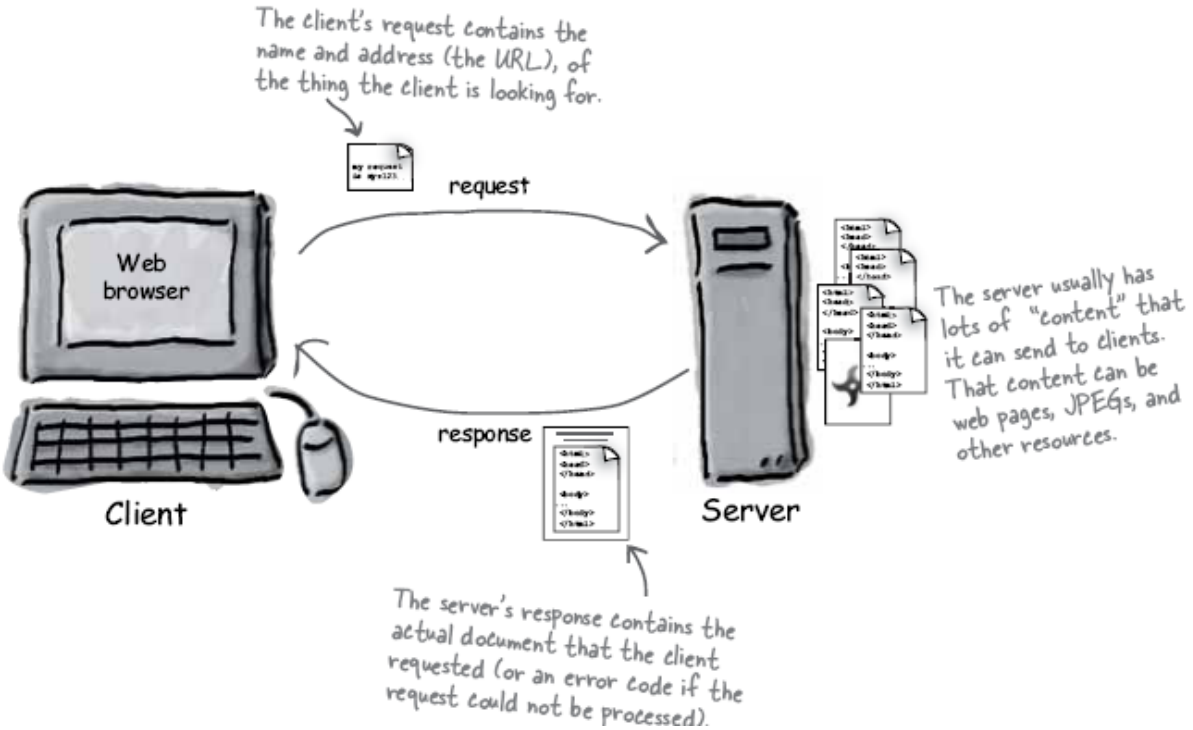
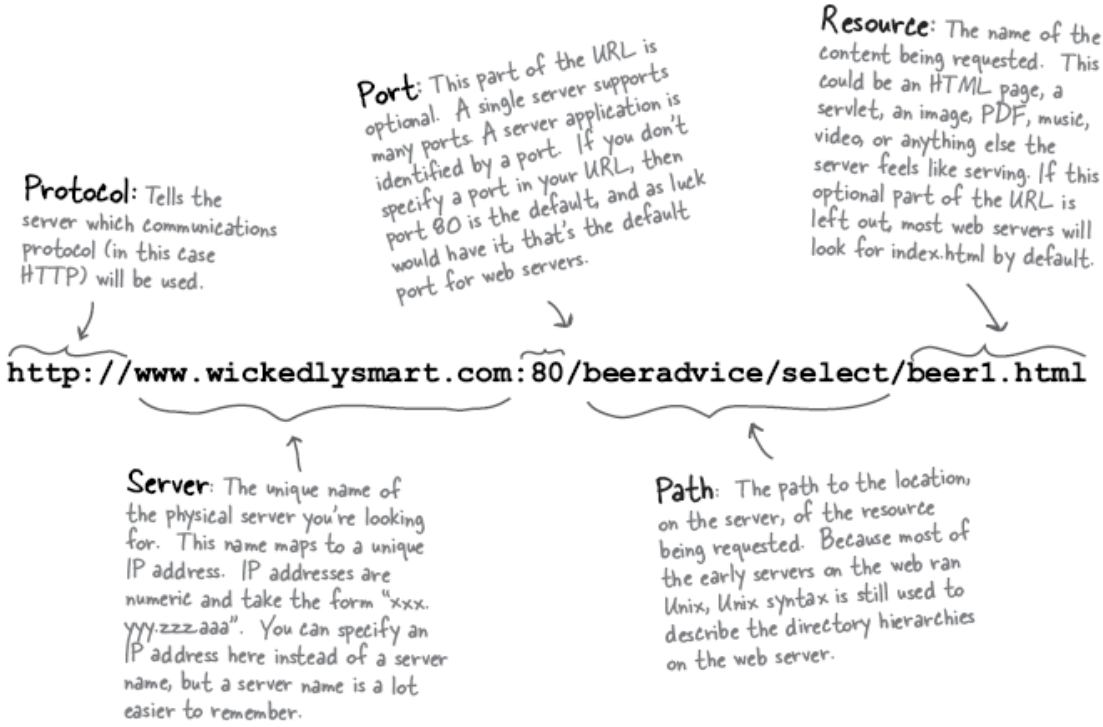
4. Tehnologii Java de programare a aplicatiilor Web

4.1. Introducere in Web (protocolul HTTP, adresele URL, limbajul HTML)

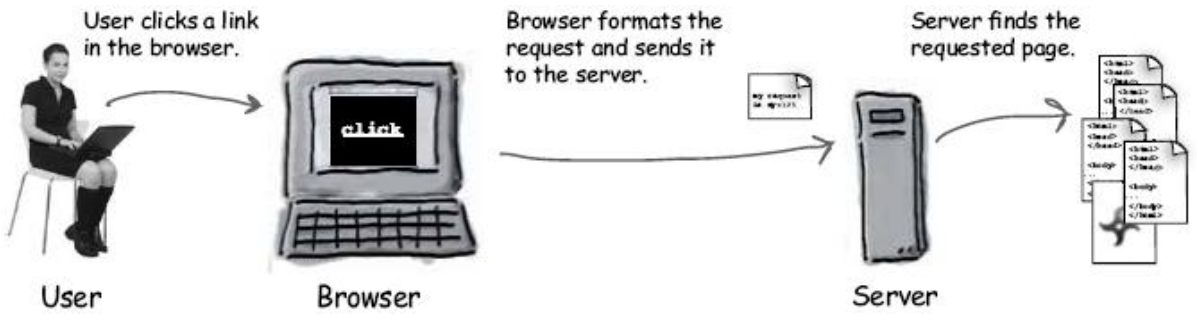
HTTP

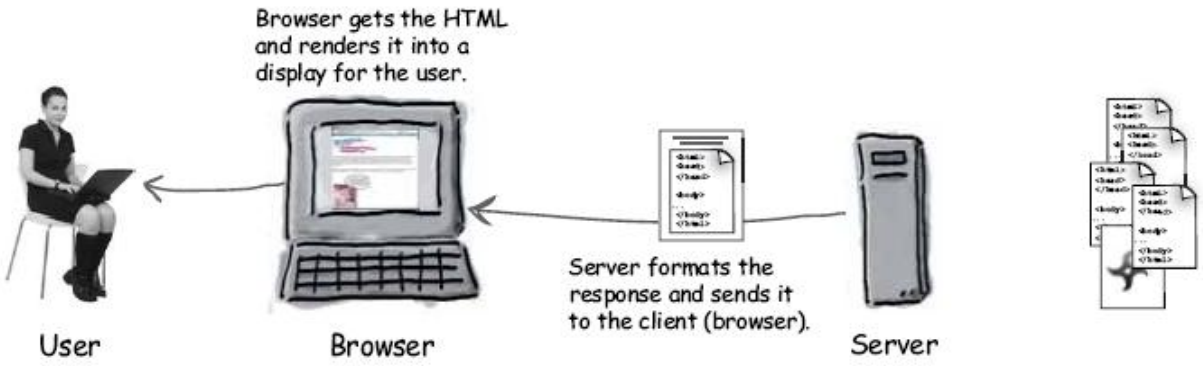


URL

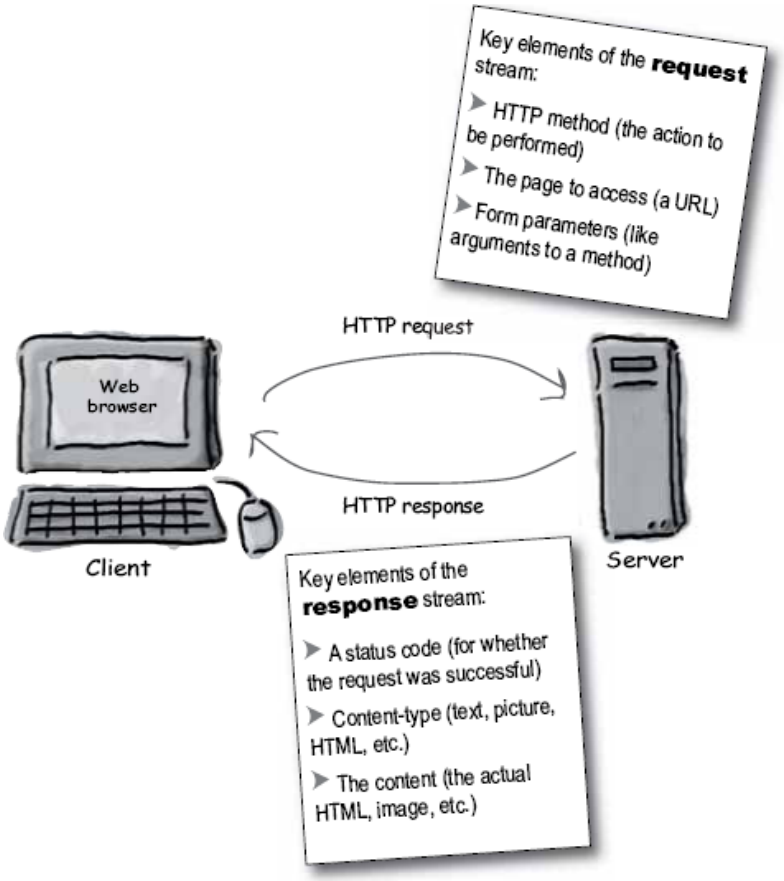


HTTP request

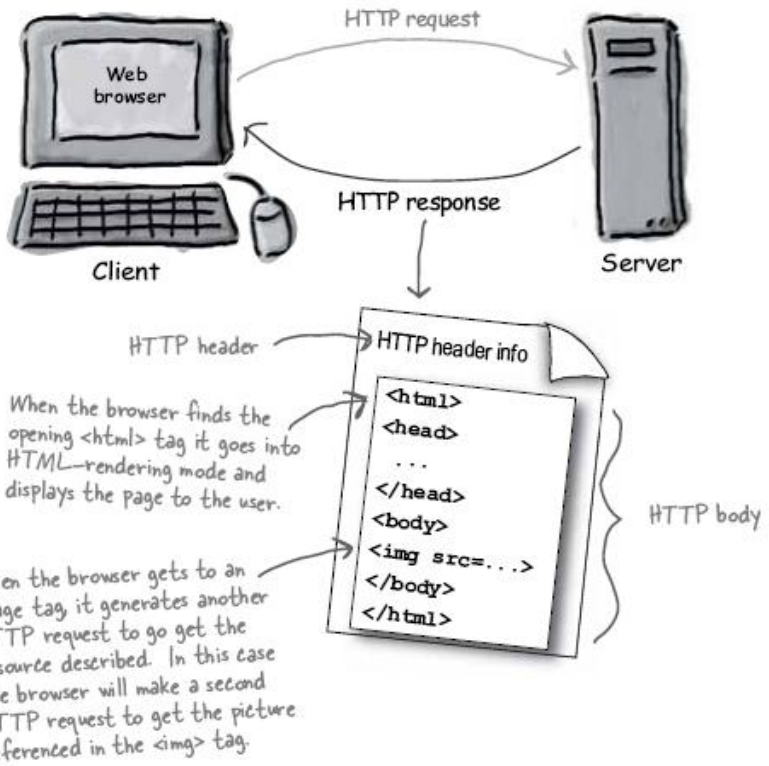




HTTP response



HTTP request si **HTTP response**



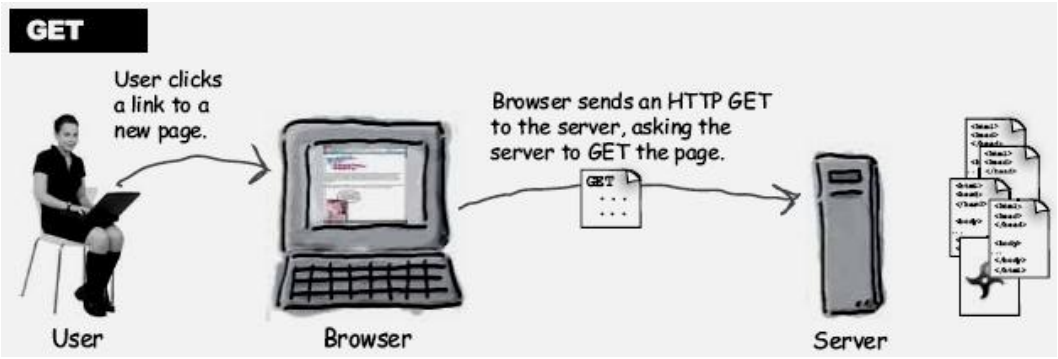
Elemente de limbaj HTML

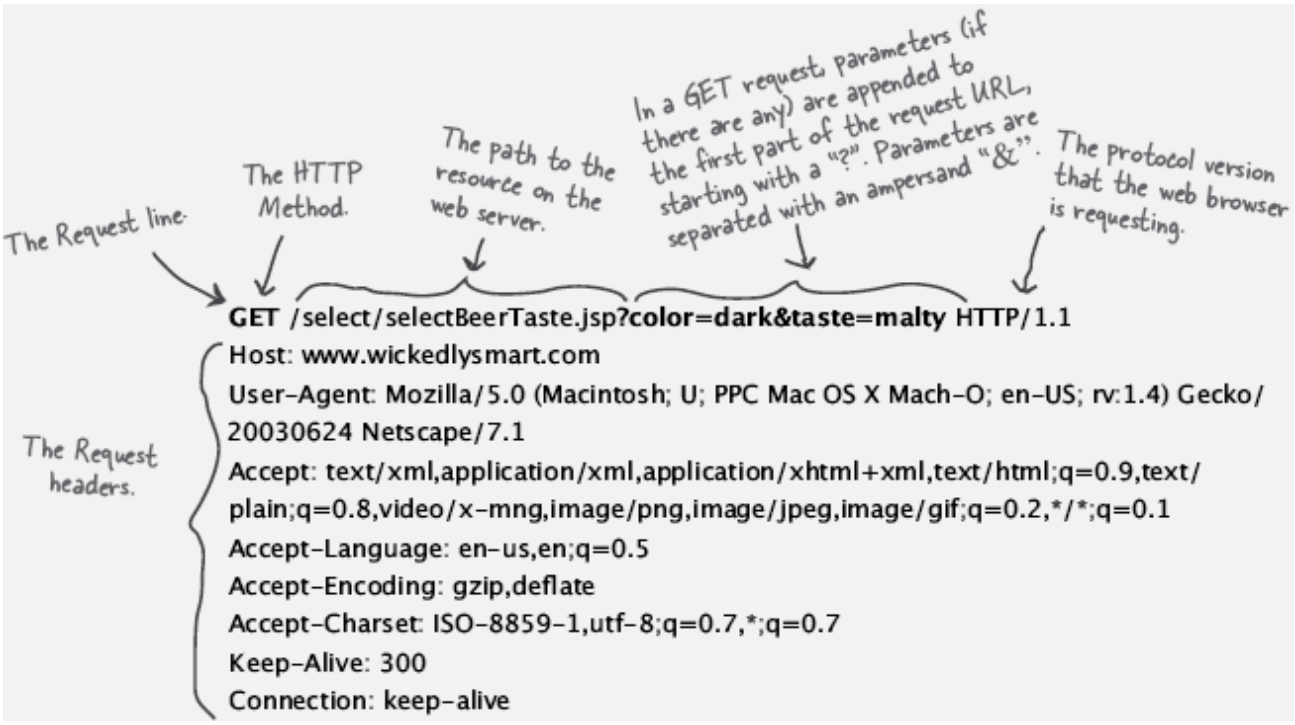
Tag	Description
`<!-- -->`	where you put your *comments*
`<a>`	*anchor* - usually for putting in a hyperlink
`<align>`	*align* the contents left, right, centered, or justified
`<body>`	define the boundaries of the document's *body*
` `	a *line break*
`<center>`	*center* the contents
`<form>`	define a *form* (which usually provides input fields)
`<h1>`	the first level *heading*
`<head>`	define the boundaries of the document's *header*
`<html>`	define the boundaries of the HTML *document*
`<input type>`	defines an *input widget* to a form

(Technically, the and <align> tags have been deprecated in HTML 4.0, but we're using them in some of our examples because it's simpler to read than the alternative, and you're not here to learn HTML anyway.)

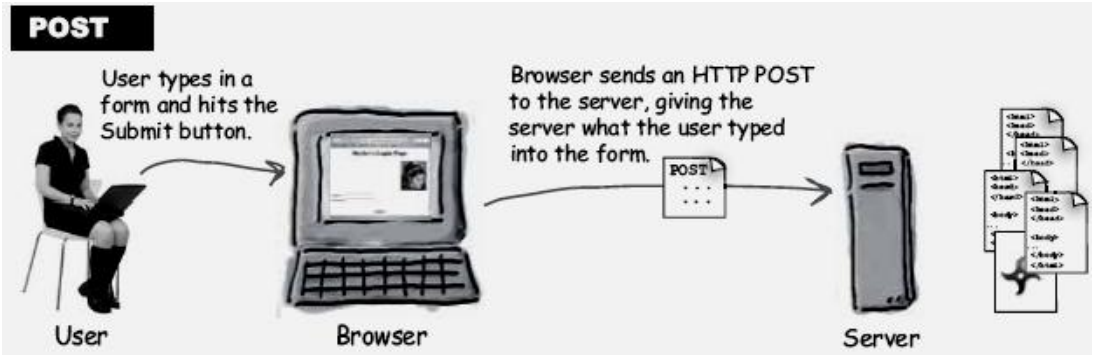
Exemple de cereri HTTP

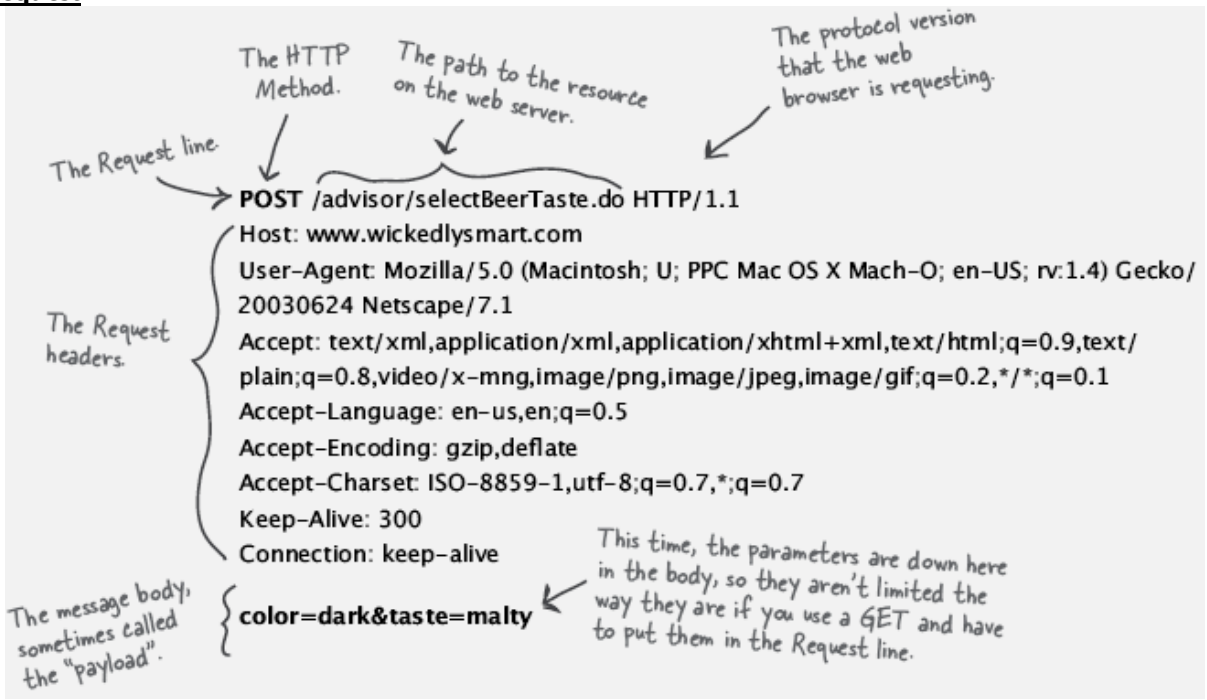
- GET request



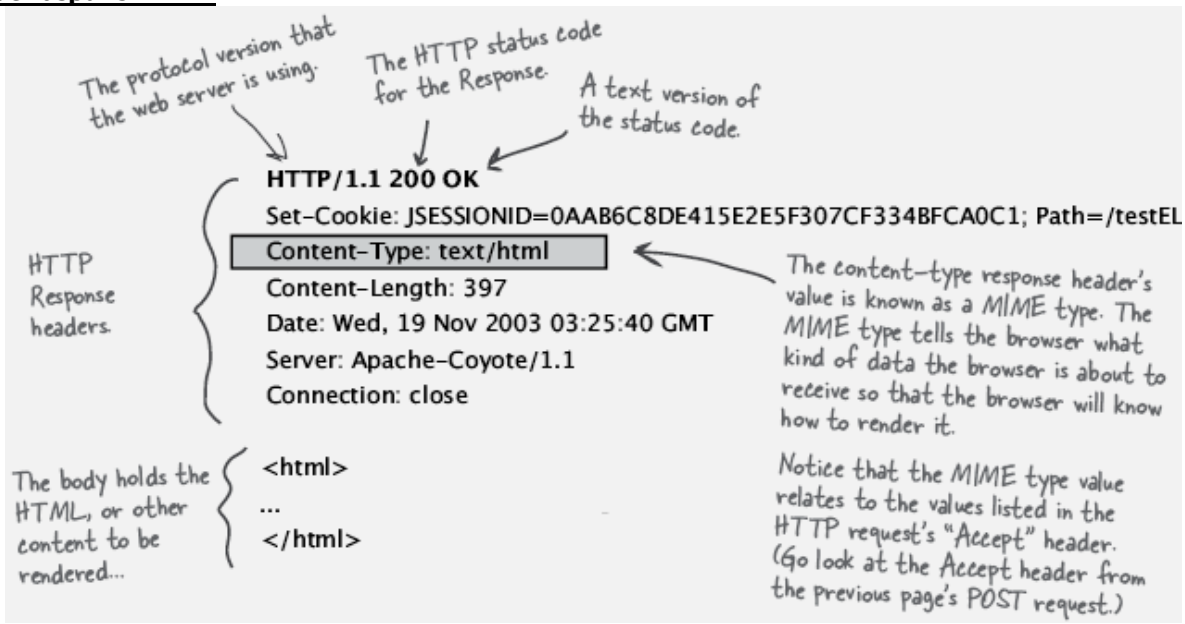


- **POST request**

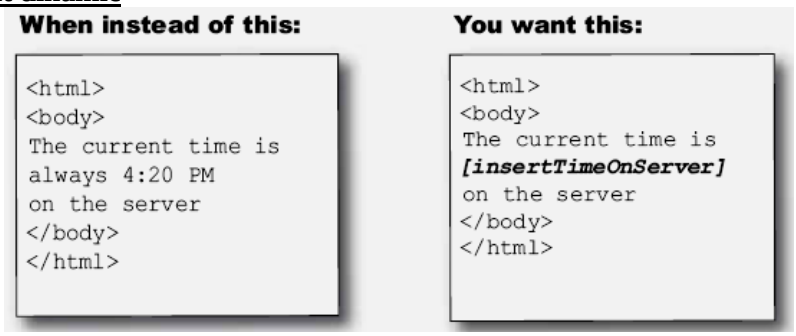




Exemple de raspuns HTTP



Continut static vs **continut dinamic**



4.2. Tehnologii client. Miniaplicatii Java (applet-uri)

4.2.1. Caracteristicile *applet-urilor* Java

Applet-urile sau miniaplicatiile Java sunt portiuni de cod Java care mostenesc clasa **`Applet`**. Prin plasarea lor in *browser-e*, *applet-urile* devin panouri frontale ale serviciilor distribuite oferite de *Web*.

Applet-urile sunt mai intai incarcate in *browser-e*, fiind apoi executate in mediul de executie oferit de acesta.

Applet-urile nu sunt aplicatii complete, ci componente care ruleaza in mediul *browser-ului*.

Browser-ul* actioneaza ca un *framework pentru executia *applet-urile* (componentelor Java).

Browser-ul* informeaza *applet-ul asupra evenimentelor care se petrec pe durata de viata a *applet-ului*. **Serviciile oferite de *browser*** sunt:

- **controlul total al ciclului de viata** al *applet-ului*,
- **furnizarea informatiilor privind atributele** din *tag-ul* APPLET,
- **functia de program/proces principal** din care se executa *applet-urile* (ofera functia **main()**).

4.1.2. Ciclul de viata al *applet-urilor* Java

Clasa **`Applet`** interfata **`Runnable`** definesc metode pe care un *browser* le poate invoca pe durata ciclului de viata al unui *applet*. *Browser-ul* invoca*:*

- **`init()`** cand ***incarca applet-ul prima oara***;
- **`start()`** cand ***un utilizator intra*** sau ***reintra in pagina care contine applet-ul***;
- **`stop()`** cand ***utilizatorul iese din pagina***;
- **`destroy()`** ***inaintea terminarii normale.***

Invocarea ultimelor doua metode conduce la **"omorarea" tuturor firelor de executie** ale *applet-ului* si la **eliberarea tuturor resurselor** *applet-ului*.

