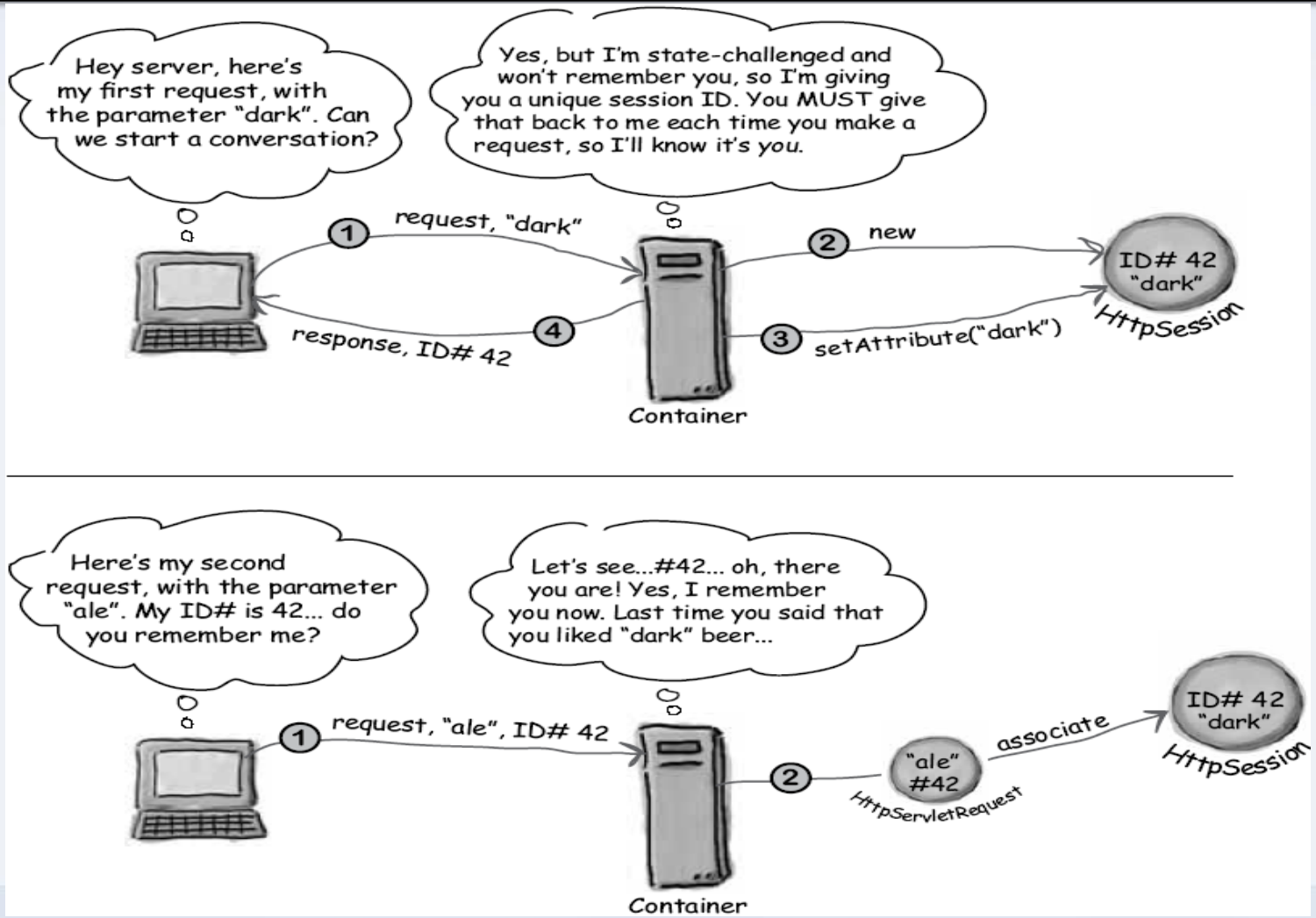




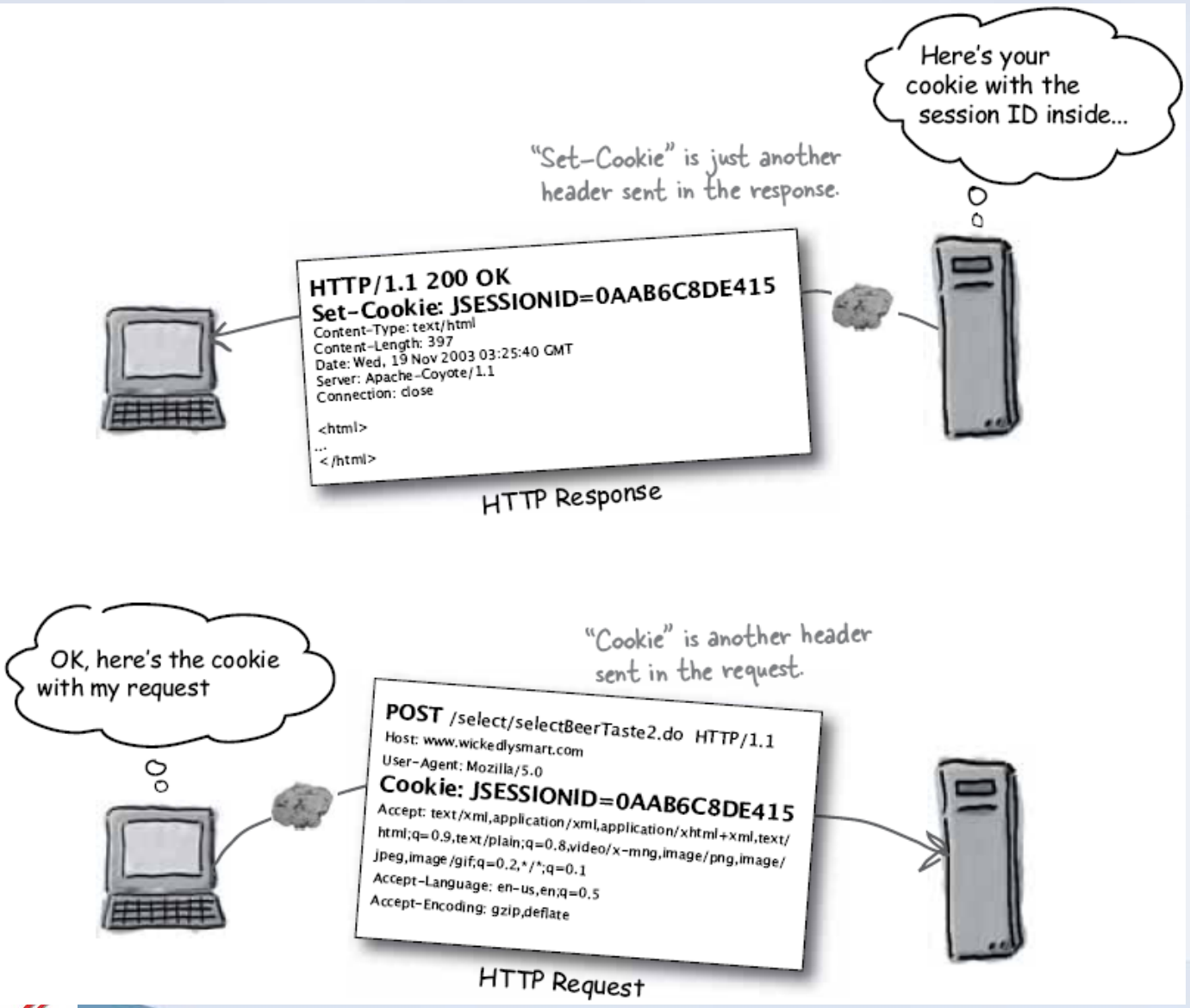
Daca un alt client acceseaza acelasi servlet, el primeste o alta sesiune:



Fiecare client primeste o sesiune identificata printr-un *session ID*:



Modul de lucru cu sesiunile HTTP



Session ID este trimis de container spre browser si regasit cand browserul acceseaza din nou servlet-ul:

Modul de lucru cu sesiunile HTTP

Varianta in care sesiunea este creata daca nu exista deja

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("test session attributes<br>");


    HttpSession session = request.getSession();


    if (session.isNew()) {
       out.println("This is a new session.");
    } else {
       out.println("Welcome back!");
    }
}
```

getSession() returns a session no matter what... but you can't tell if it's a new session unless you ask the session.

isNew() returns true if the client has not yet responded with this session ID.

Modul de lucru cu sesiunile HTTP

Varianta in care se foloseste doar o sesiune preexistenta:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("test sessions<br>");

    HttpSession session = request.getSession(false);

    if (session==null) {

       out.println("no session was available");
       out.println("making one...");
       session = request.getSession();
    } else {
       out.println("there was a session!");
    }
}
```

Passing "false" means the method returns a pre-existing session, or null if there was no session associated with this client.

Now we can test for whether there was already a session (the no-arg getSession() would NEVER return null).

Here we KNOW we're making a new session.

clasa care seteaza un cookie

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class CookieTest extends HttpServlet {
   public void doPost(HttpServletRequest request, HttpServletResponse response)
                                                 throws IOException, ServletException {
      response.setContentType("text/html");

      String name = request.getParameter("username");

      Cookie cookie = new Cookie("username", name);

      cookie.setMaxAge(30*60);

      response.addCookie(cookie);



      RequestDispatcher view = request.getRequestDispatcher("cookieresult.jsp");
      view.forward(request, response);
   }
}
```

Get the user's name submitted in the form.

Make a new cookie so store the user's name.

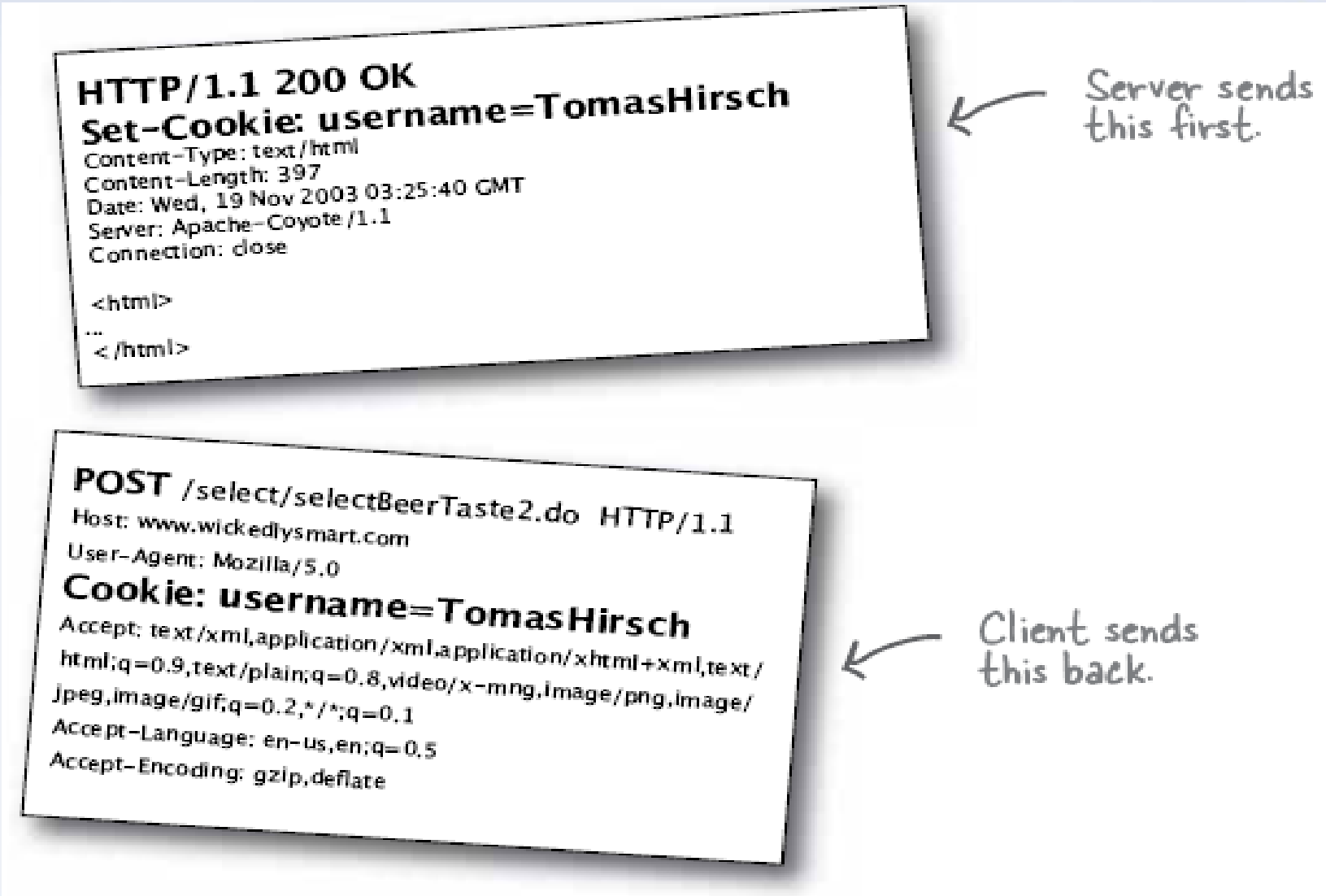
Keep it alive on the client for 30 minutes.

Add the cookie as a "Set-Cookie" response header.

Let a JSP make the response page.

Lucrul cu cookie-uri

Lucrul cu cookie-uri



clasa care seteaza un cookie

