Java Servlet

Objectives

- Explain the nature of a servlet and its operation
- Use the appropriate servlet methods in a web application
- Code the extraction of environment entries within a servlet
- Handle HTML forms within a servlet
- Explain the significance of web application state
- Explain the purpose and operation of HTTP cookies and their role in state management
- Develop java web application with MVC model

Free Servlet and JSP Engines (Servlet/JSP Containers)

- Apache Tomcat
  - http://jakarta.apache.org/tomcat/
- IDE: NetBeans, Eclipse
  - https://netbeans.org/
  - https://eclipse.org/
- Some Tutorials:
  - Creating Servlet in Netbeans: http://www.studytonight.com/servlet/creating-servlet-in-netbeans.php
  - Java Servlet Example: http://w3processing.com/index.php?subMenuId=170

Compiling and Invoking Servlets

- Put your servlet classes in proper location
  - Locations vary from server to server. E.g.,
    - *tomcat_install_dir*/webapps/ROOT/**WEB-INF/classes**
- Invoke your servlets (HTTP request)
  - http://localhost/servlet/ServletName
  - Custom URL-to-servlet mapping (via web.xml)

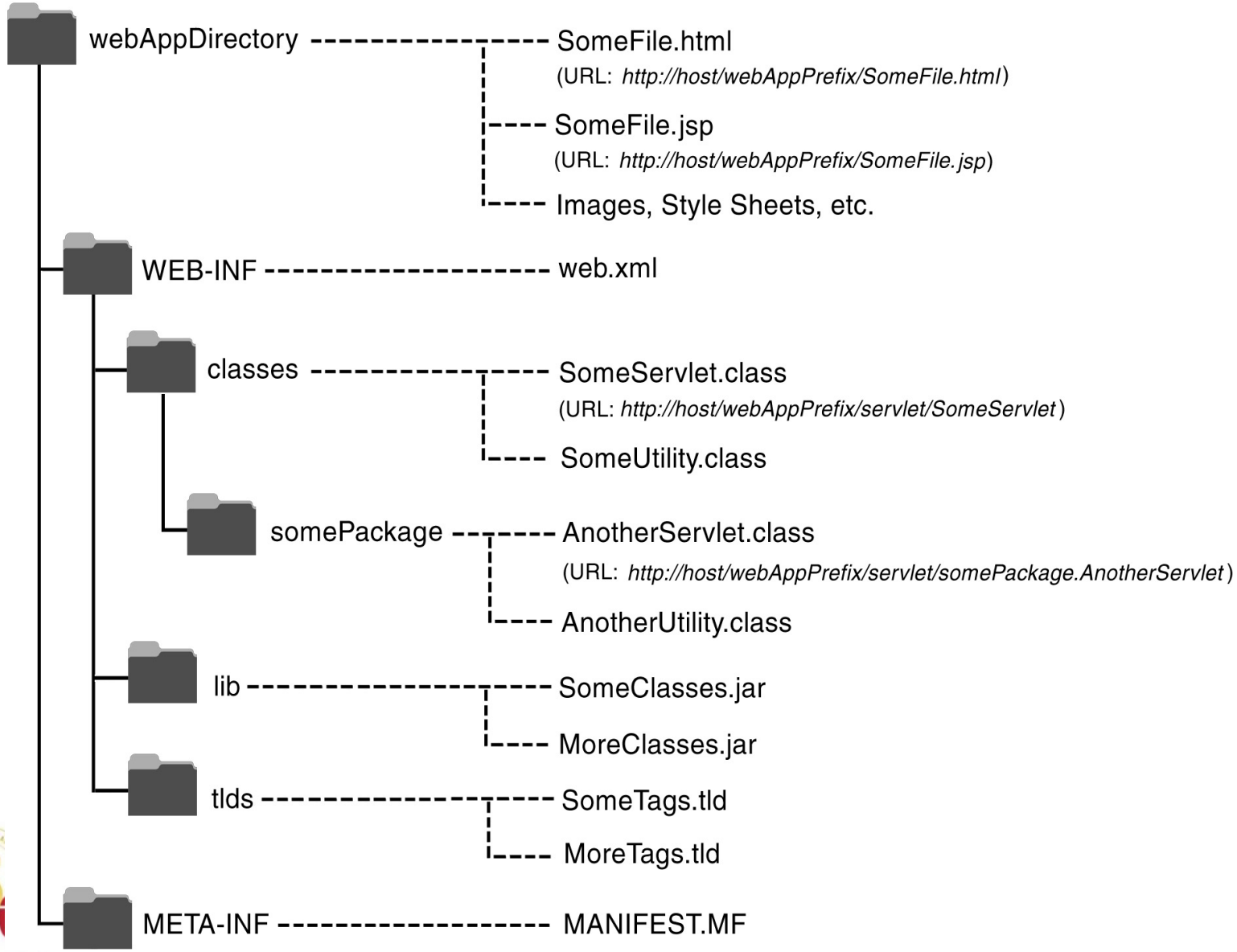
Purposes of Web Applications (A single WAR file)

- Organization
  - Related files grouped together in a single directory hierarchy.
    - HTML files, JSP pages, servlets, beans, images, etc.
- Portability
  - Most servers support Web apps.
  - Can redeploy on new server by moving a single file.
- Separation
  - Each Web app has its own:
    - ServletContext, Class loader
    - Sessions, URL prefix, Directory structure

Structure of a Web Application

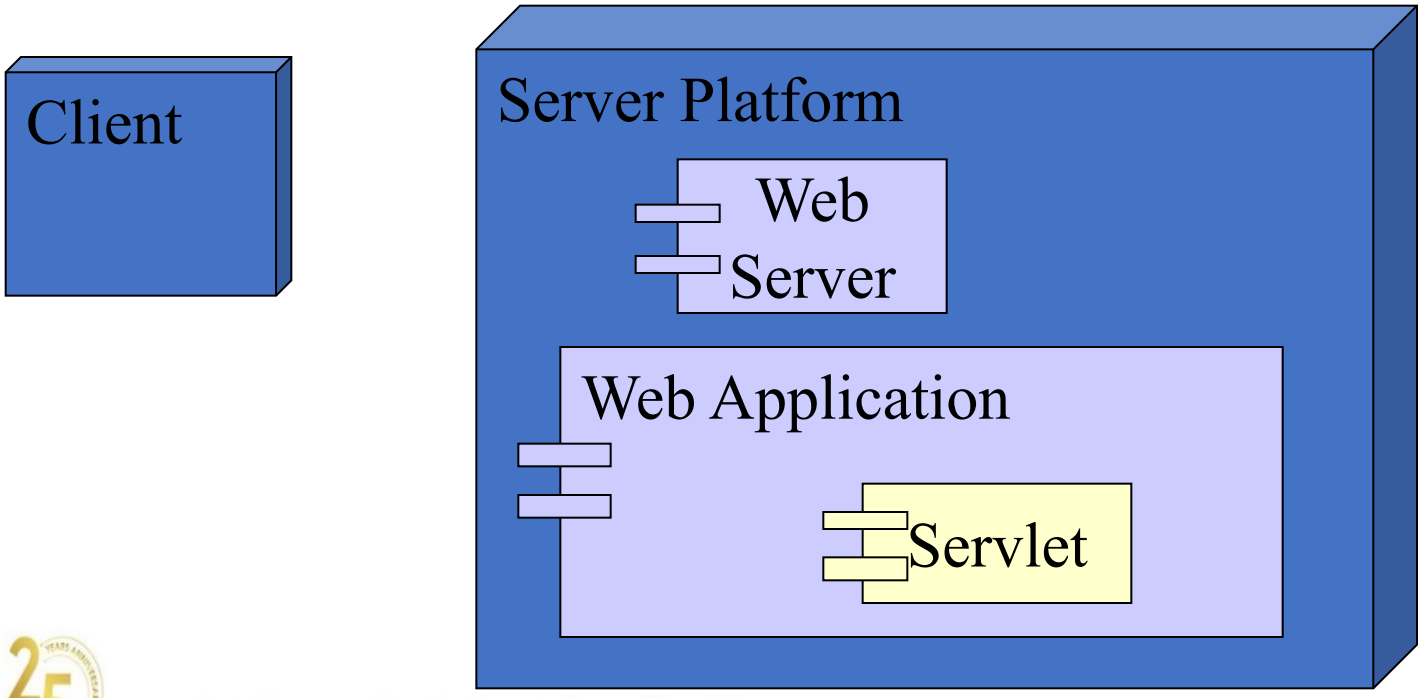
- JSP and regular Web content (HTML, style sheets, images, etc.):
  - Main directory or a subdirectory thereof.
- Servlets:
  - WEB-INF/classes (if servlet is unpackaged – i.e. in default package)
  - A subdirectory thereof that matches the package name.
- JAR files:
  - WEB-INF/lib.
- web.xml:
  - WEB-INF
- Tag Library Descriptor files:
  - WEB-INF or subdirectory thereof
- Files in WEB-INF <u>not</u> directly accessible to outside clients