Urmatorul *applet* simplu permite urmarirea fazelor ciclului de viata ale unui *applet***:**

```
 1  import                                                  java.applet.Applet;
 2  import java.awt.Graphics;
 3
 4  public class Simple extends Applet {
 5
 6      StringBuffer buffer;
 7
 8      public void init() {
 9          buffer = new StringBuffer();
10
11          addItem("initializing... ");
12      }
13
14      public void start() {
15          addItem("starting... ");
16      }
17
18      public void stop() {
19          addItem("stopping... ");
20      }
21
22      public void destroy() {
23          addItem("preparing for unloading...");
24      }
25
26      void addItem(String newWord) {
27
28          System.out.println(newWord); // afisare in Java Console a browser-ului
29
30          buffer.append(newWord);
31
32          repaint(); // apeleaza paint()
33      }
34
35      public void paint(Graphics g) {
36          //Draw a Rectangle around the applet's display area.
37          g.drawRect(0, 0, size().width - 1, size().height - 1);
38
39          //Draw the current string inside the rectangle.
40          g.drawString(buffer.toString(), 5, 15);
41      }
42  }
```

Programul **`ExtensieInteractivaJApplet`** ilustreaza:
- crearea unei miniaplicatii (*applet*) prin **extinderea clasei `JApplet`**, si
- tratarea evenimentului « actionare » pentru componentele buton

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4
 5  public class ExtensieInteractivaJApplet extends JApplet {
 6      public void init() {
 7        // Obtinerea panoului de continut (content pane) creat de browser pentru
 8        // executia appletului (container in care vor fi plasate componentele)
 9        Container container = getContentPane();
10        // Stabilirea layout-ului panoului, BorderLayout cu spatiu 10 pixeli
11        container.setLayout(new BorderLayout(10, 10));
12
13        // Adaugarea a 5 butoane la panoul appletului
14        // Referintele create vor fi necesare si inregistrarii ascultatorilor
15        JButton b1 = new JButton("Est (Dreapta)");
16        JButton b2 = new JButton("Sud (Jos)");
17        JButton b3 = new JButton("Vest (Stanga)");
18        JButton b4 = new JButton("Nord (Sus)");
19        JButton b5 = new JButton("Centru");
20        container.add(b1, BorderLayout.EAST);
21        container.add(b2, BorderLayout.SOUTH);
22        container.add(b3, BorderLayout.WEST);
23        container.add(b4, BorderLayout.NORTH);
24        container.add(b5, BorderLayout.CENTER);
25
26        // Crearea unui obiect "ascultator" de "evenimente actionare"
27        // (pe care le trateaza)
28        ActionListener obiectAscultatorActionare = new ActionListener() {
29
30          // Tratarea actionarii unui buton
31          public void actionPerformed(ActionEvent ev) {
32
33            // Mesaj informare in consola Java
34            System.out.println("A fost apasat butonul " + ev.getActionCommand());
35            // Mesaj informare in bara de stare
36            showStatus("Apasat butonul " + ev.getActionCommand());
37          }
38        };
39
40        // Inregistrarea "ascultatorului" de "evenimente actionare" la "sursele"
41        // de evenimente
42        b1.addActionListener(obiectAscultatorActionare);
43        b2.addActionListener(obiectAscultatorActionare);
44        b3.addActionListener(obiectAscultatorActionare);
45        b4.addActionListener(obiectAscultatorActionare);
46        b5.addActionListener(obiectAscultatorActionare);
47      }
48  }
```

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

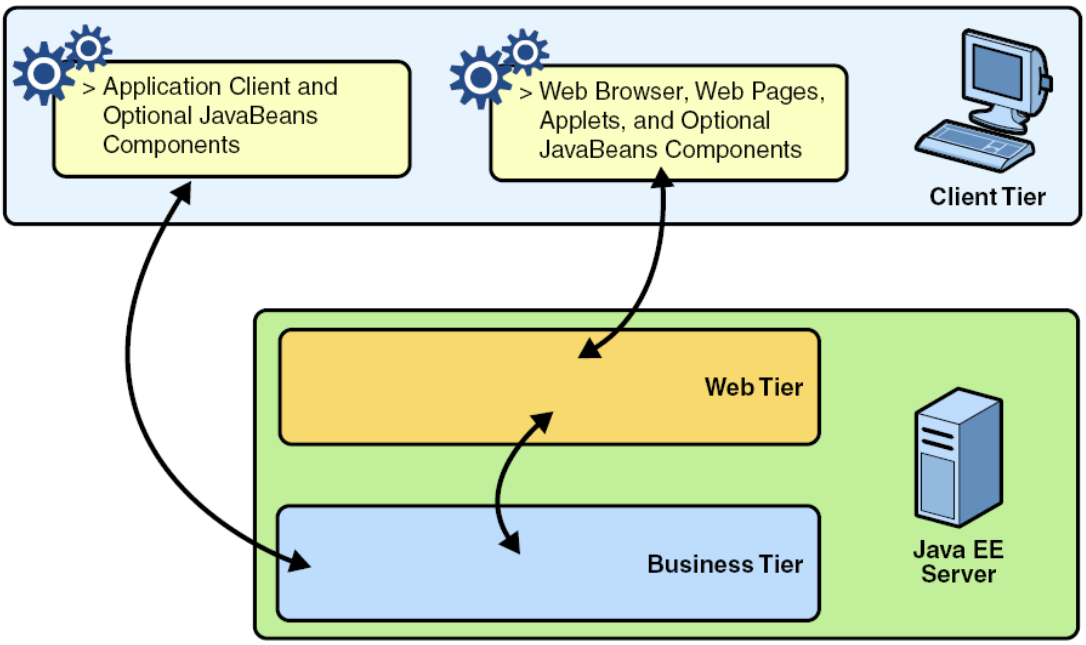


FIGURE 1–2 Server Communication

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

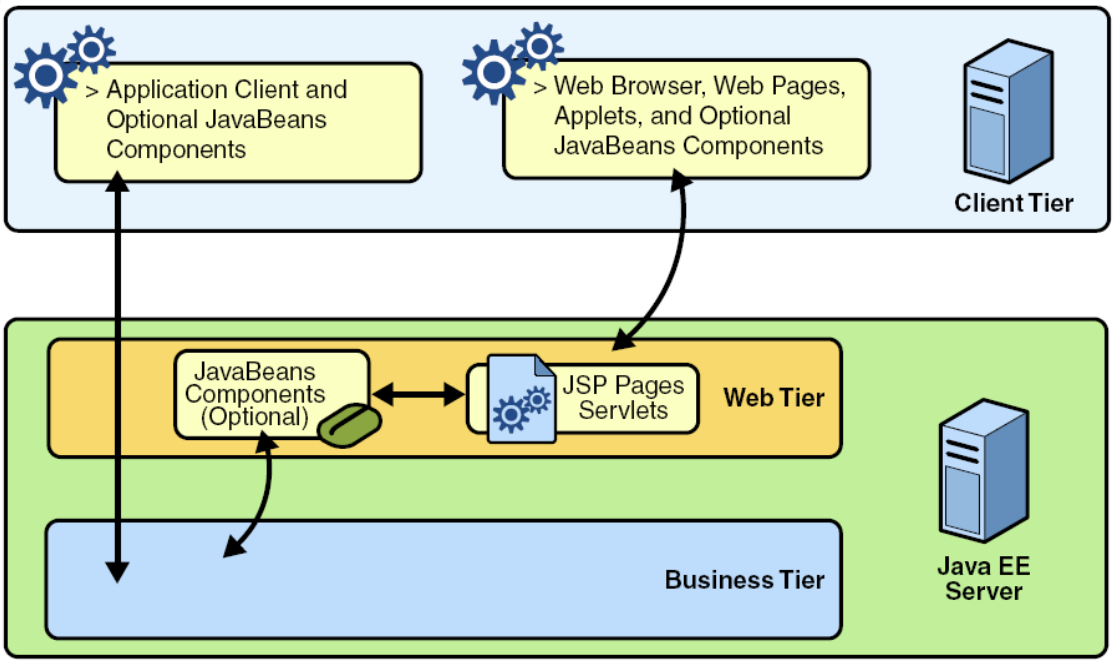


FIGURE 1–3 Web Tier and Java EE Applications

- un subsistem (care poate fi distribuit) care indeplineste sarcinile aplicatiei (***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)

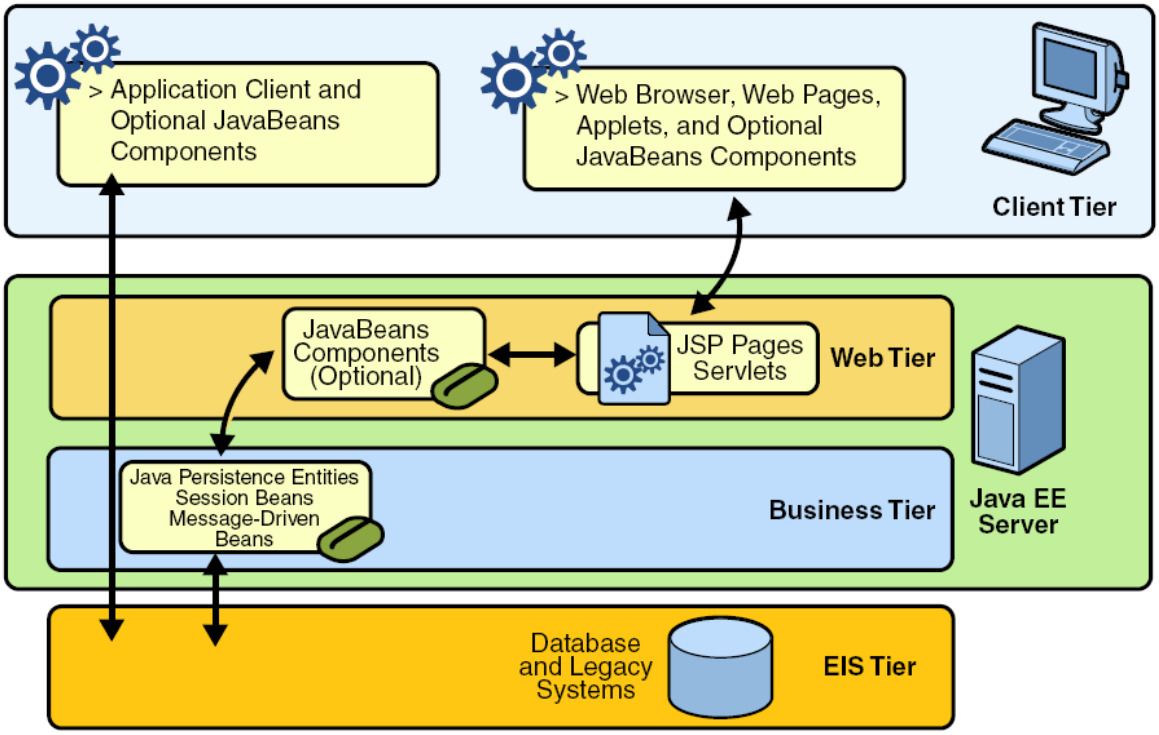


FIGURE 1–4 Business and EIS Tiers

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**:

- **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

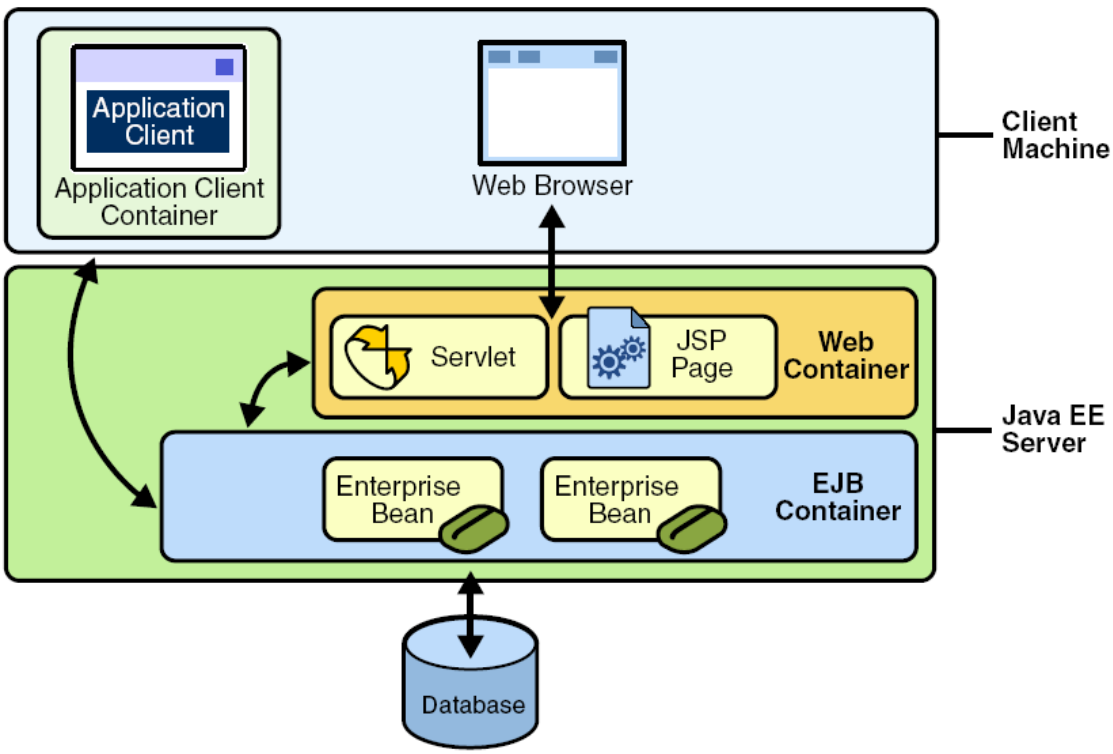
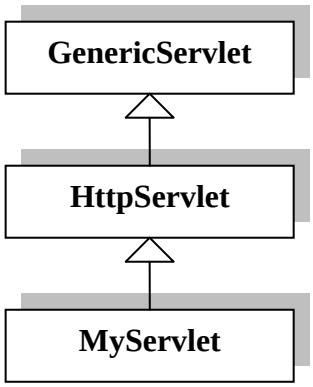


FIGURE 1–5 Java EE Server and Containers

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.



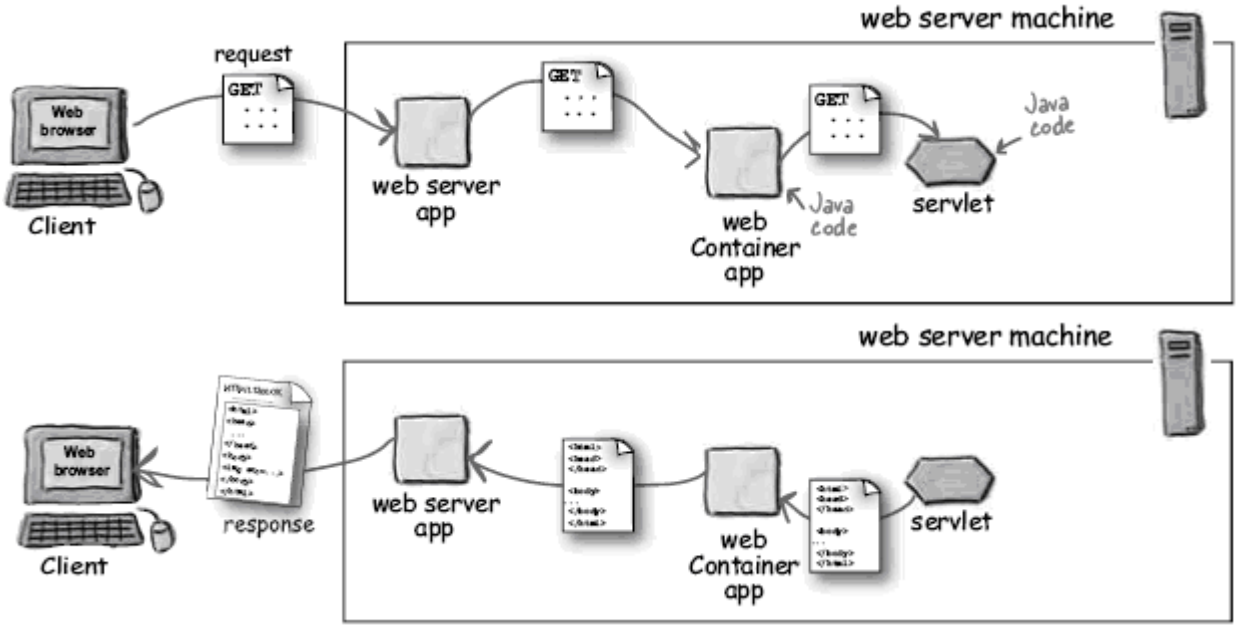
Functionalitate generica adaugata server-ului

Functionalitate asociata protocolului HTTP

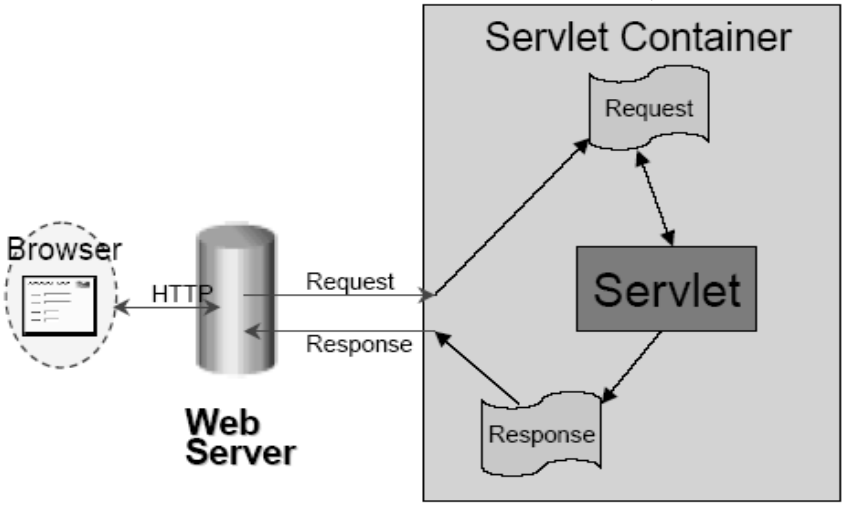
Functionalitate specifica aplicatiei

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

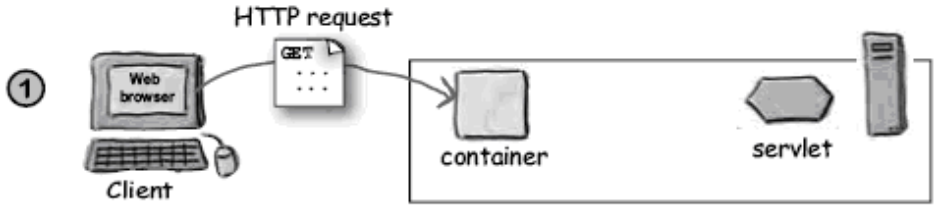
Servlet-urile Web sunt **componente** care **se executa intr-un container Web** (*Web container* sau *Web engine*), tot asa cum applet-urile sunt executate intr-un browser Web.



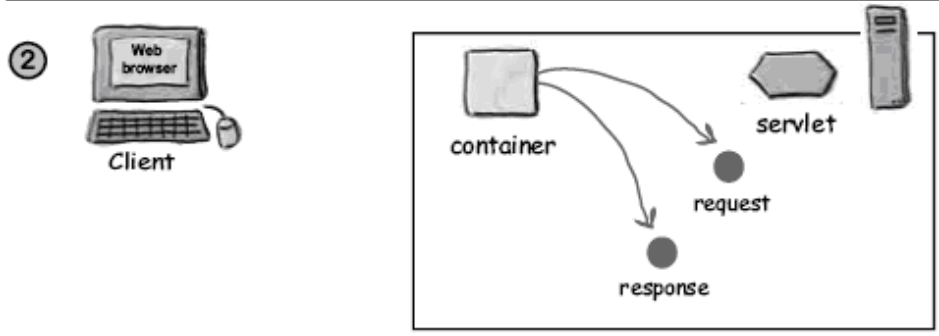
Astfel, operatiile care tin de ciclul de viata al servlet-ului (apelul metodelor **`init()`**, **`destroy()`**, **`service()`**) sunt realizate de catre container in momentele in care acestea sunt necesare (initializare, incarcare, etc.).



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



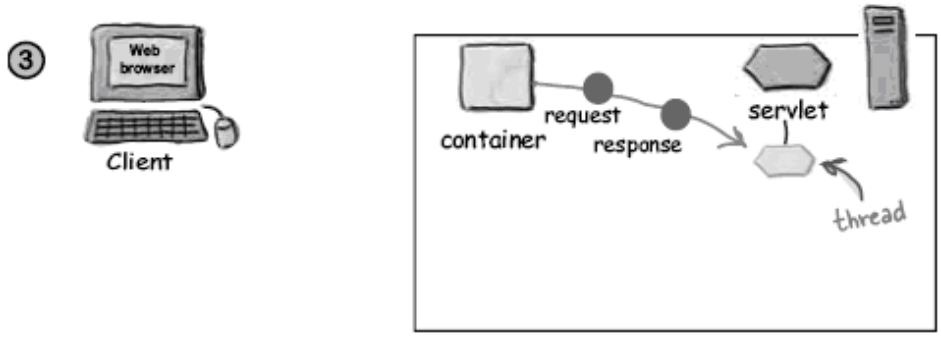
User clicks a link that has a URL to a servlet instead of a static page.



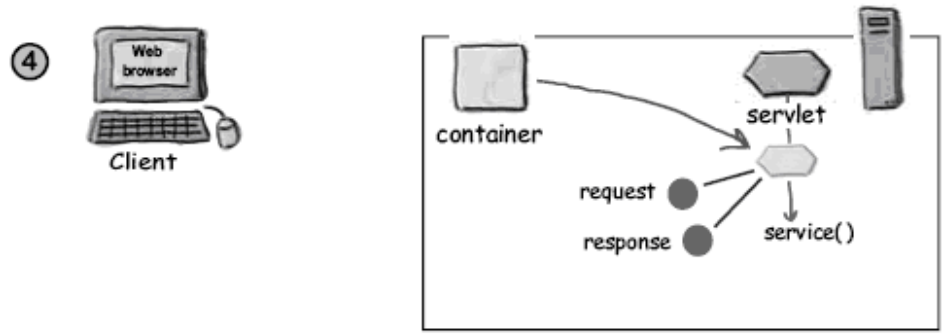
The container "sees" that the request is for a servlet, so the container creates two objects:

1) HttpServletResponse

2) HttpServletRequest

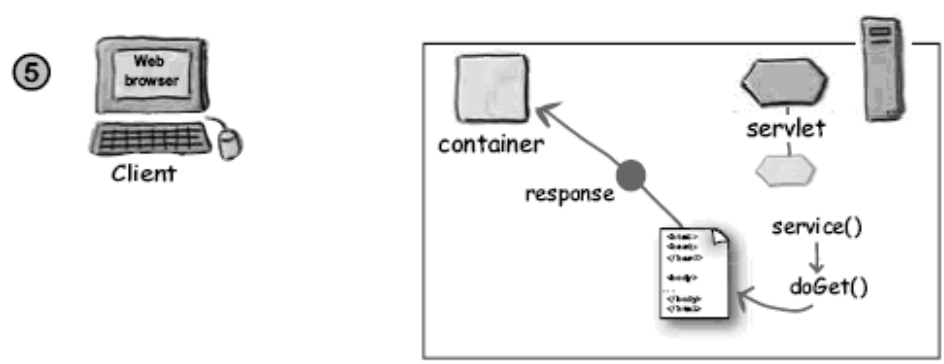


The container finds the correct servlet based on the URL in the request, creates or allocates a thread for that request, and passes the request and response objects to the servlet thread.

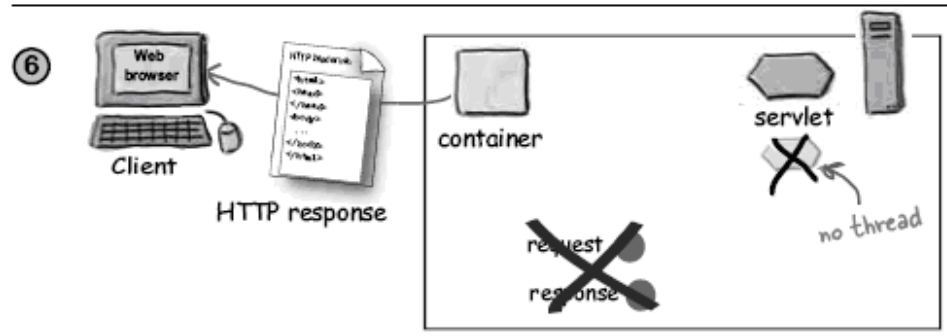


The container calls the servlet's service() method. Depending on the type of request, the service() method calls either the doGet() or doPost() method.

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

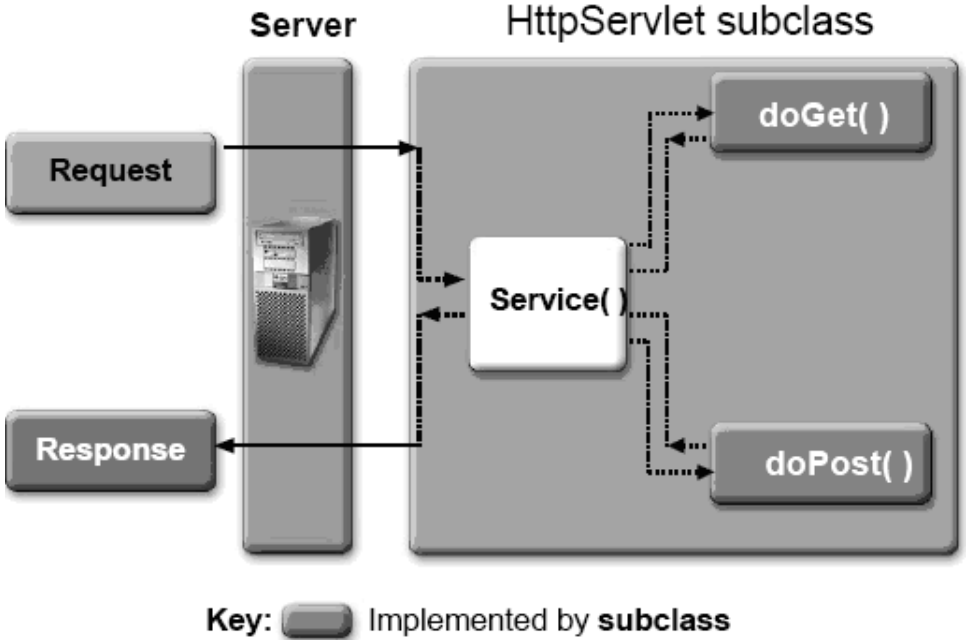


The doGet() method generates the dynamic page and stuffs the page into the response object. Remember, the container still has a reference to the response object!

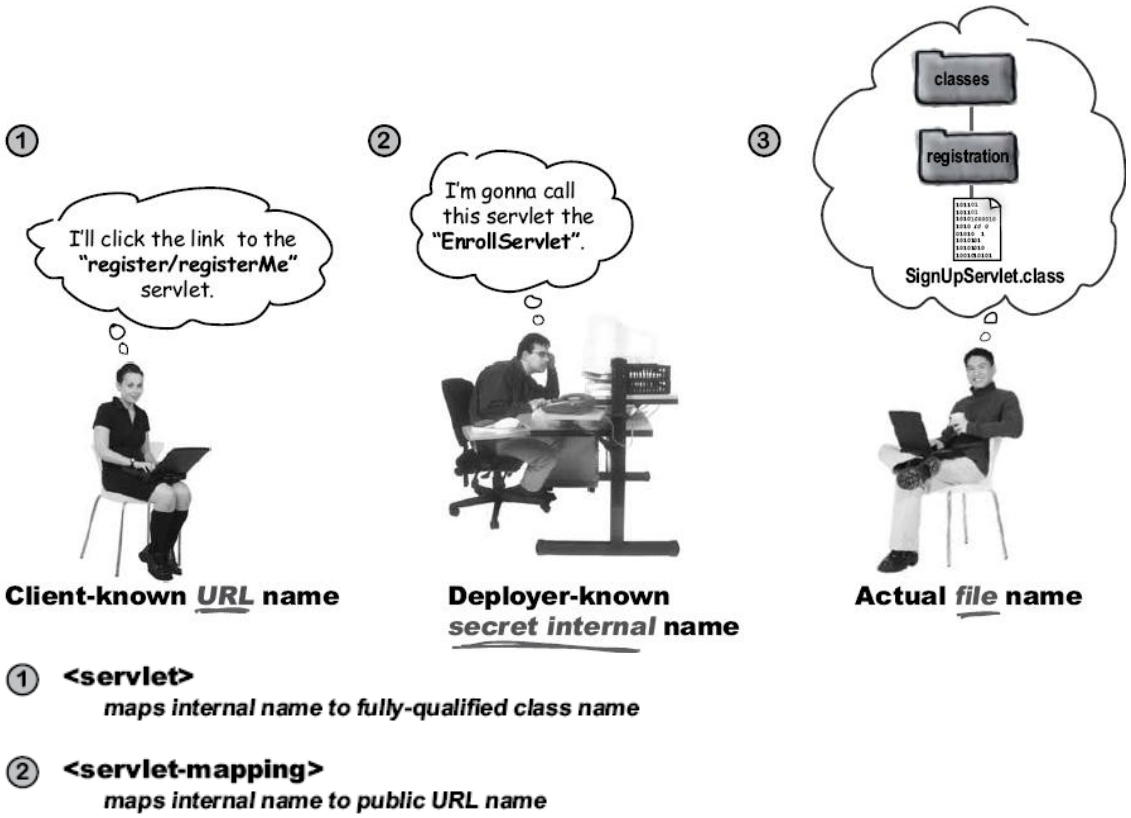


The thread completes, the container converts the response object into an HTTP response, sends it back to the client, then deletes the request and response objects.

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 la care se afla cu adevarat fisierul cu codul sursa al servlet-ului**.



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).

De exemplu, urmatorul continut al unui fisier **web.xml** specifica:

- **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`**>),

```
    <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>
```

Rolurile pe care componentele Web (servlet-urile dar si paginile JSP) le pot juca sunt:

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).

Un posibil 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
      }
    }
```

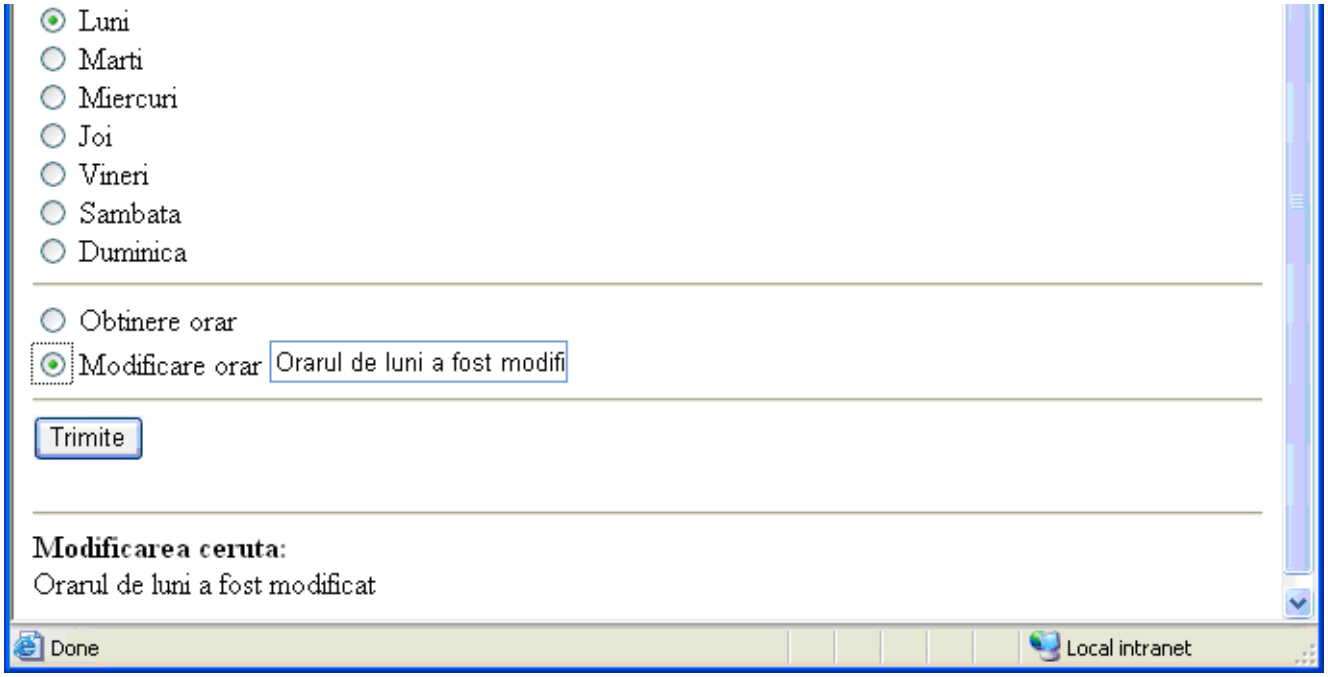
Pentru exemplificare, vom folosi din clasa Orar, accesata la distanta prin intermediul servlet-urilor:

```
 1  public class Orar {
 2      private String[] orar; // camp ascuns (starea obiectului)
 3      public Orar() {
 4          orar = new String[7]; // alocarea dinamica a spatiului pentru tablou
 5          orar[0] = "Luni este curs TPI la seriile D si E si laborator la seria E.";
 6          orar[1] = "Marti nu sunt ore de TPI.";
 7          orar[2] = "Miercuri este laborator TPI la seriile D si E.";
 8          orar[3] = "Joi este laborator TPI la seria D.";
 9          orar[4] = "Vineri este laborator TPI la seria D.";
10          orar[5] = "Sambata nu sunt ore de TPI.";
11          orar[6] = "Duminica nu sunt ore de TPI."; // popularea tabloului cu valori
12      }
13      public String getOrar(int zi) { // metoda accesor - getter
14          return orar[zi];            // returneaza referinta la tablou
15      }
16      public void setOrar(int zi, String text) { // metoda accesor - setter
17          orar[zi] = text;                       // inlocuieste un element
18      }
19  }
```

Vom incepe cu un servlet simplu care permite accesul la obiecte Orar:

```
1  import java.io.*;
2  import java.net.*;
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5
6  public class ServletOrarInitial extends HttpServlet {
7
8     // Metoda utilitara catre care doGet() si doPost() deleaga executia
9     protected void processRequest(HttpServletRequest request,
10           HttpServletResponse response) throws ServletException, IOException {
11       response.setContentType("text/html;charset=UTF-8");
12       PrintWriter out = response.getWriter();
13
14       // Generarea formularului pentru accesul recursiv la servicii
15       out.println("<html>");
16       out.println("<head>");
17       out.println("<title>Acces orar</title>");
18       out.println("</head>");
19       out.println("<body>");
20       out.println("<h1>Acces orar (forma initiala) - generat de servlet</h1>");
21       out.println("<hr><form name=\"input\" action=\"AccesInitial\""
22                + " method=\"get\">");
23       out.println("<input type=\"radio\" name=\"zi\" checked=\"checked\""
24                + " value=\"0\"> Luni");
25       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"1\"> Marti");
26       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"2\"> Miercuri");
27       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"3\"> Joi");
28       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"4\"> Vineri");
29       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"5\"> Sambata");
30       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"6\"> Duminica");
31       out.println("<hr>");
32       out.println("<input type=\"radio\" name=\"serviciu\" checked=\"checked\""
33                + " value=\"getOrar\"> Obtinere orar");
34       out.println("<br><input type=\"radio\" name=\"serviciu\" value=\"setOrar\">"
35                + " Modificare orar");
36       out.println("<input type=\"text\" name=\"modificare\" value=\"\">");
37       out.println("<hr><input type=\"submit\" value=\"Trimite\">");
38       out.println("</form><hr>");
39
40       Orar orar = new Orar();
41
42       // Obtinerea parametrilor introdusi de utilizator in formular
43       int zi = Integer.parseInt(request.getParameter("zi"));
44
45       // Daca serviciu cerut e obtinere orar
46       if (request.getParameter("serviciu").equals("getOrar")) {
47          out.println("<b>Orarul cerut:</b> <br>" + orar.getOrar(zi));
48       }
49       // Daca serviciu cerut e modificare orar
50       else if (request.getParameter("serviciu").equals("setOrar")) {
51          String modificare = request.getParameter("modificare");
52          orar.setOrar(zi, modificare);
53          out.println("<b>Modificarea ceruta:</b> <br>" + orar.getOrar(zi));
54       }
55       out.println("</body>");
56       out.println("</html>");
57       out.close();
58    }
59    // Metoda corespunzatoare cererilor HTTP de tip GET
60    protected void doGet(HttpServletRequest request, HttpServletResponse response)
61           throws ServletException, IOException {
62       processRequest(request, response);
63    }
64    // Metoda corespunzatoare cererilor HTTP de tip POST
65    protected void doPost(HttpServletRequest request, HttpServletResponse response)
66           throws ServletException, IOException {
67       processRequest(request, response);
68    }
69 }
```

Formularul generat de servlet contine in prima parte (separate prin linii orizontale) urmatoarele sectiuni:
- 7 butoane radio denumite "zi" (dintre care primul selectat - *checked*),
- 2 butoane radio denumite "serviciu" (dintre care primul selectat),
- un buton de tip "submit" cu eticheta "Trimite"
- un text generat dinamic in functie de serviciul cerut.