```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class CheckCookie extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                throws IOException, ServletException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      Cookie[] cookies = request.getCookies();

      for (int i = 0; i < cookies.length; i++) {
         Cookie cookie = cookies[i];
         if (cookie.getName().equals("username")) {
            String userName = cookie.getValue();
            out.println("Hello " + userName);
            break;
         }
      }
   }
}
```

Get the cookies from the request.

Loop through the cookie array looking for a cookie named "username". If there is one, get the value and print it.

Tehnologia Java Servlet

HTTP Cookies - Servlet care citeste un *cookie*

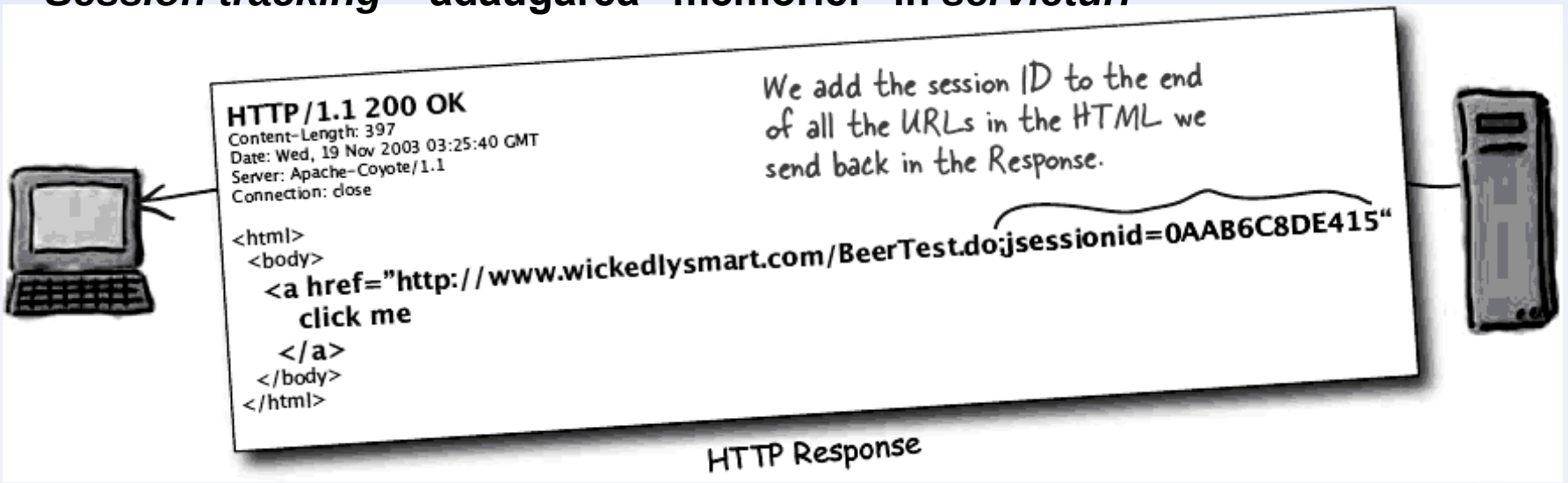
```
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
public class CheckCookie extends HttpServlet {
   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                          throws IOException, ServletException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();

      Cookie[] cookies = request.getCookies();

      for (int i = 0; i < cookies.length; i++) {
         Cookie cookie = cookies[i];
         if (cookie.getName().equals("username")) {
            String userName = cookie.getValue();
            out.println("Hello " + userName);
            break;
         }
      }
   }
}
```

Get the cookies from the request.

Loop through the cookie array looking for a cookie named "username". If there is one, get the value and print it.

Tehnologia Java Servlet

Session tracking* – adaugarea "memoriei" in *servleturi




```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();

    out.println("<html><body>");
    out.println("<a href=\"" + response.encodeURL("/BeerTest.do") + "\">click me</a>");
    out.println("</body></html>");
}
```

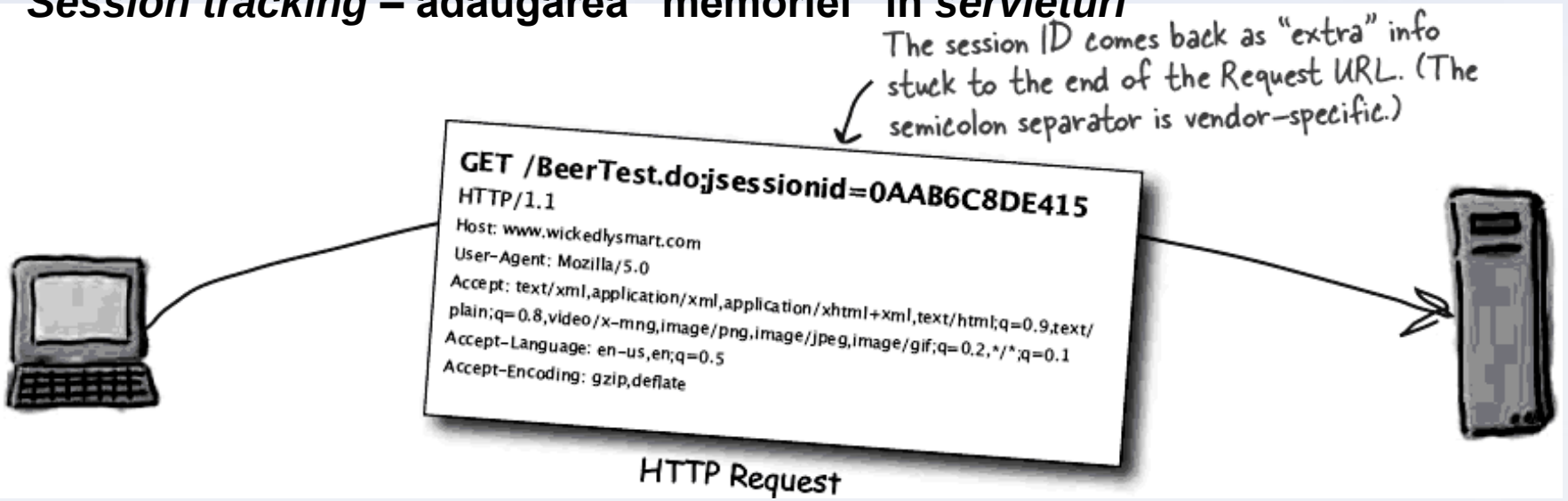
get a session

Add the extra session ID info to this URL.

Pentru cazul in care sunt **dezactivate *cookieurile*** se poate folosi **rescrierea URL-ului**

Tehnologia Java Servlet

Session tracking* – adaugarea "memoriei" in *servleturi




```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    HttpSession session = request.getSession();