Example Structure



Java Servlets

- A servlet is a Java program that is invoked by a web server in response to a request



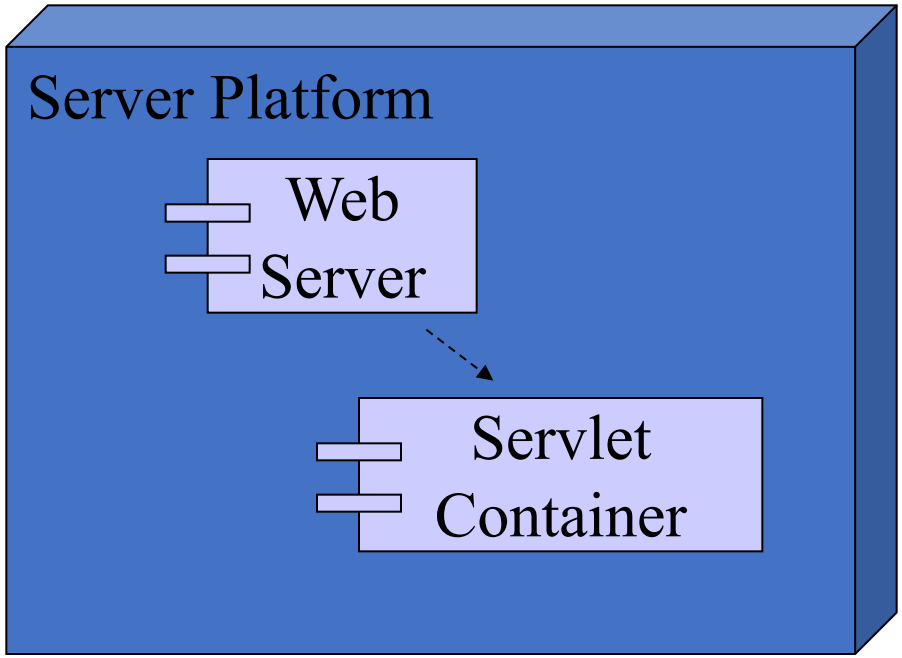
Java Servlets

- Together with web pages and other components, servlets constitute part of a web application

- Servlets can
  - create dynamic (HTML) content in response to a request
  - handle user input, such as from HTML forms
  - access databases, files, and other system resources
  - perform any computation required by an application

Java Servlets

- Servlets are hosted by a servlet container, such as Apache Tomcat*



The web server handles the HTTP transaction details

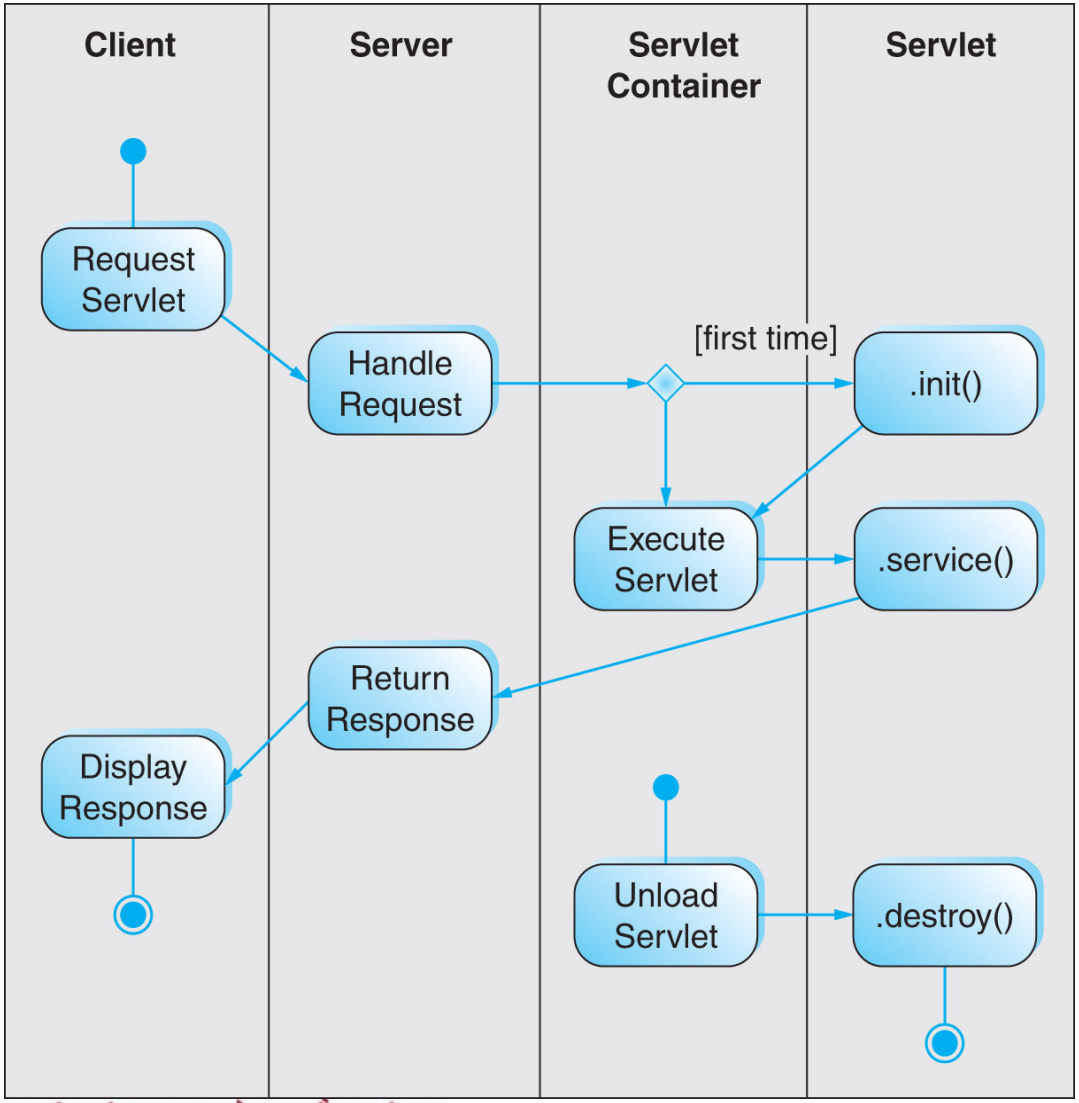
The servlet container provides a Java Virtual Machine for servlet execution

*Apache Tomcat can be both a web server and a servlet container

Environment For Developing and Testing Servlets

- Compile:
  - Need Servlet.jar. Available in Tomcat package
- Setup testing environment
  - Install and start Tomcat web server
  - Place compiled servlet at appropriate location

Servlet Operation



Servlet Methods

- Servlets have three principal methods

  `.init()`

  invoked once, when the servlet is loaded by the servlet container (upon the first client request)

  `.service(HttpServletRequest req, HttpServletResponse res)`

  invoked for each HTTP request
  parameters encapsulate the HTTP request and response

  `.destroy()`

  invoked when the servlet is unloaded
  (when the servlet container is shut down)

Servlet Methods

- The default .service() method simply invokes method-specific methods
  - depending upon the HTTP request method

```
                          .doGet()
.service()  ──────────►   .doPost()
                          .doHead()
                          … etc.
```

Methods

Methods	HTTP Requests	Comments
doGet	GET, HEAD	Usually overridden
doPost	POST	Usually overridden
doPut	PUT	Usually not overridden
doOptions	OPTIONS	Almost never overridden
doTrace	TRACE	Almost never overridden

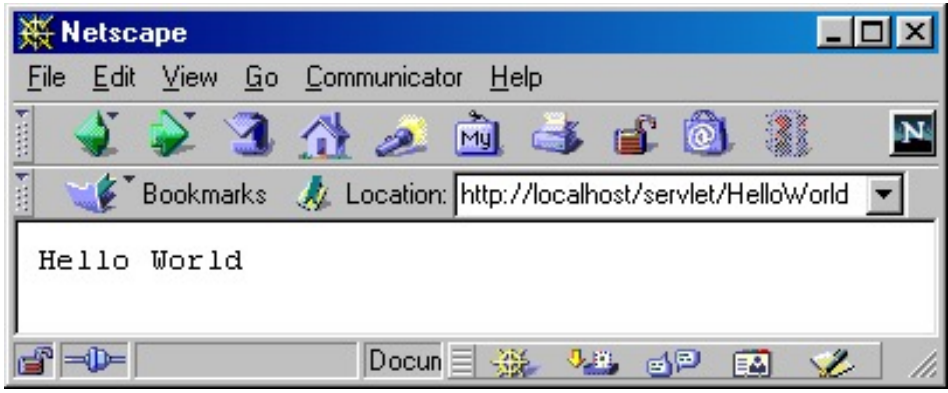
Servlet Example 1

- This servlet will say "Hello!" (in HTML)

```
package servlet;
import javax.servlet.http.*;

public class HelloServlet extends HttpServlet {
  public void service(HttpServletRequest req,
      HttpServletResponse res) throws IOException {
    PrintWriter htmlOut = res.getWriter();
    res.setContentType("text/html");
    htmlOut.println("<html><head><title>" +
      "Servlet Example Output</title></head><body>" +
      "<p>Hello!</p>" + "</body></html>");
    htmlOut.close();
  }
}
```

Servlet Example 2




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

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
      throws ServletException, IOException {
    PrintWriter out = response.getWriter();
    out.println("Hello World");
  }
}
```

Servlet Configuration

- The web application configuration file, web.xml, identifies servlets and defines a mapping from requests to servlets

An identifying name for the servlet (appears twice)

```
<servlet>
  <servlet-name>HelloServlet</servlet-name>
  <servlet-class>servlet.HelloServlet</servlet-
   class>
</servlet>
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

The servlet's package and class names

The pathname used to invoke the servlet (relative to the web application URL)

Environment Entries

- Servlets can obtain configuration information at run-time from the configuration file (web.xml)
  - a file name, a database password, etc.