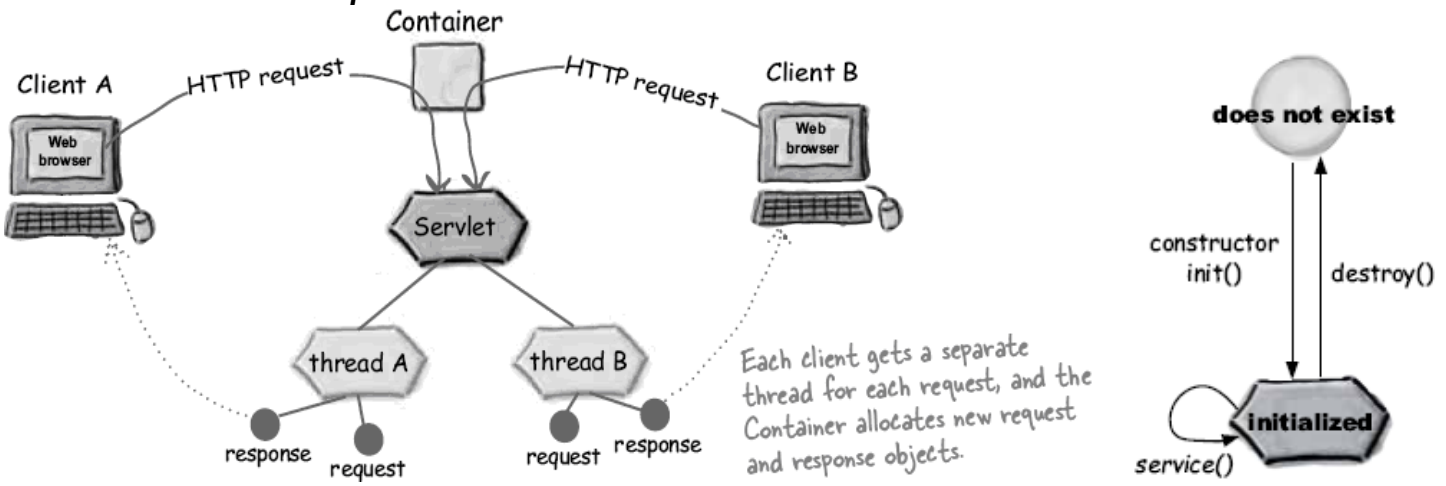
Pentru exemplul de mai sus continutul HTML generat poate fi urmatorul:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Acces orar</title>
    </head>
    <body>
      <h1> Acces orar (forma initiala) - generat de servlet </h1>
         <hr><form name="input" action="AccesInitial" method="get">
         <input type="radio" name="zi" checked="checked" value="0"> Luni
         <br> <input type="radio" name="zi" value="1"> Marti
         <br> <input type="radio" name="zi" value="2"> Miercuri
         <br> <input type="radio" name="zi" value="3"> Joi
         <br> <input type="radio" name="zi" value="4"> Vineri
         <br> <input type="radio" name="zi" value="5"> Sambata
         <br> <input type="radio" name="zi" value="6"> Duminica
         <hr><input type="radio" name="serviciu" checked="checked" value="getOrar">
               Obtinere orar
         <br> <input type="radio" name="serviciu" value="setOrar"> Modificare orar
         <input type="text" name="modificare" value="">
         <input type="submit" value="Trimite">
      </form><hr>
      <b>Modificarea ceruta:</b> <br>
      Orarul de luni a fost modificat
    </body>
</html>
```

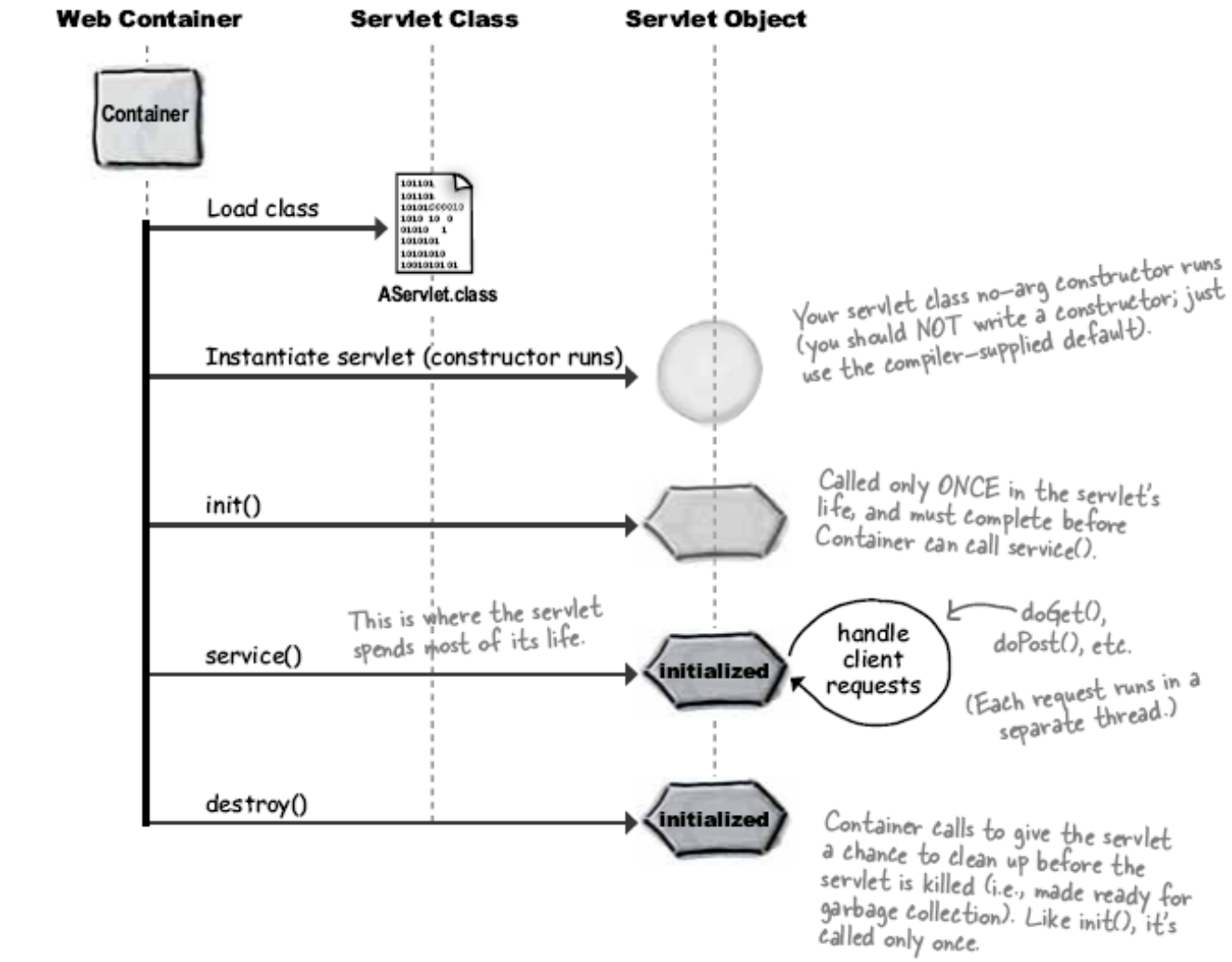
Continutul fisierului web.xml in acest caz poate fi:

```
    <?xml version="1.0" encoding="UTF-8"?>
    <web-app version="2.4" 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
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
        <servlet>
            <servlet-name>servletinitial</servlet-name>
            <servlet-class>ServletOrarInitial</servlet-class>
        </servlet>
        <servlet-mapping>
            <servlet-name>servletinitial</servlet-name>
            <url-pattern>/AccesInitial</url-pattern>
        </servlet-mapping>
        <welcome-file-list>
             <welcome-file>
                index.jsp
            </welcome-file>
        </welcome-file-list>
    </web-app>
```

Protocolul HTTP nu are stari (este *stateless*) asa incat serverul HTTP nu retine informatii privind cererile anterioare. In plus, **pentru ca servlet-urile sa fie accesate eficient de catre mai multi clienti in acelasi timp containerul de servlet-uri formeaza un asa-numit *thread pool* cu fire de executie ale servlet-ului din care alege unul oarecare pentru fiecare client.** De aceea **declararea obiectului de tip Orar ca variabila instanta nu este o solutie *thread safe*.**



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.

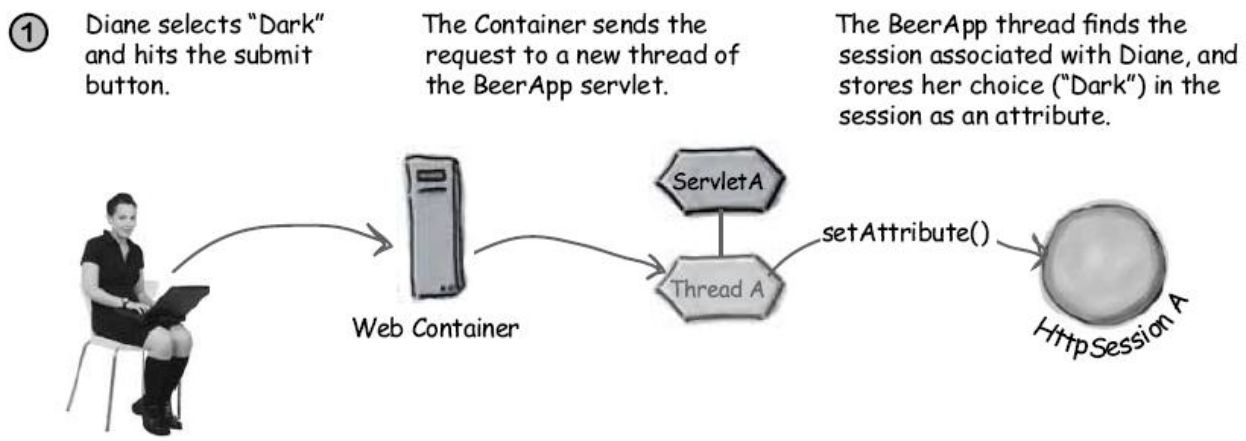


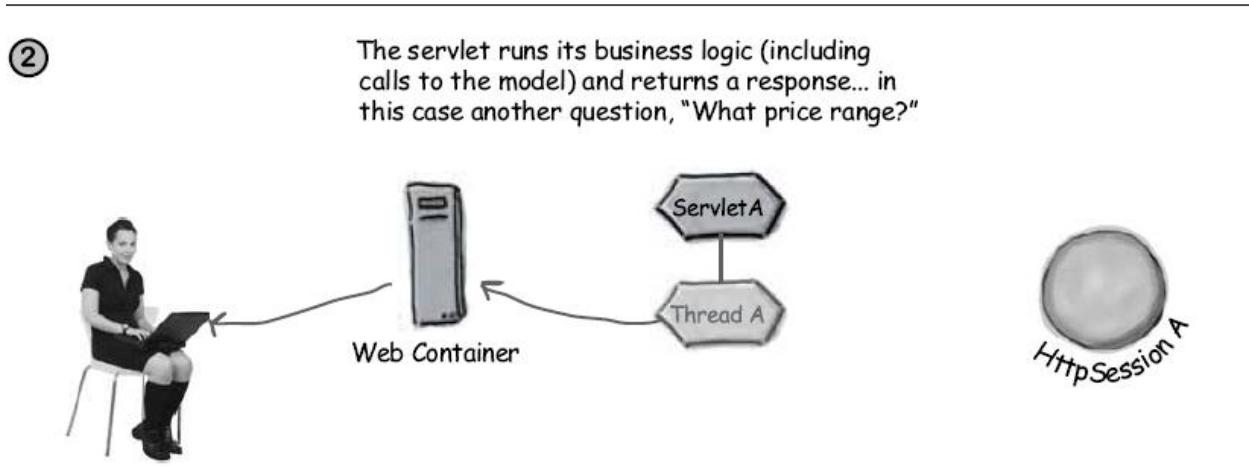
Obiectele din clasa `HttpSession` gestionate de **containerul de servlet-uri** permit **pastrarea referintelor catre obiecte ale aplicatiei**, numite **atribute**, **si regasirea acestora**, prin intermediul metodelor `setAttribute()` si `getAttribute()`.

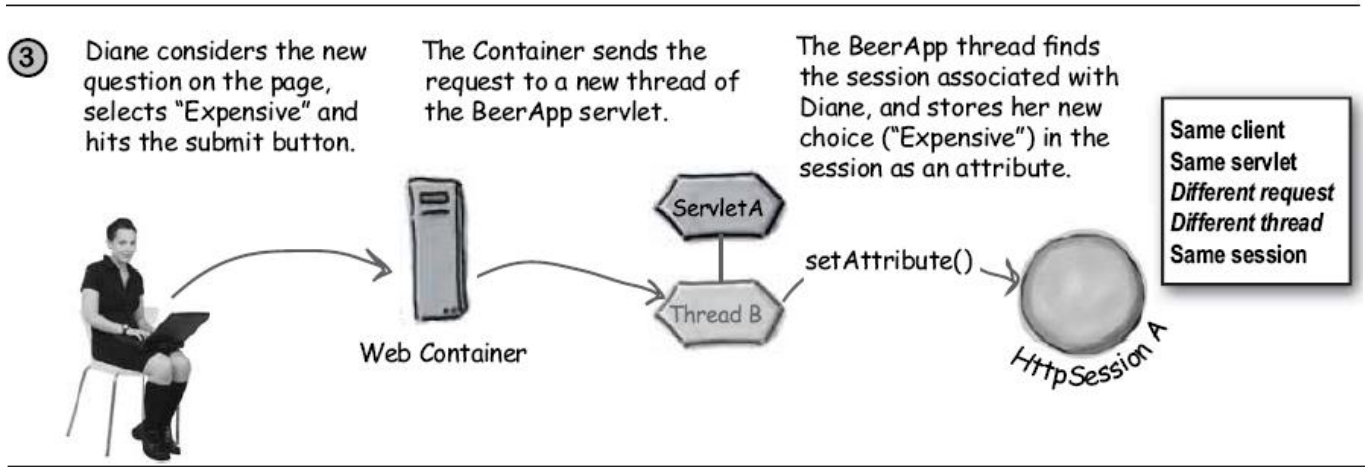
Vom modifica servletul pentru a crea si utiliza o sesiune.

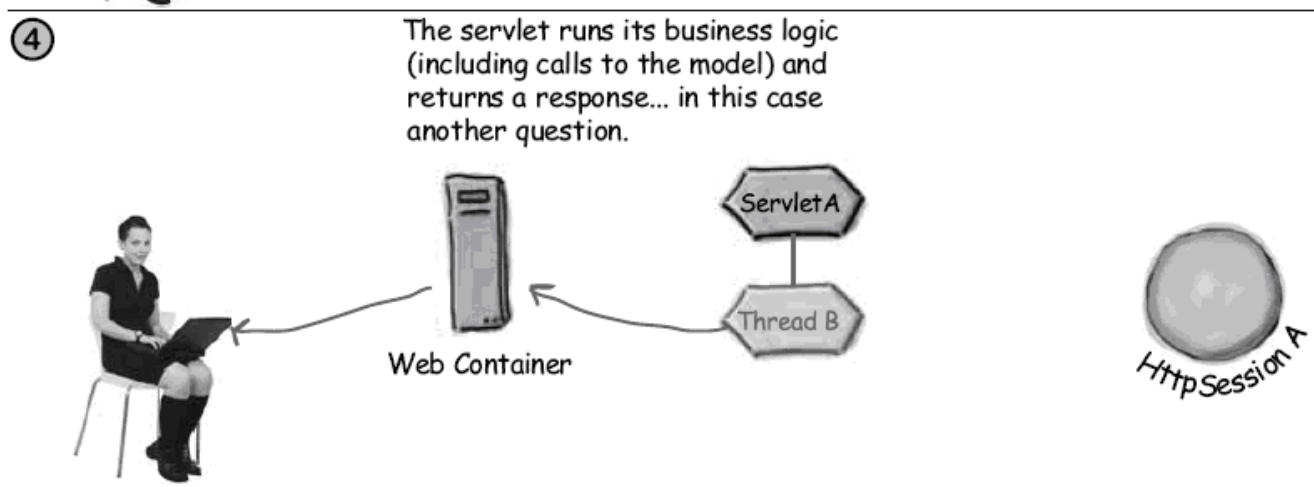
```
1  import java.io.*;
2  import java.net.*;
3  import javax.servlet.*;
4  import javax.servlet.http.*;
5
6  public class ServletOrarFinal extends HttpServlet {
7
8     protected void processRequest(HttpServletRequest request,
9           HttpServletResponse response) throws ServletException, IOException {
10       response.setContentType("text/html;charset=UTF-8");
11       PrintWriter out = response.getWriter();
12
13       // Generarea formularului pentru accesul recursiv la servicii
14       out.println("<html>");
15       out.println("<head>");
16       out.println("<title>Acces orar</title>");
17       out.println("</head>");
18       out.println("<body>");
19       out.println("<h1>Acces orar (forma finala) - generat de servlet</h1>");
20       out.println("<hr><form name=\"input\" action=\"AccesFinal\""
21                + " method=\"get\">");
22       out.println("<input type=\"radio\" name=\"zi\" checked=\"checked\""
23                + " value=\"0\"> Luni");
24       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"1\"> Marti");
25       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"2\"> Miercuri");
26       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"3\"> Joi");
27       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"4\"> Vineri");
28       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"5\"> Sambata");
29       out.println("<br> <input type=\"radio\" name=\"zi\" value=\"6\"> Duminica");
30       out.println("<hr>");
31       out.println("<input type=\"radio\" name=\"serviciu\" checked=\"checked\""
32                + " value=\"getOrar\"> Obtinere orar");
33       out.println("<br><input type=\"radio\" name=\"serviciu\" value=\"setOrar\">"
34                + " Modificare orar");
35       out.println("<input type=\"text\" name=\"modificare\" value=\"\">");
36       out.println("<hr><input type=\"submit\" value=\"Trimite\">");
37       out.println("</form><hr>");
38
39       // Transformarea obiectului orar in atribut al sesiunii curente pentru
40       // salvarea starii lui
41       HttpSession ses = request.getSession();
42       Orar orar = (Orar) ses.getAttribute("orar");
43       if (orar == null) { // Daca nu exista orarul salvat ca atribut al sesiunii
44          orar = new Orar();
45          ses.setAttribute("orar", orar);
46       }
47
48       // Obtinerea parametrilor introdusi de utilizator in formular
49       int zi = Integer.parseInt(request.getParameter("zi"));
50
51       // Daca serviciu cerut e obtinere orar
52       if (request.getParameter("serviciu").equals("getOrar")) {
53          out.println("<b>Orarul cerut:</b> <br>" + orar.getOrar(zi));
54       }
55       // Daca serviciu cerut e modificare orar
56       else if (request.getParameter("serviciu").equals("setOrar")) {
57          String modificare = request.getParameter("modificare");
58          orar.setOrar(zi, modificare);
59          out.println("<b>Modificarea ceruta:</b> <br>" + orar.getOrar(zi));
60       }
61       out.println("</body>");
62       out.println("</html>");
63       out.close();
64    }
65    protected void doGet(HttpServletRequest request, HttpServletResponse response)
66             throws ServletException, IOException {
67       processRequest(request, response);
68    }
69    protected void doPost(HttpServletRequest request, HttpServletResponse response)
70             throws ServletException, IOException {
71       processRequest(request, response);
72    }
73 }
```

Modul de lucru cu sesiunile:

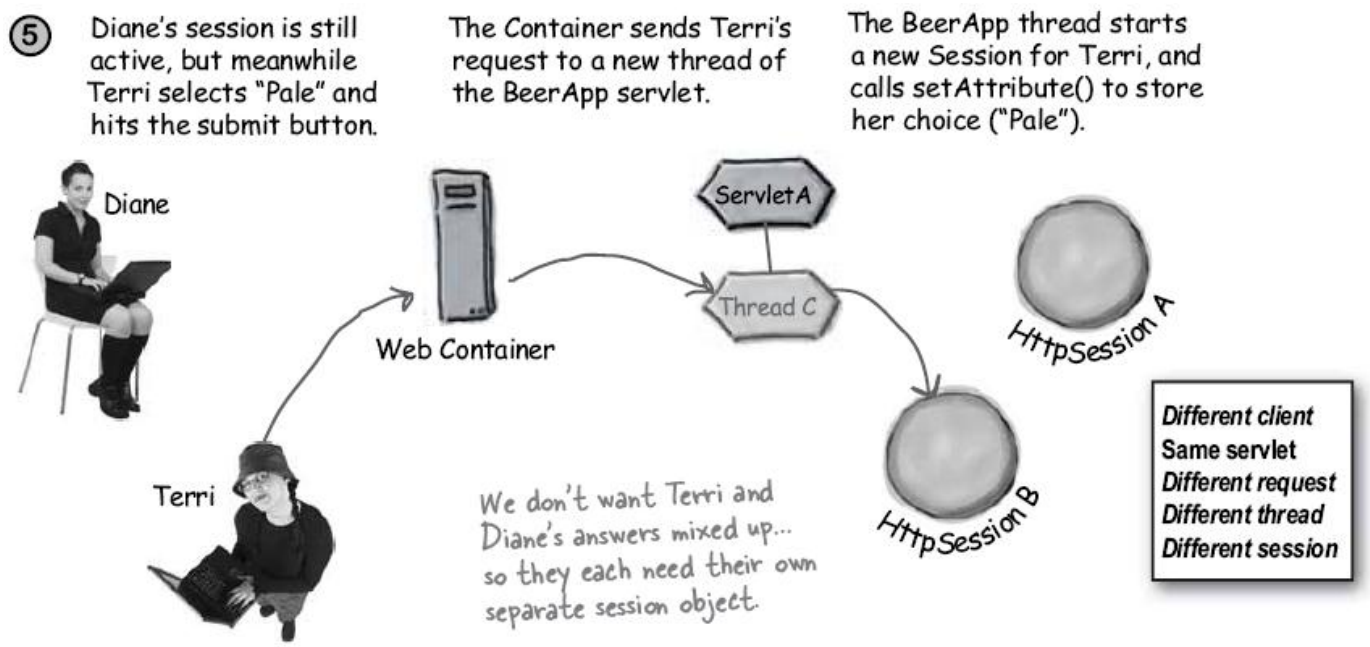




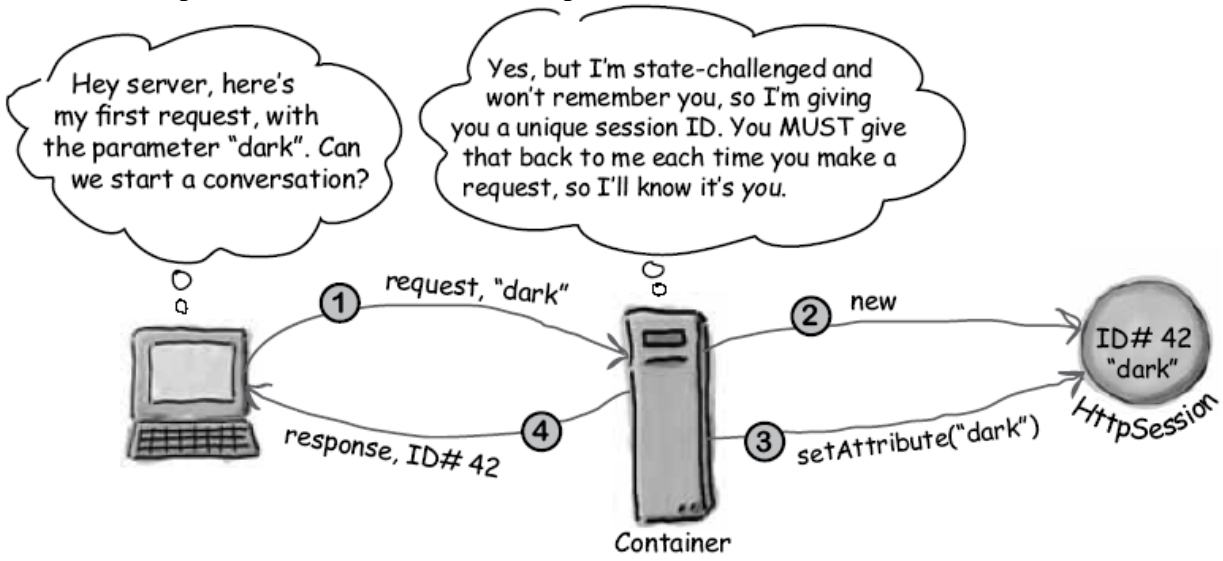


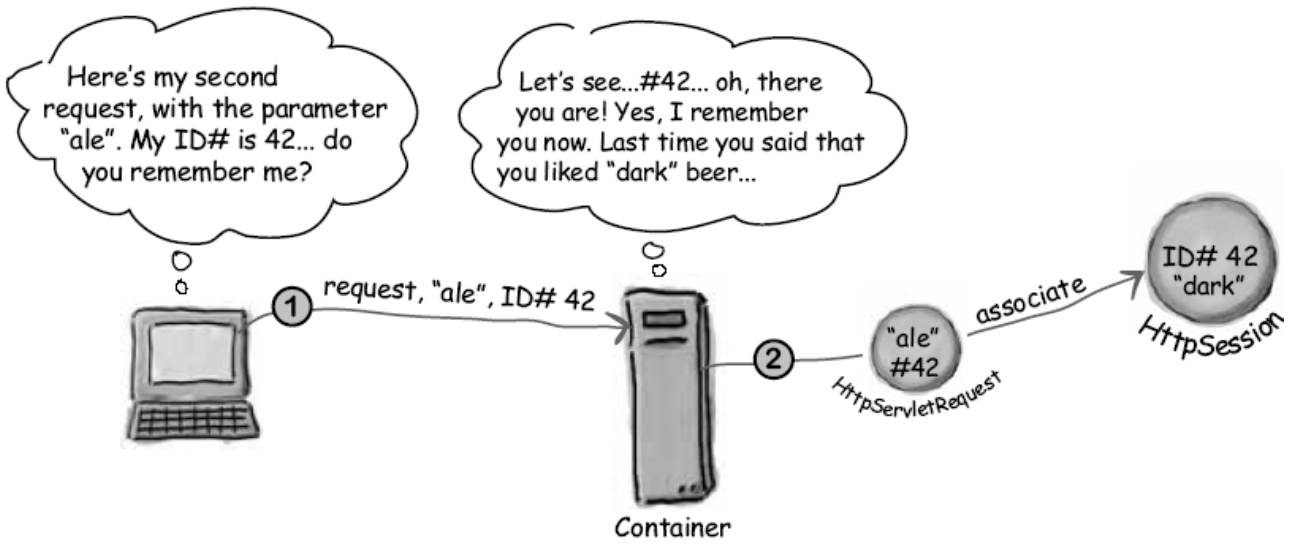


Daca un alt client acceseaza acelasi servlet, el primeste o alta sesiune:

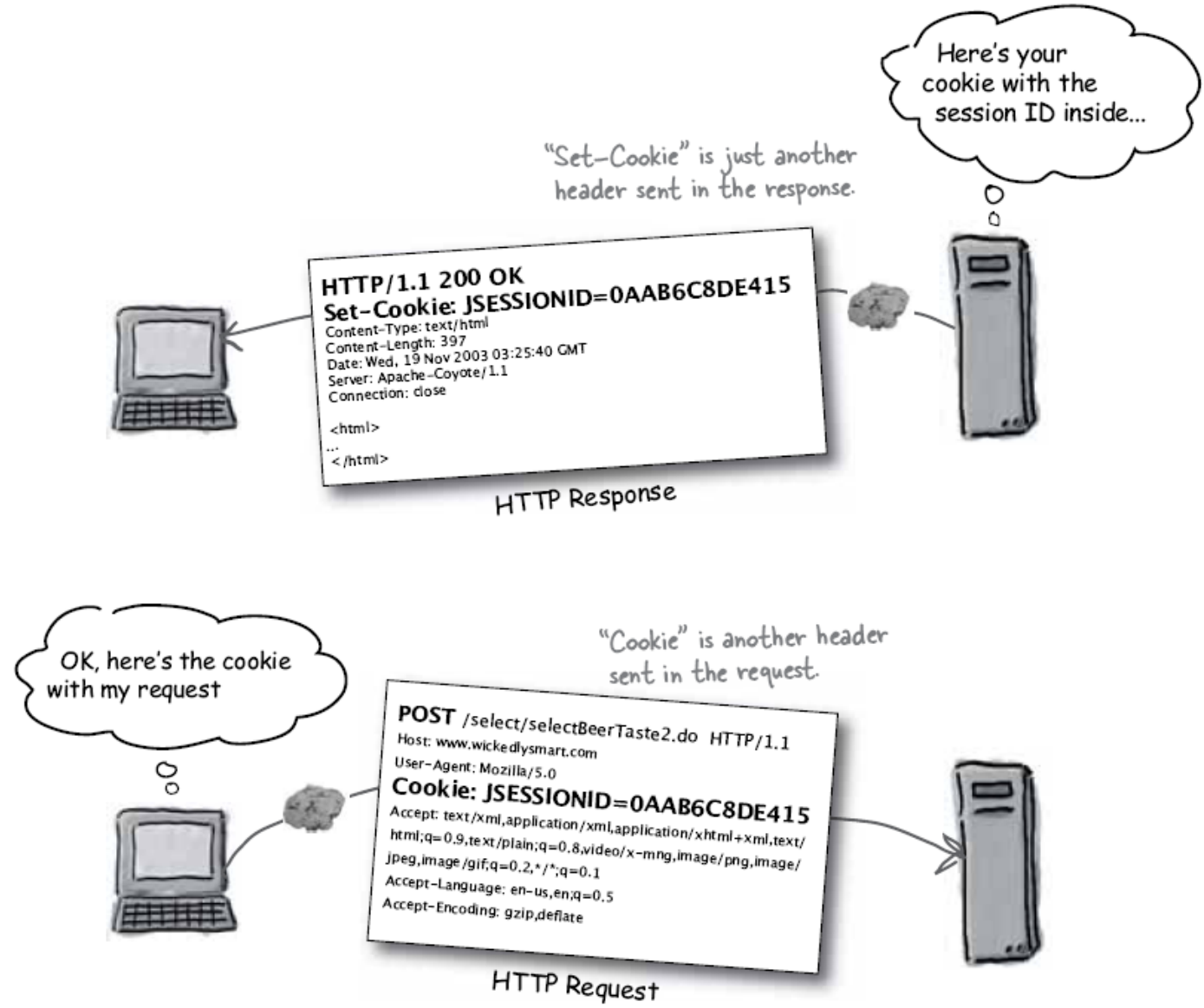


Fiecare client primeste o sesiune identificata printr-un *session ID*:





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



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.

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.

Pentru cazul in care pot fi dezactivate cookie-urile se foloseste SI rescrierea URL-ului:

realizata daca programatorul specifica o codare a URL-ului:

```
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.

Lucrul cu cookie-uri: - 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.

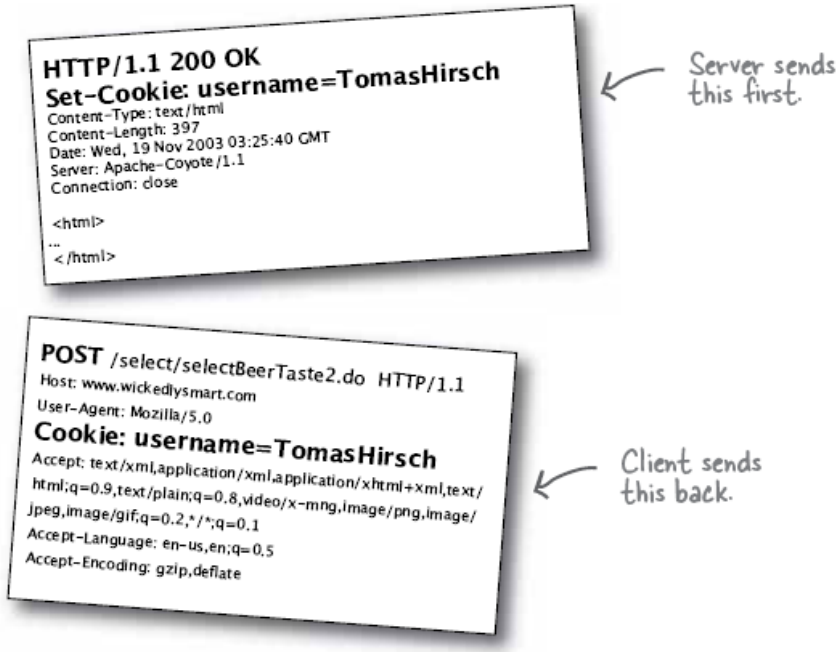
Let a JSP make the response page.

Lucrul cu cookie-uri: - clasa 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.



Servlet-ul urmator 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**"):

```
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.

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

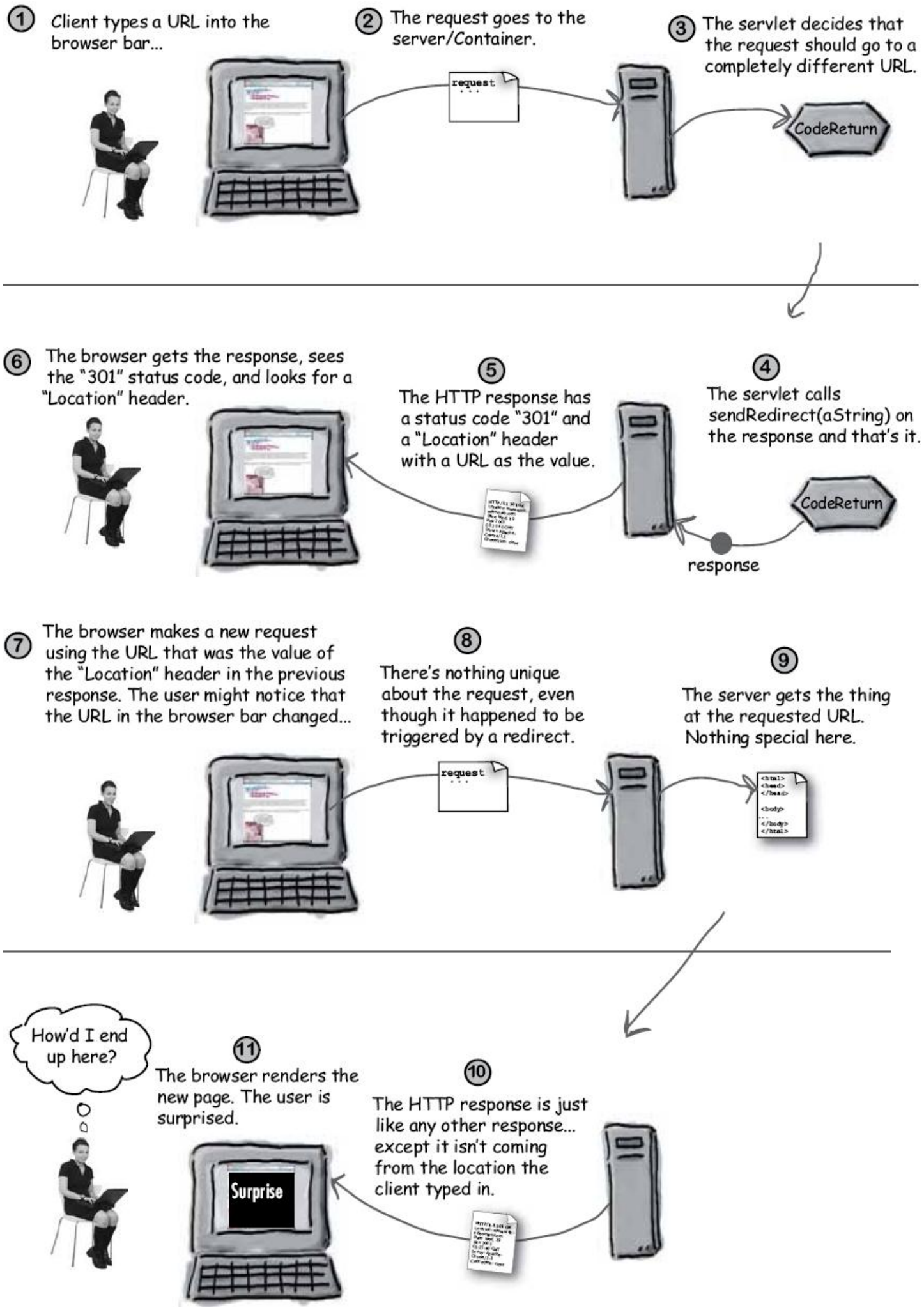
```
PrintWriter writer = response.getWriter();

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

sau fluxuri de octeti:

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

out.write(aByteArray);
```

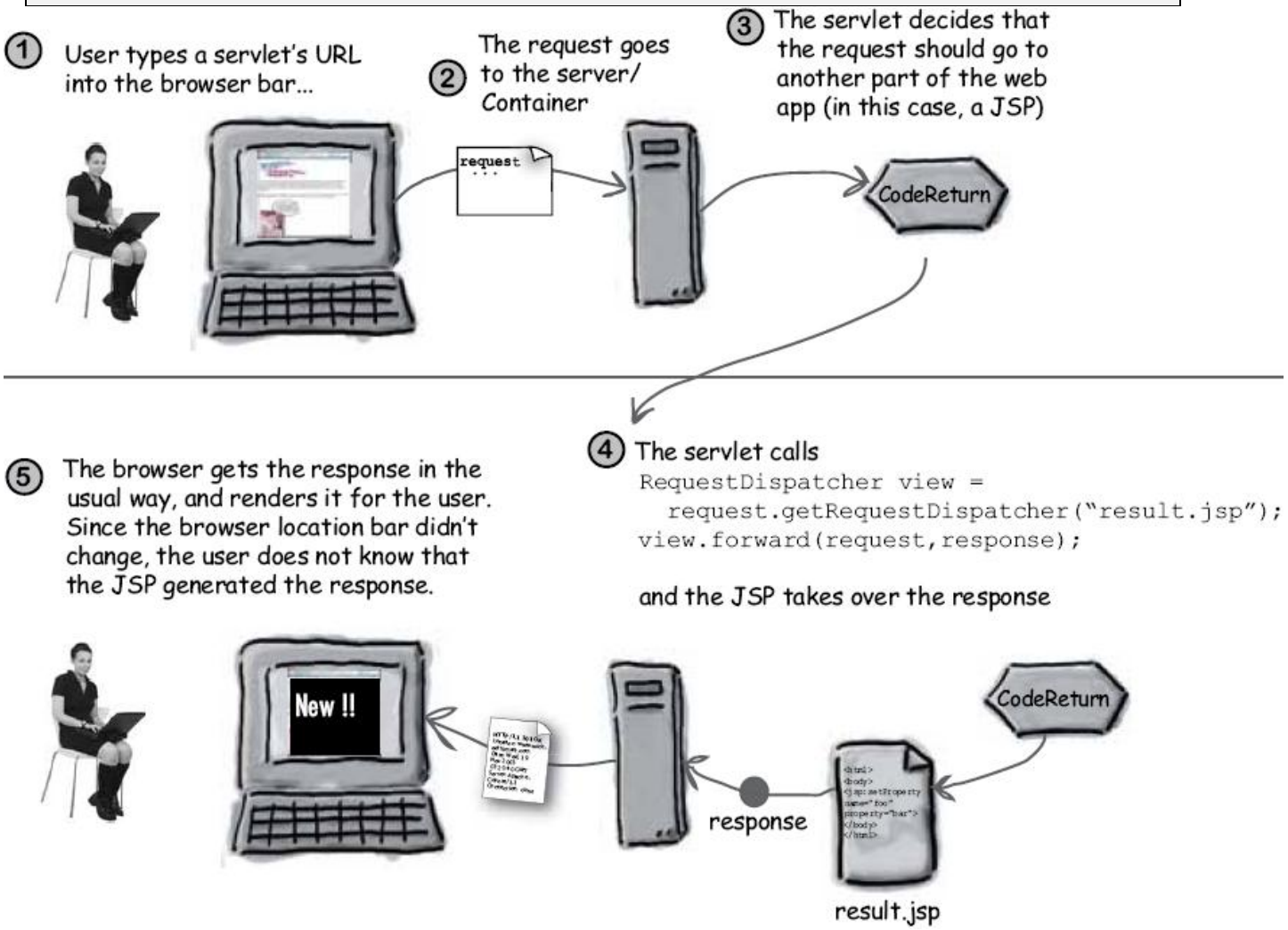
Servlet-urile pot redirecta browser-ul catre un alt URL folosind metoda **sendRedirect()**:

```
if (worksForMe) {
  // handle the request
} else {
   response.sendRedirect("http://www.oreilly.com");
}
```


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.
request
9 The server gets the thing at the requested URL. Nothing special here.
10 The HTTP response is just like any other response... except it isn't coming from the location the client typed in.
11 The browser renders the new page. The user is surprised.
Surprise
How'd I end up here?

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>
```

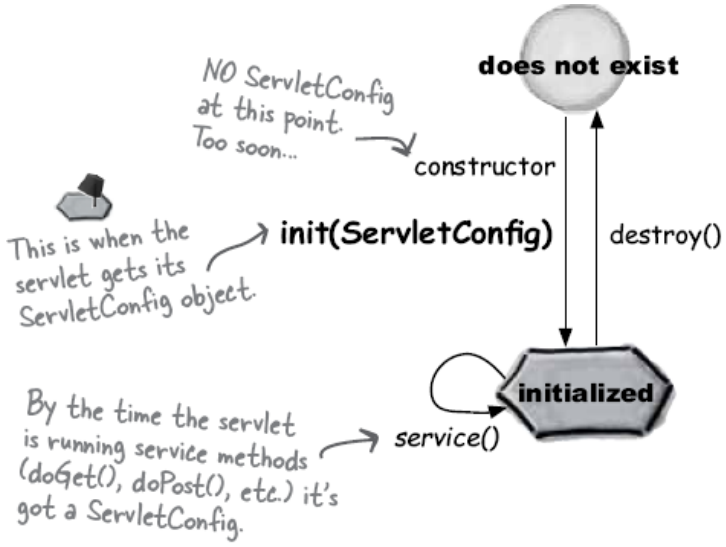
You give it a param-name and a param-value. Simple. Just make sure it's INSIDE the <servlet> element in the DD.

prin intermediul obiectului **ServletConfig asociat servlet-ului**:

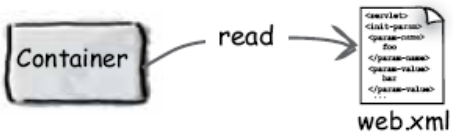
```
out.println(getServletConfig().getInitParameter("adminEmail"));
```

Every servlet inherits a getServletConfig() method.

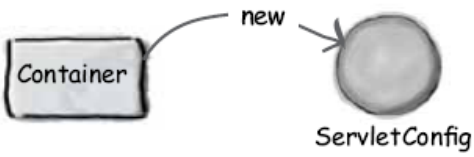
The getServletConfig() method returns a... wait for it... ServletConfig. And one of its methods is getInitParameter().



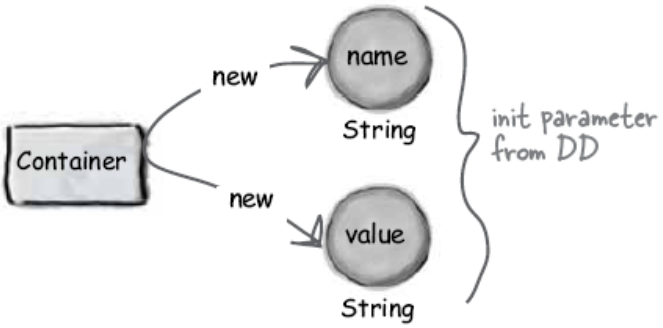
1. Container reads the Deployment Descriptor for this servlet, including the servlet init parameters (<init-param>).



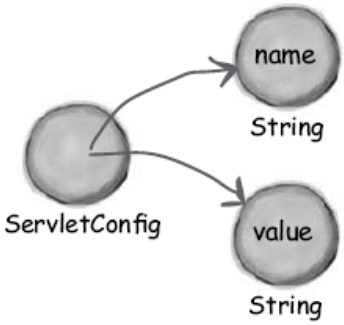
2. Container creates a new ServletConfig instance for this servlet.



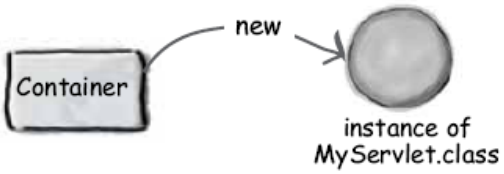
3. Container creates a name/value pair of Strings for each servlet init parameter. Assume we have only one.



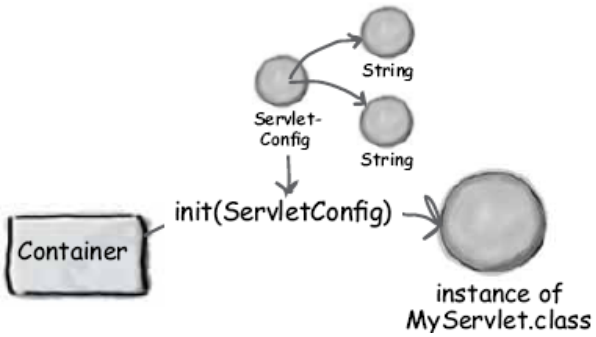
4. Container gives the ServletConfig references to the name/value init parameters.



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.



Exemplu de **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 aplicatiei Web in ansamblu**:

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

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

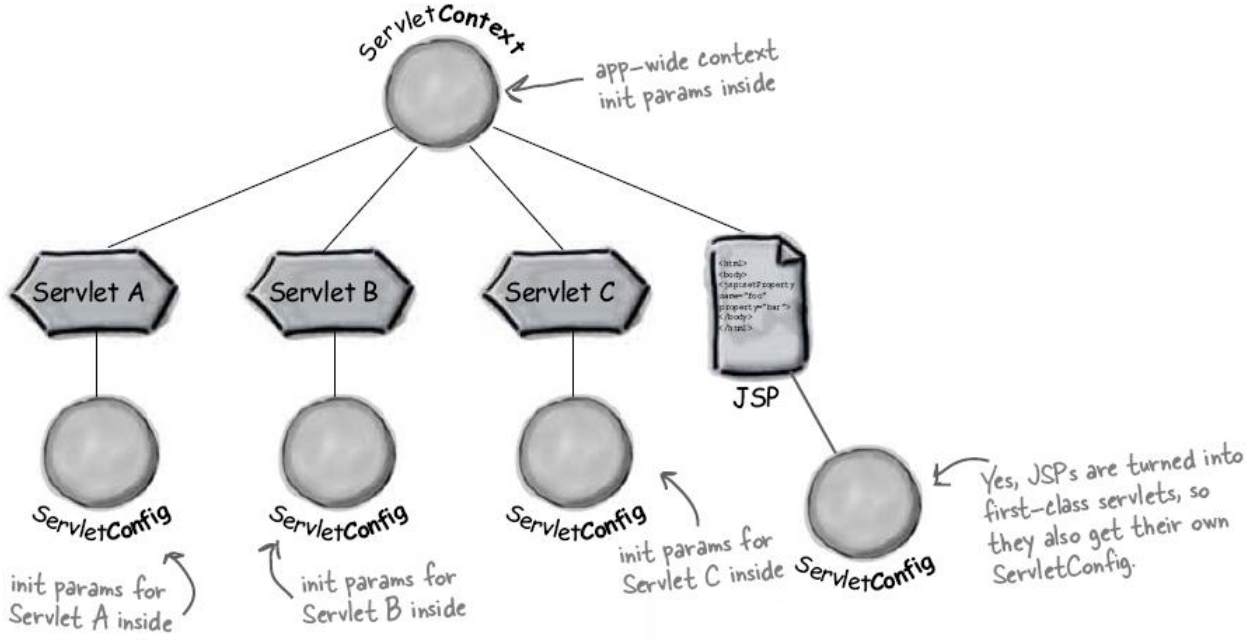
The getServletContext() method returns, surprisingly, a ServletContext object. And one of its methods is getInitParameter().

Comparatia modurilor de initializare:

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.)



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.

Clasa atribut (resursa facuta disponibila intregii aplicatii):

```
package com.example;

public class Dog {
   private String breed;

   public Dog(String breed) {
     this.breed = breed;
   }

   public String getBreed() {
      return breed;
   }
}
```

Nothing special here. Just a plain old Java class.

(We'll use the context init parameter as the argument for the Dog constructor.)

Our servlet will get the Dog from the context (the Dog that the listener sets as an attribute), call the Dog's getBreed() method, and print out the breed in the response so we can see it in the browser.

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.

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.

don't forget the cast!!

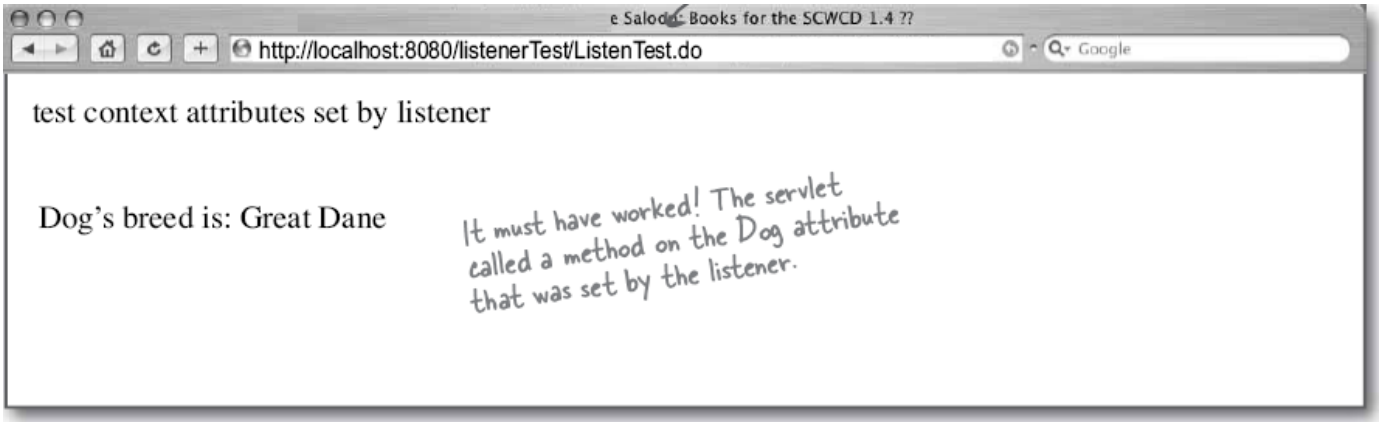
If things didn't work, THIS is where we'll find out... we'll get a big fat NullPointerException if we try to call getBreed() and there's no Dog.

Continutul **web.xml**:

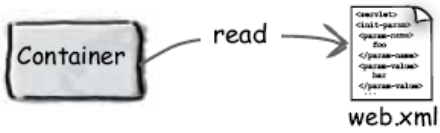
```
<web-app xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd"
    version="2.4">

  <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.

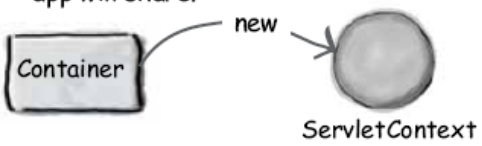
Register this class as a listener. IMPORTANT: the <listener> element does NOT go inside a <servlet> element. That wouldn't work because a context listener is for a ServletContext (which means application-wide) event. The whole point is to initialize the app BEFORE any servlets are initialized.



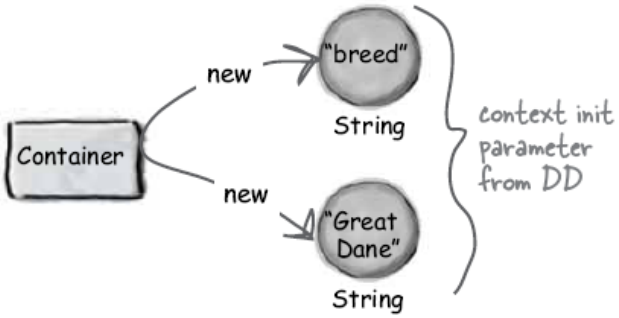
1. Container reads the Deployment Descriptor for this app, including the <listener> and <context-param> elements.



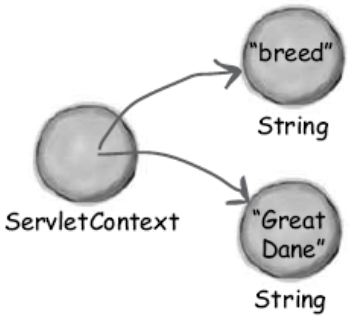
2. Container creates a new ServletContext for this application, that all parts of the app will share.



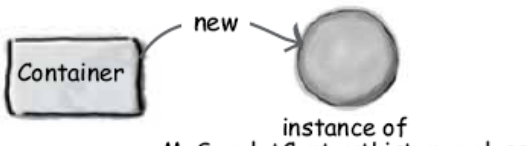
3. Container creates a name/value pair of Strings for each context init parameter. Assume we have only one.



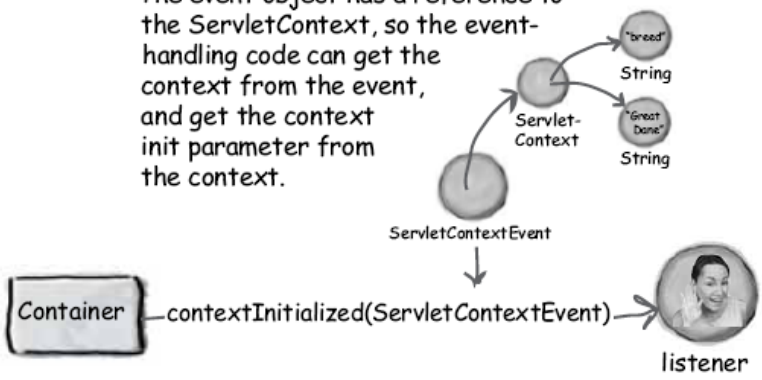
4. Container gives the ServletContext references to the name/value parameters.



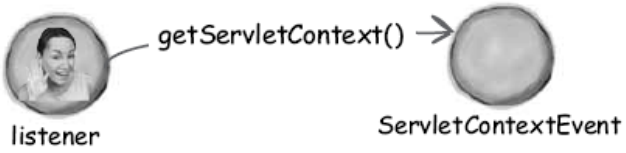
5. Container creates a new instance of the MyServletContextListener class.



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.



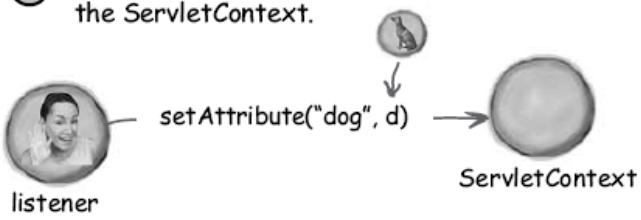
⑦ Listener asks ServletContextEvent for a reference to the ServletContext.



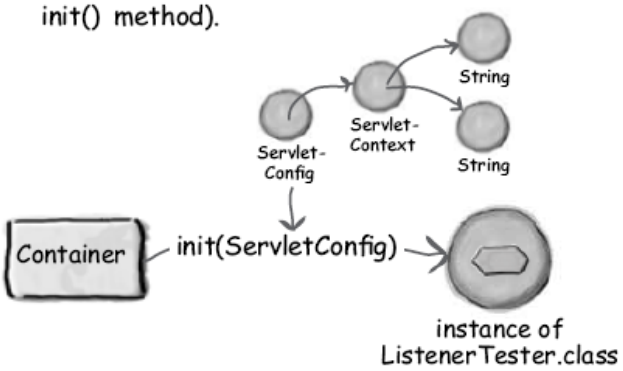
⑧ Listener asks ServletContext for the context init parameter "breed" .

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

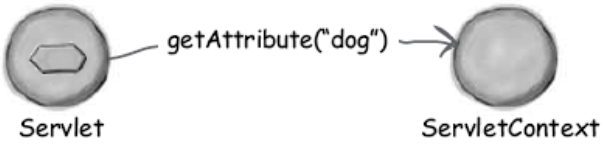
⑩ Listener sets the Dog as an attribute in the ServletContext.



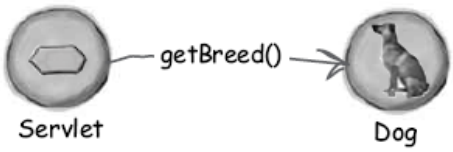
⑪ Container makes a new Servlet (i.e., makes a new ServletConfig with init parameters, gives the ServletConfig a reference to the ServletContext, then calls the Servlet's init() method).



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).

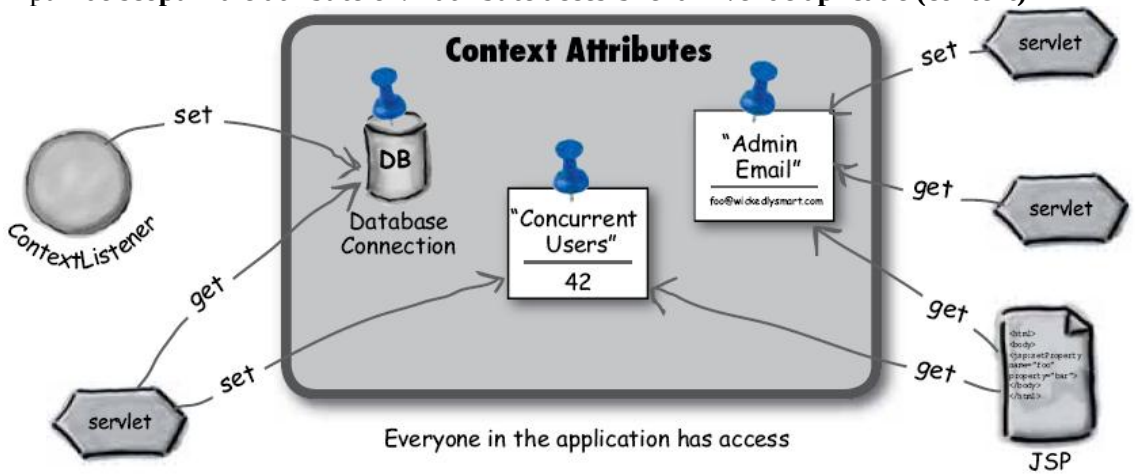


	Attributes	Parameters
Types	Application/context Request Session There is no servlet-specific attribute (just use an instance variable).	Application/context init parameters Request parameters Servlet init parameters No such thing as session parameters!
Method to set	setAttribute(String name, Object value)	You CANNOT *set* Application and Servlet init parameters—they're set in the DD, remember? (With Request parameters, you *can* adjust the query String, but that's different.)
Return type	Object	String ← Big difference!
Method to get	getAttribute(String name) Don't forget that attributes must be cast, since the return type is Object.	getInitParameter(String name)

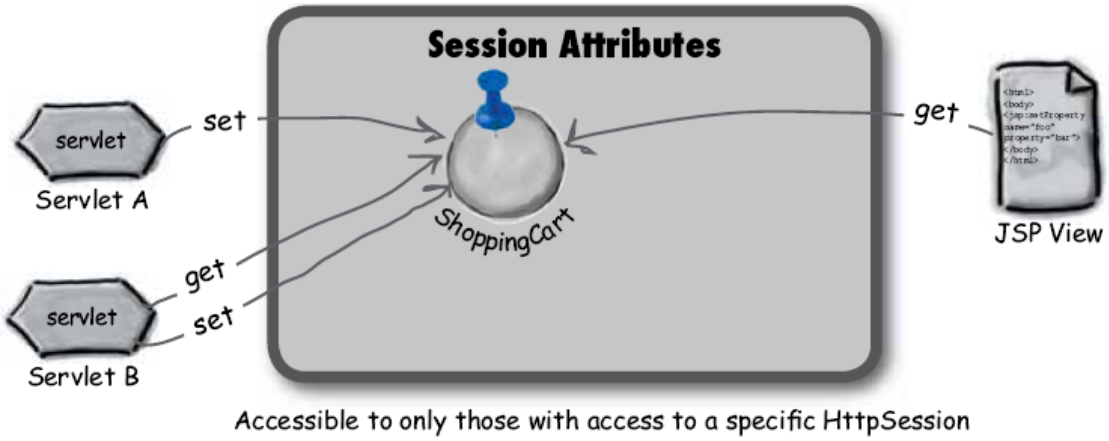
Tipuri de ascultatori de evenimente si **evenimentele asociate**

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* Watch out for this naming inconsistency! The Event for HttpSessionAttributeListener is NOT what you expect (you expect HttpSessionAttributeEvent).	HttpSessionBindingEvent
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)**

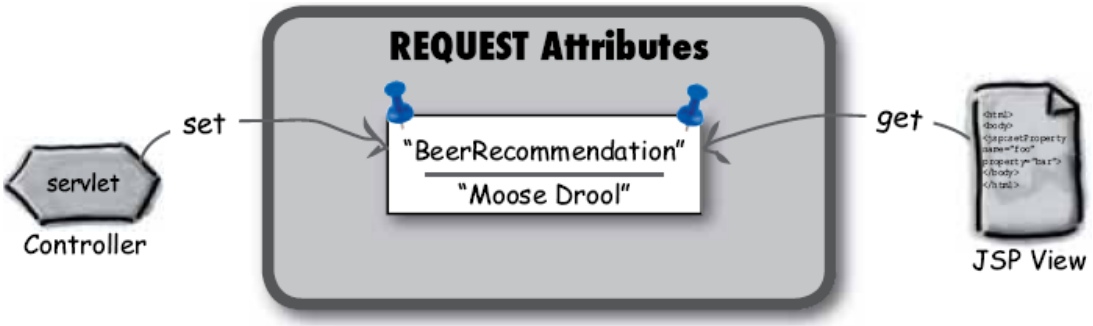


- atribute accesibile la nivel de sesiune:



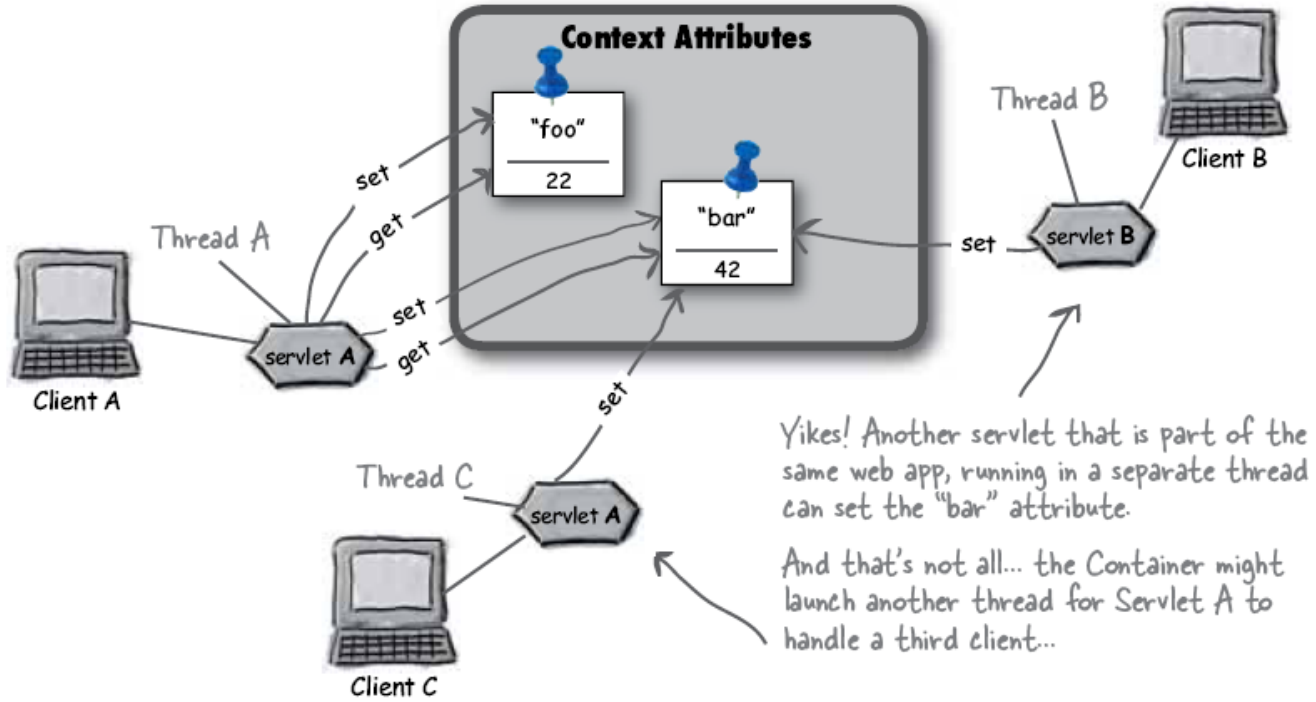
Accessible to only those with access to a specific HttpSession

- atribute accesibile la nivel de cerere:



Accessible to only those with access to a specific ServletRequest

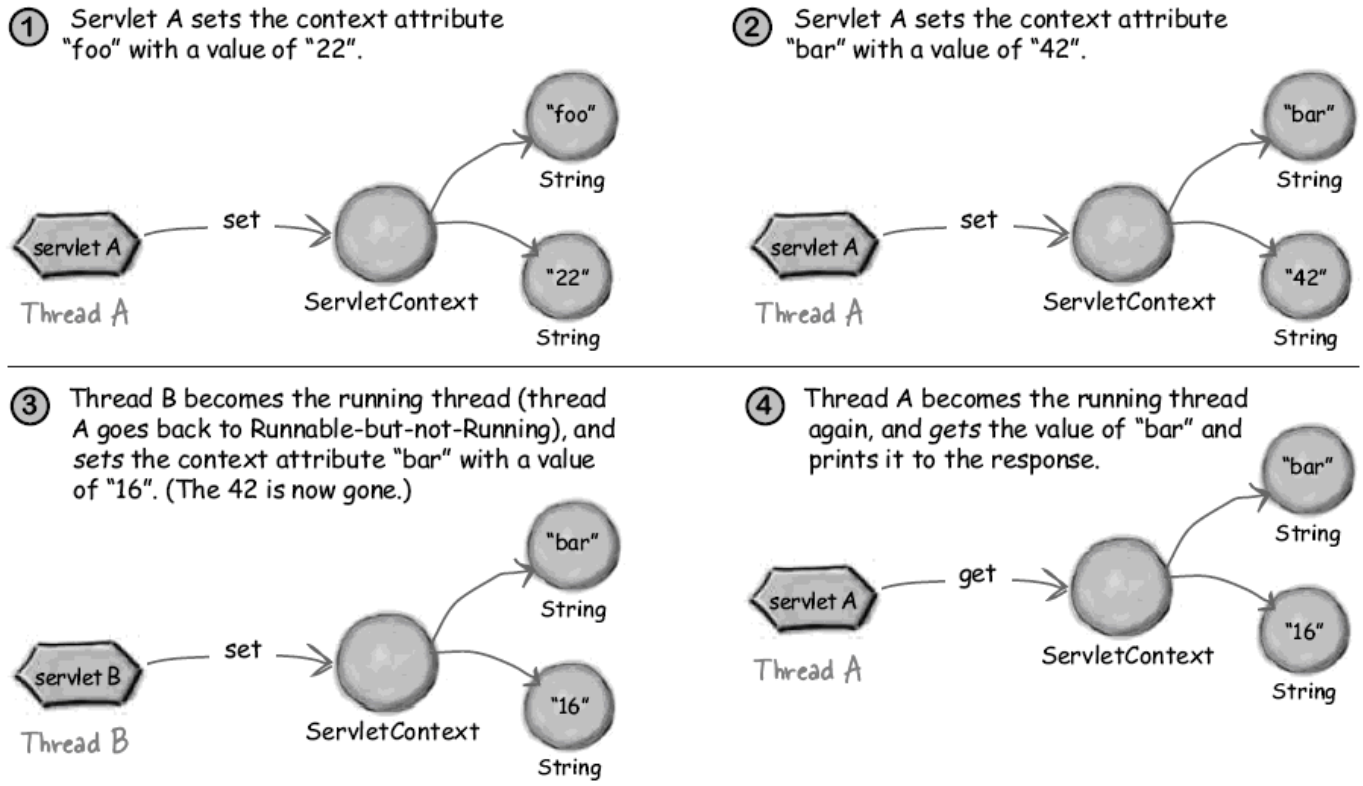
Atributele la nivel de context nu sunt *thread safe*.



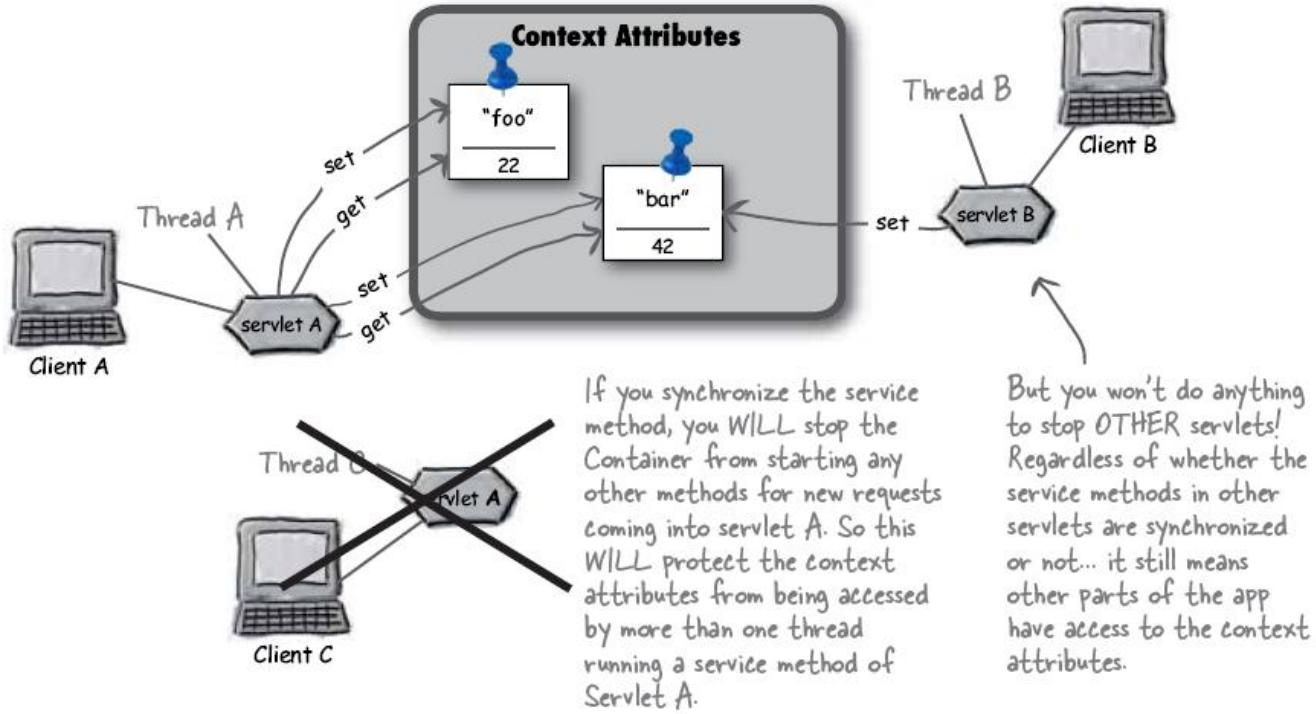

```
getServletContext().setAttribute("foo", "22");
getServletContext().setAttribute("bar", "42");

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".



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).)

4.5. Tehnologia Java ServerPages (JSP)

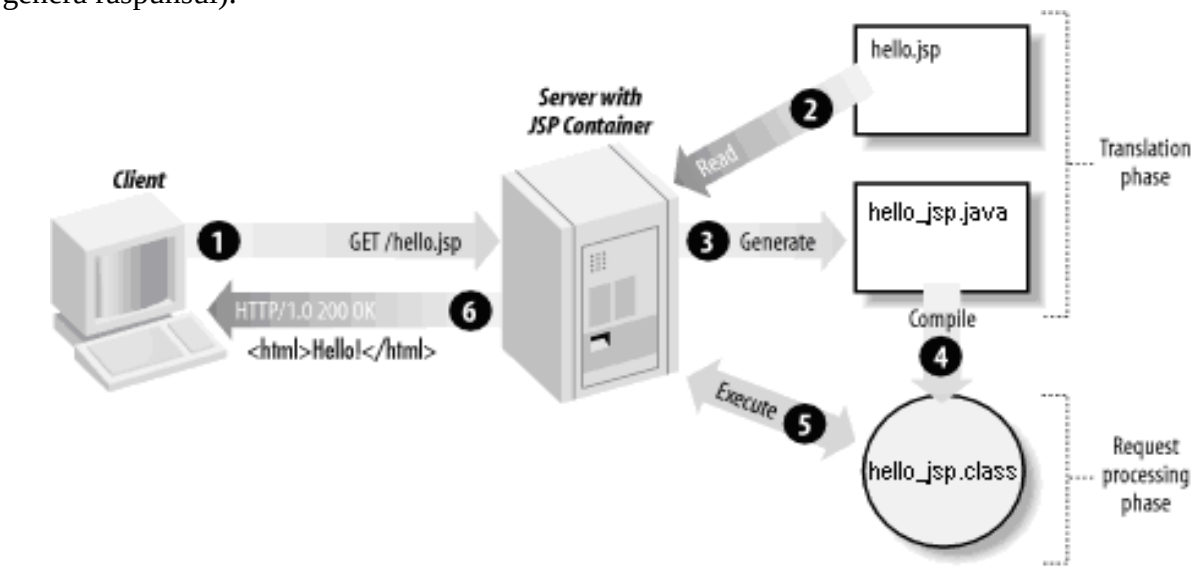
Servlet-urile au unele dezavantaje. Trebuie scrise String-uri complexe (care includ caractere escape, de exemplu `\"` in locul fiecarei ghilimele care trebuie trimisa pe flux pentru a face parte din continutul HTML) pentru fiecare linie de cod HTML care urma sa ii fie trimisa clientului. Sunt necesare cunostinte de Java pentru a scrie intreg codul unui servlet.

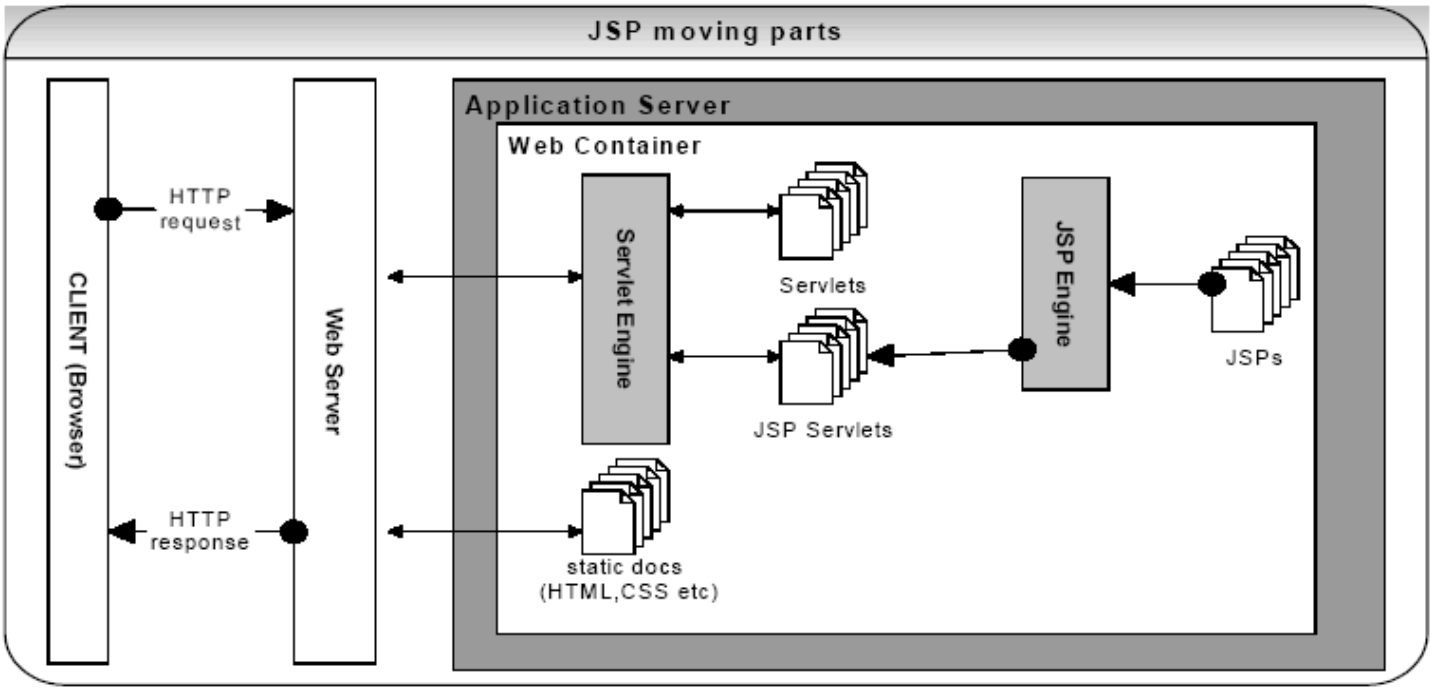
Pentru ca puterea servlet-urilor Java sa fie pusa la dispozitia celor ce se ocupa de dezvoltare web fara a-i obliga sa invete Java a aparut specificatia *Java ServerPages* (JSP), care combina puterea si extensibilitatea limbajului Java cu simplitatea si usurinta de folosire a scripturilor pe baza de etichete.

O pagina JSP este un document text ce contine 2 tipuri de text: static, ce poate fi exprimat in orice tip de format bazat pe text (HTML, WML, XML, etc.), si continutul JSP propriu-zis altfel spus dinamic.

Pentru a se ajunge de la continutul unei pagini JSP la continut generat dinamic se parcurg trei etape:

- **translatia**, in care pagina JSP este transformata de catre containerul de JSP-uri intr-un servlet (de ex. **pagina.jsp** este translatata in **pagina_jsp.java**) – realizata doar atunci cand servlet-ul nu a fost anterior generat sau atunci cand servlet-ul este mai vechi decat pagina JSP
- **compilarea**, in care servletul obtinut la primul pas este compilat de catre containerul de JSP-uri (de ex. Din **pagina_jsp.java** se obtine prin compilare **pagina_jsp.class**) – realizata doar atunci cand containerul a generat un nou servlet
- **executia** (prelucrarea cererii), in care cererile catre pagina JSP sunt directionate catre servlet (care va genera raspunsul).





Comentariile din paginile JSP sunt de doua feluri:

- comentarii **JSP propriu-zise**, care nu ajung in paginile generate dinamic:

```
<%-- Comentariu JSP propriu-zis --%>
```

- comentarii SGML (**HTML**, WML, XML), care ajung in paginile generate dinamic:

```
<!-- Comentariu HTML -->
```

Elementele constitutive ale sintaxei JSP sunt **tiparul** (*template text* – HTML, WML, XML) si elementele JSP: **directivele**, **elementele de scripting** si **actiunile**.

```
<%@ page language="java" contentType="text/html" %>          — JSP element

<html>                                                       — template text
 <body bgcolor="white">

 <jsp:useBean                                                — JSP element
  id="userInfo"
  class="com.ora.jsp.beans.userinfo.UserInfoBean">
  <jsp:setProperty name="userInfo" property="*"/>
 </jsp:useBean>

  The following information was saved:                       — template text
  <ul>
   <li>User Name:

    <jsp:getProperty name="userInfo"                         — JSP element
     property="userName"/>

   <li>Email Address:                                        — template text

    <jsp:getProperty name="userInfo"                         — JSP element
    property="emailAddr"/>

  </ul>                                                      — template text
 </body>
</html>
```

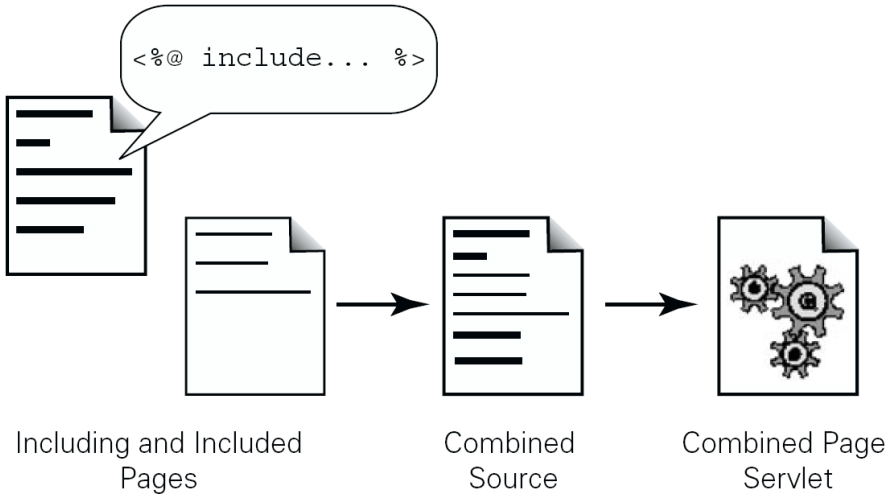
Directivele sunt elemente JSP care furnizeaza informatii globale pentru faza de translatie (se adreseaza containerului). De exemplu, directiva **page** specifica atribute ale paginii generate, cum ar fi bibliotecile importate sau tipul de continut generat.

```
<%-- import clasa biblioteca --%>
<%@ page import="java.util.Date" %>

<%-- tip de continut generat --%>
<%@ page contentType="text/html" %>
```

Directiva **include** specifica **includerea statica a continutului unui fisier in continutul celui curent** in faza de translatie (anterioara compilarii):

```
<%-- includere continut fisier statica (in timpul translatiei sau compilarii) --%>
<%@ include file="altJSP.jsp" %>
```

Static include <%@ include %>	Dynamic include <jsp:include …..>
During the translation or compilation phase all the included JSP pages are compiled into a single Servlet.	The dynamically included JSP is compiled into a separate Servlet. It is a separate resource, which gets to process the request, and the content generated by this resource is included in the JSP response.
No run time performance overhead.	Has run time performance overhead.



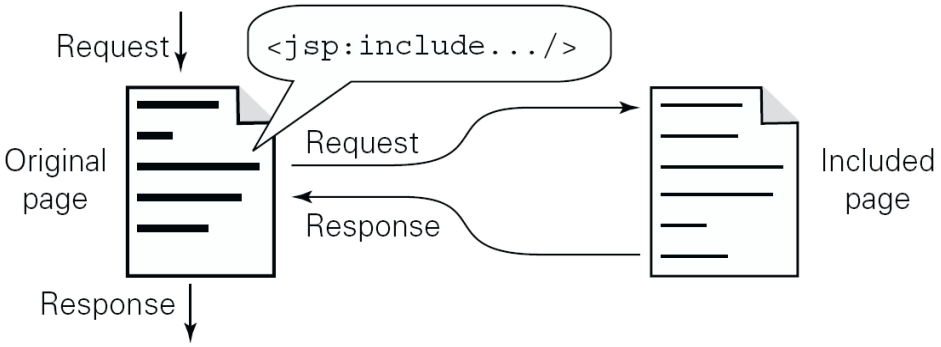
Elementele JSP, inclusiv directivele, pot fi specificate si **folosind sintaxa de tip XML**:

```
<%-- includere continut fisier statica - format XML --%>
<jsp:directive.include file="altJSP.jsp" %>
```

Actiunile sunt elemente JSP care furnizeaza informatii pentru faza de executie (se adreseaza containerului). De exemplu, actiunea **include** specifica includerea continutului unui fisier in continutul celui curent in faza de executie (dupa compilare):

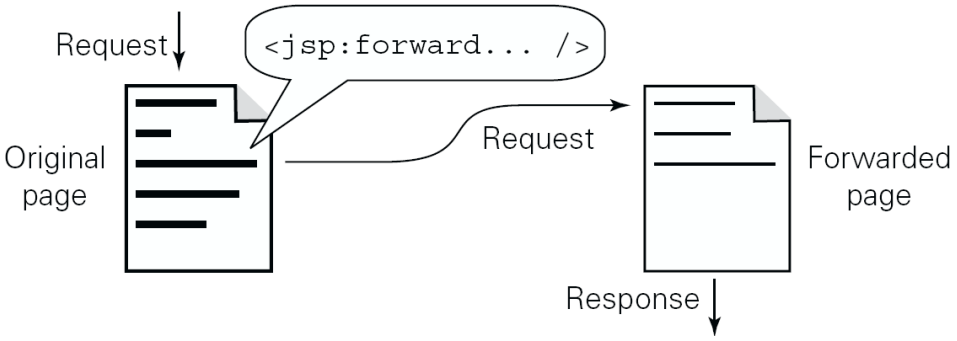
```
<%-- includere continut fisier dinamica (in timpul executiei) --%>
<jsp:include page="altJSP.jsp" %>
```

Includerea este in acest caz dinamica (raspunsul este inclus, nu codul, rezultand doua servlet-uri distincte) in faza executiei.