    out.println("<html><body>");
    out.println("<a href=\"" + response.encodeURL("/BeerTest.do") + "\">click me</a>");
    out.println("</body></html>");
}
```

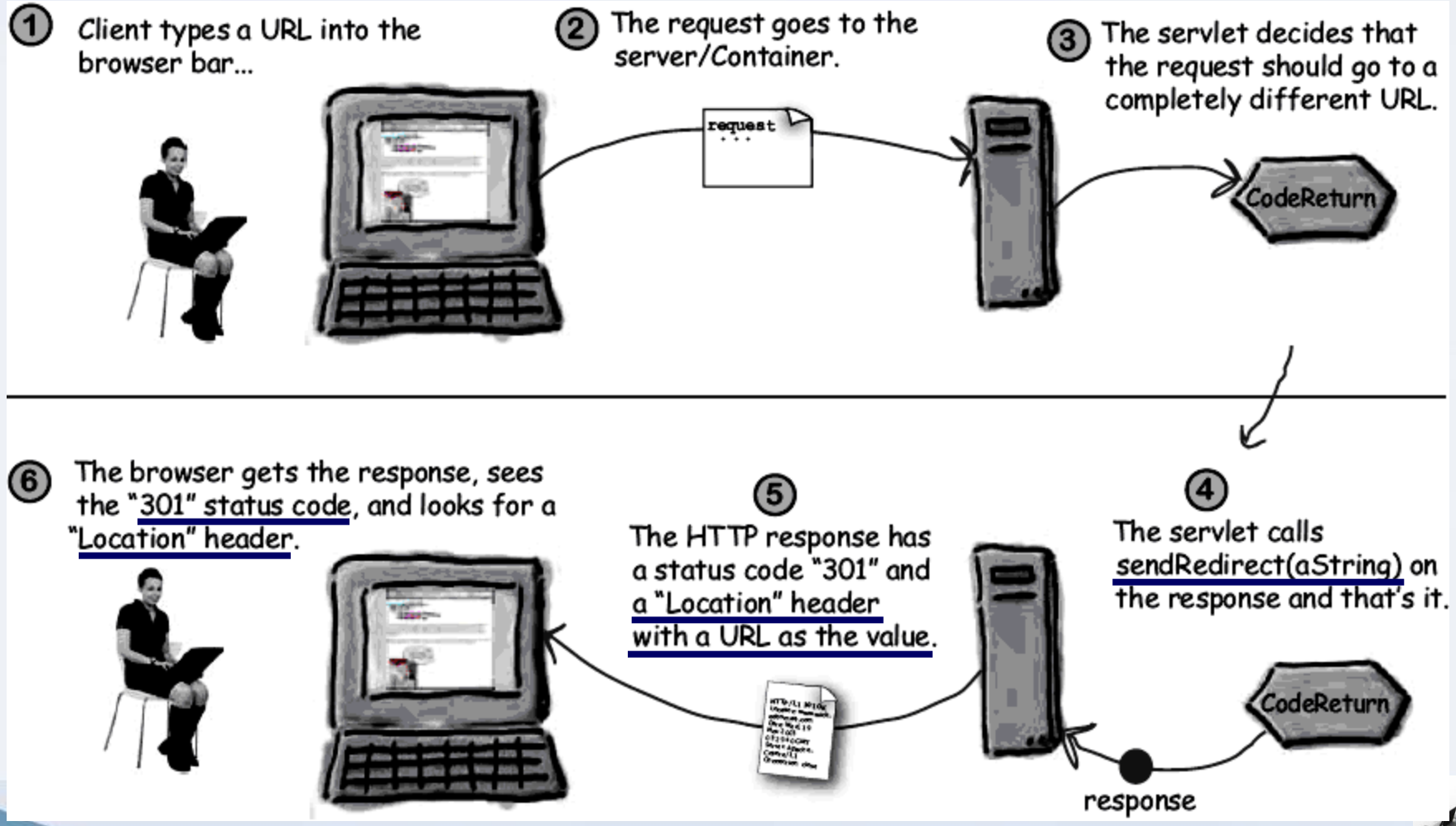
get a session

Add the extra session ID info to this URL.

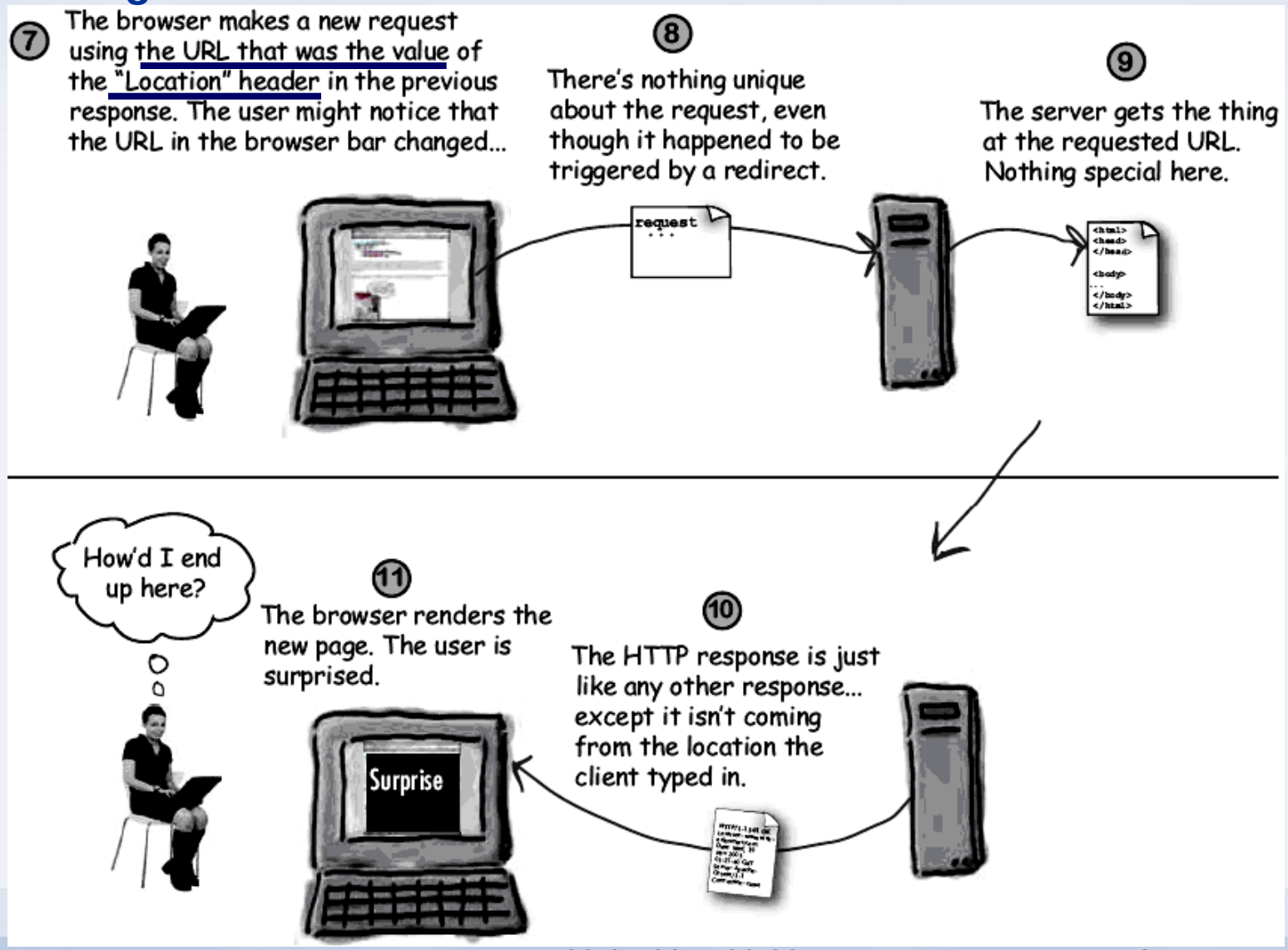
Pentru cazul in care sunt **dezactivate *cookieurile*** se poate folosi **rescrierea URL-ului**

Tehnologia Java Servlet

***Servleturile* pot redirecta *browserul* catre alt URL folosind sendRedirect()**

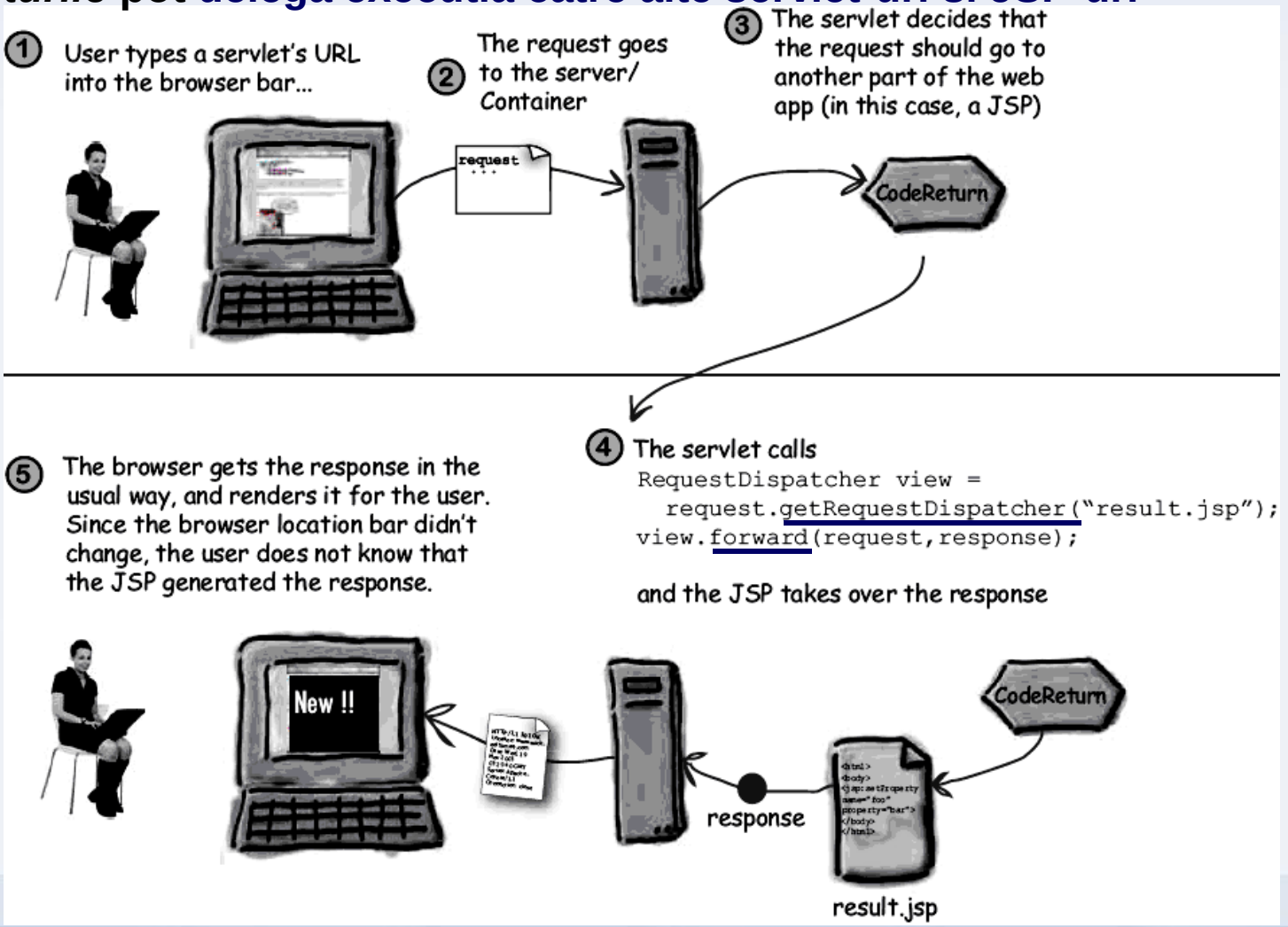


Tehnologia Java Servlet



Tehnologia Java Servlet

***Servleturile* pot delega executia catre alte servlet-uri si JSP-uri**



Tehnologia Java Servlet

Pentru a-si transmite informatii (inclusiv parametri de initializare)

- ***servleturile*** si **paginile JSP** care **deleaga executia**

  - pot crea **atribute ale cererii**

    - in cazul **paginilor JSP** un **obiect implicit** numit "**request**"

    - in cazul **servleturilor** obiectul **request** de tip **HTTPServletRequest**

  carora le dau ca valori **informatiile de transmis**

Sintaxa pentru **atasarea informatiilor** obiectului **cerere**

```
request.setAttribute("raspuns", "Comanda a fost trimisa");
```

Sintaxa pentru **obtinerea informatiilor** de la obiectul **cerere**

```
String raspuns = request.getAttribute("raspuns");
```

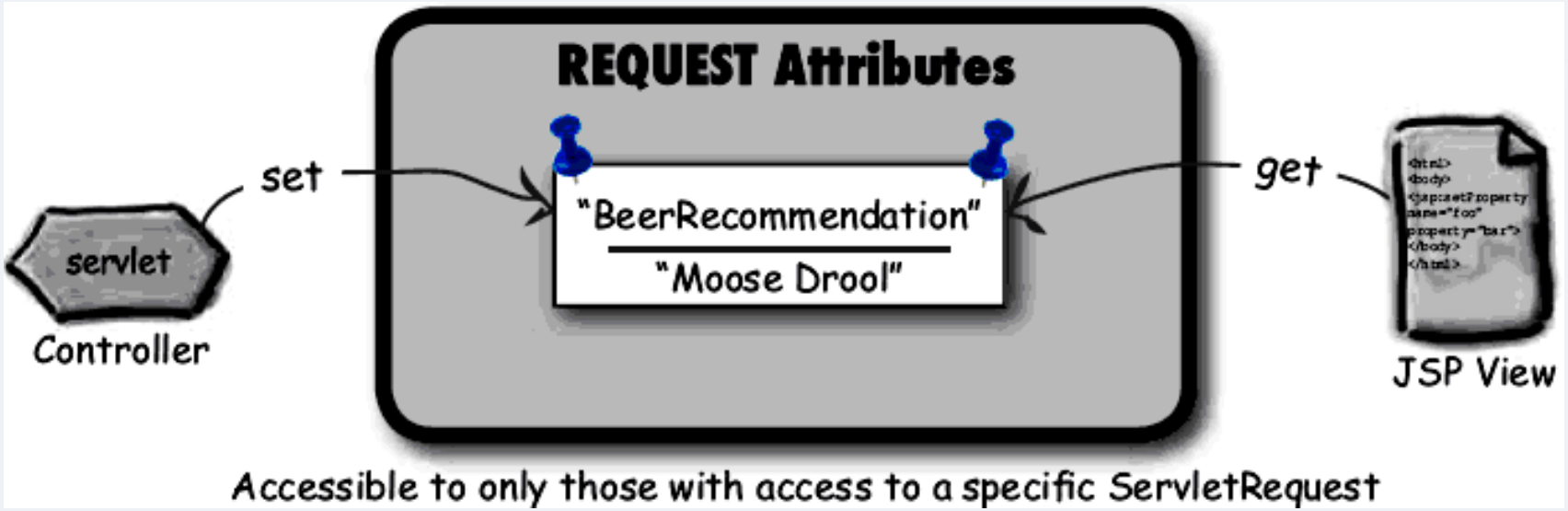
Tehnologia Java Servlet

Sintaxa pentru **atasarea informatiilor** obiectului **cerere**

```
request.setAttribute("raspuns", "Comanda a fost trimisa");
```

Sintaxa pentru **obtinerea informatiilor** de la obiectul **cerere**

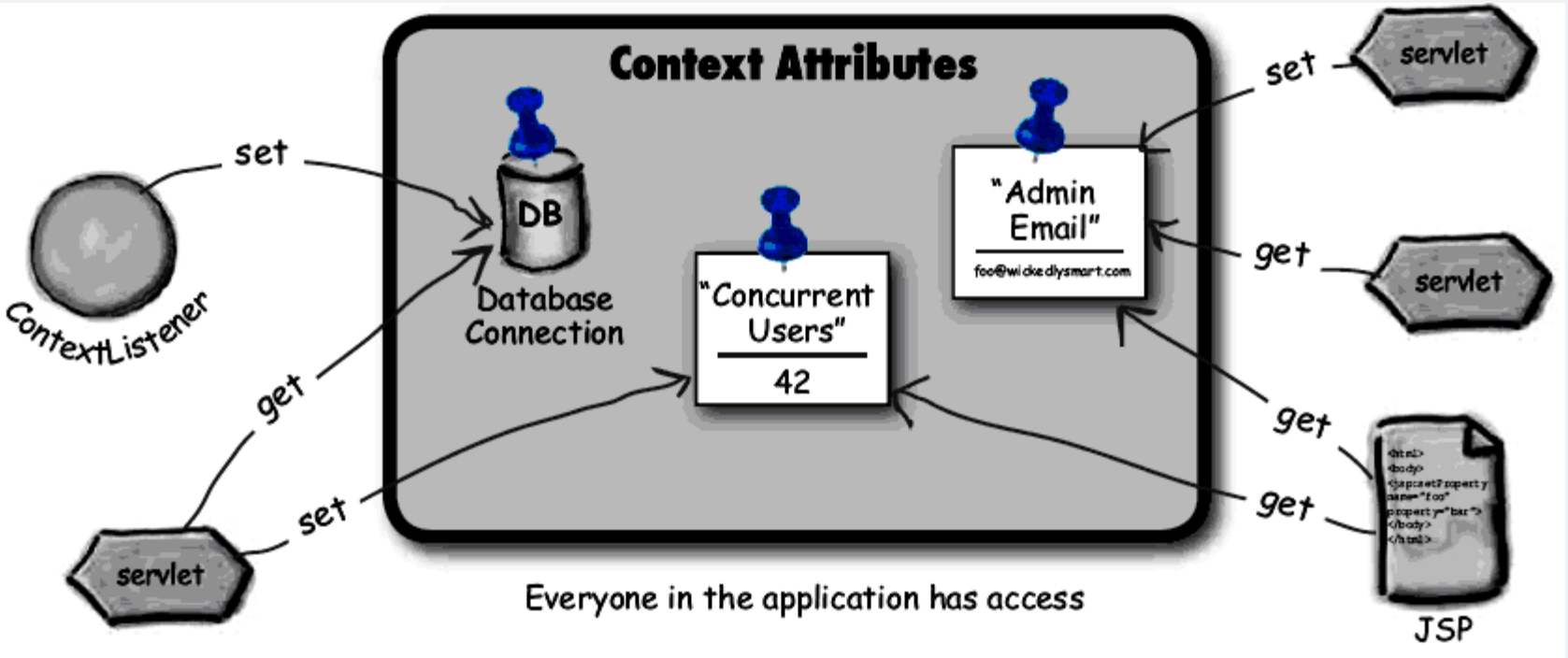
```
String raspuns = request.getAttribute("raspuns");
```



Tehnologia Java Servlet

Pentru comunicatia intre toate componentele Web ale unei aplicatii Web formata din mai multe **servleturi** si **pagini JSP**

- pot fi de asemenea definite atribute, numite **atribute context**



Operatii de I/O in cadrul Servlet-ului