- in web.xml:

```
<env-entry-description>password</env-entry-
   description>
<env-entry>
  <env-entry-name>UserId</env-entry-name>
<env-entry-value>Xy87!fx9*</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```

Environment Entries

- in the init() method of the servlet:

```
try {
  Context envCtx = (Context)
    (new InitialContext()).lookup("java:comp/env");
  password = (String) envCtx.lookup("password");
} catch (NamingException e) {
  e.printStackTrace();
} catch (ClassNotFoundException e) {
  e.printStackTrace();
}
```

Handling HTML Forms

- An HTML form can be sent to a servlet for processing
- The action attribute of the form must match the servlet URL mapping

```
<form method="post" action="hello" />
```

```
<servlet-mapping>
  <servlet-name>HelloServlet</servlet-name>
  <url-pattern>/hello</url-pattern>
</servlet-mapping>
```

Simple Form Servlet

```
<form action="hello" method="post" >
  <p>User Id:<input type="text" name="userid" /></p>
  <p><input type="submit" value="Say Hello" /></p>
</form>

public class HelloServlet extends HttpServlet {
  public void doPost(HttpServletRequest req,
      HttpServletResponse res) throws IOException {
    PrintWriter out = res.getWriter();
    res.setContentType("text/html");
    String userId = req.getParameter("userid");

   out.println("<html><head><title>Hello</title></he
   ad>"
      + "<body><p>Hello, " + userId
      + "!</p></body></html>");
    out.close();
}
```

State Management

- session: a series of transaction between user and application
- session state: the short-term memory that the application needs in order to maintain the session
  - e.g., shopping cart, user-id
- cookie: a small file stored by the client at the instruction of the server

Cookies

- The Set-Cookie: header in an HTTP response instructs the client to store a cookie

```
Set-Cookie: SESSIONID=B6E98A; Path=/slms;
   Secure
```

- After a cookie is created, it is returned to the server in subsequent requests as an HTTP request Cookie: header