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

```
<jsp:forward page="localURL" />
```



Actiunea **useBean** conduce la instantierea unui obiect dintr-o clasa JavaBean (cu rol de *helper*, model, etc.) specificata (prin `class`) si cu nume al referintei specificat (prin `id`) care va putea fi utilizat apoi prin intermediul referintei:

```
<%-- instantiere obiect JavaBean --%>
<jsp:useBean id="orar" class="model.Orar" %>
```

Elementele de scripting sunt elemente JSP cu ajutorul carora se include cod Java in pagina (urmand ca acesta sa ajunga nemodificat in codul servletului obtinut prin translatie). Exista 3 categorii de astfel de elemente: **declaratiile, expresiile si scriptlet-urile**.

Declaratiile introduc metode (situatie rar intalnita) si campuri (variabile instanta - care **nu sunt *thread safe***) ale servlet-ului (sintaxa include caracterul **!** care poate fi interpretat ca "atentie"):

```
<%-- declaratie variabila instanta a servlet-ului --%>
<%! private String s; %>

<%-- declaratie metoda a servlet-ului --%>
<%! public String getS() {return s;} %>
```

Expresiile Java ajung sa fie evaluate in codul servlet-ului obtinut prin translatie (spre deosebire de declaratii si scrptlets, **sintaxa nu include caracterul ;**):

```
<%-- expresie --%>
<%= 2*a*b %>
```

Un scriptlet este o secventa de instructiuni Java care ajung sa fie incluse nemodificate in codul servlet-ului obtinut prin translatie (**variabilele declarate in interiorul lor sunt locale**):

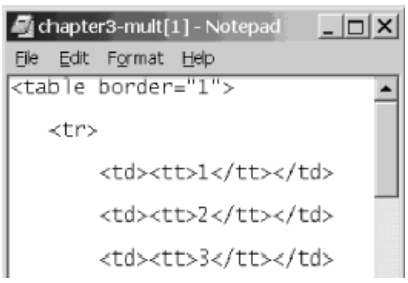
```
<%-- scpriptlet --%>
<%
    String user = null;                      // variabila locala
    username = request.getParameter("user"); // "request" este un obiect implicit
%>
```

Exemplu de scriptlet care utilizeaza decizii si bucle:

```
<html>
   <body>
      <% for (int i = 0; i < 5; i++) { %>
      <p> Your virtual coin has landed on

      <% if (Math.random() < 0.5) { %>
      heads.

      <% } else { %>
      tails.
      <% } %>

      </p>
      <% } %>
   </body>
</html>
```

Exemplu de scriptlet care genereaza dinamic un tablou:

```
<table border="1">
<% for (int row = 1; row < 11; row++) { %>
  <tr>
  <% for (int column = 1; column < 11; column++) { %>
    <td><tt><%= row * column %></tt></td>
  <% } %>
  </tr>
<% } %>
</table>
```



Obiectele implicite au rolul (asemanator celor din limbajul JavaScript) de a oferi acces la diferite resurse (cererea, raspunsul, contextul aplicatiei, sesiunea curenta, configuratia servlet-ului care rezulta, pagina JSP):

Implicit object	Scope	comment
request	Request	Refers to the current request from the client.
response	Page	Refers to the current response to the client.
pageContext	Page	Refers to the page's environment.
session	Session	Refers to the user's session.
application	Application	Same as ServletContext. Refers to the web application's environment.
out	Page	Refers to the outputstream.
config	Page	same as ServletConfig. Refers to the servlet's configuration.
page	Page	Refers to the page's Servlet instance.

exception	Page	exception created on this page. Used for error handling. Only available if it is an errorPage with the following directive:
		`<%@ page isErrorPage="true" %>`
		The "exception" implicit object is not available for global error pages declared through web.xml. You can retrieve the java.lang.Throwable object as follows:
		`<%= request.getAttribute("javax.servlet.error.exception") %>`

Tag-urile JSP standard (implementate ca clase Java) au rolul de a inlocui cat mai mult din codul Java cu elemente asemanatoare celor HTML / XML:

Using scriptlets

```
<html>
   <head>
     <title>simple  example<title>
   </head>
   <body>
        <%
           for(int i=0; i<5; i++) {
        %>
              <%= i %> <br/>

         <% } %>
   </body>
</html>
```

The above JSP code is hard to read and maintain.

Using JSTL tags

```
<%@ taglib prefix="c"
                uri="http//java.sun.com/jstl/core">

<html>
  <head><title>simple example<title></head>
  <body>
    <c:forEach var="i" begin="1" end="5" step="1">
       <c:out value="${i}"> <br/>
    </c:forEach>
   </body>
</html>
```

The above JSP code consists entirely of HTML & JSTL tags (in bold).

Principalele biblioteci de tag-uri standard:

Description	Tag Prefix (recommended)	Example

Core Tag Library – looping, condition evaluation, basic input, output etc.	c	`<c:out value="${hello}" />`
		`<c:if test="${param.name='Peter'}"> …`
		`<c:forEach items="${addresses}" var="address"> …`

Formatting/Internationalization Tag Library – parse data such as number, date, currency etc	fmt	`<fmt:formatNumber value="${now.time}" />`

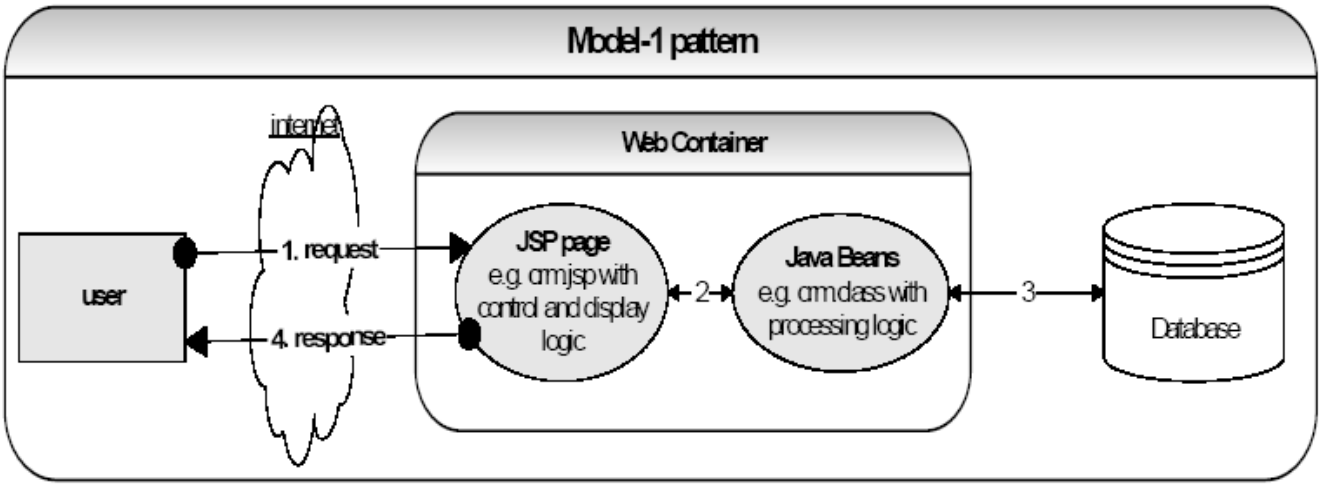
XML Tag Library – tags to access XML elements.	x	`<x:forEach select="$doc/books/book" var="n">`
		`<x:out select="$n/title" />`
		`</x:forEach>`

Database Tag Library – tags to	sql	`<sql:query var="emps" sql="SELECT * FROM Employee">`

Principalul avantaj al JSP-urilor consta in introducerea template-urilor de continut static (posibil de creat in formate non-HTML: WML, XML) care pot fi realizate de dezvoltatori specializati in proiectarea interfetelor Web. Exemplu WML:

```
<%@ page contentType="text/vnd.wap.wml;charset=UTF-8"
      import="java.text.*, java.util.*"
%><?xml version="1.0"?>
<%
  SimpleDateFormat df = new SimpleDateFormat("hh:mm a");
%>

<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN"
"http://www.wapforum.org/DTD/wml_1.1.xml">
<wml>
  <card id="time" title="Time">
    <p>It's <%= df.format(new Date()) %>.</p>
    <p>(Do you know where your laptop is?)</p>
  </card>
</wml>
```

Deoarece insa partea de prelucrare a informatiei necesara generarii de continut dinamic este mai greu de scris in JSP, si este preferabil sa fie separata pentru a fi scrisa de programatori Java, s-a trecut rapid de la **lucrul exclusiv cu pagini JSP** (arhitectura numita "**model-0**" sau "***model-less***") la **delegarea sarcinilor de stocare si prelucrare catre coduri Java** care pot fi clase Java clasice (POJO – *Plain Old Java Objects*) sau componente JavaBeans. Arhitectura care a rezultat poarta numele de "**model-1**".



O clasa JavaBean simpla (reprezentand cod reutilizabil ce contribuie la realizarea sarcinilor aplicatiei):

```
public class HelloBean implements java.io.Serializable {
   private String name = "world";       // proprietate

   public String getName() {            // metoda accesor (getter)
      return name;
   }
   public void setName(String name) {   // metoda accesor (setter)
      this.name = name;
   }
}
```

Utilizarea bean-ului intr-o pagina JSP:

```
<jsp:useBean id="hello" class="HelloBean"/>
<jsp:setProperty name="hello" property="name"/>
Hello, <jsp:getProperty name="hello" property="name"/>!
```

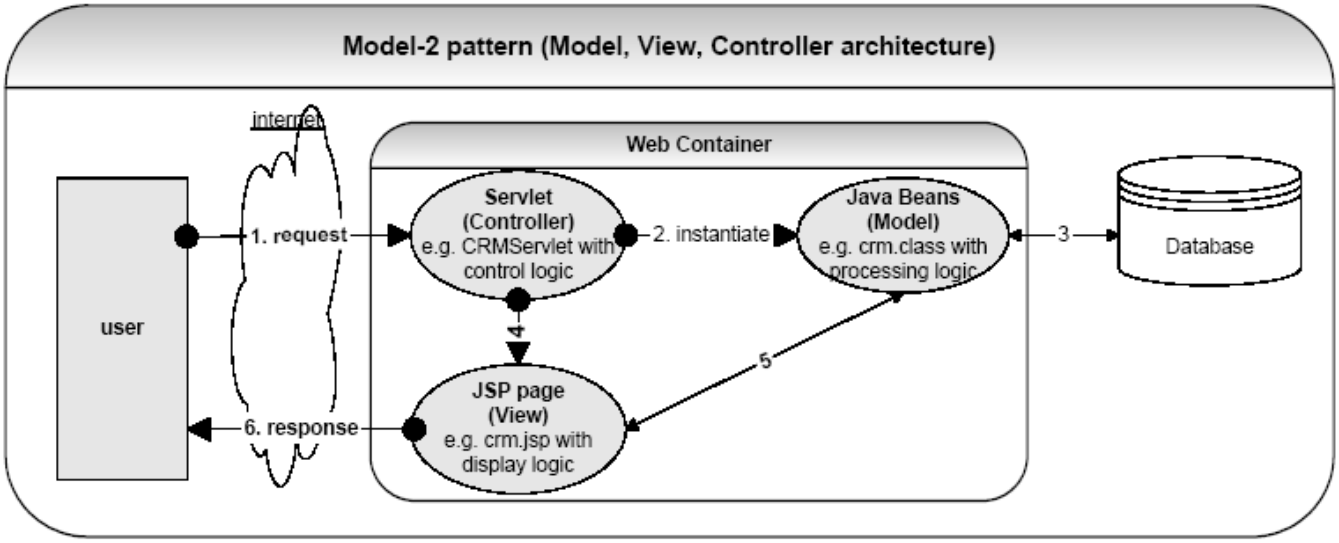
O forma de **delegare a sarcinilor de stocare si prelucrare catre coduri Java** a fost folosita si in cazul aplicatiei Web **cu servlet-uri** (clasa Orar incorporand parte din prelucrarea si stocarea datelor).

Aceasta arhitectura este potrivita pentru aplicatii Web mici. O mai buna gestiune a sistemului, o mai buna separare a responsabilitatilor de dezvoltare si o mai buna specializare a tehnologiilor utilizate sunt necesare pentru aplicatii mari. Acest avans a fost obtinut prin introducerea arhitecturii care poarta numele de "**model-2**" sau **MVC** (***model-view-controler***) in care se folosesc 3 categorii de componente realizate cu tehnologii diferite:

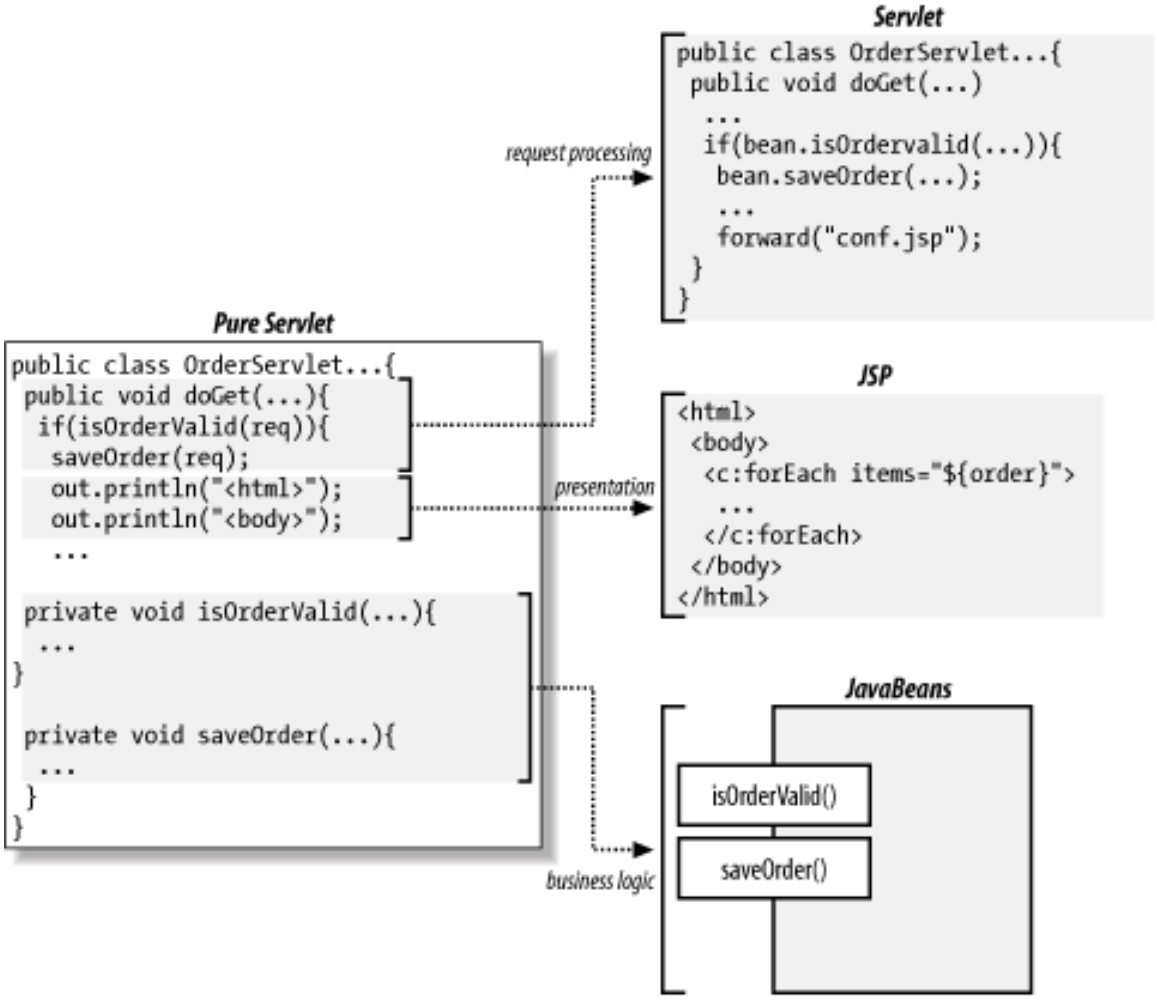
- **controlerul** (realizat in general cu un **servlet**) primeste **cererile**, apeleaza la model pentru a realiza actualizarea datelor de stare si prelucrarile necesare aplicatiei, si in final deleaga prezentarea catre componente specializate

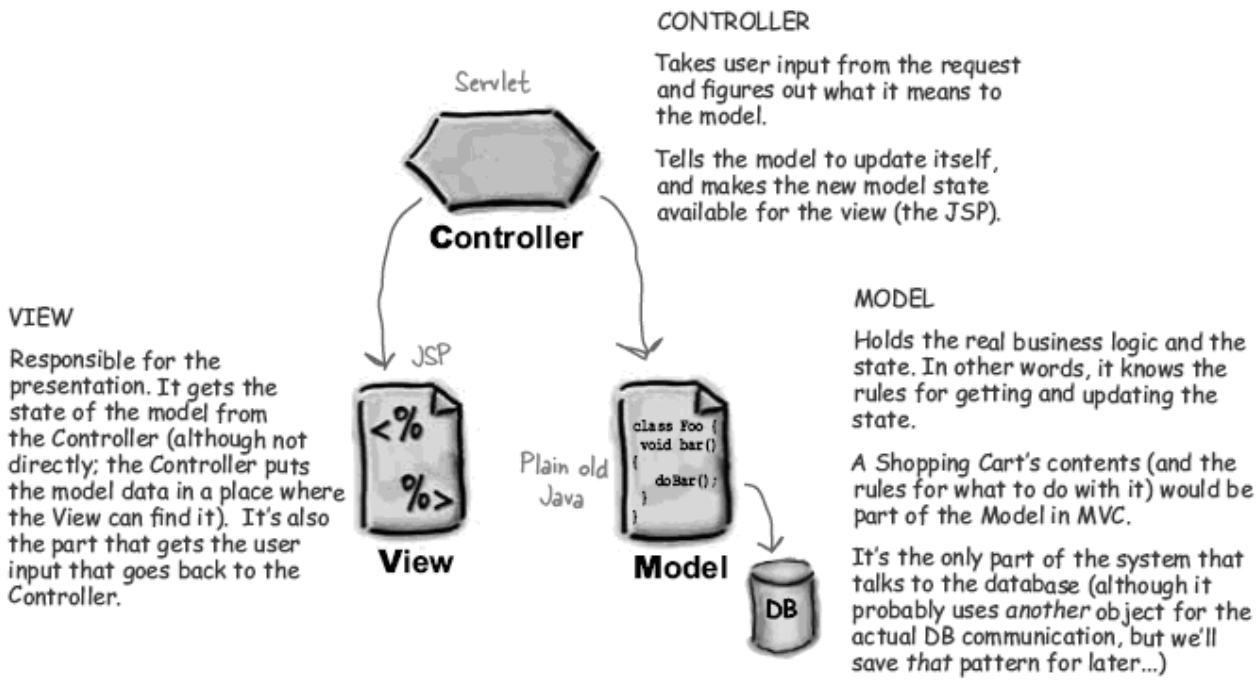
- **modelul** (realizat sub forma de componente **JavaBeans**) se ocupa de pastrarea **datelor** (starii) si de **prelucrarile** necesare,

- **view-ul** este format din componente de **prezentare** (in general pagini **JSP**) care sunt folosite pentru a genera **continutul raspunsului**.

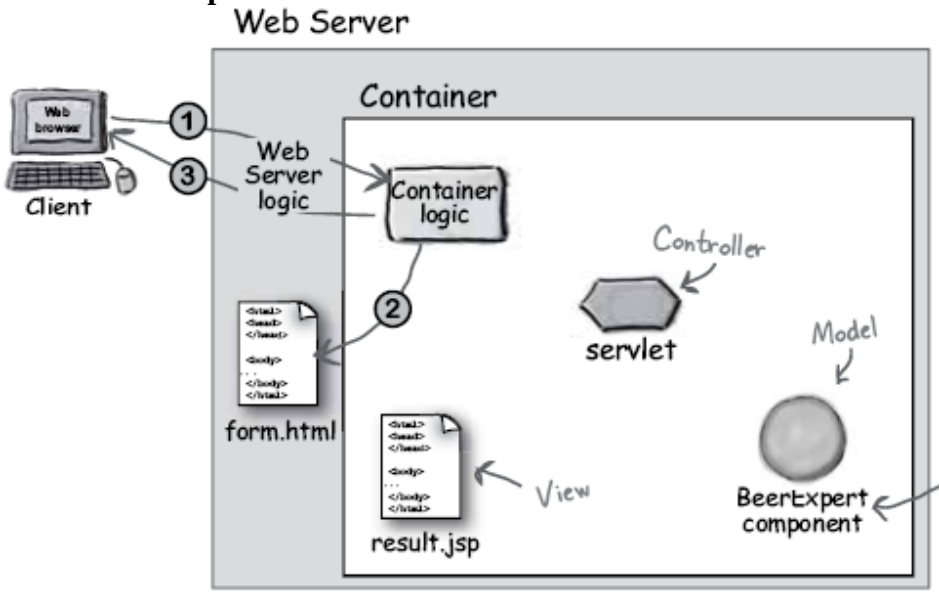


Dezvoltarea unei astfel de aplicatii poate fi realizata modular de specialisti in servlet-uri, specialisti in proiectare Web si scripting JSP (inclusiv utilizare si dezvoltare de biblioteci de tag-uri) si specialisti in proiectare si programare orientata spre obiecte.





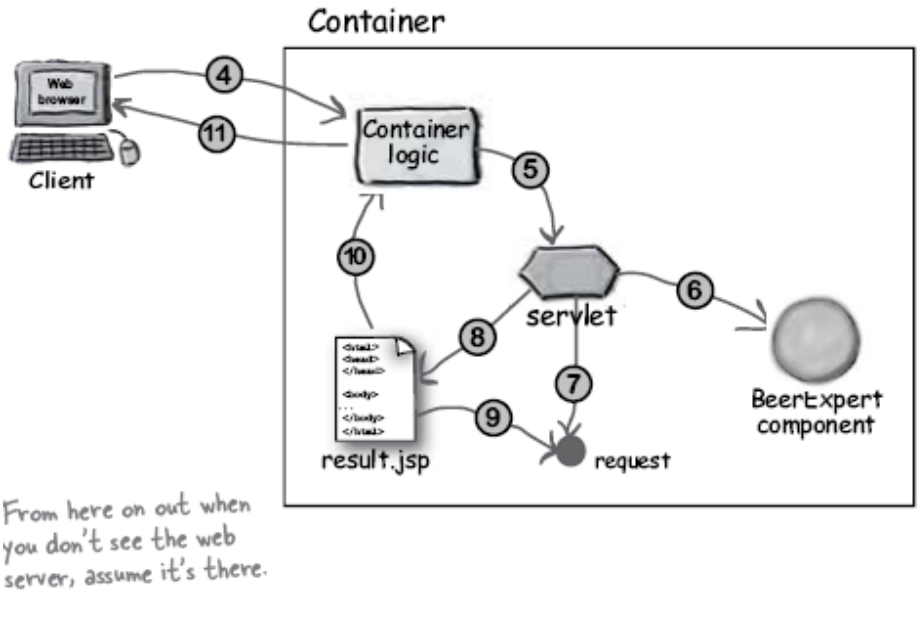
Comportamentul unei aplicatii care utilizeaza arhitectura MVC:



1 - The client makes a request for the *form.html* page.

2 - The Container retrieves the *form.html* page.

3 - The Container returns the page to the browser, where the user answers the questions on the form and...



From here on out when you don't see the web server, assume it's there.

4 - The browser sends the request data to the container.

5 - The Container finds the correct servlet based on the URL, and passes the request to the servlet.

6 - The servlet calls the BeerExpert for help.

7 - The expert class returns an answer, which the servlet adds to the request object.

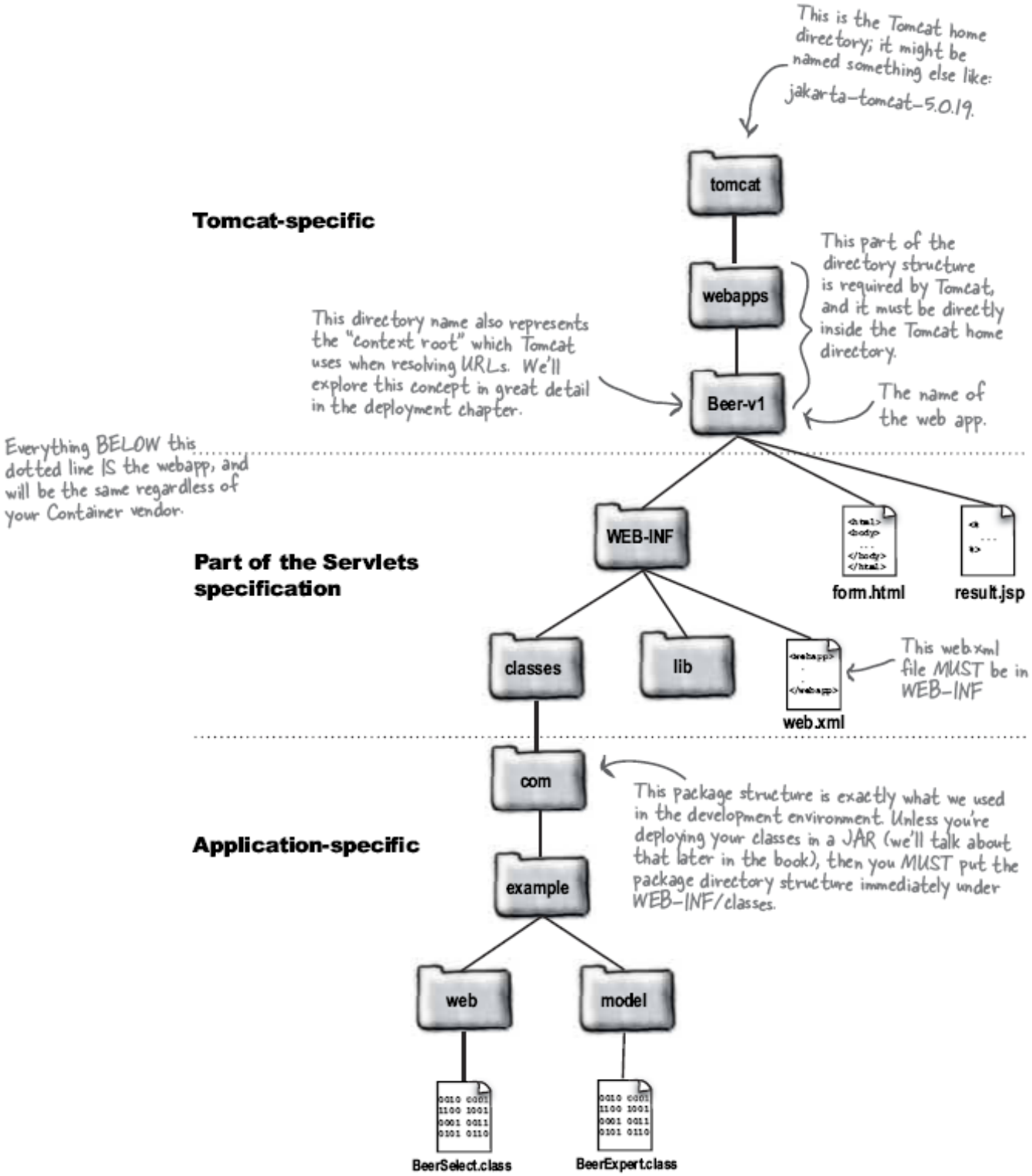
8 - The servlet forwards the request to the JSP.

9 - The JSP gets the answer from the request object.

10 - The JSP generates a page for the Container.

11 - The container returns the page to the happy user.

Structura de directoare si fisiere in cazul utilizarii serverului-container Tomcat:



Clasa **model**:

```
package com.example.model;
import java.util.*;

public class BeerExpert {
  public List getBrands(String color) {
    List brands = new ArrayList();
    if (color.equals("amber")) {
       brands.add("Jack Amber");
       brands.add("Red Moose");
    }
    else {
       brands.add("Jail Pale Ale");
       brands.add("Gout Stout");
    }
    return(brands);
  }
}
```

Clasa servlet **controler**:

```
package com.example.web;

import com.example.model.*;
import javax.servlet.*;
import javax.servlet.http.*;
import java.io.*;
import java.util.*;

public class BeerSelect extends HttpServlet {

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

       // response.setContentType("text/html");

       // remove the old test output
       // PrintWriter out = response.getWriter();
       // out.println("Beer Selection Advice<br>");

       String c = request.getParameter("color");

       // out.println("<br>Got beer color " + c);

       BeerExpert be = new BeerExpert();
       List result = be.getBrands(c);


       request.setAttribute("styles", result);


        RequestDispatcher view =
             request.getRequestDispatcher("result.jsp");

        view.forward(request, response);
    }
}
```

Now that the JSP is going to produce the output, we should remove the test output from the servlet. We commented it out so that you could still see it here.

Add an attribute to the request object for the JSP to use. Notice the JSP is looking for "styles".

Instantiate a request dispatcher for the JSP.

Use the request dispatcher to ask the Container to crank up the JSP, sending it the request and response.

Pagina JSP **view**:

```
<%@ page import="java.util.*" %>

<html>
<body>
<h1 align="center">Beer Recommendations JSP</h1>
<p>

<%
  List styles = (List)request.getAttribute("styles");
  Iterator it = styles.iterator();
  while(it.hasNext()) {
    out.print("<br>try: " + it.next());
  }
%>

</body>
</html>
```

This is a "page directive" (we're thinking it's pretty obvious what this one does).

Some standard HTML (which is known as "template text" in the JSP world).

Here we're getting an attribute from the request object. A little later in the book, we'll explain everything about attributes and how we managed to get the request object...

Some standard Java sitting inside <% %> tags (this is known as scriptlet code).