```
public class CodeReturn extends HttpServlet {

   public void doGet(HttpServletRequest request, HttpServletResponse response)
                                        throws IOException, ServletException {

      response.setContentType("application/jar");

      ServletContext ctx = getServletContext();
      InputStream is = ctx.getResourceAsStream("/bookCode.jar");

      int read = 0;
      byte[] bytes = new byte[1024];

      OutputStream os = response.getOutputStream();
      while ((read = is.read(bytes)) != -1) {
         os.write(bytes, 0, read);
      }
      os.flush();
      os.close();
   }
}
```

We want the browser to recognize that this is a JAR, not HTML, so we set the content type to "application/jar".

This just says, "give me an input stream for the resource named bookCode.jar".

Here's the key part, but it's just plain old I/O!! Nothing special, just read the JAR bytes, then write the bytes to the output stream that we get from the response object.

Ex: Servlet-ul foloseste fluxuri pentru a citi continutul unui fisier .jar (bookCode.jar) si a trimite continutul lui prin intermediul raspunsului HTTP (pentru asta este deschis fisierul sursa cu getResourceAsStream() si este stabilit tipul continutului cu setContentType() la "application/jar"):

Operatii de I/O in cadrul Servlet-ului

Servlet-urile pot deschide catre raspunsul HTTP fluxuri de caractere:

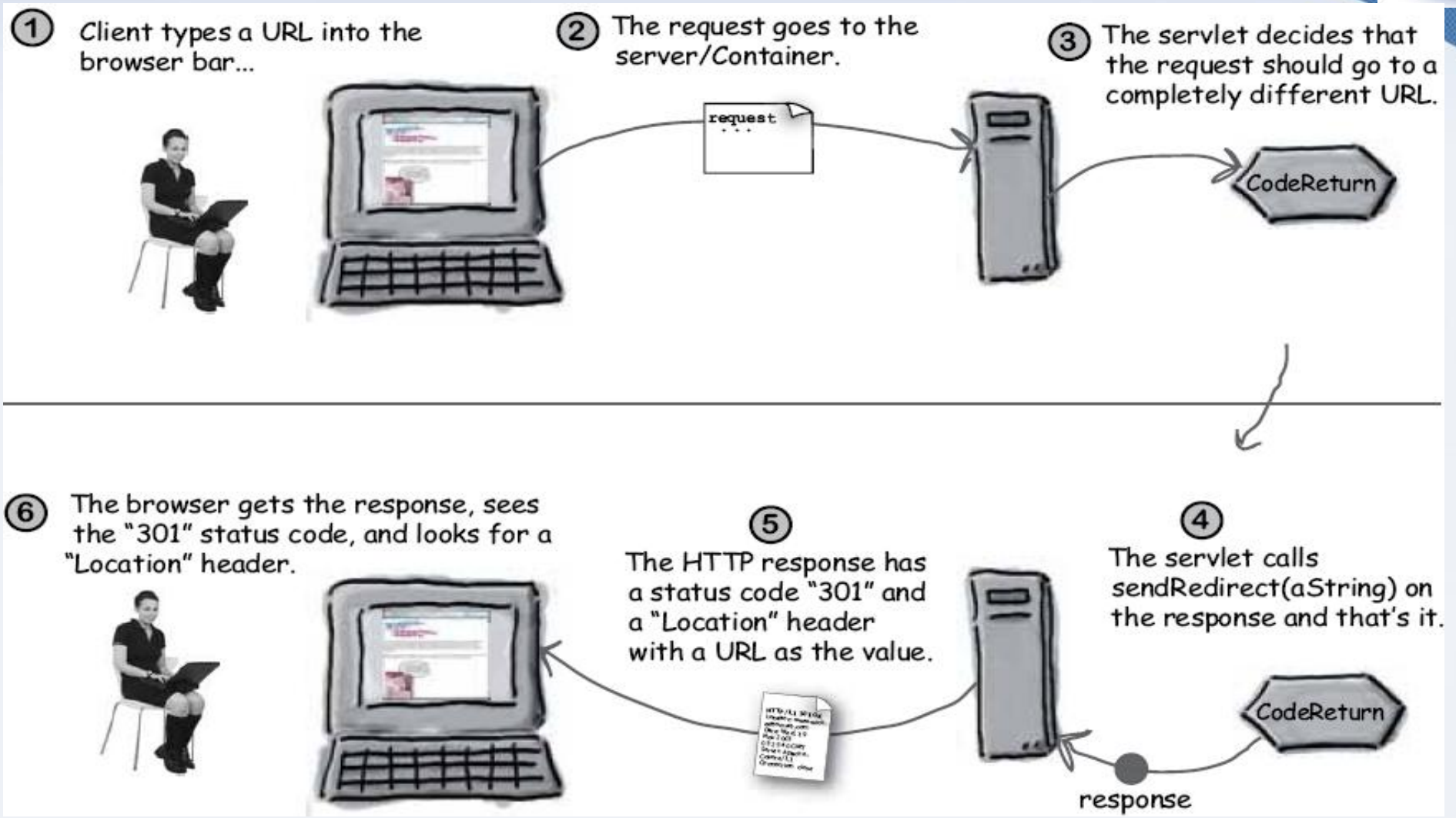
```
PrintWriter writer = response.getWriter();

writer.println("some text and HTML");
```

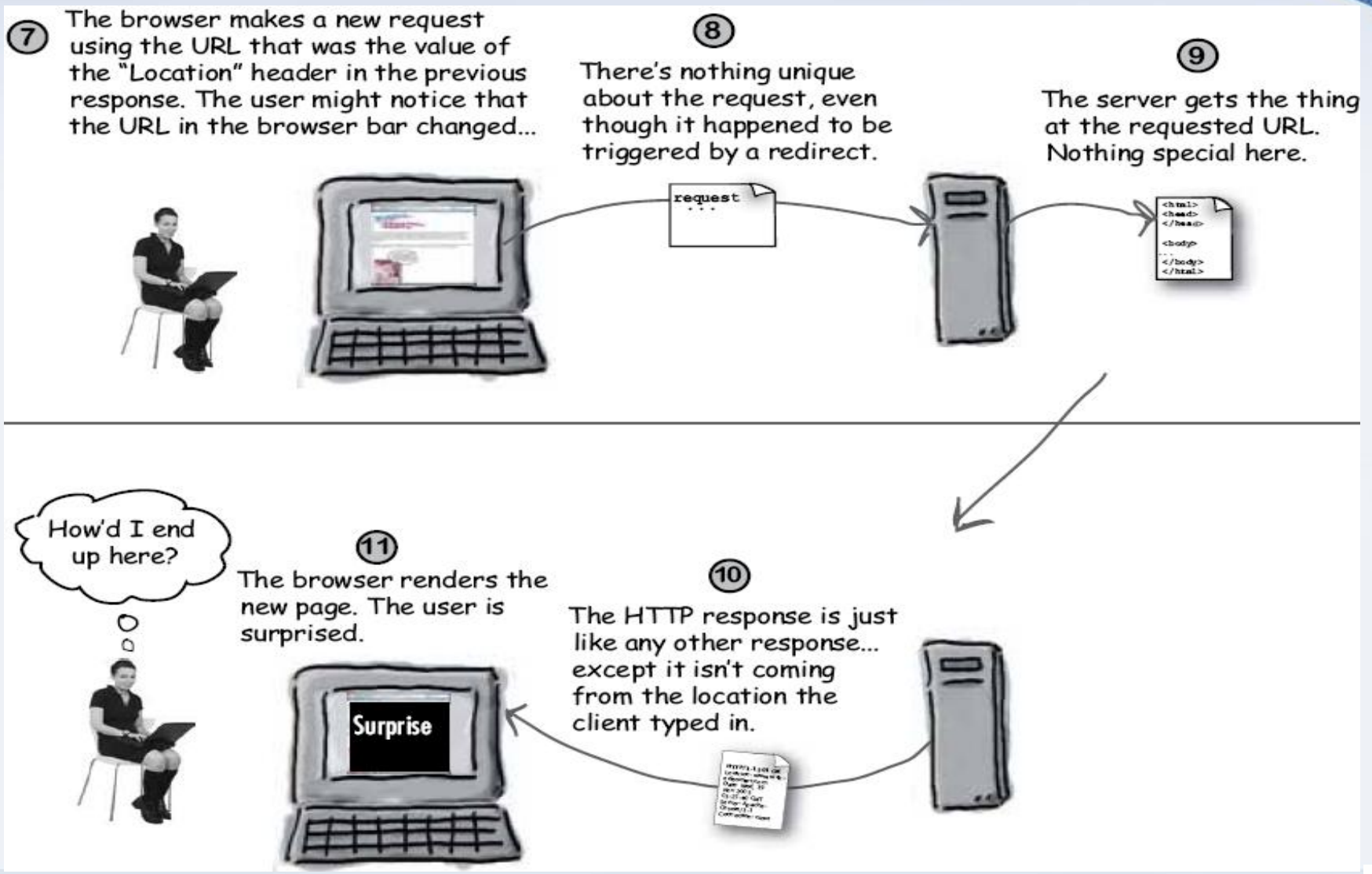
sau fluxuri de octeti:

```
ServletOutputStream out = response.getOutputStream();

out.write(aByteArray);
```

```
import java.io.*;
import java.net.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class ServletOrarFinal extends HttpServlet {
   protected void processRequest(HttpServletRequest request,
         HttpServletResponse response) throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();
      // Generarea formularului pentru accesul recursiv la servicii
      out.println("<html>");
      out.println("<head>");
      out.println("<title>Acces orar</title>");
      out.println("</head>");
      out.println("<body>");
      out.println("<h1>Acces orar (forma finala) - generat de servlet</h1>");
      out.println("<hr><form name=\"input\" action=\"AccesFinal\""
               + " method=\"get\">");
      out.println("<input type=\"radio\" name=\"zi\" checked=\"checked\""
               + " value=\"0\"> Luni");
      out.println("<br> <input type=\"radio\" name=\"zi\" value=\"1\"> Marti");
      out.println("<br> <input type=\"radio\" name=\"zi\" value=\"2\"> Miercuri");
      out.println("<br> <input type=\"radio\" name=\"zi\" value=\"3\"> Joi");
      out.println("<br> <input type=\"radio\" name=\"zi\" value=\"4\"> Vineri");
      out.println("<br> <input type=\"radio\" name=\"zi\" value=\"5\"> Sambata");
      out.println("<br> <input type=\"radio\" name=\"zi\" value=\"6\"> Duminica");
      out.println("<hr>");
      out.println("<input type=\"radio\" name=\"serviciu\" checked=\"checked\""
               + " value=\"getOrar\"> Obtinere orar");
      out.println("<br><input type=\"radio\" name=\"serviciu\" value=\"setOrar\">"
               + " Modificare orar");
      out.println("<input type=\"text\" name=\"modificare\" value=\"\">");
      out.println("<hr><input type=\"submit\" value=\"Trimite\">");
      out.println("</form><hr>");
```

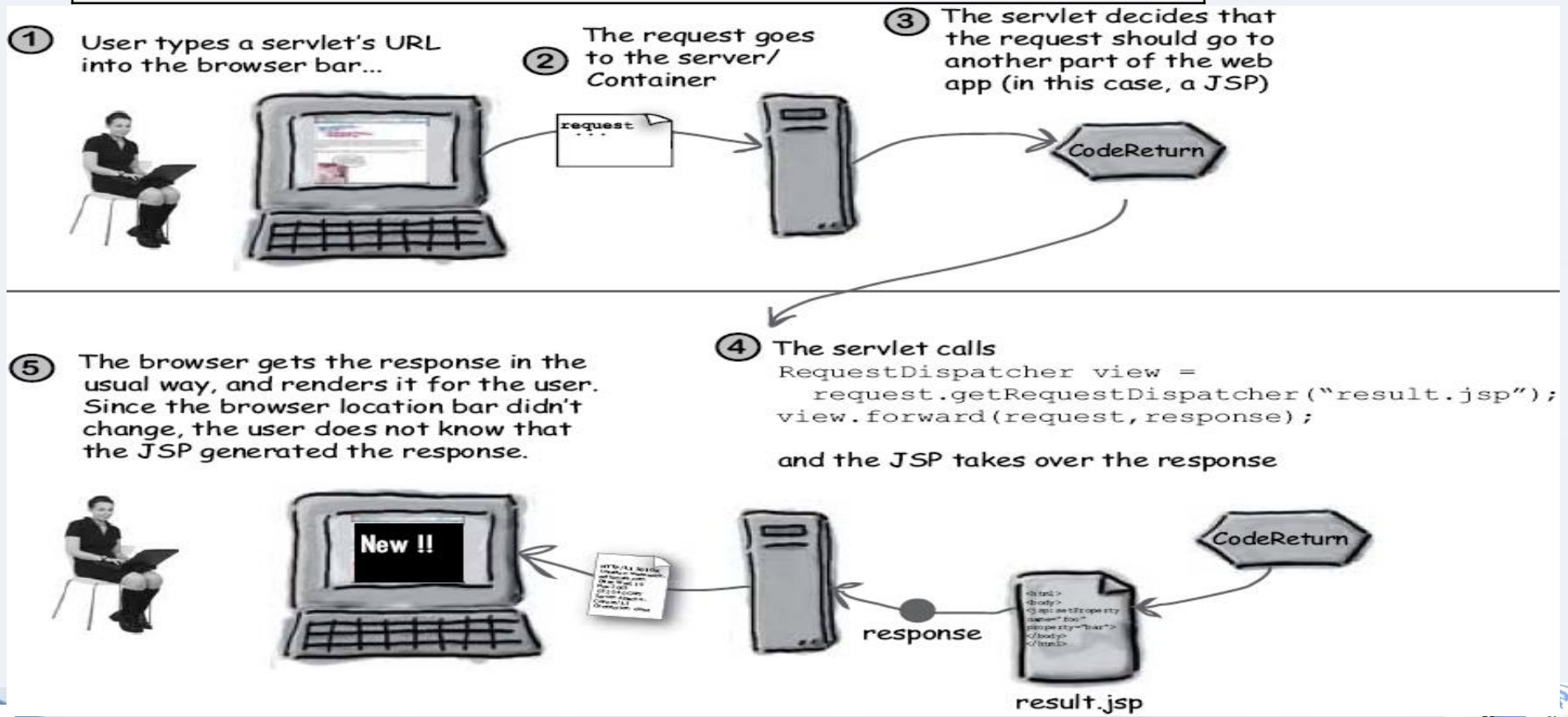
```
      // Transformarea obiectului orar in atribut al sesiunii curente pentru
      // salvarea starii lui
      HttpSession ses = request.getSession();
      Orar orar = (Orar) ses.getAttribute("orar");
      if (orar == null) { // Daca nu exista orarul salvat ca atribut al sesiunii
         orar = new Orar();
         ses.setAttribute("orar", orar);
      }
      // Obtinerea parametrilor introdusi de utilizator in formular
      int zi = Integer.parseInt(request.getParameter("zi"));