```
Cookie: SESSIONID=B6E98A
```

Cookie Attributes

- Name: the unique name associated with the cookie
- Content: value stored in the cookie
- Expiration Date: cookie lifetime
- Domain: Defines the hosts to which the cookie should be returned
- Path: Defines the resource requests with which the cookie should be returned
- Secure: if true, cookie is returned only with HTTPS requests

Cookie Example

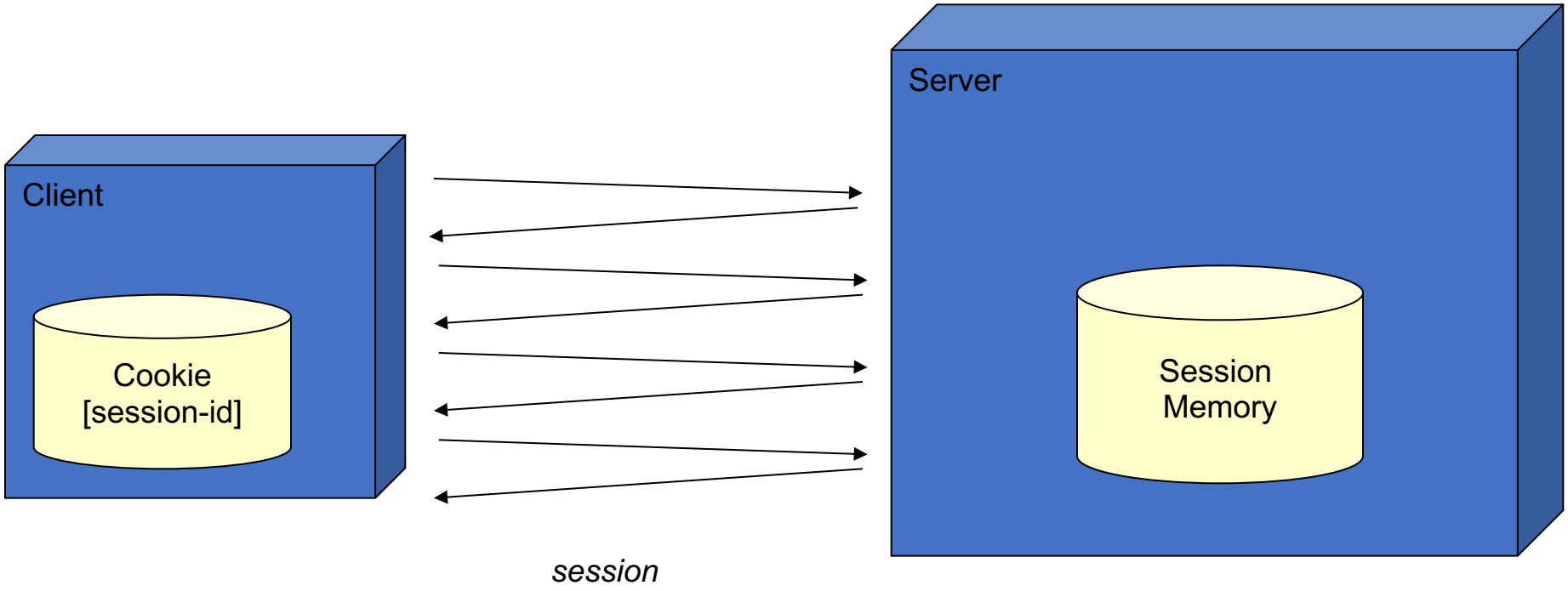
- Name: session-id
- Content: 104-1898635-929144
- Expiration Date: Monday, June 29, 2009 3:33:30 PM
- Domain: .ehsl.org
- Path: /slms
- Secure: no

- This cookie will be returned with all requests matching *.ehsl.org/slms*, through the indicated expiration date

Session Management

- HTTP is inherently stateless, i.e., there is no memory between transactions
- Applications must maintain a session memory if it is required
- Cookies are used to identify sessions, by recording a unique session-id

State Management



- At the start of a new session, the server sets a new cookie containing the session-id
- With each transaction, the client sends the session-id, allowing the server to retrieve the session

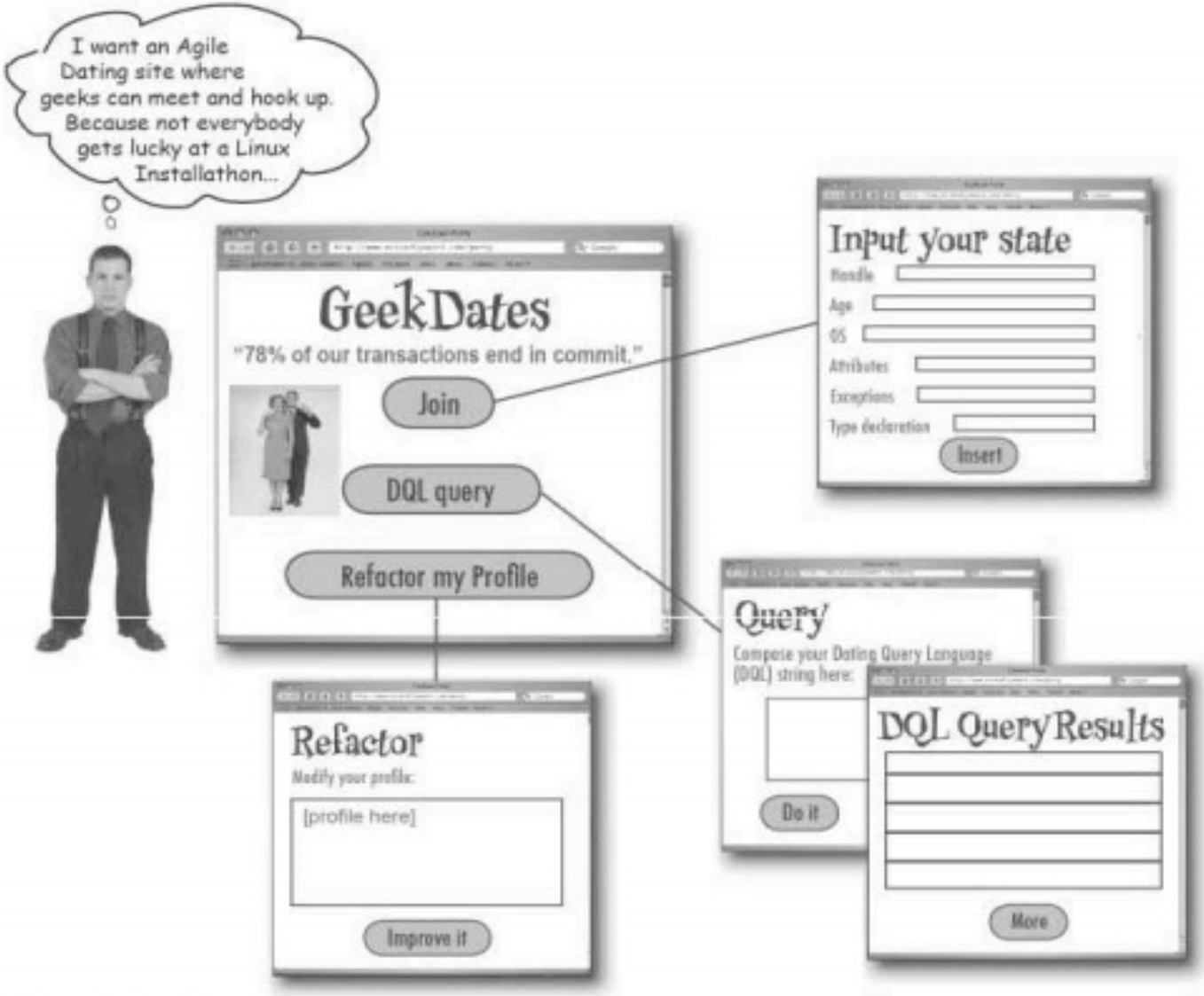
Session Attributes

- The methods
  `session.setAttribute(key, value)`
  `session.getAttribute(key)`
  store and retrieve session memory
- key is a string; value can be any object

- For example,