Efectul executiei**:**

Cazul **clasei Orar (de aceasta data componenta JavaBeans) cu rol de componenta model**:

```
1  package model;
2
3  import java.beans.*;
4  import java.io.Serializable;
5
6  public class Orar implements Serializable {
7
8      private String[] orar; // camp ascuns (starea obiectului)
9
10     public Orar() {
11         orar = new String[7]; // alocarea dinamica a spatiului pentru tablou
12         // popularea tabloului cu valori
13         orar[0] = "Luni este curs TPI la seriile D si E " +
14                                 "si laborator TPI la seria E.";
15         orar[1] = "Marti nu sunt ore de TPI.";
16         orar[2] = "Miercuri este laborator TPI la seriile D si E.";
17         orar[3] = "Joi este laborator TPI la seria D.";
18         orar[4] = "Vineri este laborator TPI la seria D.";
19         orar[5] = "Sambata nu sunt ore de TPI.";
20         orar[6] = "Duminica nu sunt ore de TPI.";
21     }
22     public String getOrar(int zi) { // metoda accesor - getter
23         return orar[zi];            // returneaza referinta la tablou
24     }
25     public void setOrar(int zi, String text) { // metoda accesor - setter
26         orar[zi] = text;                       // inlocuieste un element
27     }
28 }
```

Se poate observa ca **Orar face parte dintr-un pachet de clase intitulat model si implementeaza interfata Serializable** (asa incat starea obiectelor Orar poate fi persistata sau poate fi transferata in retea folosind RMI).

Un JSP care sa permita accesul la obiecte Orar si sa realizeze sarcinile aplicatiei (pastrand pe cat posibil formatul de la lucrarea anterioara) **ar putea fi organizat astfel**:

- un **template** (formularul pentru acces) preluat din continutul paginii HTML reprezinta partea statica,
- **obiectul clasei Orar** este instantiat cu ajutorul unei actiuni JSP si accesibil la nivel de sesiune (efectul fiind similar utilizarii HTTPSession),
- un **scriptlet** decide si genereaza continut dinamic ca efect al accesului prin formular,

```jsp
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <h1>Pagina JSP acces orar (Model 1)</h1>
        <hr><form name="input" action="PaginaJSP.jsp" method="get">
            <input type="radio" name="zi" checked="checked" value="0"> Luni
            <br> <input type="radio" name="zi" value="1"> Marti
            <br> <input type="radio" name="zi" value="2"> Miercuri
            <br> <input type="radio" name="zi" value="3"> Joi
            <br> <input type="radio" name="zi" value="4"> Vineri
            <br> <input type="radio" name="zi" value="5"> Sambata
            <br> <input type="radio" name="zi" value="6"> Duminica
            <hr>
            <input type="radio" name="serviciu" checked="checked" value="getOrar">
            Obtinere orar
            <br><input type="radio" name="serviciu" value="setOrar"> Modificare orar
            <input type="text" name="modificare" value="">
            <input type="submit" value="Trimite">
        </form>
        <hr>
        <jsp:useBean scope="session" id="orar" class="model.Orar" />
        <%
        try {
            int zi = Integer.parseInt(request.getParameter("zi"));
            // Daca serviciul cerut e obtinere orar
            if (request.getParameter("serviciu").equals("getOrar")) {
                out.println("<b>Orarul cerut:</b> <br>" + orar.getOrar(zi));
            }
            // Daca serviciul 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));
            }
        } catch (NumberFormatException ex) {}
        %>

    </body>
</html>
```

Pagina JSP se auto-refera, asa incat **in cazul primei accesari** (deoarece nu a fost in prealabil selectata nici o zi) **este necesara tratarea exceptia** de tip **`NumberFormatException`**.

Pagina HTML care contine formularul pentru accesul la servletul controler, reutilizabil si in paginile JSP (prin includere):

```html
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Acces orar</title>
    </head>
    <body>
      <h1>Formular HTML acces orar</h1>
         <hr><form name="input" action="ServletControler" method="get">
         <input type="radio" name="zi" checked="checked" value="0"> Luni
         <br> <input type="radio" name="zi" value="1"> Marti
         <br> <input type="radio" name="zi" value="2"> Miercuri
         <br> <input type="radio" name="zi" value="3"> Joi
         <br> <input type="radio" name="zi" value="4"> Vineri
         <br> <input type="radio" name="zi" value="5"> Sambata
         <br> <input type="radio" name="zi" value="6"> Duminica
         <hr><input type="radio" name="serviciu" checked="checked" value="getOrar">
               Obtinere orar
         <br> <input type="radio" name="serviciu" value="setOrar"> Modificare orar
         <input type="text" name="modificare" value="">
         <input type="submit" value="Trimite">
      </form><hr>
    </body>
</html>
```

Un servlet cu rol de controler va accesa informatiile din Orar si **le va pasa prin intermediul unui atribut** nou al cererii **catre o pagina JSP selectata in functie de valoarea unui parametru** din formular.

```
1  package controler;
2
3  import model.Orar;
4
5  import java.io.*;
6  import java.net.*;
7  import javax.servlet.*;
8  import javax.servlet.http.*;
9
10 public class ServletControler extends HttpServlet {
11
12    protected void processRequest(HttpServletRequest request,
13             HttpServletResponse response) throws ServletException, IOException {
14       // Transformarea obiectului orar in atribut al sesiunii curente pentru
15       // salvarea starii lui
16       HttpSession ses = request.getSession();
17       Orar orar = (Orar) ses.getAttribute("orar");
18       if (orar == null) { // Daca nu exista orarul salvat ca atribut al sesiunii
19          orar = new Orar();
20          ses.setAttribute("orar", orar);
21       }
22
23       // Obtinerea parametrilor introdusi de utilizator in formular
24       int zi = Integer.parseInt(request.getParameter("zi"));
25
26       RequestDispatcher view;
27
28       // Daca serviciul cerut e obtinere orar
29       if (request.getParameter("serviciu").equals("getOrar")) {
30          view = request.getRequestDispatcher("RezultatObtinereOrar.jsp");
31       }
32       // Daca serviciul cerut e modificare orar
33       else if (request.getParameter("serviciu").equals("setOrar")) {
34          String modificare = request.getParameter("modificare");
35          orar.setOrar(zi, modificare);
36          view = request.getRequestDispatcher("RezultatModificareOrar.jsp");
37       }
38       // Daca serviciul cerut nu e recunoscut
39       else {
40          view = request.getRequestDispatcher("ServiciuNeimplementat.jsp");
41       }
42
43       request.setAttribute("raspuns", orar.getOrar(zi));
44
45       view.forward(request, response);
46    }
47    protected void doGet(HttpServletRequest request, HttpServletResponse response)
48            throws ServletException, IOException { processRequest(request, response);
49    }
50    protected void doPost(HttpServletRequest request, HttpServletResponse response)
51            throws ServletException, IOException { processRequest(request, response);
52    }
53 }
```

Paginile JSP vor include formularul pentru accesul la servlet, cate una pentru fiecare dintre cele 3 cazuri:

```
<%@page contentType="text/html"%>
<%@page pageEncoding="UTF-8"%>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>JSP Page</title>
    </head>
    <body>
        <%@ include file="PaginaHTMLAcces.html" %>

        <b>Orarul cerut:</b> <br> <%=request.getAttribute("raspuns")%>
    </body>
</html>
```

In codurile de mai sus se regasesc **avantajele arhitecturii MVC** si sunt exemplificate mai multe elemente sintactice si mecanisme oferite de paginile JSP (delegare servlet catre JSP, includere de continut static, etc.).

Exemplu – codul controlerului – servlet

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

   String name = request.getParameter("userName");
   request.setAttribute("name", name);

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

Use the request parameter from the form to set a request-scoped attribute that the JSP will use.

Forward the request to the view.

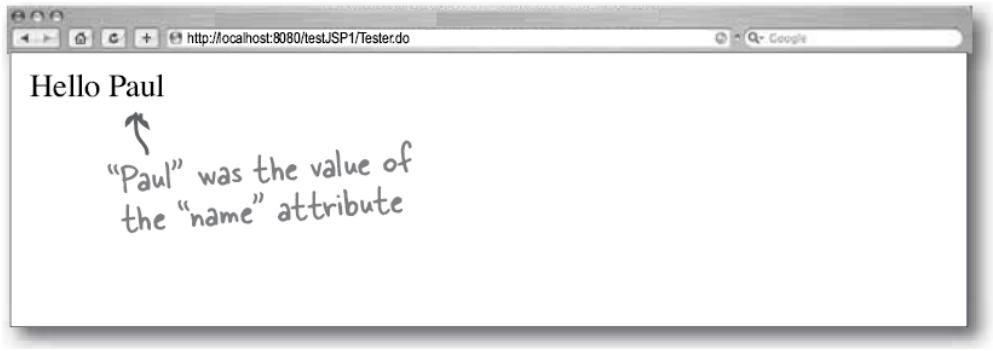
Exemplu – codul view-ului – JSP

```
<html><body>
Hello
<%= request.getAttribute("name") %>
</body></html>
```

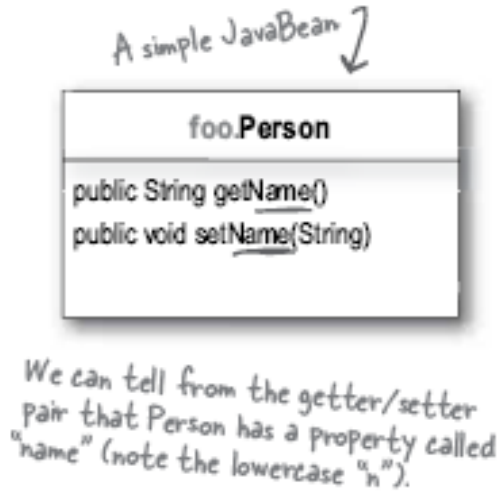
Use a scripting expression to get the attribute and print it to the response.

(Remember: scripting expressions are ALWAYS the argument to the out.write() method.)

Efectul:



Daca atributul este (in loc de String) un obiect Person (JavaBean avand proprietatea name)



Codul servletului

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

   foo.Person p = new foo.Person();
   p.setName("Evan");
   request.setAttribute("person", p);

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

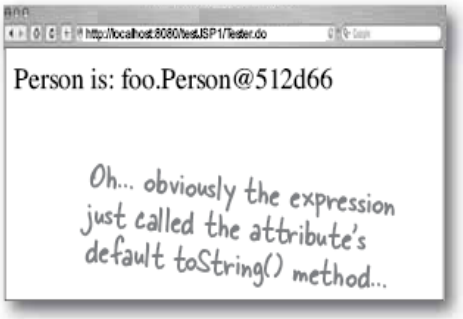
Codul JSP-ului

What does getAttribute() return?

```
<html><body>
Person is: <%= request.getAttribute("person") %>
</body></html>
```

What we WANT:

What we GOT:



Rescrierea codului JSP-ului pentru a obtine numele persoanei – <u>varianta cu scriptlet</u>

```
<html><body>

<% foo.Person p = (foo.Person) request.getAttribute("person"); %>
Person is: <%= p.getName() %>

</body></html>
```

Print the result of getName().

Rescrierea codului JSP-ului pentru a obtine numele persoanei – <u>varianta cu expresie</u>

```
<html><body>
Person is:
<%= ((foo.Person) request.getAttribute("person")).getName() %>

</body></html>
```

Person este un JavaBean, asa incat - putem utiliza <u>actiuni standard legate de bean-uri</u>

NO Java code here! No scripting, just two standard action tags.

```
<html><body>
<jsp:useBean id="person"  class="foo.Person" scope="request" />
Person created by servlet: <jsp:getProperty name="person"  property="name" />
</body></html>
```

<u>Declararea si initializarea unui atribut al unui bean cu <jsp:useBean></u>

```
<jsp:useBean id="person"  class="foo.Person" scope="request" />
```

Identifies the standard action.

Declares the identifier for the bean object. This corresponds to the name used when the servlet code said:

request.setAttribute("person", p);

Declares the class type (fully-qualified, of course) for the bean object.

Identifies the attribute scope for this bean object.

Obtinerea proprietatii unui atribut al unui bean cu <jsp:getProperty>

```
<jsp:getProperty name="person"  property="name" />
```

Identifies the standard action.

Identifies the actual bean object. This will match the "id" value from the <jsp:useBean> tag.

Identifies the property name (in other words, the thing with the getter and setter in the bean class).

Note: this "name" property has nothing to do with the name="person" part of this tag. The property is called "name" simply because of the way the Person class is defined.

Daca <jsp:useBean> nu gaseste un atribut al unui bean numit "person" il creaza:

- **tagul** `<jsp:useBean id="person"  class="foo.Person" scope="request" />`

- **se transforma in codul**

```
foo.Person person = null;
```
Declare a variable based on the value of id. This variable is what lets other parts of your JSP (including other bean tags) refer to that variable.

```
synchronized (request) {
```
Tries to get the attribute at the scope you defined in the tag, and assigns the result to the id variable.

```
  person = (foo.Person)_jspx_page_context.getAttribute("person", PageContext.REQUEST_SCOPE);

  if (person == null){
```
BUT, if there was NOT an attribute with that name at that scope...

```
     person = new foo.Person();
```
Make one, and assign it to the id variable.

```
     _jspx_page_context.setAttribute("person", person, PageContext.REQUEST_SCOPE);
  }
}
```
Finally, set the new object as an attribute at the scope you defined.

Stabilirea valorii proprietatii unui atribut al unui bean cu <jsp:setProperty>

```
<jsp:useBean id="person"  class="foo.Person" scope="request" />
<jsp:setProperty name="person" property="name" value="Fred" />
```

Daca insa se doreste ca <jsp:setProperty> sa stabileasca valoarea proprietatii doar pentru un bean NOU:

```
<jsp:useBean id="person" class="foo.Person" scope="page" >
```
There's no slash!

This is the body.

```
  <jsp:setProperty name="person" property="name" value="Fred" />

</jsp:useBean >
```
Any code inside the body of <jsp:useBean > is CONDITIONAL. It runs ONLY if the bean isn't found and a new one is created.

Finally we close off the tag. Everything between the opening and closing tags is the body.

- ceea ce <u>se transforma in codul</u>

```
foo.Person person = null;
person = (foo.Person) _jspx_page_context.getAttribute("person", PageContext.PAGE_SCOPE);
if (person == null){
   person = new foo.Person();
   _jspx_page_context.setAttribute("person", person, PageContext.PAGE_SCOPE);

   org.apache.jasper.runtime.JspRuntimeLibrary.introspecthelper(
        _jspx_page_context.findAttribute("person"), "name", "Fred", null, null, false);

}
```

Declare the reference variable.

Look for an existing attribute with the name and scope from the tag.

If there isn't one, make a new instance.

Bind the new bean object to the specified scope.

THIS is the part that's new. It's here ONLY when useBean has a body.

You were expecting:

person.setName("Fred");

but that's what this code does. Except it uses a generic property-setting method that takes the attribute, the property, and the value as arguments. The end result is still the same: ultimately it invokes setName() on the Person object.

(Remember you aren't expected to know the Tomcat implementation code...only the end result.)

Presupunand ca <u>un formular aceseaza direct un JSP</u> TestBean.jsp:

```
<html><body>

<form action="TestBean.jsp">
  name: <input type="text" name="userName">
  ID#: <input type="text" name="userID">
  <input type="submit">
 </form>

</body></html>
```

The request goes STRAIGHT to the JSP.

- <u>putem stabili valoarea unei proprietati a unui bean</u> pe baza unui parametru al cererii printr-o combinatie de actiune standard si scripting:

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee"/>
<% person.setName(request.getParameter("userName")); %>
```

- <u>sau</u> doar pe baza unor actiuni standard:

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee">

<jsp:setProperty name="person" property="name"
                 value="<%= request.getParameter("userName") %>" />
</jsp:useBean>
```

Yes, you ARE seeing an expression INSIDE the <jsp:setProperty> tag (which happens to be inside the body of a <jsp:useBean> tag)

And yes, it DOES look bad.

- sau putem folosi atributul param al actiunii standard <jsp:setProperty>:

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee">
    <jsp:setProperty name="person" property="name" param="userName" />
</jsp:useBean>
```

```
<html><body>

<form action="TestBean.jsp">
  name: <input type="text" name="userName">
  ID#: <input type="text" name="userID">
  <input type="submit">
 </form>

</body></html>
```

The param value "userName" comes from the name attribute of the form's input field.

- daca insa schimbam si numele parametrului din formular din "userName" in "name", astfel incat acesta sa fie acelasi cu numele proprietatii bean-ului:

```
<html><body>

<form action="TestBean.jsp">
  name: <input type="text" name="name">
  ID#: <input type="text" name="userID">
  <input type="submit">
 </form>

</body></html>
```

obtinem acelasi efect utilizand (fara sa mai fie nevoie sa specificam "param"):

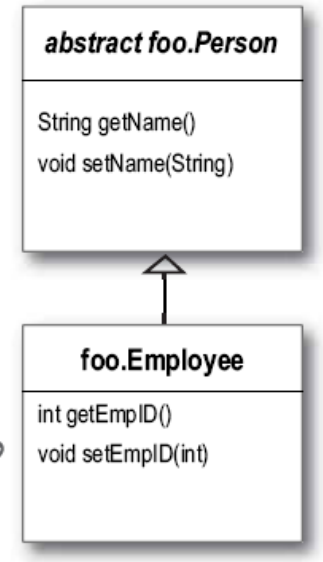
```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee">
    <jsp:setProperty name="person" property="name" />
</jsp:useBean>
```

We didn't specify ANY value!

Presupunand urmatorul formular HTML si bean-urile din figura:

Now BOTH parameters match the property names of the bean.

```
<html><body>

<form action="TestBean.jsp">
  name: <input type="text" name="name">
  ID#: <input type="text" name="empID">
  <input type="submit">
 </form>

</body></html>
```



putem folosi un wildcard (*) pentru a popula ambele proprietati cu valori:

```
<jsp:useBean id="person" type="foo.Person" class="foo.Employee">
    <jsp:setProperty name="person" property="*" />
</jsp:useBean>
```

How cool is that??

Initializarea unui JSP – <u>configurarea unui parametru de initializare a servlet-ului</u>

```
<web-app ...>
  ...
   <servlet>
     <servlet-name>MyTestInit</servlet-name>
     <jsp-file>/TestInit.jsp</jsp-file>
     <init-param>
       <param-name>email</param-name>
       <param-value>ikickedbutt@wickedlysmart.com</param-value>
     </init-param>
   </servlet>
  ...
</web-app>
```

This is the only line that's different from a regular servlet. It basically says, "apply everything in this <servlet> tag to the servlet created from this JSP page..."

Initializarea unui JSP – <u>suprascrierea metodei jspInint()</u>

Override the jspInit() method using a declaration.

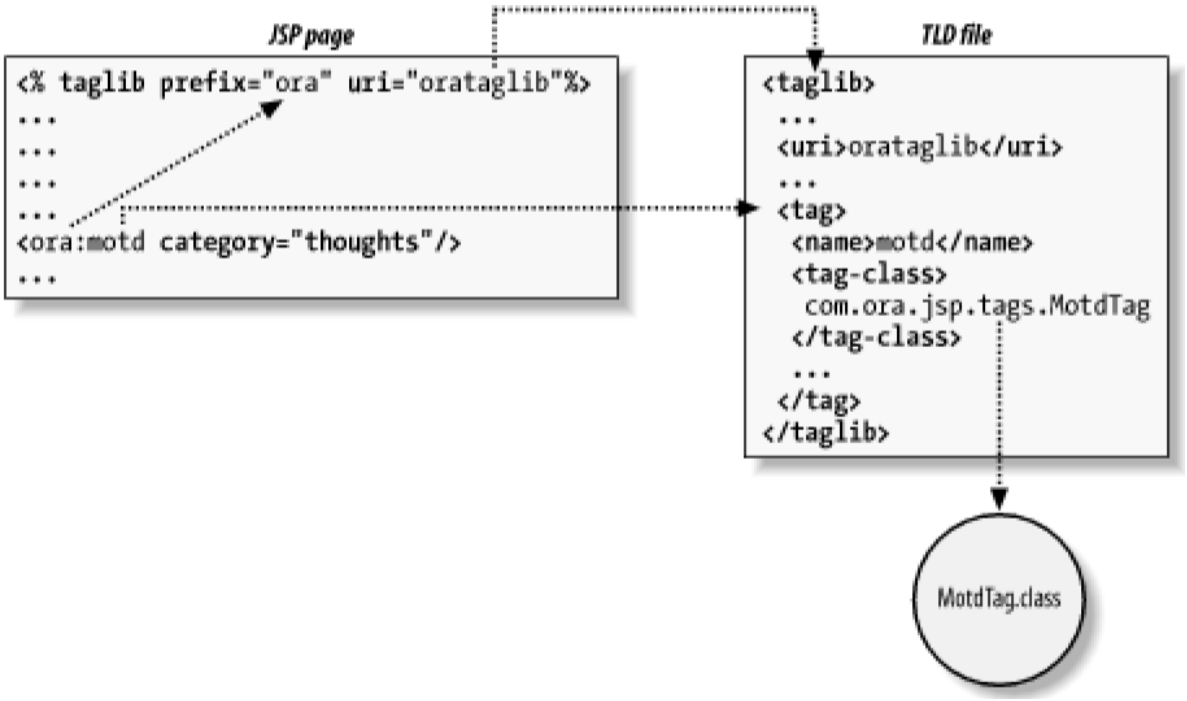
You're in a servlet, so you can call your inherited getServletConfig() method!

```
<%!
  public void jspInit() {

    ServletConfig sConfig = getServletConfig();

    String emailAddr = sConfig.getInitParameter("email");

    ServletContext ctx = getServletContext();

    ctx.setAttribute("mail", emailAddr);
  }
%>
```

This is EXACTLY what you'd do in a normal servlet.

Get a reference to the ServletContext and set an application-scope attribute.

Relatia dintre **directiva taglib**, descriptorul **TLD**, si **implementarea** (**tag handler**) **actiunilor create de programator (custom, non-standard)**



`

4.6. Accesul la baze de date prin tehnologii Java

JDBC (Java DataBase Connectivity) este o interfata standard SQL de acces la baze de date. Acesta ne furnizeaza un acces uniform la baze de date relationale. JDBC este constituit dintr-un set de clase si interfete scrise în Java, furnizând un API standard pentru proiectantii de aplicatii baze de date. Acest lucru face posibila scrierea aplicatiilor de baze de date folosind un API Java pur.

Folosind JDBC este usor sa transmitem secvente SQL catre baze de date relationale. Cu alte cuvinte, nu este necesar sa scriem un program pentru a accesa o baza de date Oracle, alt program pentru a accesa o baza de date Sybase si asa mai departe. Este de ajuns sa scriem un singur program folosind API-ul JDBC si acesta va fi capabil sa trimita secvente SQL bazei de date dorite. Bineînteles, scriind codul sursa în Java, ne este asigurata portabilitatea programului. Deci, iata doua motive puternice care fac combinatia Java - JDBC demna de luat în seama.

JDBC face trei lucruri:

- stabileste o conexiune cu o baza de date;
- trimite secvente SQL;
- prelucreaza rezultatele.

5.2.1 Structura unei aplicatii JDBC - Instalarea unui driver

Primul lucru pe care trebuie sa faca o aplicatie care lucreaza cu baze de date este **instalarea unui driver specific bazei de date**:

```
public static String dbdriver = "org.apache.derby.jdbc.ClientDriver";
…
Class.forName(dbdriver)
```

Prin apelul acestor metode se va incarca un driver JDBC. In clasa specifica driverului esista o metoda statica care va inregistra existenta sa cu *DriverManager*. Dupa incarcarea driverului trebuie creata legatura cu baza de date.

Se ridica o serie de probleme:

- unde se gaseste pe Internet calculatorul caruia apartine baza de date.?
- pe ce numar de port asculta RDBMS-ul cererile?

Aceasta problema este rezolvata prin introducerea *URL*-urilor (Uniform resource Locator). Structura unui url este: *protocol//nume_host:port//cale.* Un URL pentru o baza de date ar avea forma: *jdbc:<protocol_secundar>:<nume_secundar>//nume_host:port//nume_baza_de_date.*

```
public static String url = "jdbc:derby://localhost:1527/orar";
```

5.2.2 Structura unei aplicatii JDBC - Crearea conexiunii

La crearea conexiunii pe langa numele bazei de date se vor transmite si numele utilizatorului precum si parola acestuia. Daca sunt incarcate mai multe drivere pentru lucrul cu diferite baze de date, atunci se pune problema cum se alege driverul pentru conexiunea curenta.

Clasa *DriverManager* este responsabila pentru acesta intreband fiecare driver daca poate realiza legatura cu baza de date specificata prin url. De exemplu un driver Oracle ar observa imediat ca in exemplul prezentat se lucreaza cu un alt tip de baza de date si ar refuza cererea. La realizarea conexiunii se creaza un obiect de tip Connection.

De fapt Connection este o interfata, care asigura transmiterea datelor spre baza de date si obtinerea datelor din baza de date. Interfata furnizeza metode pentru a obtine informatii despre baza de date, inchide conexiunea cu baza de date sau / si asculta mesajele sosite de la baza de date.

```
Connection con = DriverManager.getConnection(url, username, password);
```

5.2.3 Structura unei aplicatii JDBC - Accesul la baza de date

Interfata *Connection* contine si metoda *createStatement()* care ne va furniza un obiect Statement printr-un apel de urmatoarea forma:

```
PreparedStatement prepStmt = con.prepareStatement(selectStatement);
```

sau

```
Statement stmt = con.createStatement();
```

Metodele cele mai importante ale interfetei *Statement* sunt urmatoarele: *executeQuery(String)*, *executeUpdate(String)*, *execute(String)*. Aceste metode se utilizeaza pentru executia codului SQL.

Metoda ***executeQuery()*** *executa comanda SQL si* returneaza un obiect de tip *ResultSet* si se utilizeaza pentru executia comenziilor SQL de interogare SELECT.

Metoda ***executeUpdate()*** executa comanda SQL primita ca parametru si returneaza numarul randurilor tabelei modificate (update count). Se utilizeaza pentru comenzile SQL de manipularea datelor: INSERT, UPDATE, DELETE si pentru comenzi de definire a datelor CREATE/DROP TABLE. In cazul comenzilor de definirea datelor valoarea returnata este 0.

Ultima metoda **execute()** poate fi privita ca fiind generalizarea celorlaltor doua metode. Se utilizeaza daca comanda SQL poate returna deodata mai multe rezultate sau nu se cunoaste rezultatul executiei.