      // Daca serviciu cerut e obtinere orar
      if (request.getParameter("serviciu").equals("getOrar")) {
         out.println("<b>Orarul cerut:</b> <br>" + orar.getOrar(zi));
      }
      // Daca serviciu cerut e modificare orar
      else if (request.getParameter("serviciu").equals("setOrar")) {
         String modificare = request.getParameter("modificare");
         orar.setOrar(zi, modificare);
         out.println("<b>Modificarea ceruta:</b> <br>" + orar.getOrar(zi));
      }
      out.println("</body>");
      out.println("</html>");
      out.close();
   }
   protected void doGet(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
   }
   protected void doPost(HttpServletRequest request, HttpServletResponse response)
         throws ServletException, IOException {
      processRequest(request, response);
   }
}
```


1 Client types a URL into the browser bar...
2 The request goes to the server/Container.
request
3 The servlet decides that the request should go to a completely different URL.
CodeReturn
4 The servlet calls sendRedirect(aString) on the response and that's it.
CodeReturn
response
5 The HTTP response has a status code "301" and a "Location" header with a URL as the value.
6 The browser gets the response, sees the "301" status code, and looks for a "Location" header.


7
The browser makes a new request using the URL that was the value of the "Location" header in the previous response. The user might notice that the URL in the browser bar changed...
8
There's nothing unique about the request, even though it happened to be triggered by a redirect.
9
The server gets the thing at the requested URL. Nothing special here.
request
How'd I end up here?
11
The browser renders the new page. The user is surprised.
Surprise
10
The HTTP response is just like any other response... except it isn't coming from the location the client typed in.

Servlet-urile si JSP-urile pot delega executia catre alte servlet-uri si JSP-uri folosind interfata RequestDispatcher si o sintaxa de genul:

```
RequestDispatcher view = request.getRequestDispatcher("pagina.jsp");
view.forward(request, response);
```



Pentru a fi initializate individual servlet-urile pot obtine informatii pasate de programator in descriptorul de desfasurare web.xml:

```
<servlet>
    <servlet-name>BeerParamTests</servlet-name>
    <servlet-class>TestInitParams</servlet-class>

    <init-param>
      <param-name>adminEmail</param-name>
      <param-value>likewecare@wickedlysmart.com</param-value>
    </init-param>

</servlet>
```

You give it a param-name and a param-value. Simple. Just make sure it's INSIDE the <servlet> element in the DD.

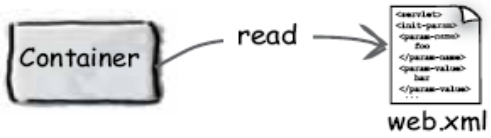
prin intermediul obiectului ServletConfig asociat servlet-ului:

```
out.println(getServletConfig().getInitParameter("adminEmail"));
```

Every servlet inherits a getServletConfig() method.

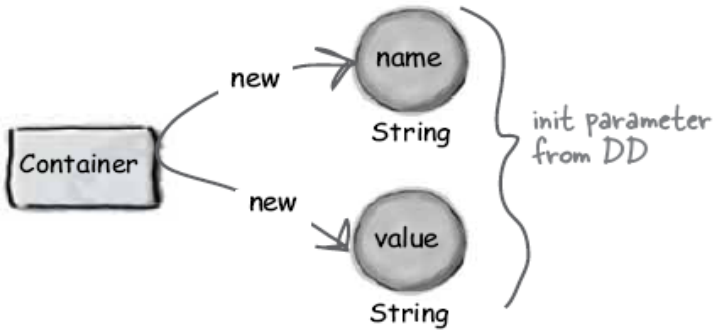
The getServletConfig() method returns a... wait for it... ServletConfig. And one of its methods is getInitParameter().

1. Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).

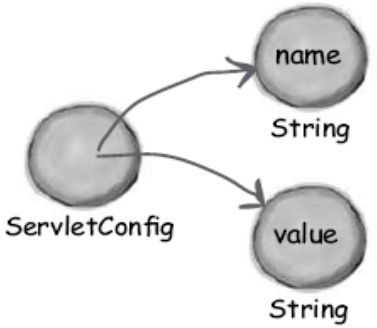


2. Container creates a new ServletConfig instance for this servlet.

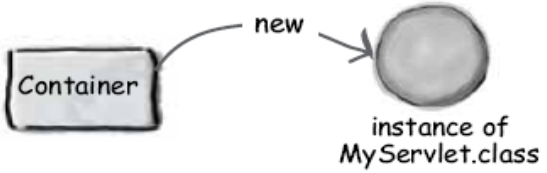
3. Container creates a name/value pair of Strings for each servlet init parameter. Assume we have only one.



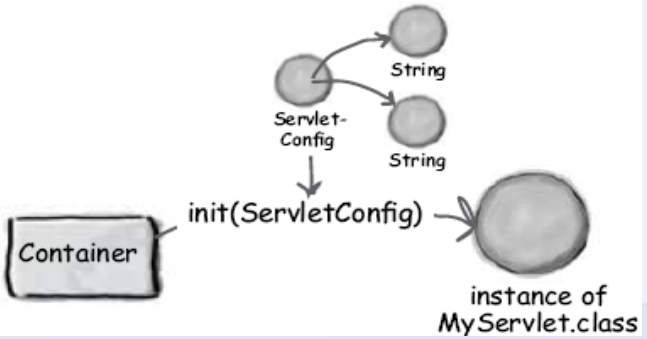
4. Container gives the ServletConfig references to the name/value init parameters.



5. Container creates a new instance of the servlet class.



6. Container calls the servlet's init() method, passing in the reference to the ServletConfig.



Web xml

```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">
  <servlet>
    <servlet-name>BeerParamTests</servlet-name>
    <servlet-class>com.example.TestInitParams</servlet-class>
    <init-param>
      <param-name>adminEmail</param-name>
      <param-value>likewecare@wickedlysmart.com</param-value>
    </init-param>
    <init-param>
      <param-name>mainEmail</param-name>
      <param-value>blooper@wickedlysmart.com</param-value>
    </init-param>
  </servlet>
  <servlet-mapping>
    <servlet-name>BeerParamTests</servlet-name>
    <url-pattern>/Tester.do</url-pattern>
  </servlet-mapping>
</web-app>
```

Exemplu de servlet initializat prin ServletConfig

```
package com.example;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;

public class TestInitParams extends HttpServlet {

  public void doGet(HttpServletRequest request, HttpServletResponse response)
                                                    throws IOException, ServletException {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("test init parameters<br>");

        Enumeration e = getServletConfig().getInitParameterNames();
        while(e.hasMoreElements()) {
           out.println("<br>param name = " + e.nextElement() + "<br>");
        }
        out.println("main email is " + getServletConfig().getInitParameter("mainEmail"));
        out.println("<br>");
        out.println("admin email is " + getServletConfig().getInitParameter("adminEmail"));
    }
  }
```

Pentru a-si transmite informatii (inclusiv parametri de initializare), servlet-urile si JSP-urile care deleaga executia pot crea atribute ale cererii (in cazul JSP-ului obiectul implicit numit request, in cazul servlet-ului obiectul request de tip HTTPServletRequest) carora le dau valori:

```
request.setAttribute("raspuns", "Comanda a fost trimisa");
```

si respectiv servlet-urile si JSP-urile carora li se deleaga executia pot obtine valorile acestor atribute:

```
String raspuns = request.getAttribute("raspuns");
```

Pentru a fi initializate in ansamblu aplicatiile Web obtin informatii pasate de programator in web.xml

```
<context-param>
    <param-name>adminEmail</param-name>
    <param-value>clientheaderror@wickedlysmart.com</param-value>
</context-param>
```

You give it a param-name and param-value just like with servlet init parameters, except this time it's in the <context-param> element instead of <init-param>.

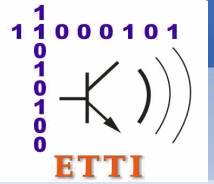
IMPORTANT!! The <context-param> is for the WHOLE app, so its not nested inside an individual <servlet> element!! Put <context-param> inside the <web-app> but OUTSIDE any <servlet> declaration.

prin intermediul obiectului ServletContext asociat <u>aplicatiei Web in ansamblu</u>:

```
out.println(getServletContext().getInitParameter("adminEmail"));
```

Every servlet inherits a getServletContext() method (and JSPs have special access to a context as well).

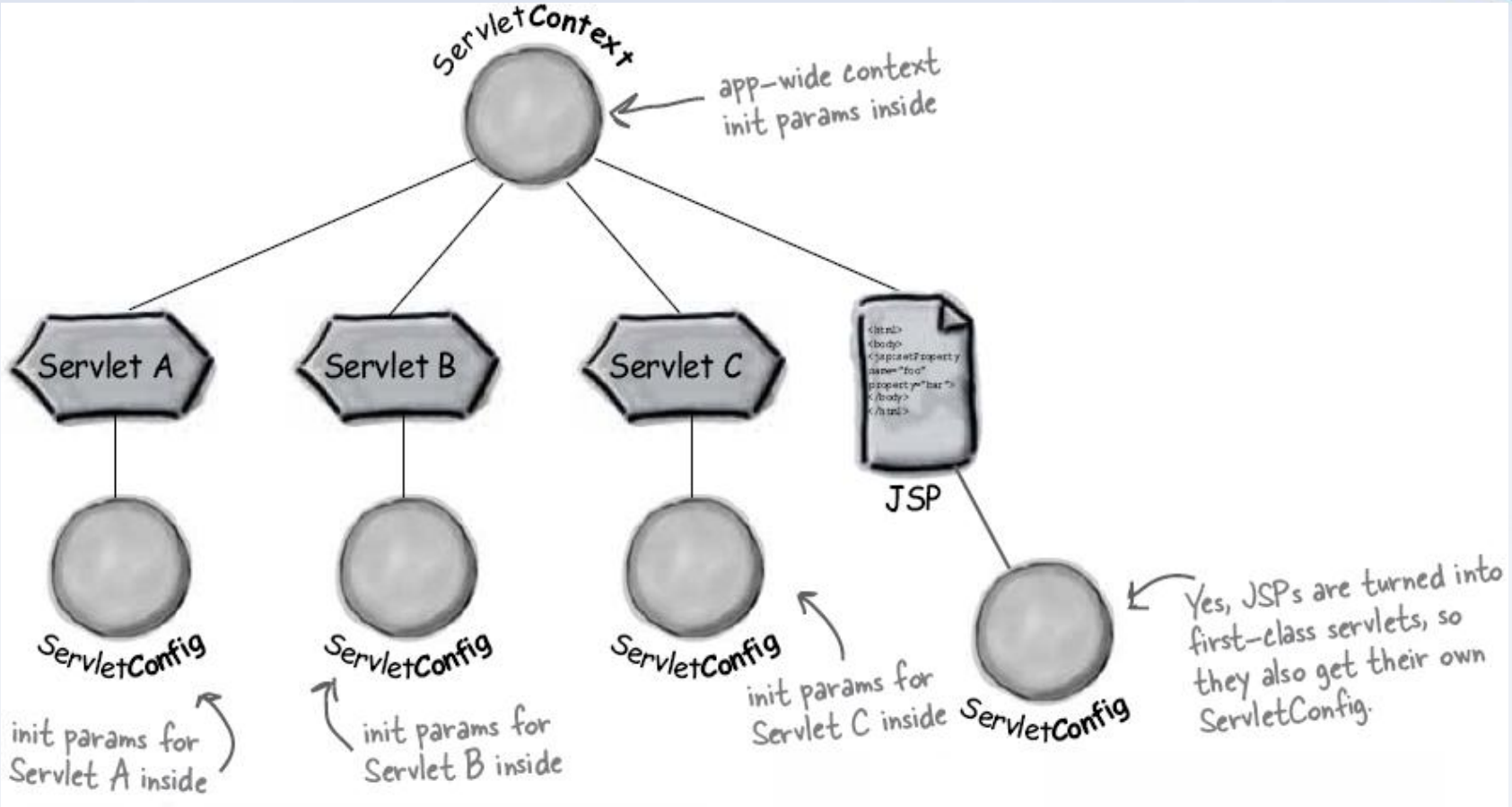
The getServletContext() method returns, surprisingly, a ServletContext object. And one of its methods is getInitParameter().




Context init parameters	Servlet init parameters
Deployment Descriptor	
Within the <web-app> element but NOT within a specific <servlet> element	Within the <servlet> element for each specific servlet

`<web-app ...>`
`  <context-param>`
`  <param-name>foo</param-name>`
`  <param-value>bar</param-value>`
`  </context-param>`
 	
`  <!-- other stuff including servlet declarations -->`
`</web-app>`
	`<servlet>`
	`  <servlet-name>`
	`  BeerParamTests`
	`  <servlet-name>`
	`  <servlet-class>`
	`  TestInitParams`
	`  </servlet-class>`
	`  <init-param>`
	`  <param-name>foo</param-name>`
	`  <param-value>bar</param-value>`
	`  </init-param>`
	
	`  <!-- other stuff -->`
	`</servlet>`

Notice it doesn't say "init" anywhere in the DD for context init parameters, the way it does for servlet init parameters.	
Servlet Code	
`getServletContext().getInitParameter("foo");`	`getServletConfig().getInitParameter("foo");`
It's the same method name!	
Availability	
To any servlets and JSPs that are part of this web app.	To only the servlet for which the <init-param> was configured. (Although the servlet can choose to make it more widely available by storing it in an attribute.)



ServletContext
app-wide context
init params inside
Servlet A
Servlet B
Servlet C
JSP
ServletConfig
ServletConfig
ServletConfig
ServletConfig
init params for
Servlet A inside
init params for
Servlet B inside
init params for
Servlet C inside
Yes, JSPs are turned into
first-class servlets, so
they also get their own
ServletConfig.

Pentru a putea initializa o resursa (de exemplu o conexiune cu o baza de date) si a o face disponibila intregii aplicatii Web se poate folosi tratarea unui eveniment asociat contextului aplicatiei Web:

```
import javax.servlet.*;

public class MyServletContextListener implements ServletContextListener {

    public void contextInitialized(ServletContextEvent event) {
      //code to initialize the database connection
      //and store it as a context attribute
    }

    public void contextDestroyed(ServletContextEvent event) {
      //code to close the database connection
    }
}
```

ServletContextListener is in javax.servlet package.

A context listener is simple: implement ServletContextListener.

These are the two notifications you get. Both give you a ServletContextEvent.

Exemplu complex – clasa ascultator al evenimentului (evenimentul creare context, adica creare aplicatie Web):

```
package com.example;

import javax.servlet.*;

public class MyServletContextListener implements ServletContextListener {
   public void contextInitialized(ServletContextEvent event) {
      ServletContext sc = event.getServletContext();
      String dogBreed = sc.getInitParameter("breed");
      Dog d = new Dog(dogBreed);
      sc.setAttribute("dog", d);
   }

  public void contextDestroyed(ServletContextEvent event) {
    // nothing to do here
  }
}
```

Implement javax.servlet.ServletContextListener.

Ask the event for the ServletContext.

Use the context to get the init parameter.

Make a new Dog.

Use the context to set an attribute (a name/object pair) that is the Dog. Now other parts of the app will be able to get the value of the attribute (the Dog).

We don't need anything here. The Dog doesn't need to be cleaned up... when the context goes away, it means the whole app is going down, including the Dog.

Exemplu complex – Clasa servlet

```
package com.example;

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

public class ListenerTester extends HttpServlet {

  public void doGet (HttpServletRequest request, HttpServletResponse response)
                                          throws IOException, ServletException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();

    out.println("test context attributes set by listener<br>");

    out.println("<br>");

    Dog dog = (Dog) getServletContext().getAttribute("dog");

    out.println("Dog's breed is: " + dog.getBreed());
  }
}
```

Nothing special so far... just a regular servlet.

don't forget the cast!!

Now we get the Dog from the ServletContext. If the listener worked, the Dog will be there BEFORE this service method is called for the first time.

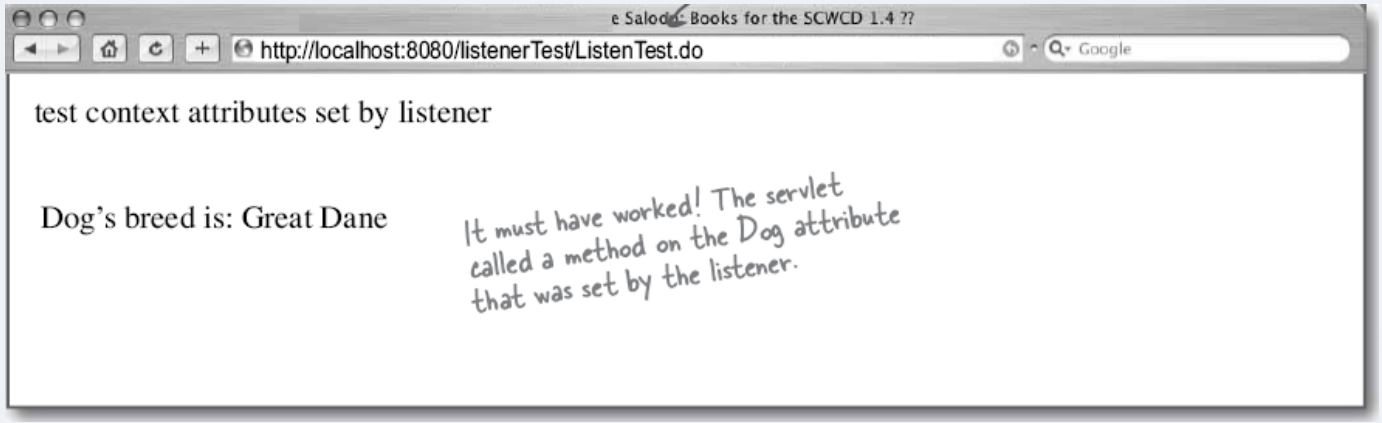
If things didn't work, THIS is where we'll find out... we'll get a big fat NullPointerException if we try to call getBreed() and there's no Dog.

Exemplu complex – web xml

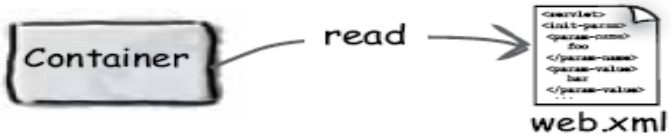
```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <servlet>
    <servlet-name>ListenerTester</servlet-name>
    <servlet-class>com.example.ListenerTester</servlet-class>
  </servlet>

  <servlet-mapping>
    <servlet-name>ListenerTester</servlet-name>
    <url-pattern>/ListenTest.do</url-pattern>
  </servlet-mapping>

  <context-param>
    <param-name>breed</param-name>
    <param-value>Great Dane</param-value>
  </context-param>

  <listener>
    <listener-class>
       com.example.MyServletContextListener
    </listener-class>
  </listener>