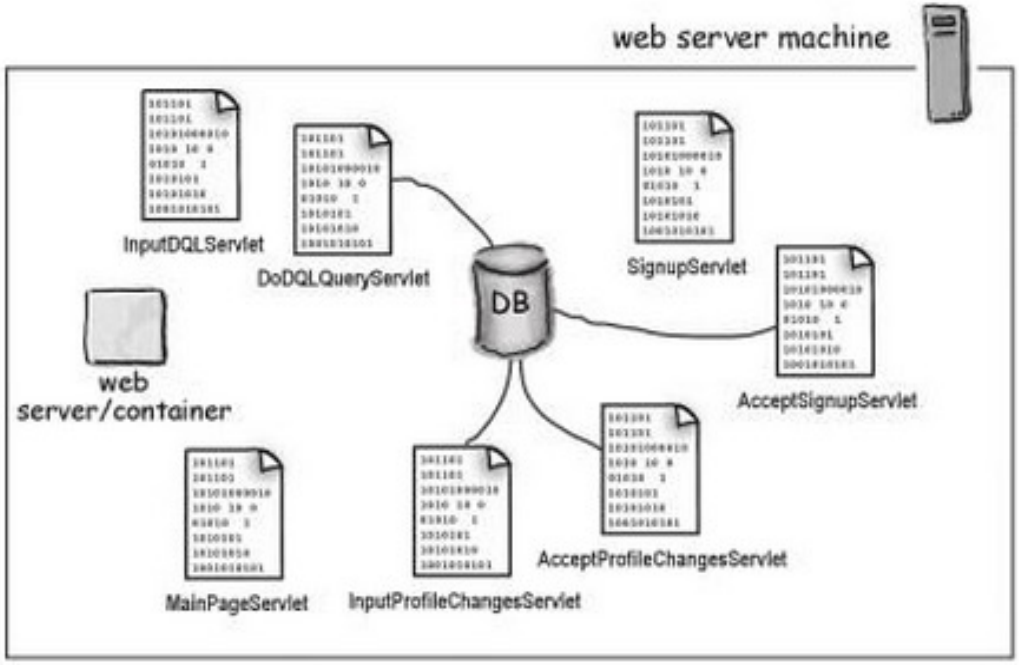
```
session.setAttribute("userid", userId);
String userId =
   (String)session.getAttribute("userid");
```

Problem

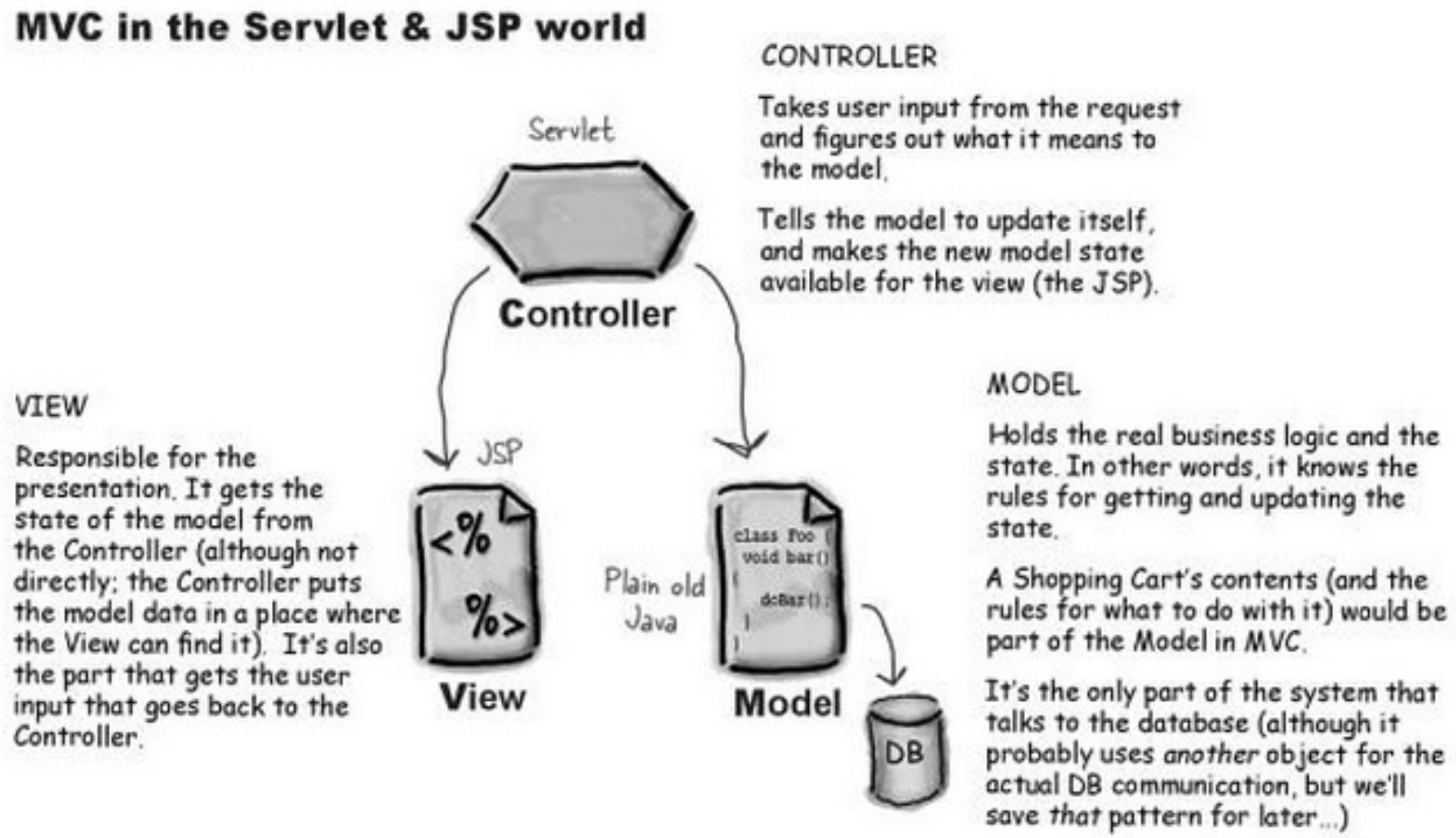

I want an Agile Dating site where geeks can meet and hook up. Because not everybody gets lucky at a Linux Installathon...
GeekDates
"78% of our transactions end in commit."
Join
DQL query
Refactor my Profile
Input your state
Handle
Age
OS
Attributes
Exceptions
Type declaration
Insert
Query
Compose your Dating Query Language (DQL) string here:
Do it
DQL Query Results
More
Refactor
Modify your profile:
[profile here]
Improve it

Initial Solution

- Develop a number of servlets
- Each servlet plays the role of one function (a.k.a business logic)



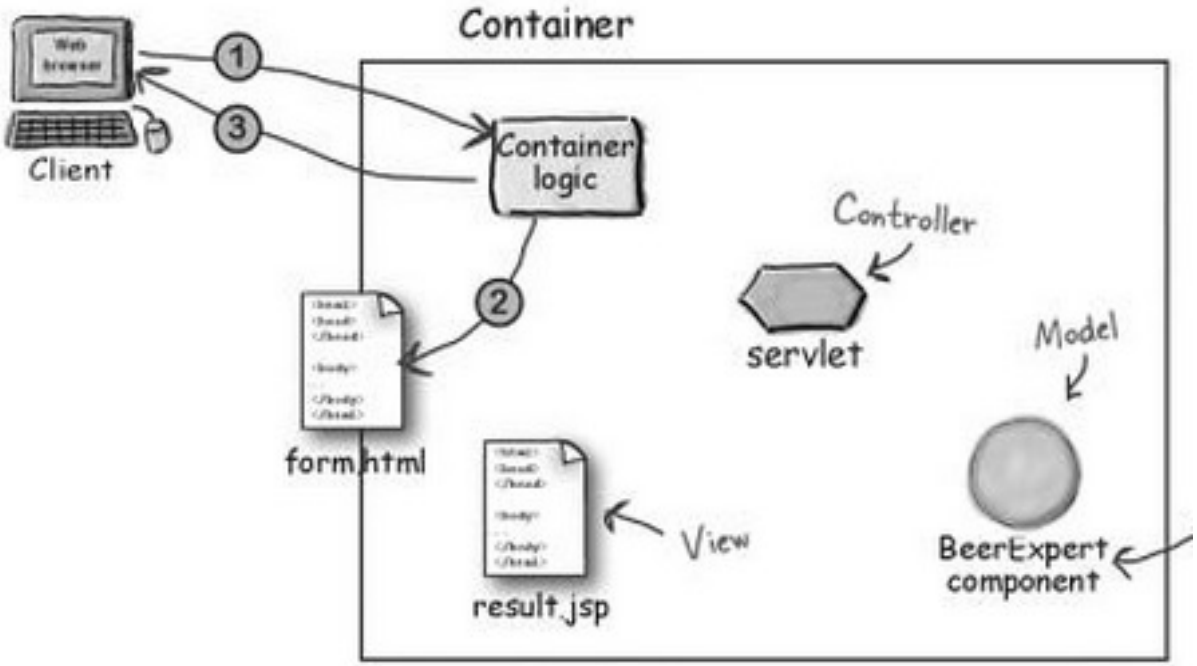
Better Solution: Using MVC

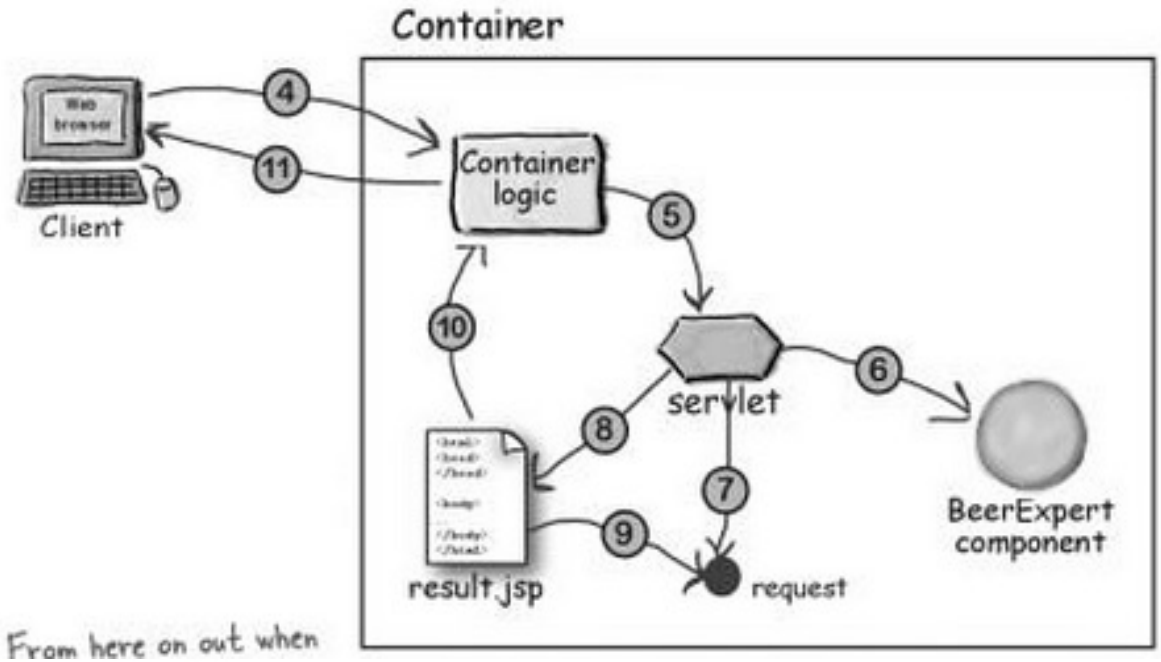


Example 1: Beer Recommendation

HTML page

JSP page





From here on out when

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...

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.

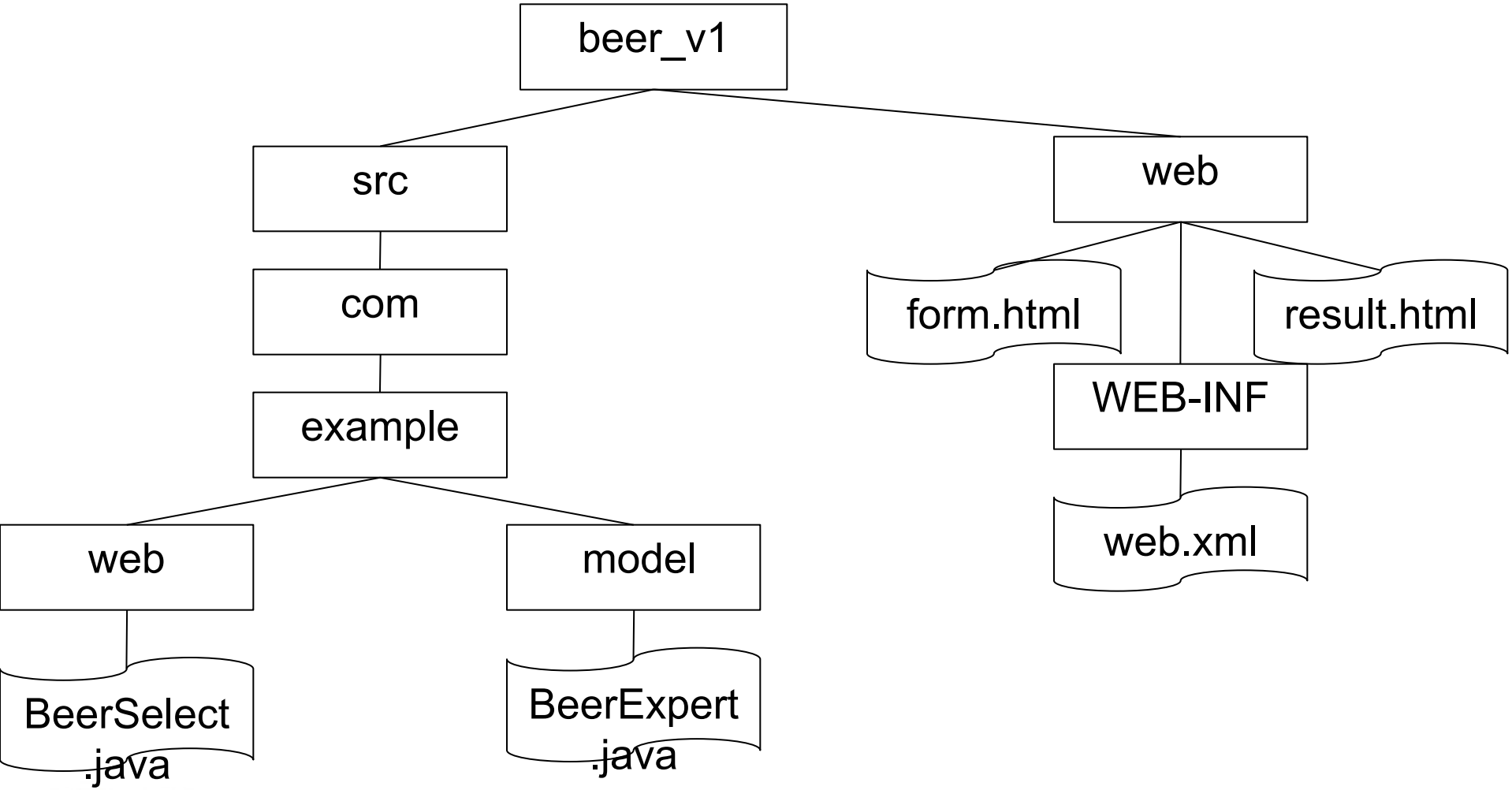
6 - The servlet calls the BeerExpert for help.

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

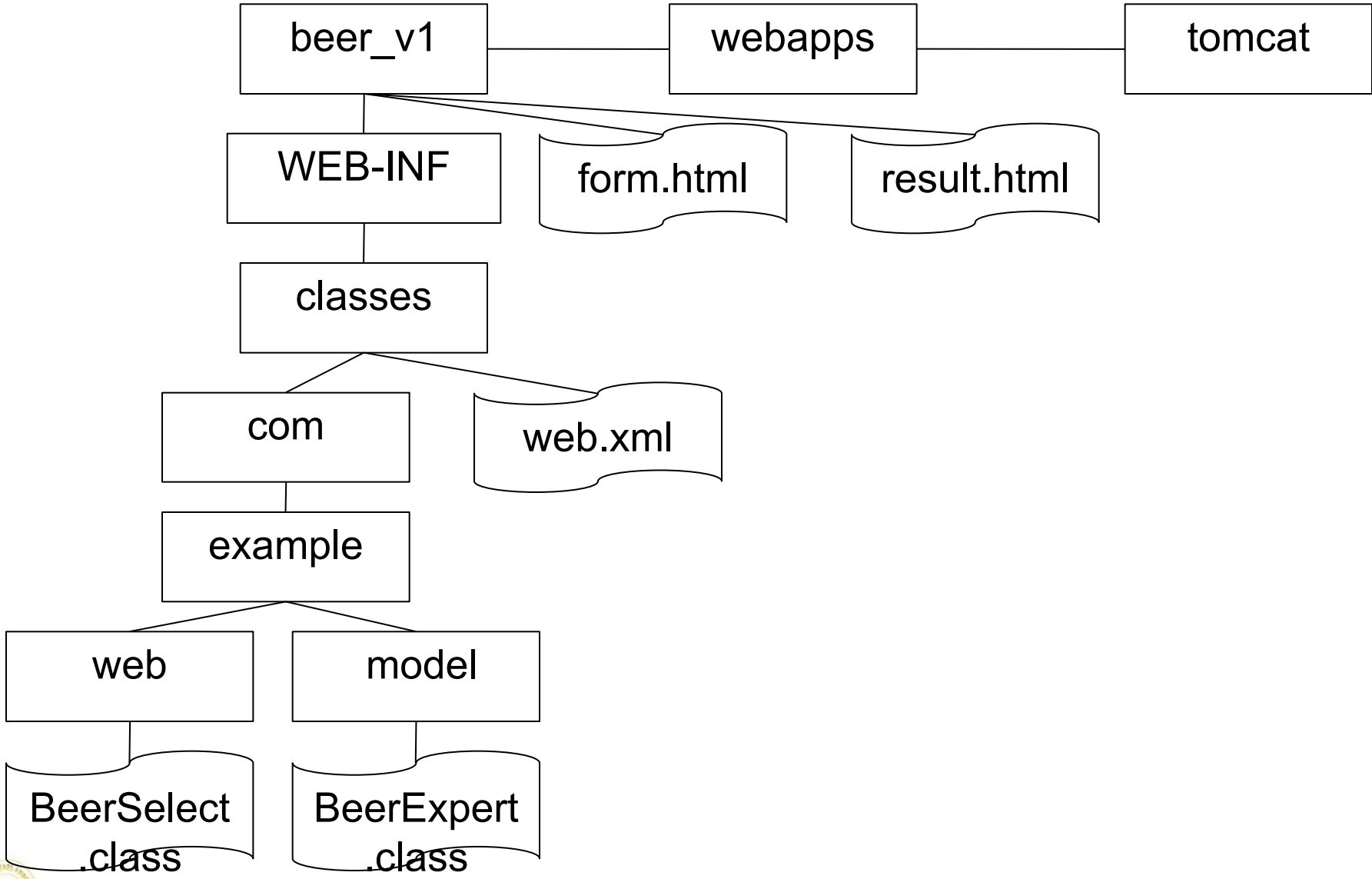
8 - The servlet forwards the request to the JSP.

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

Application Programming Structure



Structure of Folder Development



form.html


