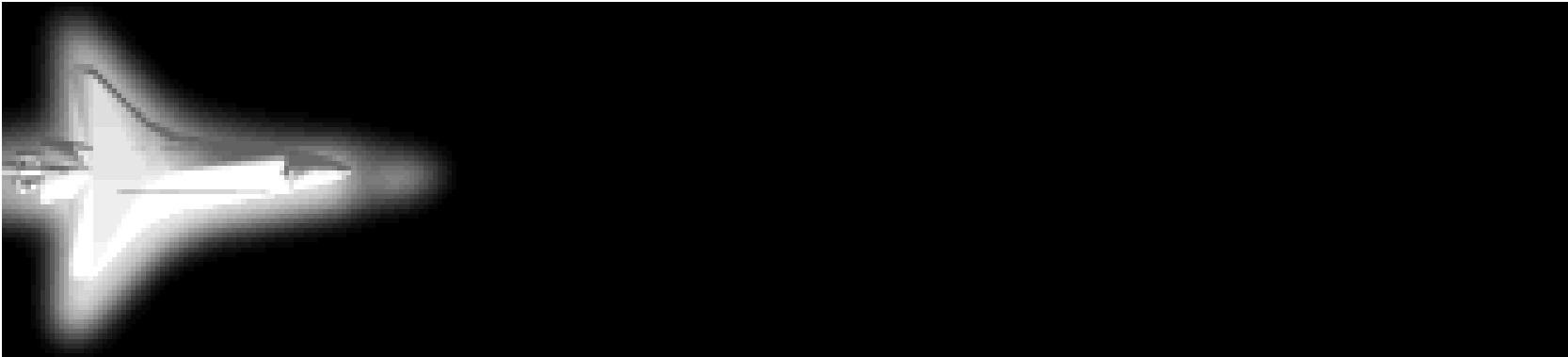
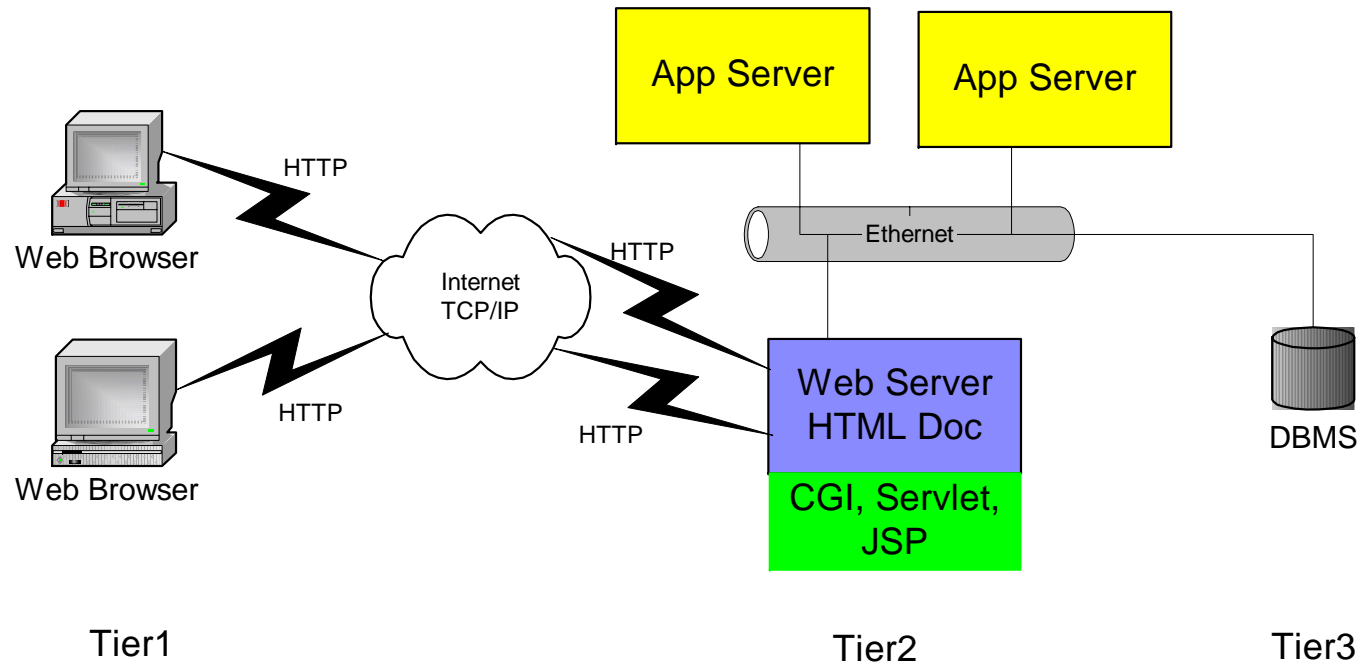