</web-app>
```

We need a context init parameter for the app. The listener needs this to construct the Dog.

Register this class as a listener. IMPORTANT: the <listener> element does NOT go inside a <servlet> element. That wouldn't work because a context listener is for a ServletContext (which means application-wide) event. The whole point is to initialize the app BEFORE any servlets are initialized.

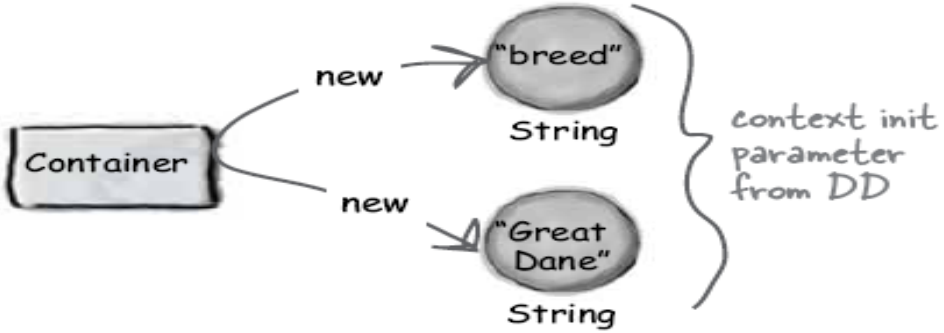

e Saloon: Books for the SCWCD 1.4 ??
http://localhost:8080/listenerTest/ListenTest.do
Google
test context attributes set by listener
Dog's breed is: Great Dane
It must have worked! The servlet called a method on the Dog attribute that was set by the listener.

1. Container reads the Deployment Descriptor for this app, including the <listener> and <context-param> elements.

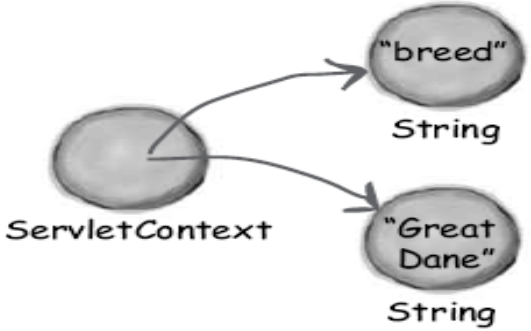


2. Container creates a new ServletContext for this application, that all parts of the app will share.

3. Container creates a name/value pair of Strings for each context init parameter. Assume we have only one.



4. Container gives the ServletContext references to the name/value parameters.

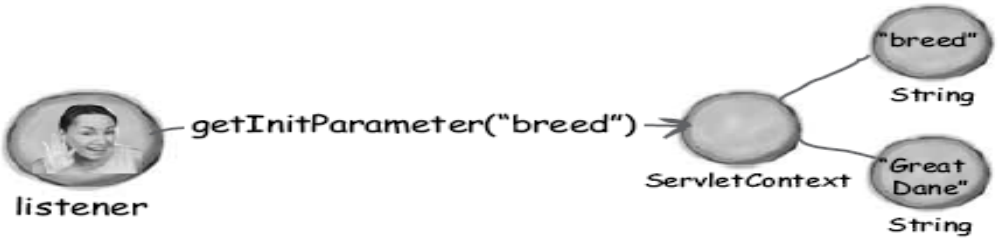


5. Container creates a new instance of the MyServletContextListener class.

6. Container calls the listener's contextInitialized() method, passing in a new ServletContextEvent. The event object has a reference to the ServletContext, so the event-handling code can get the context from the event, and get the context init parameter from the context.

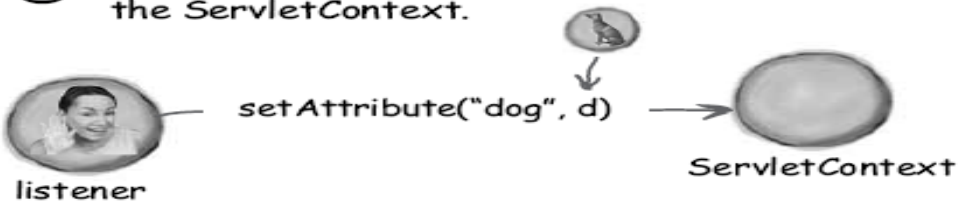
7 Listener asks ServletContextEvent for a reference to the ServletContext.

8 Listener asks ServletContext for the context init parameter "breed" .

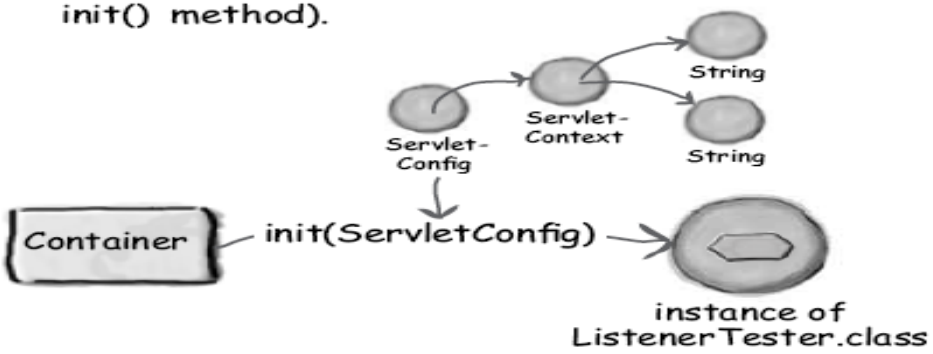


9 Listener uses the init parameter to construct a new Dog object.

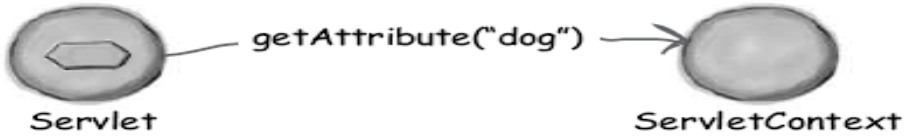
10 Listener sets the Dog as an attribute in the ServletContext.



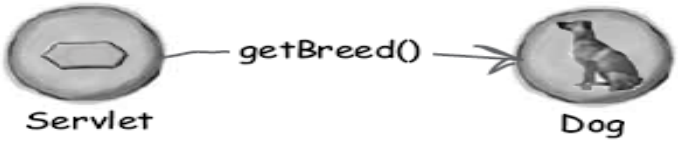
11 Container makes a new Servlet (i.e., makes a new ServletConfig with init parameters, gives the ServletConfig a reference to the ServletContext, then calls the Servlet's init() method).



12 Servlet gets a request, and asks the ServletContext for the attribute "dog".

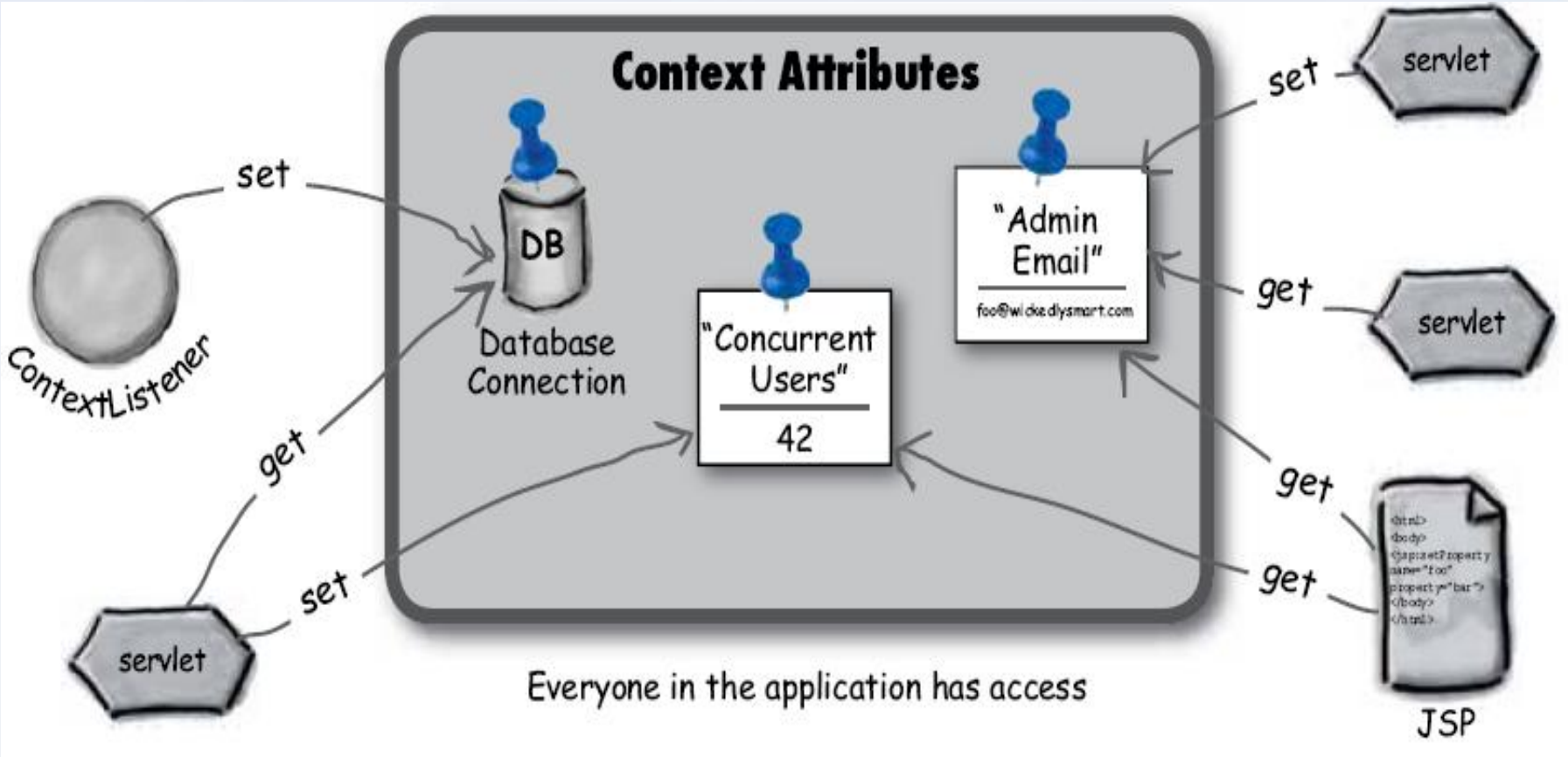


13 Servlet calls getBreed() on the Dog (and prints that to the HttpResponse).

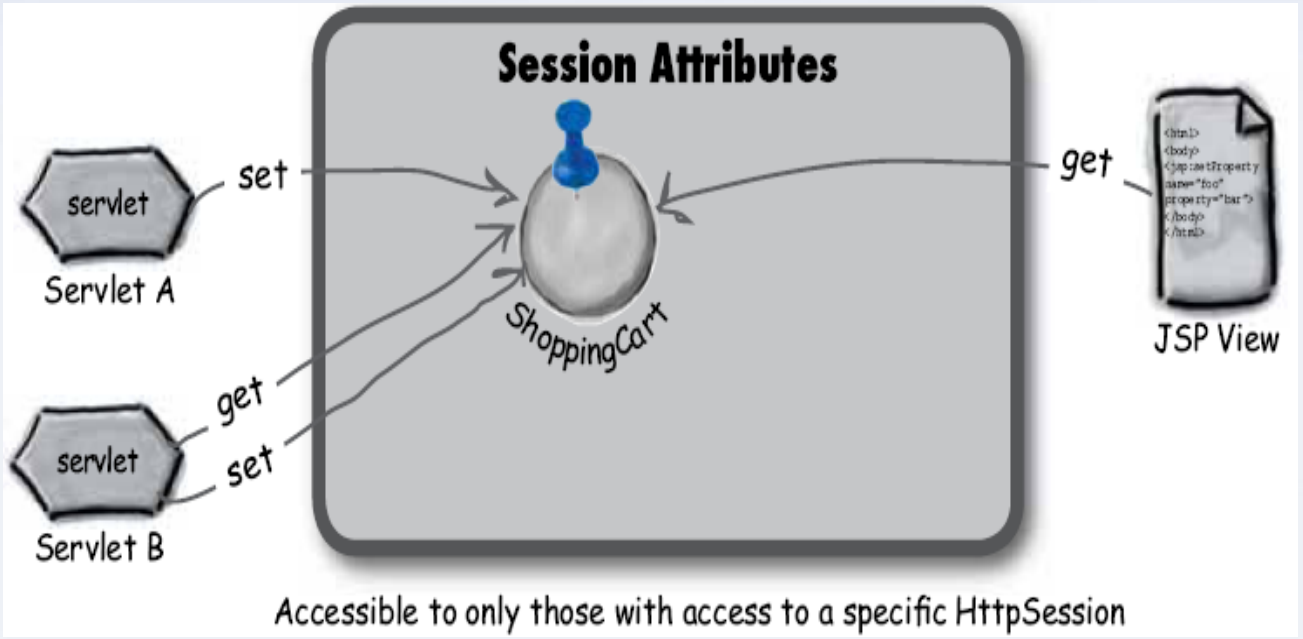


Scenario	Listener interface	Event type
You want to know if an attribute in a web app context has been added, removed, or replaced.	javax.servlet.**ServletContextAttributeListener** *attributeAdded* *attributeRemoved* *attributeReplaced*	ServletContextAttributeEvent
You want to know how many concurrent users there are. In other words, you want to track the active sessions. (We cover sessions in detail in the next chapter).	javax.servlet.http.**HttpSessionListener** *sessionCreated* *sessionDestroyed*	HttpSessionEvent
You want to know each time a request comes in, so that you can log it.	javax.servlet.**ServletRequestListener** *requestInitialized* *requestDestroyed*	ServletRequestEvent
You want to know when a request attribute has been added, removed, or replaced.	javax.servlet.**ServletRequestAttributeListener** *attributeAdded* *attributeRemoved* *attributeReplaced*	ServletRequestAttributeEvent
You have an attribute class (a class for an object that will stored as an attribute) and you want objects of this type to be notified when they are bound to or removed from a session.	javax.servlet.http.**HttpSessionBindingListener** *valueBound* *valueUnbound*	HttpSessionBindingEvent
You want to know when a session attribute has been added, removed, or replaced.	javax.servlet.http.**HttpSessionAttributeListener** *attributeAdded* *attributeRemoved* *attributeReplaced*	HttpSessionBindingEvent Watch out for this naming inconsistency! The Event for HttpSessionAttributeListener is NOT what you expect (you expect HttpSessionAttributeEvent).
You want to know if a context has been created or destroyed.	javax.servlet.**ServletContextListener** *contextInitialized* *contextDestroyed*	ServletContextEvent
You have an attribute class, and you want objects of this type to be notified when the session to which they're bound is migrating to and from another JVM.	javax.servlet.http.**HttpSessionActivationListener** *sessionDidActivate* *sessionWillPassivate*	HttpSessionEvent It's NOT "HttpSessionActivationEvent"

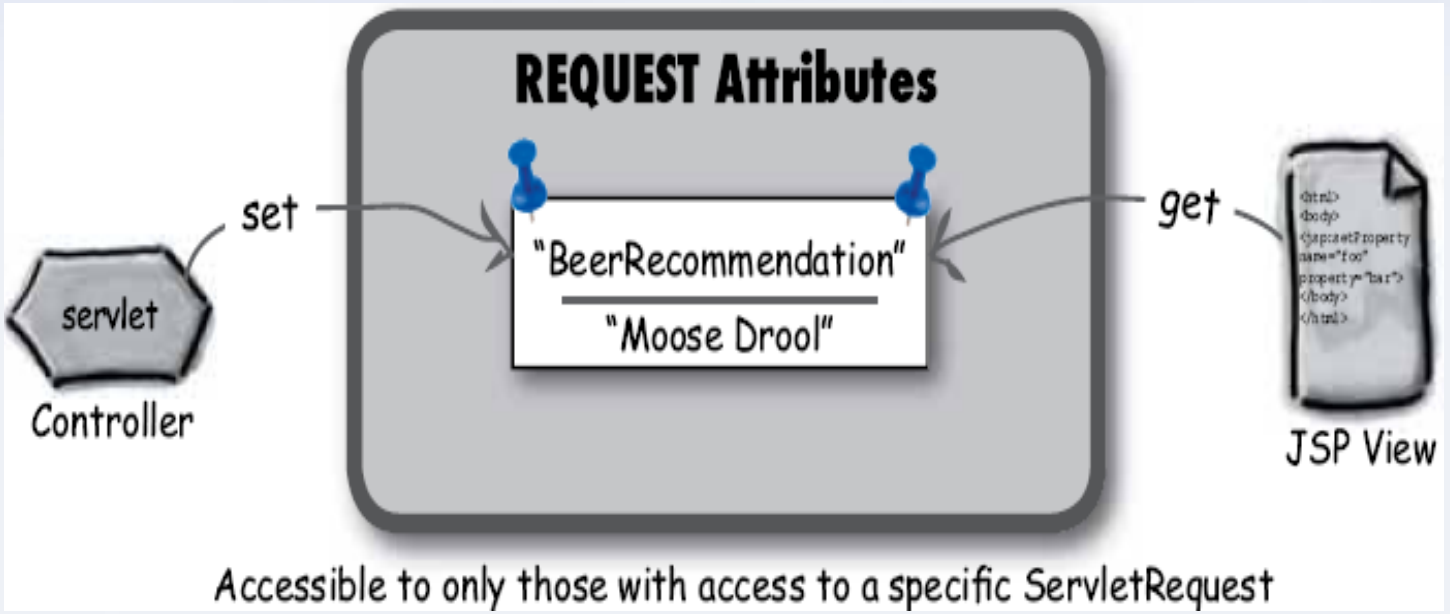
Tipuri de scopuri ale atributelor:
- atribute accesibile la nivel de aplicatie (context)



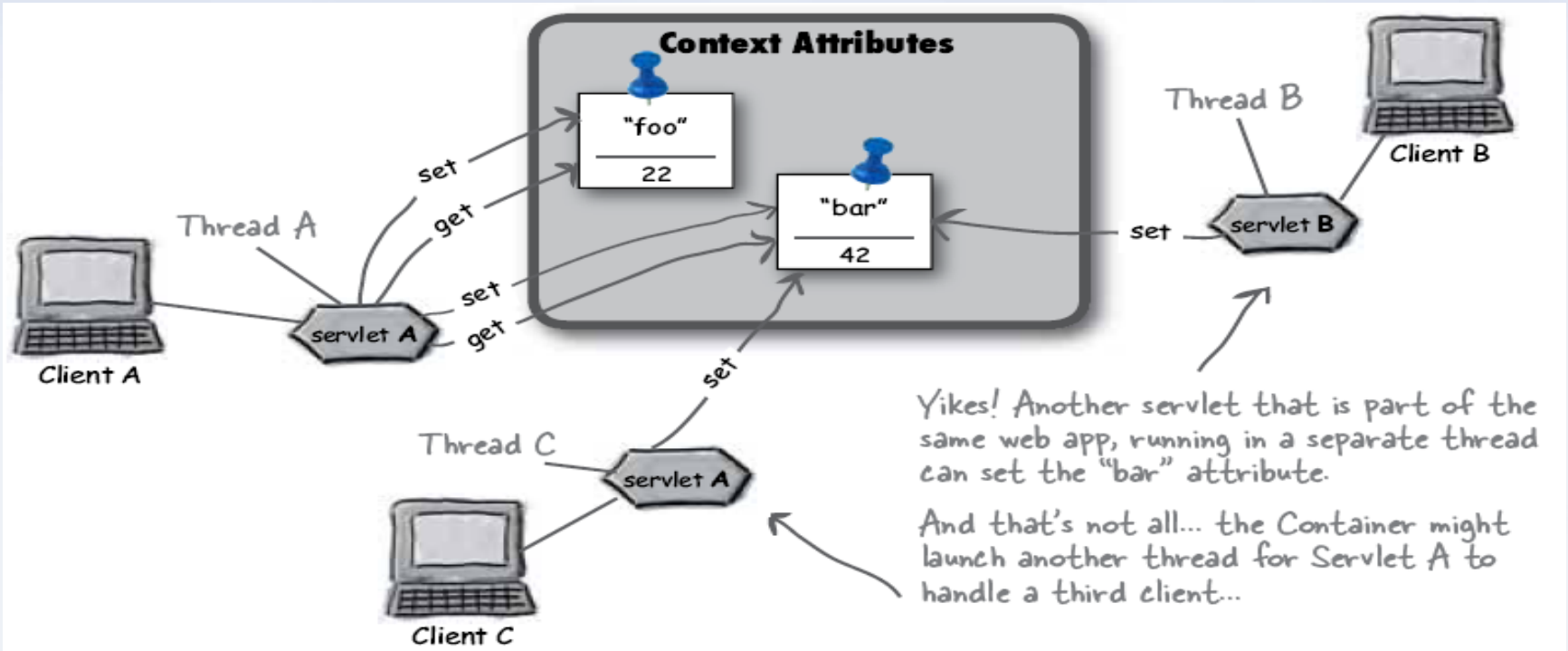
atribute accesibile la nivel de sesiune:



- atribute accesibile la nivel de cerere:



Atributele la nivel de context nu sunt *thread safe*.

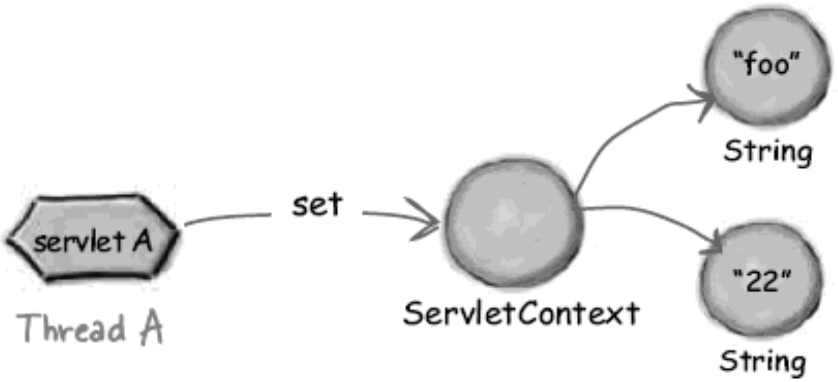