```
<form method="POST"
action="SelectBeer.do">
 Select beer characteristics
 <p>Color:
  <select name="color" size="1">
  <option value="light">light</option>
  <option value="amber">amber</option>
  <option value="brown">brown</option>
  <option value="dark">dark</option>
 </select>
 <center> <input type="SUBMIT">  </center>
</form>
```

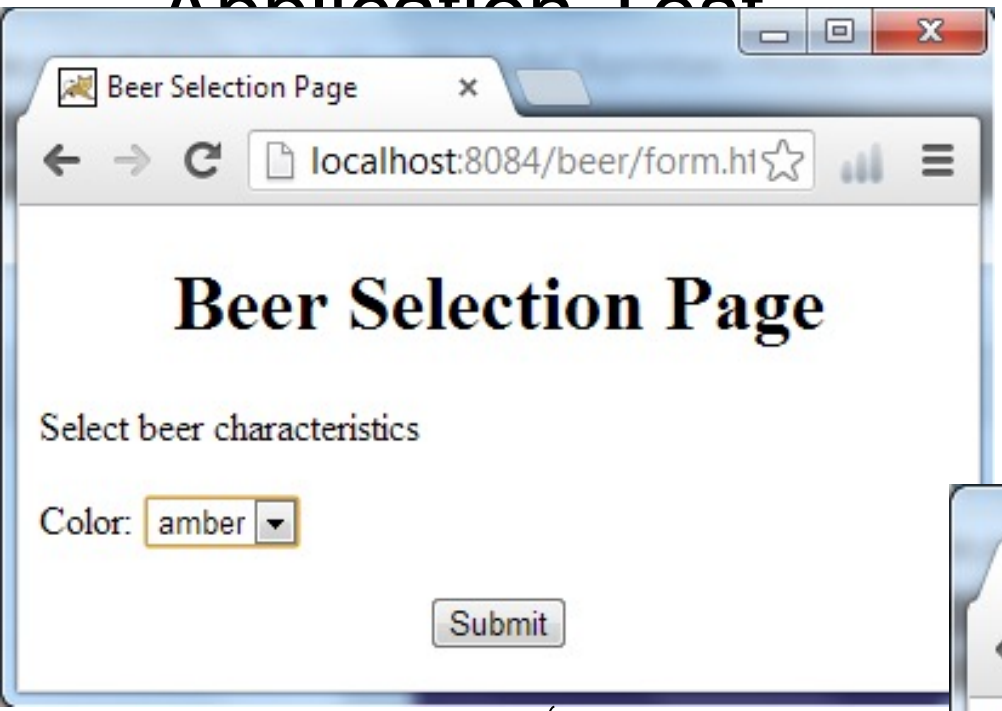
web.xml

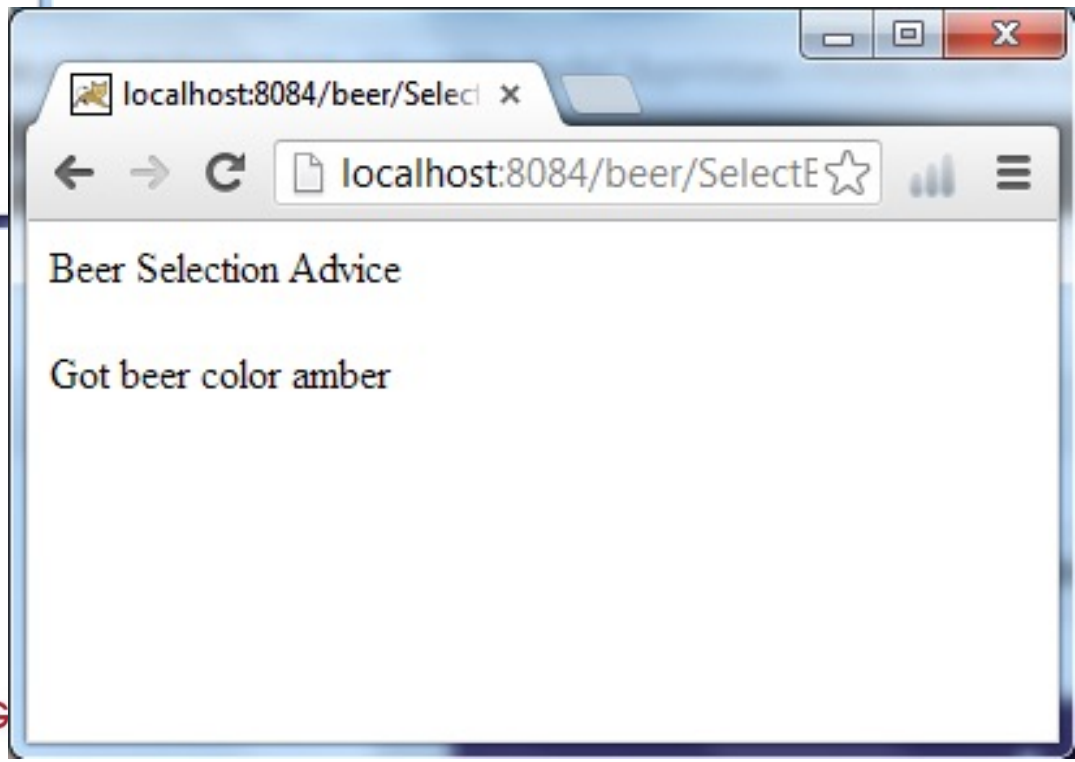
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app …>
   <servlet>
       <servlet-name>ServletBeer</servlet-name>
       <servlet-class>com.example.web.BeerSelect
                                  </servlet-class>
    </servlet>
    <servlet-mapping>
        <servlet-name>ServletBeer</servlet-name>
        <url-pattern>/SelectBeer.do</url-pattern>
    </servlet-mapping>
</web-app>
```