Introduction to Internet Computing

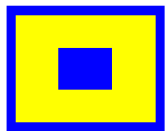


Typical Web Application



Presentation layer: CGI, Servlet, JSP, ASP

Business logic layer: EJB, COM+, CORBA



Hypertext Transaction Protocol

- ◆ HTTP is an application protocol that allows Web clients and servers to communicate
- ◆ Main function of HTTP is to download HTTP documents specified by URLs
- ◆ HTTP is built on top of TCP/IP
- ◆ HTTP is stateless
- ◆ HTTP is Web's version of RPC
- ◆ Current dominant version is HTTP 1.0

HTTP Request

- ◆ A Web client sends an HTTP request to a Web server specified by a URL
- ◆ An HTTP request has a *request line*, a few *request header fields*, and an *entity body*

<method> <resource ID> <HTTP version>

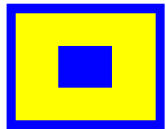
[<header> : <value>]

.....

[<header> : <value>]

(blank line)

[Entity body]



HTTP Request Example

GET /path/file.html HTTP/1.0

Accept: text/html

Accept: audio/x

User-agent: MacWeb

HTTP Response

- ◆ Web server processes the HTTP request, and sends back an HTTP response
- ◆ HTTP response has a *response header line*, a few *response header fields*, and an *entity body*

<HTTP Version> <result code> [<explanation>]

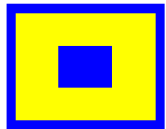
[<header> : <value>]

.....

[<header> : <value>]

(blank line)

[Entity body]



HTTP Response Example

HTTP/1.0 200 OK

Server: NCSA/1.3

Mime_version: 1.0

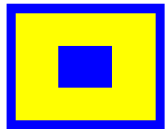
Content_type: text/html

Content_length: 2000

<HTML>

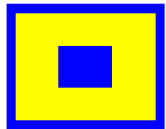
.....

</HTML>



Multipurpose Internet Mail Extensions

- ◆ MIME was designed to support extensible mechanisms for transmitting multimedia email
- ◆ HTTP uses MIME to specify its contents type
- ◆ Common HTTP content-types
 - Text/html
 - Text/plain
 - Image/gif
 - Image/jpeg
 - Video/mpeg



Composition of a URL

- ◆ A sample URL may look like

`http://www.server.dom/app/resource?name=J&age=18`

- ◆ *http://www.server.dom* identifies a Web server
- ◆ */app/resource* identifies a resource of a Web application *app* on the Web server
- ◆ The substring after the ? mark is the query string. It was originally designed to support database queries. But it is mainly used to pass name/value pairs to the Web server

GET

- ◆ GET is mainly for retrieving a file from a Web server
- ◆ When you enter a URL into the address field of a Web browser, or when you click on a hyperlink, the Web browser issues a GET HTTP request to the target HTTP server
- ◆ The URL for a GET may contain a query-string, which may be used to pass arguments
- ◆ The documents retrieved by GETs are usually cached

GET Example

- ◆ Submit data from a form

```
<html>
```

```
<body>
```

```
<form method="get" action="http://server/app/input">
```

```
  Enter your name: <input type="text" name="user">
```

```
  <input type="submit" value="Submit">
```

```
</form>
```

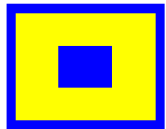
```
</body>
```

Enter your name:

Submit

```
</html>
```

- ◆ After entering “Ada” and clicking on Submit, URL becomes
`http://server/app/input?user=Ada`

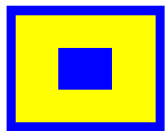


Drawbacks of GET

- ◆ Even though GET is the default HTTP method, it should not be used to submit form data for two reasons
 - All user data goes to URL query string
 - It is hard to control how much data a user will submit
 - All user data will be submitted through a query_string environment variable, which has limited memory capacity
 - If user data is larger than environment memory capacity, the data may be truncated, or the server may crash

POST

- ◆ POST is similar to GET, but user data is transmitted to the Web server in the *entity body*. The server processor (like CGI) can read these data through `System.in`
- ◆ It is safer to use POST to submit data to a Web server
- ◆ If POST is used, environment variable `query_string` will not be set on the server

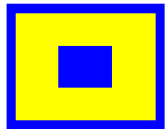


How to Support a Session?

- ◆ Typical e-commerce applications need to maintain user data for a user session
- ◆ HTTP is stateless: successive GET/POST calls don't share information through HTTP: a socket is opened to connect to the server, submit data, process data, and return data. Then socket connection terminates
- ◆ Two basic patches to solve the problem
 - Hidden form fields
 - Cookie

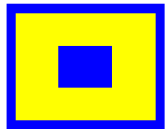
Hidden Form Fields

- ◆ Inside an HTTP form, some data fields are declared and made invisible:
 - `<input type="hidden" name="x" value="100">`
- ◆ Session data is stored in hidden fields
- ◆ Web browsers submit form data to Web server, the server set value of some hidden fields of the following form, and send the new form back to the Web browser
- ◆ Main problem: insecure. Hackers can use View Source to read the values of the hidden fields, and modify them



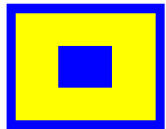
Cookies

- ◆ A cookie is a small piece of session data to be stored on client's machine: $x=1, y=hi$
- ◆ After a Web browser submits data to a Web server, the server may create a sequence of cookies in form of a sequence of *name=value* pairs. Server may set the max age of the cookies
- ◆ The cookies are returned to the Web browser as part of the response, and saved in files on the client machine



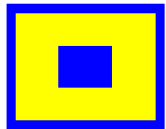
Cookies ...