```
String selectStatement =
        "select * from \"APP\".\"orars\" where \"APP\".\"orars\".\"id\" = ?";

PreparedStatement prepStmt = con.prepareStatement(selectStatement);
prepStmt.setInt(1, id.intValue());
ResultSet rs = prepStmt.executeQuery();
```

5.2.4 Structura unei aplicatii JDBC - Prelucrarea rezultatelor

ResultSet este tot o interfata. Obiectul ResultSet contine rezultatul interogarii bazei de date, insa pointerul atasat acestei tabele puncteaza inaintea primului rand din tabela:

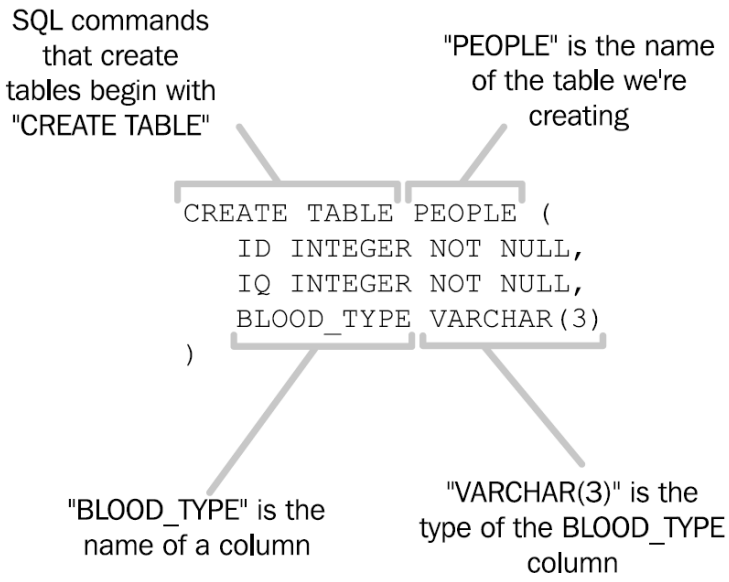
```
ResultSet rs = prepStmt.executeQuery();

while (rs.next()) {
      orar =  new OrarEntity();
      orar.setId(new Integer(rs.getString(1)));
      orar.setDescription(rs.getString(2));
}
```

Daca rezultatul interogarii este vid atunci constructia de mai sus va genera exceptie *SQLException*. *ResultSet* contine metode pentru accesul datelor din rezultat. Astfel metodele *getXXX(int)* si *getXXX(String)* returneaza valoarea dintr-o coloana specificata prin parametru si linia curenta. In locul XXX se pune un tip predfinit Java (tip primitiv-ex. int sau clasa de baza-ex. String).

Tabela PEOPLE in baza de date:

NAME	IQ	BLOOD_TYPE
1	106	O
2	82	A-
3	164	B+
4	143	
5	128	AB+

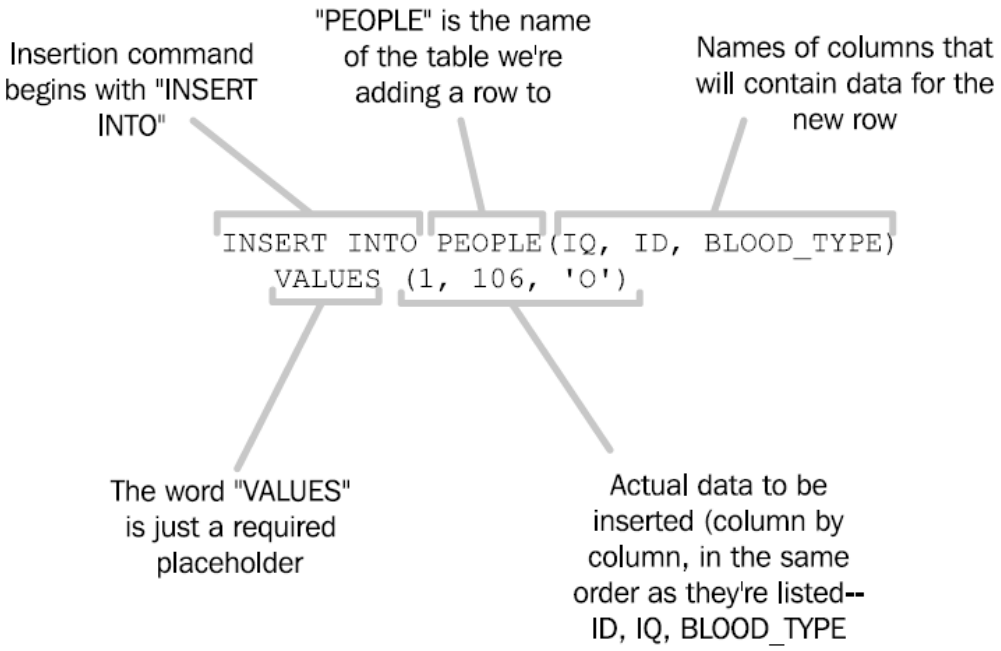
Crearea tabelei cu comanda SQL CREATE:



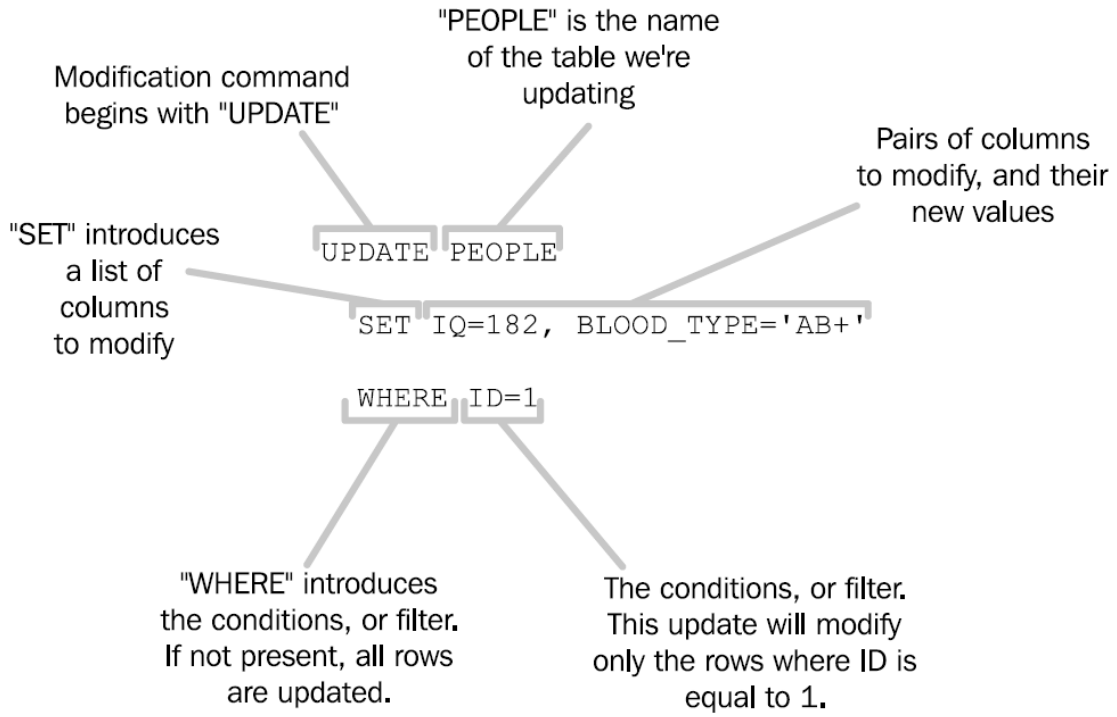
Tipuri de date SQL:

SQL data type	Description	Sample value
INTEGER	Integer	6510
REAL	Floating-point number	6.02
DATE	Date only (no time)	January 20, 1986
TIME	Time only (no date)	2:05 a.m.
TIMESTAMP	Date and time	January 20, 1985 2:05 a.m.
VARCHAR(x)	String of up to x characters	"Where did I leave my hat?"

Inserarea datelor (unei linii - inregistrari) in tabela cu comanda SQL INSERT:



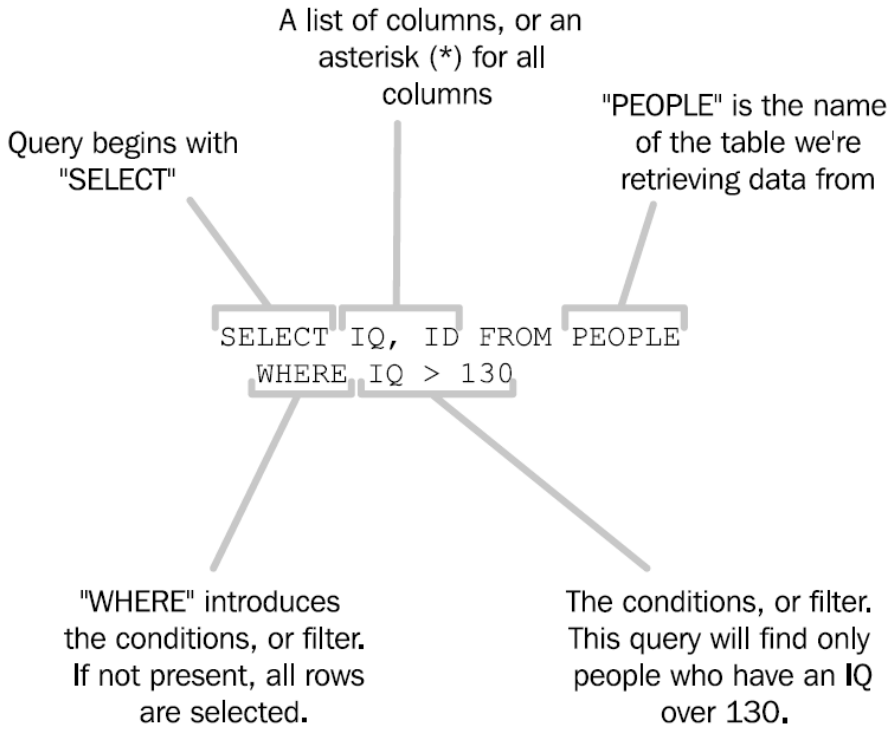
Modificarea datelor din tabela cu comanda SQL UPDATE:



Stergerea conditionata a datelor din tabela cu comanda SQL DELETE si WHERE:

```
DELETE FROM PEOPLE
WHERE AGE < 18
```

Obtinerea selectiva a datelor din tabela cu comanda SQL SELECT:



Pentru exemplificare am folosit din nou **clasa Orar** (aflata in pachetul **model**) cu modificari minore (pastrand astfel nemodificate componentele **view**) si am rescris codul servletului ServletControler (aflat in pachetul **controler**) doar pentru a adauga persistenta datelor obiectelor Orar in baza de date.

```
1  package model;
2
3  import java.beans.*;
4  import java.io.Serializable;
5
6  public class Orar implements Serializable {
7      private String[] orar; // camp ascuns (starea obiectului)
8
9      public Orar() {
10         orar = new String[7];
11         orar[0] = "Luni este curs TPI la seriile D si E si laborator TPI la seria E.";
12         orar[1] = "Marti nu sunt ore de TPI.";
13         orar[2] = "Miercuri este laborator TPI la seriile D si E.";
14         orar[3] = "Joi este laborator TPI la seria D.";
15         orar[4] = "Vineri este laborator TPI la seria D.";
16         orar[5] = "Sambata nu sunt ore de TPI.";
17         orar[6] = "Duminica nu sunt ore de TPI.";
18     }
19     public String getOrar(int zi) { // metoda accesor - getter
20         return orar[zi];            // returneaza referinta la tablou
21     }
22     public void setOrar(int zi, String text) { // metoda accesor - setter
23         orar[zi] = text;                       // inlocuieste un element
24     }
25     // Returnare mai eficienta a tuturor datelor orarului
26     public Iterator<String> getItems(){
27         return Arrays.asList(orar).iterator();
28     }
29 }
```

Clasa **model.dao.jdbc.ApplicationJDBCService** are rolul de a salva (persista) sau obtine informatiile dorite din baza de date. Datele persistate / extrase din baza de date vor fi incapsulate in cadrul claselor din pachetul model.entity, fiind disponibile in cadrul afisarii sau procesarii informatiilor din cadrul jsp-urilor sau sevlet-urilor (**controler.ServlerController**).

Clasa OrarEntry (aflata in pachetul **model.entity**) este o reprezentare Java a structurii tabelei ORARENTRY. Astfel, fiecarei zile de (luni, marti… duminica) ii este asociat cate un obiect al clasei OrarEntry, continand o descriere (prin atributul **String description**) a orarului din ziua reprezentata in cadrul atributului **Integer day.**

```
1  package model.entity;
2
3  import java.io.Serializable;
4
5  public class OrarEntry implements Serializable {
6      private Integer id;
7      private OrarEntity orar;
8      private Integer day;
9      private String description;
10
11     public OrarEntry(OrarEntity orar, Integer day, String description){
12         this.orar = orar;
13         this.day = day;
14         this.description = description;
15     }
16     public OrarEntry(){
17     }
18         public Integer getId() {
19         return id;
20     }
21     public void setId(Integer id) {
22         this.id = id;
23     }
24     public String getDescription() {
25         return description;
26     }
27     public void setDescription(String description) {
28         this.description = description;
29     }
30     public OrarEntity getOrar() {
31         return orar;
32     }
33     public void setOrar(OrarEntity orar) {
34         this.orar = orar;
35     }
```

```
36      public Integer getDay() {
37          return day;
38      }
39      public void setDay(Integer day) {
40          this.day = day;
41      }
42  }
```

Aceasta clasa **contine** deasemenea **o referinta catre OrarEntity** (prin atributul **OrarEntity orar**) care **indeplineste astfel in clasa Java rolul campului orar_id** (foreign key (FK)) din tabela ORARENTRY.

Astfel se realizeaza o **asociere unica dintre detalierea orarului pentru o anumita zi si orarul ca entitate parinte**. Atributul **Integer id** va contine cheia primara (primary key (PK)), unica, atibuita coloanei **id** din cadrul tabelei ORARENTRY.

Clasa OrarEntity (aflata in pachetul **model.entity**) este o **reprezentare Java a structurii tabelei ORARENTITY**. Pentru fiecare entitate OrarEntity avem **o colectie de obiecte OrarEntry care stocheaza informatii despre detalierea orarului pentru fiecare zi din saptamana**.

Atributul List<OrarEntry> orarEntry reprezinta o **colectie** de **model.entity.OrarEntry**, cu referinte catre descrierea orarului pentru fiecare zi.

Atributul Integer id va contine **cheia primara** (primary key (PK)) asociata orarului in tabela ORARENTITY. Se realizeaza astfel o **dependenta unica dintre fiecare obiect OrarEntry si un obiect de tipul OrarEntity**.

```
1   package model.entity;
2
3   import java.io.Serializable;
4   import java.util.List;
5
6   public class OrarEntity implements Serializable {
7
8       private Integer id;
9
10      private String description;
11
12      // Lista de intrari in orar
13      private List<OrarEntry> orarEntry;
14
15      public Integer getId() {
16          return id;
17      }
18      public void setId(Integer id) {
19          this.id = id;
20      }
21
22      public String getDescription() {
23          return description;
24      }
25      public void setDescription(String description) {
26          this.description = description;
27      }
28
29      // Obtinerea listei de intrari in orar
30      public List<OrarEntry> getOrarEntry() {
31          return orarEntry;
32      }
33      // Stabilirea listei de intrari in orar
34      public void setOrar(List<OrarEntry> orarEntry) {
35          this.orarEntry = orarEntry;
36      }
37  }
```

Clasa **OrarUtil** (aflata in pachetul **model**) reprezinta o **clasa utilitara** a carei rol este de **conversie in ambele sensuri** dintre **structura model.Orar** (entitate folosita in cadrul formarii paginilor jsp) si **obiectele care descriu structura tabelelor din baza de date**.

Se evidenteaza **decuplarea** dintre **obiectele participante in cadrul construirii paginii html** care va fi afisata clientului si **obiectele care fac parte din structura de baza a modului de control**, respectiv, **de persistenta**:

```
1  package model;
2
3  import model.entity.OrarEntity;
4  import model.entity.OrarEntry;
5
6  public class OrarUtil {
7
8      public static Orar convertFromOrarEntity(OrarEntity orar){
9
10        Orar orarBean = new Orar();
11
12        // Iterare prin intrarile in orar si populare
13        for (OrarEntry entry : orar.getOrarEntry()){
14             orarBean.setOrar(entry.getDay().intValue(), entry.getDescription());
15        }
16        return orarBean;
17     }
18
19     public static OrarEntity convertFromOrar(Orar orar){
20
21        OrarEntity orarEntity = new OrarEntity();
22        orarEntity.setDescription("434D");
23
24        List<OrarEntry> orarEntries = new ArrayList<OrarEntry>(7);
25
26        Iterator it = orar.getItems();
27
28        for(int i=0; it.hasNext();i++){
29            String orarDescription = (String)it.next();
30            orarEntries.add(new OrarEntry(orarEntity, i, orarDescription));
31        }
32        orarEntity.setOrar(orarEntries);
33        return orarEntity;
34     }
35 }
```

Interfata ApplicationDAOService (aflata in pachetul **model.dao**) reprezinta interfata dintre **structura de decizie si control** al aplicatiei web (controler.ServletControler) (**Controler**) si **modulul care se ocupa de persistenta** obiectelor entitate (**Model**).

Prin utlizarea acestei interfete **se « ascunde » astfel implementarea propriu-zisa a modului de persistenta cu baza de date**, asigurand **transparenta fata de diferitele medii de persistanta** pentru care s-ar putea aplica. Este **simplificata posibilitatea modificarii modulului de persistanta** fara a fi necesara modificarea zonelor de program care apeleaza metode din acest modul.

Metoda **public List<OrarEntity> getOrars()** permite **obtinerea tuturor orariilor inscrise in baza de date**. Aceasta metoda **poate returna o multitudine de valori (in cazul in care in baza de date se afla multe orarii inscrise)** ceea ce ar afecta performanat aplicatiei.

Metoda **public OrarEntity getOrar(Integer id)** returneaza un obiect de tipul OrarEntity pe baza valorii cheii primare unice (id).

Metoda **public OrarEntity saveUpdateOrar(OrarEntity orar)** folosita **pentru a persista un nou orar** sau **pentru a modifica valorile unui orar deja persistat in modulul de persistenta**.

```
1  package model.dao;
2
3  import model.entity.OrarEntity;
4
5  public interface ApplicationDAOService {
6
7      public OrarEntity saveUpdateOrar(OrarEntity orar)throws OrarNotFoundException;
8
9      public OrarEntity getOrar(Integer id) throws OrarNotFoundException;
10
11     public Iterator<OrarEntity> getIteratedOrars() throws OrarNotFoundException;
12
13     public List<OrarEntity> getOrars() throws OrarNotFoundException;
14
15     public void remove();
16 }
```

Clasa **ApplicationJDBCService** (aflata in pachetul **model.dao.jdbc**) **implementeaza metodele descrise de catre model.dao.ApplicationDAOService.**

Ea **pune la dispozitie modulului ServletControler (Controller) metode de access / persistenta** la baza de date a **obiectelor de tip entity (OrarEntity, OrarEntry).**

Totodata clasa realizeaza o **conversie, in cadrul metodelor de interogare / salvare**, conversie dintre obiectele amintite de tip entity cu structura tabelelor asociata.

```
 1  package model.dao.jdbc;
 2
 3  import java.sql.*;
 4  import javax.sql.*;
 5  import javax.naming.*;
 6  import java.util.*;
 7
 8  import model.dao.ApplicationDAOService;
 9  import model.dao.OrarNotFoundException;
10  import model.entity.OrarEntity;
11  import model.entity.OrarEntry;
12
13  public class ApplicationJDBCService implements ApplicationDAOService {
14      private Connection con;
15
16      // Database configuration
17      public static String url = "jdbc:derby://localhost:1527/orar";
18      public static String dbdriver = "org.apache.derby.jdbc.ClientDriver";
19      public static String username = "APP";
20      public static String password = "APP";
21
22      public ApplicationJDBCService() throws Exception {
23          try {
24              Class.forName(dbdriver);
25              con = DriverManager.getConnection(url, username, password);
26          } catch (Exception ex) {
27              System.out.println("Exception in ApplicationJDBCService: " + ex);
28              throw new Exception("Couldn't open connection to database: " +
29                      ex.getMessage());
30          }
31      }
32      public void remove() {
33          try {
34              con.close();
35          } catch (SQLException ex) {
36              System.out.println(ex.getMessage());
37          }
38      }
39
40      public OrarEntity getOrar(Integer id) throws OrarNotFoundException{
41          OrarEntity orar = null;
42          PreparedStatement prepStmt = null;
43
44          try {
45              String selectStatement =
46                  "select * from \"APP\".\"orars\" where \"APP\".\"orars\".\"id\" = ?";
47              prepStmt = con.prepareStatement(selectStatement);
48              prepStmt.setInt(1, id.intValue());
49
50              ResultSet rs = prepStmt.executeQuery();
51
52              while (rs.next()) {
53                  orar =  new OrarEntity();
54                  orar.setId(new Integer(rs.getString(1)));
55                  orar.setDescription(rs.getString(2));
56              }
57
58              if (orar!=null) {
59                  List<OrarEntry> entrys = getOrarEntrys(orar);
60                  orar.setOrar(entrys);
61              }else{
62                  throw new OrarNotFoundException("Orar not found");
63              }
64          } catch (SQLException ex) {
65              ex.printStackTrace();
66          }finally{
```

```
67              try {
68                  prepStmt.close();
69              } catch (SQLException ex) {
70              }
71          }
72          return orar;
73      }
74
75      public List<OrarEntity> getOrars() throws OrarNotFoundException{
76
77          OrarEntity orar = null;
78          PreparedStatement prepStmt = null;
79          List<OrarEntity> orars = new ArrayList<OrarEntity>(2);
80
81          try {
82              String selectStatement = "select * from \"APP\".\"orars\"";
83              prepStmt = con.prepareStatement(selectStatement);
84
85              ResultSet rs = prepStmt.executeQuery();
86
87              while (rs.next()) {
88                  orar =  new OrarEntity();
89                  orar.setId(new Integer(rs.getString(1)));
90                  orar.setDescription(rs.getString(2));
91
92                  if (orar!=null) {
93                      List<OrarEntry> entrys = getOrarEntrys(orar);
94                      orar.setOrar(entrys);
95                  }
96                  orars.add(orar);
97             }
98          } catch (SQLException ex) {
99              ex.printStackTrace();
00          }finally{
 1              try {
 2                  prepStmt.close();
 3              } catch (SQLException ex) {
 4              }
 5          }
 6          return orars;
 7      }
 8
 9      public Iterator<OrarEntity> getIteratedOrars() throws OrarNotFoundException{
10          //TODO implementeaza aceasta metoda
11          return null;
12      }
13
14       private List<OrarEntry> getOrarEntrys(OrarEntity orar){
15
16           List<OrarEntry> orarEntryList = new ArrayList<OrarEntry>(7);
17           PreparedStatement prepStmt = null;
18
19          try {
20              String selectStatement = "select * " +
21             "from \"APP\".\"orarentry\" where \"APP\".\"orarentry\".\"id_orar\" = ?";
22              prepStmt = con.prepareStatement(selectStatement);
23              prepStmt.setInt(1, orar.getId());
24
25              ResultSet rs = prepStmt.executeQuery();
26
27              while (rs.next()) {
28                  OrarEntry orarEntry =  new OrarEntry();
29                  orarEntry.setId(rs.getInt(1));
30                  orarEntry.setDay(rs.getInt("day"));
31                  orarEntry.setOrar(orar);
32                  orarEntry.setDescription(rs.getString("description"));
33
34                  orarEntryList.add(orarEntry);
35              }
36
37          } catch (SQLException ex) {
38              ex.printStackTrace();
39          }catch (Exception e){
40
41          }finally{
42              try {
```

```
43                  prepStmt.close();
44              } catch (SQLException ex) {
45              }
46          }
47          return orarEntryList;
48       }
49
50       public OrarEntity saveUpdateOrar(OrarEntity orar) throws OrarNotFoundException{
51           if (orar.getId()!=null){
52               updateOrarEntity(orar);
53           }else{
54               saveOrarEntity(orar);
55           }
56           for (OrarEntry entry: orar.getOrarEntry()){
57               if (entry.getId()!=null){
58                    updateOrarEntry(entry);
59               }else{
60                    saveOrarEntry(entry);
61                }
62           }
63           return null;
64       }
65
66       private OrarEntity updateOrarEntity(OrarEntity orar) throws OrarNotFoundException{
67           PreparedStatement prepStmt = null;
68           try {
69               String updateStatement =
70                   "update \"APP\".\"orars\" set \"APP\".\"orars\".\"description\" = ? "
71                   + " where \"APP\".\"orars\".\"id\" = ?";
72               prepStmt = con.prepareStatement(updateStatement);
73               prepStmt.setString(1, orar.getDescription());
74               prepStmt.setInt(2, orar.getId());
75
76               prepStmt.executeUpdate();
77
78           } catch (SQLException e){
79              e.printStackTrace();
80           }finally{
81               try {
82                   prepStmt.close();
83               } catch (SQLException ex) {
84               }
85           }
86           return null;
87       }
88
89      private void updateOrarEntry(OrarEntry entry){
90          PreparedStatement prepStmt = null;
91          try {
92               String updateStatement =
93            "UPDATE \"APP\".\"orarentry\" SET \"APP\".\"orarentry\".\"id_orar\" = ? ," +
94                       "\"APP\".\"orarentry\".\"description\" = ?, "+
95                       "\"APP\".\"orarentry\".\"day\" = ? "+
96                       "WHERE \"APP\".\"orarentry\".\"id\" = ?";
97
98               prepStmt = con.prepareStatement(updateStatement);
99               prepStmt.setInt(1, entry.getOrar().getId());
00               prepStmt.setString(2, entry.getDescription());
 1               prepStmt.setInt(3, entry.getDay());
 2               prepStmt.setInt(4,entry.getId());
 3
 4               prepStmt.executeUpdate();
 5
 6           } catch (SQLException e){
 7              e.printStackTrace();
 8           }finally{
 9               try {
10                   prepStmt.close();
11               } catch (SQLException ex) {
12               }
13           }
14      }
15
16      private OrarEntity saveOrarEntity(OrarEntity orar){
17          //TODO .... implementeaza
18          return null;
```

```
19      }
20
21      private void saveOrarEntry(OrarEntry entry){
22           PreparedStatement prepStmt = null;
23          try {
24               String updateStatement =
25             "INSERT INTO \"APP\".\"orarentry\" (\"APP\".\"orarentry\".\"id_orar\", " +
26                      "\"APP\".\"orarentry\".\"description\", "+
27                      "\"APP\".\"orarentry\".\"day\") VALUES (?, ?, ? )";
28
29               prepStmt = con.prepareStatement(updateStatement);
30               prepStmt.setInt(1, entry.getOrar().getId());
31               prepStmt.setString(2, entry.getDescription());
32               prepStmt.setInt(3, entry.getDay());
33
34               prepStmt.executeUpdate();
35          } catch (SQLException e){
36              e.printStackTrace();
37          }finally{
38              try {
39                  prepStmt.close();
40              } catch (SQLException ex) {
41              }
42          }
43      }
44  }
```

Clasa ServletControler (aflata in pachetul **model.dao.jdbc**) **pastreaza functionalitatile prezentate in laboratorul anterior**.

In actuala abordare, aspectele noi aduse se refera la **interfatarea utilizatorului cu modulul de acces la baza de date** (Model). In acest context, **fiecarei sesiuni este asociat un obiect Orar care va fi folosit pentru popularea jsp-urilor** cu rol de afisare a detaliilor de orar si de modificare a acestuia.

In cazul in care nu exista nici o entitate de tipul orar persistata in modulul de persistenta (DAO – Data AccessObject), ServletControler va **decide persistarea (salvarea) entitatii obtinuta prin metodele prezentate in laboratorul anterior**.

Se observa introducerea, prin folosirea interfetei **`ApplicationDAOService`**, **a** unui nivel abstract DAO ce **« ascunde » folosirea explicita a metodelor de access la baza de date** de catre clasa **`ApplicationJDBCService`**.

```
1   package controler;
2
3   import java.io.*;
4   import java.net.*;
5   import javax.servlet.*;
6   import javax.servlet.http.*;
7
8   import model.Orar;
9   import model.OrarUtil;
10  import model.dao.ApplicationDAOService;
11  import model.dao.OrarNotFoundException;
12  import model.entity.OrarEntity;
13  import model.entity.OrarEntry;
14
15  public class ServletControler extends HttpServlet {
16     protected void processRequest(HttpServletRequest request,
17             HttpServletResponse response) throws ServletException, IOException {
18
19        ApplicationDAOService serviceDAO =
20            (ApplicationDAOService) getServletContext().getAttribute("orarDBAO");
21
22        // Transformarea obiectului orar in atribut al sesiunii curente pentru
23        // salvarea starii lui
24        HttpSession ses = request.getSession();
25        Orar orar = (Orar) ses.getAttribute("orar");
26
27        OrarEntity orarEntity = null;
28        try {
29           List<OrarEntity> orars = serviceDAO.getOrars();
30           //INFO: consideram primul orar gasit in baza de date
31           orarEntity = serviceDAO.getOrar(new Integer(2));
32        } catch (OrarNotFoundException ex) {
33           System.err.println(ex.getMessage());
34        }
```

```
35
36          if (orarEntity!=null){
37             orar = OrarUtil.convertFromOrarEntity(orarEntity) ;
38          }else{
39             orar = new Orar();
40
41             try {
42                 //INFO: salveaza (persista) orar in baza de date
43                 serviceDAO.saveUpdateOrar(OrarUtil.convertFromOrar(orar));
44             } catch (OrarNotFoundException ex) {
45                 ex.printStackTrace();
46             }
47          }
48          ses.setAttribute("orar", orar);
49
50          // Obtinerea parametrilor introdusi de utilizator in formular
51          int zi = Integer.parseInt(request.getParameter("zi"));
52
53          RequestDispatcher view;
54          // Daca serviciul cerut e obtinere orar
55          if (request.getParameter("serviciu").equals("getOrar")) {
56             view = request.getRequestDispatcher("RezultatObtinereOrar.jsp");
57          }
58
59          // Daca serviciul cerut e modificare orar
60          else if (request.getParameter("serviciu").equals("setOrar")) {
61             String modificare = request.getParameter("modificare");
62             orar.setOrar(zi, modificare);
63
64             try {
65                //salvare orar in baza de date
66                if (orarEntity!=null){
67
68                   boolean found = false;
69                   for (OrarEntry entry: orarEntity.getOrarEntry()){
70                      if (entry.getDay()==zi){
71                         entry.setDescription(modificare);
72                         found = true;
73                      }
74                   }
75                   if (!found){
76                      orarEntity.getOrarEntry().add(
77                                new OrarEntry(orarEntity, zi, modificare));
78                   }
79
80                   serviceDAO.saveUpdateOrar(orarEntity);
81                }else{  //TODO scrie un mesaj de eroare  }
82             } catch (OrarNotFoundException ex) {
83                ex.printStackTrace();
84             }
85             view = request.getRequestDispatcher("RezultatModificareOrar.jsp");
86          } else {
87             view = request.getRequestDispatcher("ServiciuNeimplementat.jsp");
88          }
89          request.setAttribute("raspuns", orar.getOrar(zi));
90          view.forward(request, response);
91       }
92       protected void doGet(HttpServletRequest request, HttpServletResponse response)
93                throws ServletException, IOException {
94          processRequest(request, response);
95       }
96       protected void doPost(HttpServletRequest request, HttpServletResponse response)
97                throws ServletException, IOException {
98          processRequest(request, response);
99       }
00    }
```

Clasa **ContextListener** (aflata in pachetul **listeners**) se apeleaza **la initializarea / distrugerea aplicatiei de catre server-ul de aplicatii.**

In **metoda contextInitialized(), in contextul aplicatiei se va salva (cu cheia orarDBAO) o instanta a clasei ApplicationJDBCService.** Aceasta instanta **contine o conexiune activa cu baza de date**. Toate cererile de acces la baza de date ce vor avea loc pe durata vietii aplicatiei vor fi efectuate prin intermediul conexiunii deja existente.

```
1  package listeners;
2
3  import javax.servlet.*;
4  import model.dao.ApplicationDAOService;
5  import model.dao.jdbc.ApplicationJDBCService;
6
7  public final class ContextListener implements ServletContextListener {
8      private ServletContext context = null;
9
10     public void contextInitialized(ServletContextEvent event) {
11         context = event.getServletContext();
12
13         try {
14             ApplicationDAOService applicationDAO = new ApplicationJDBCService();
15             context.setAttribute("orarDBAO", applicationDAO);
16         } catch (Exception ex) {
17             System.out.println("Couldn't create bookstore database bean: " +
18                 ex.getMessage());
19         }
20     }
21     public void contextDestroyed(ServletContextEvent event) {
22         context = event.getServletContext();
23
24         ApplicationDAOService applicationDAO =
25               (ApplicationDAOService) context.getAttribute("orarDBAO");
26
27         if (applicationDAO != null) {
28             applicationDAO.remove();
29         }
30
31         context.removeAttribute("orarDBAO");
32     }
33 }
```