```
getServletContext().setAttribute("foo", "22");
getServletContext().setAttribute("bar", "42");

out.println(getServletContext().getAttribute("foo"));
out.println(getServletContext().getAttribute("bar"));
```

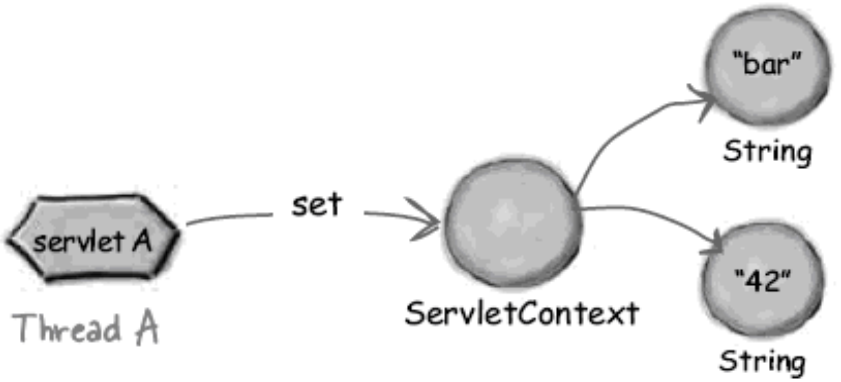
In between when servlet A set the value of "bar" and then got the value of "bar", another servlet thread snuck in and set "bar" to a different value.

So by the time servlet A printed the value of "bar", it had been changed to "16".

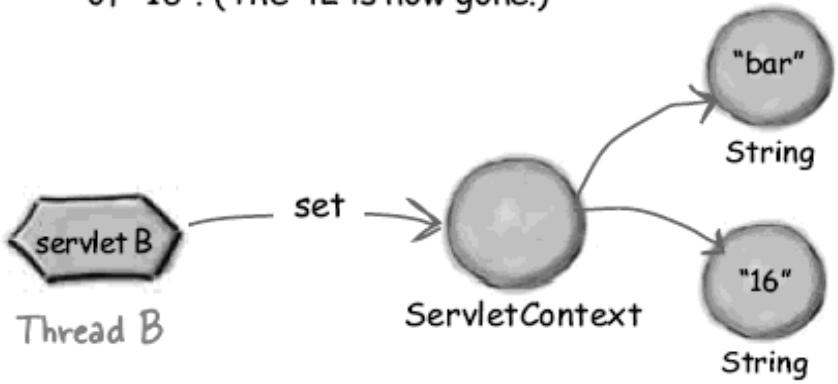
1. Servlet A sets the context attribute "foo" with a value of "22".



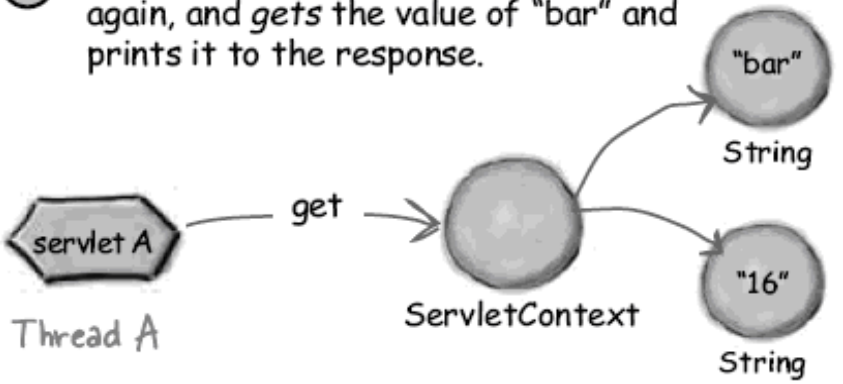
2. Servlet A sets the context attribute "bar" with a value of "42".



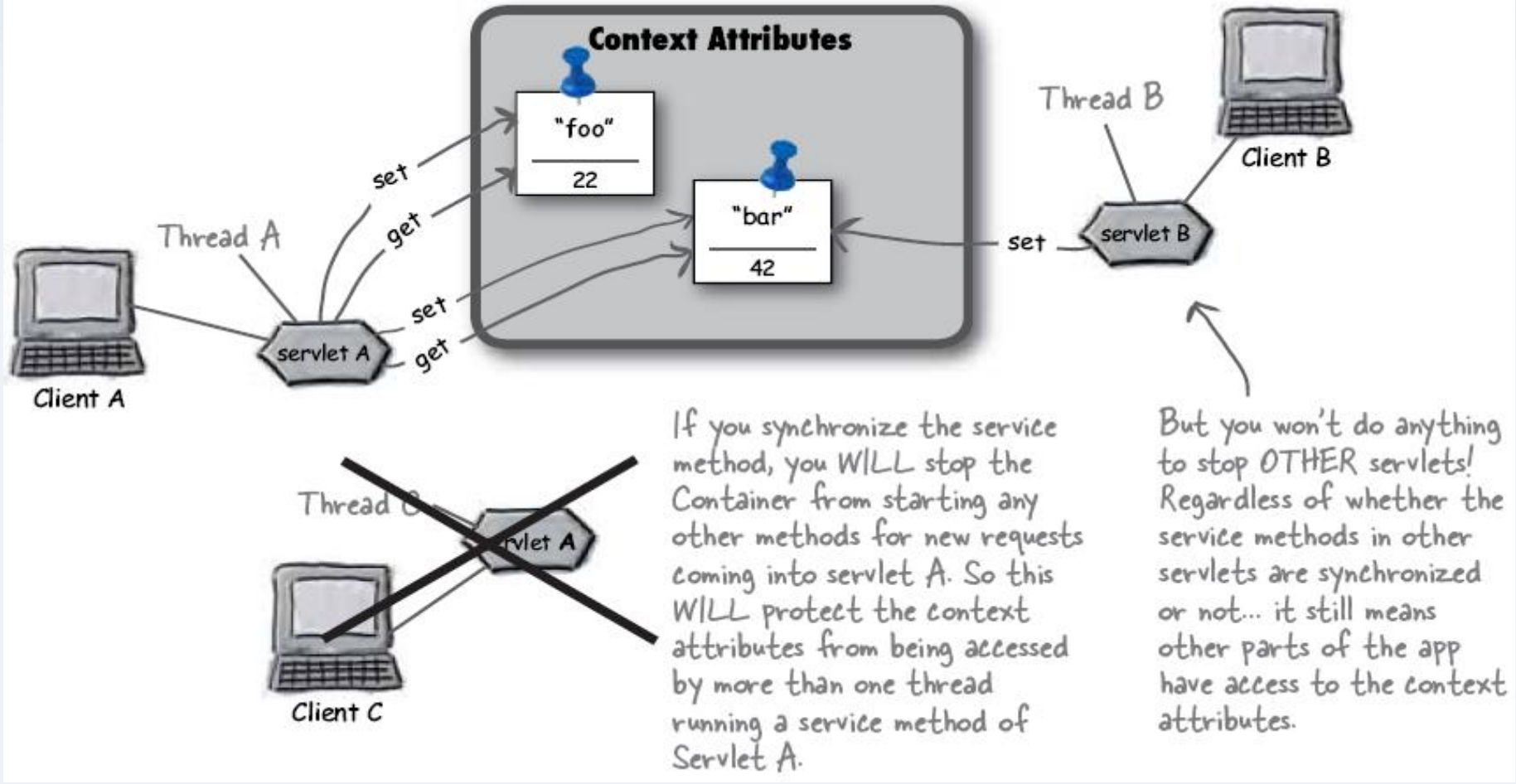
3. Thread B becomes the running thread (thread A goes back to Runnable-but-not-Running), and *sets* the context attribute "bar" with a value of "16". (The 42 is now gone.)



4. Thread A becomes the running thread again, and *gets* the value of "bar" and prints it to the response.



Sincronizarea metodelor service(), doGet(), doPost(), etc., nu rezolva problema:



Sincronizarea trebuie facuta pe context:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
                                    throws IOException, ServletException {

   response.setContentType("text/html");
   PrintWriter out = response.getWriter();

   out.println("test context attributes<br>");

   synchronized(getServletContext()) {
      getServletContext().setAttribute("foo", "22");
      getServletContext().setAttribute("bar", "42");

      out.println(getServletContext().getAttribute("foo"));
      out.println(getServletContext().getAttribute("bar"));
   }
}
```

Now we're getting the lock on the context itself!! This is the way to protect context attribute state. (You don't want synchronized(this).)