Servlet BeerSelect – Version 1

```
public class BeerSelect extends HttpServlet {
@Override
protected void doPost(HttpServletRequest request,
  HttpServletResponse response) throws
  ServletException, IOException {
      response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("Beer Selection Advice<br>");
      String c = request.getParameter("color");
      out.println("<br>Got beer color "+c);
    }
}
```

Application Test





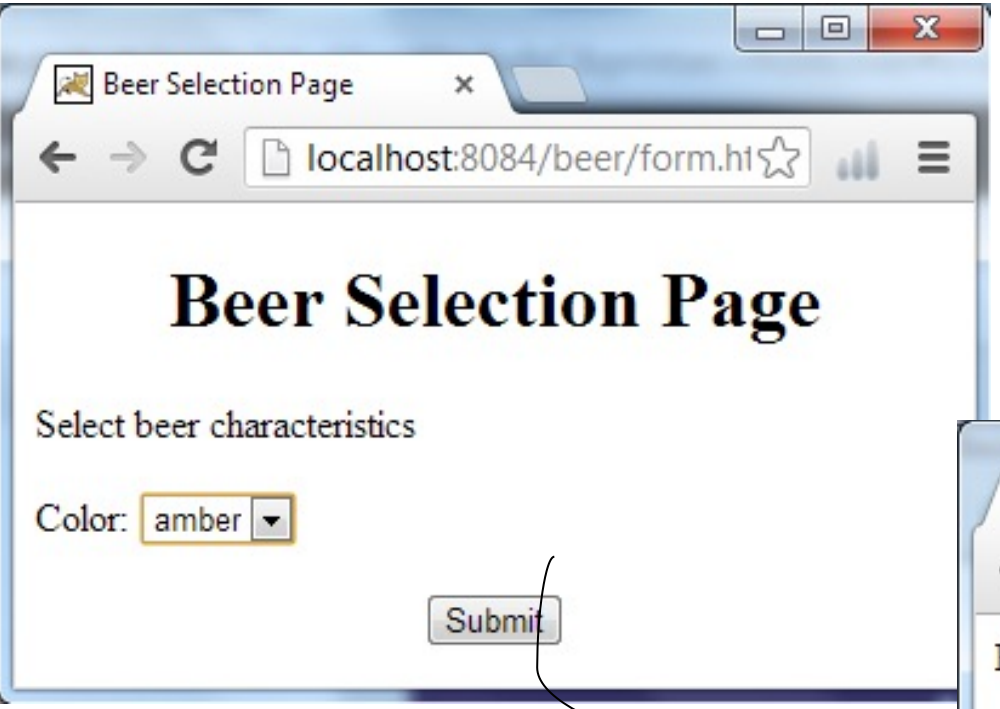
Model BeerExpert

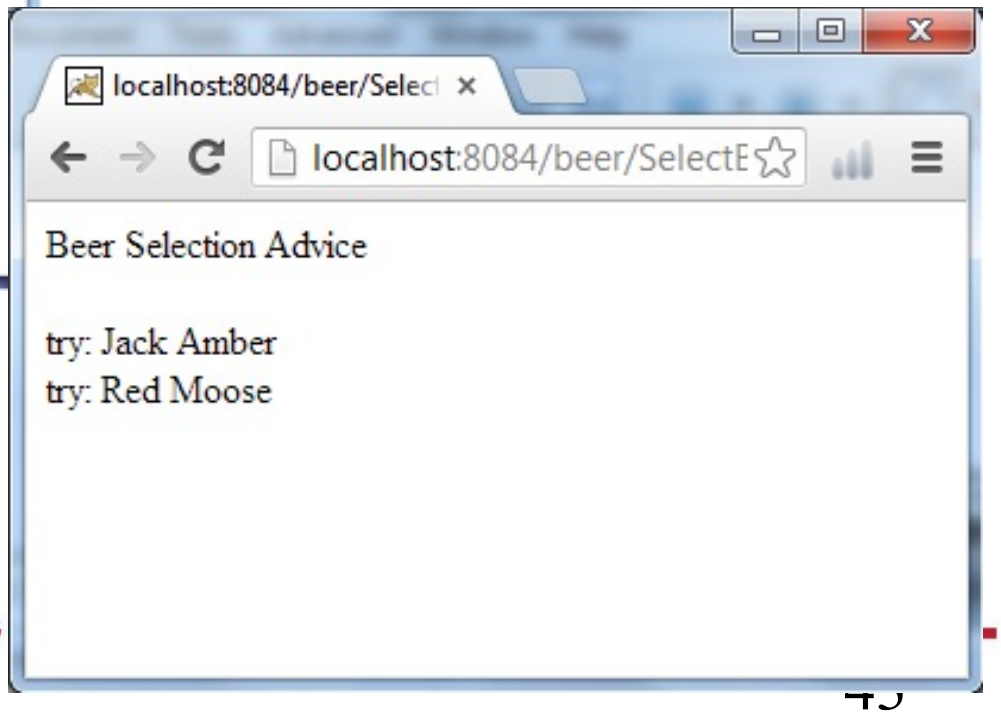
```
public class BeerExpert {
    public List getBrands(String color){
        List brands = new ArrayList();
        if(color.equals("amber")){
            brands.add("Jack Amber");
            brands.add("Red Moose");
        }
        else{
            brands.add("Jail Pale Ale");
            brands.add("Gout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

Servlet BeerSelect – Version 2

```
import package com.example.web;
…
protected void doPost(HttpServletRequest request,
   HttpServletResponse response)
            throws ServletException, IOException {
        String c = request.getParameter("color");
        BeerExpert be = new BeerExpert();
        List result = be.getBrands(c);
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        out.println("Beer Selection Advice<br>");
        Iterator it = result.iterator();
        while(it.hasNext()){
            out.print("<br>try: "+it.next());
        }
    }
```

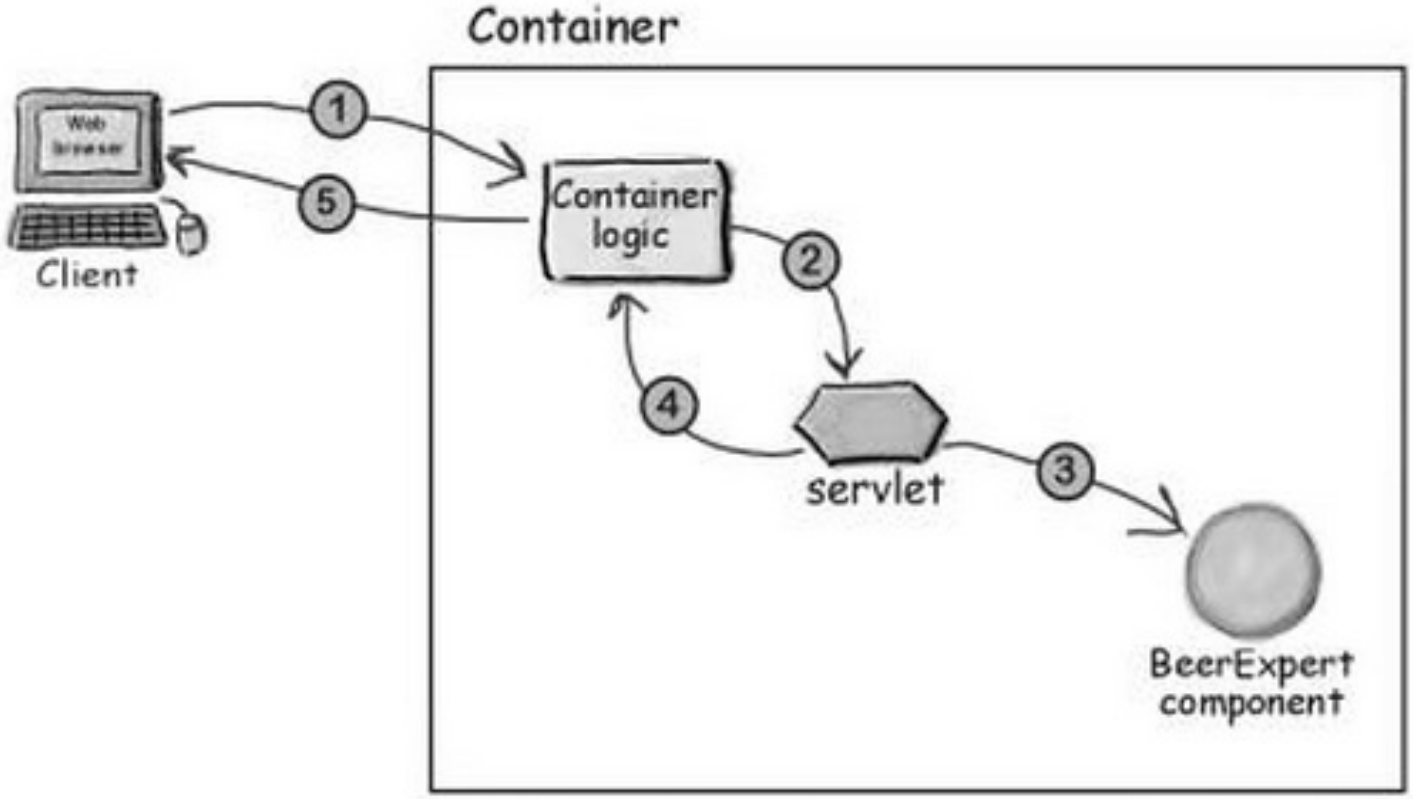
Application Test





Current Architecture of the Application

What's working so far...



1 - The browser sends the request data to the Container.

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

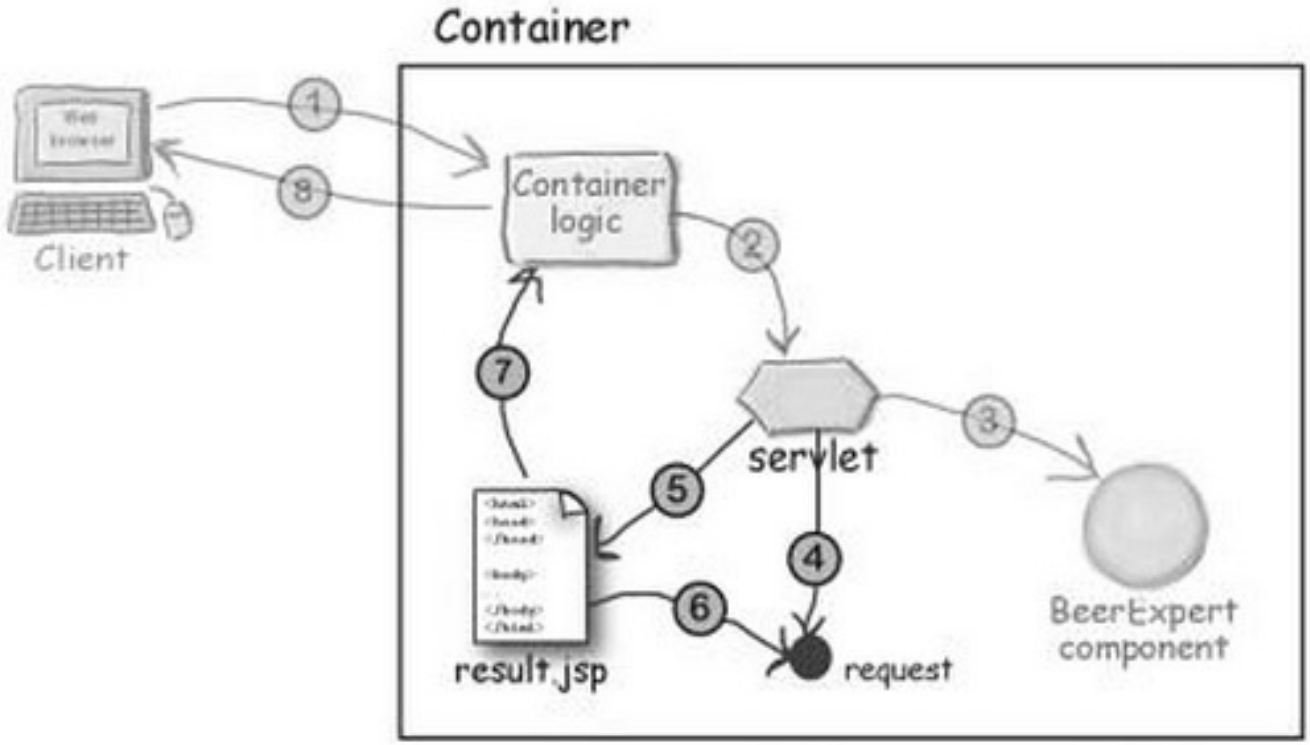
3 - The servlet calls the BeerExpert for help.

4 - The servlet outputs the response (which prints the advice).

5 - The Container returns the page to the happy user.

Desired Application Architecture

What we WANT...



1 - The browser sends the request data to the Container.

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

3 - The servlet calls the BeerExpert for help.

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

5 - The servlet forwards the request to the JSP.

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

7 - The JSP generates a page for the Container.

8 - The Container returns the page to the happy user.

Result.jsp

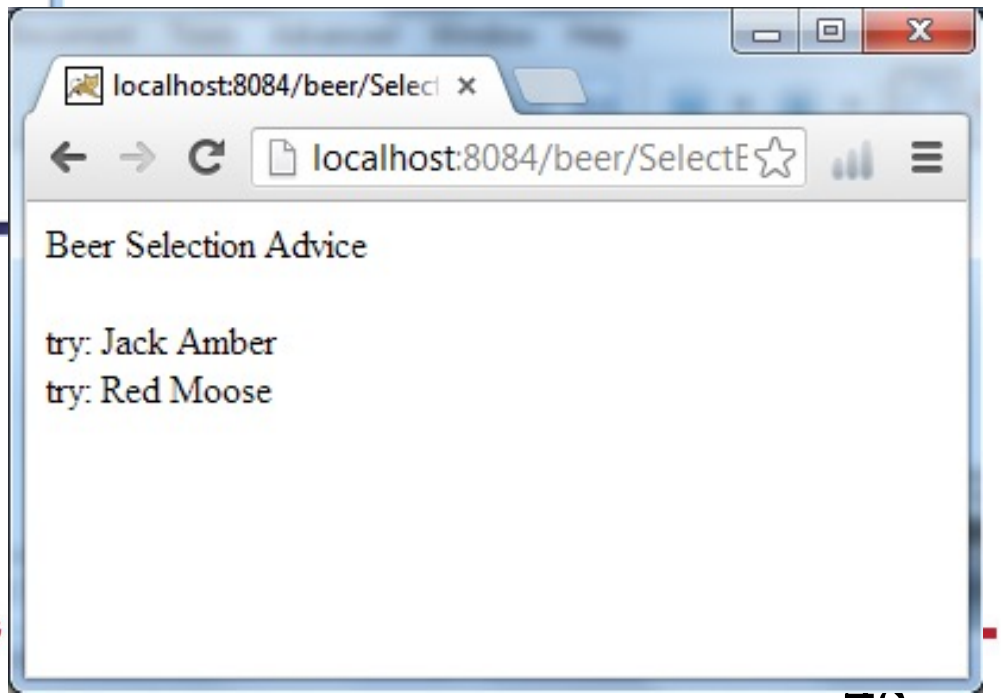
```
<%@ page import="java.util.*"%>
<!DOCTYPE html>
<html>
<body>
        <h1 align="center">Beer Recommendation </h1>
   <p>
            <%
            List styles=(List)
   request.getAttribute("styles");
            Iterator it = styles.iterator();
            while(it.hasNext()){
                out.print("<br>try: "+it.next());
            }
            %>
    </body>
</html>
```

Servlet BeerSelect – Version 3

```
import package com.example.web;
…
protected void doPost(HttpServletRequest request,
   HttpServletResponse response)
            throws ServletException, IOException {
        String c = request.getParameter("color");
        BeerExpert be = new BeerExpert();
        List result = be.getBrands(c);

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

Application Test



Review

- Java servlets
- Servlet methods and operation
- HTML forms and servlets
- HTTP cookies
- Web application state management
- Beer Recommendations with MVC model