- ◆ When the client makes the next request to the same Web server, all previous cookies sent over by this server in this session will be automatically sent back to the server
- ◆ When the client shuts down the Web browser, or if a cookie's life has come to the end, the cookie will be deleted from the client machine
- ◆ Main problems
 - Insecure
 - Inefficient



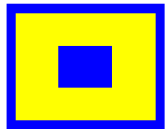
Web Server Extensions

- ◆ Web server by itself is only responsible to serve file requests
- ◆ CGI and Servlet are simple extensions to Web servers
- ◆ If a request needs data processing, the Web server will forward the request to a CGI or servlet
- ◆ CGI or servlet processes the input data, and dynamically generates an HTML file to let the Web server send back to the client Web browser
- ◆ CGI and Servlet are two standard interfaces for Web server to communicate with Web extensions



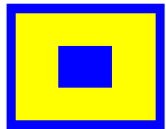
Common Gateway Interface (CGI)

- ◆ Each Web server has its own way of identifying requests for CGI applications
 - URL contains “/cgi-bin”
 - The target application has extension “.cgi”
- ◆ For each CGI request, the Web server sets values for a standard set of environment variables including `query_string`, `server_port`, `request_method`, `content_length`, and `remote_host`



Common Gateway Interface ...

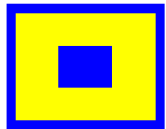
- ◆ A CGI application is a script or executable that reads request entity body through standard system input stream, reads value of environment variables, composes an HTML file, and sends the file through standard system output stream
- ◆ Main problem: Each CGI request starts a new server process, slow and not scalable



Servlets

◆ Servlets

- Like applets running in a container, but
 - Execute on server's machine, supported by most Web servers
- Communication with client via HTTP protocol
 - Client sends HTTP request
 - Server receives request, servlets process it
 - Results returned (HTML document, images, binary data)



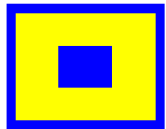
The Servlet API

◆ **Servlet** interface

- Implemented by all servlets
- Many methods invoked automatically by servlet container
 - Similar to applets (`paint`, `init`, `start`, etc.)
- **abstract** classes that implement **Servlet**
 - `GenericServlet` (`javax.servlet`)
 - `HTTPServlet` (`javax.servlet.http`)
- My example extends `HTTPServlet`

◆ **Methods**

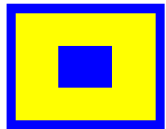
- `void init(ServletConfig config)`
 - Automatically called, argument provided



The Servlet API ...

◆ Methods

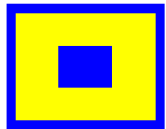
- **ServletConfig getConfig()**
 - Returns reference to config object, gives access to config info
- **void service (ServletRequest request, ServletResponse response)**
 - Key method in all servlets
 - Provide access to input and output streams
 - Read from and send to client
- **void destroy()**
 - Cleanup method, called when servlet exiting



HttpServlet Class

◆ HttpServlet

- Base class for Web-based servlets
- Overrides method **service**
 - Request methods:
 - **GET** - retrieve HTML documents or image
 - **POST** - send server data from HTML form
- Methods **doGet** and **doPost** respond to **GET** and **POST**
 - Called by **service**
 - Receive **HttpServletRequest** and **HttpServletResponse** objects



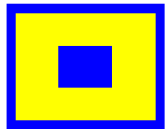
HttpServletRequest Interface

◆ **HttpServletRequest** interface

- Object passed to `doGet` and `doPost`
- Extends `ServletRequest`

◆ **Methods**

- **String** `getParameter(String name)`
 - Returns value of parameter name
- **Enumeration** `getParameterNames()`
 - Returns names of parameters
- **String[]** `getParameterValues(String name)`
 - Returns array of strings containing values of a parameter
- **Cookie[]** `getCookies()`
 - Returns array of `Cookie` objects, can identify client



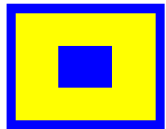
HttpServletResponse Interface

◆ HttpServletResponse

- Object passed to `doGet` and `doPost`
- Extends `ServletResponse`

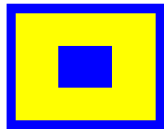
◆ Methods

- `void addCookie(Cookie cookie)`
 - Add `Cookie` to header of response to client
- `ServletOutputStream getOutputStream()`
 - Gets byte-based output stream, send binary data to client
- `PrintWriter getWriter()`
 - Gets character-based output stream, send text to client
- `void setContentType(String type)`
 - Specify MIME type of the response
 - Helps display data

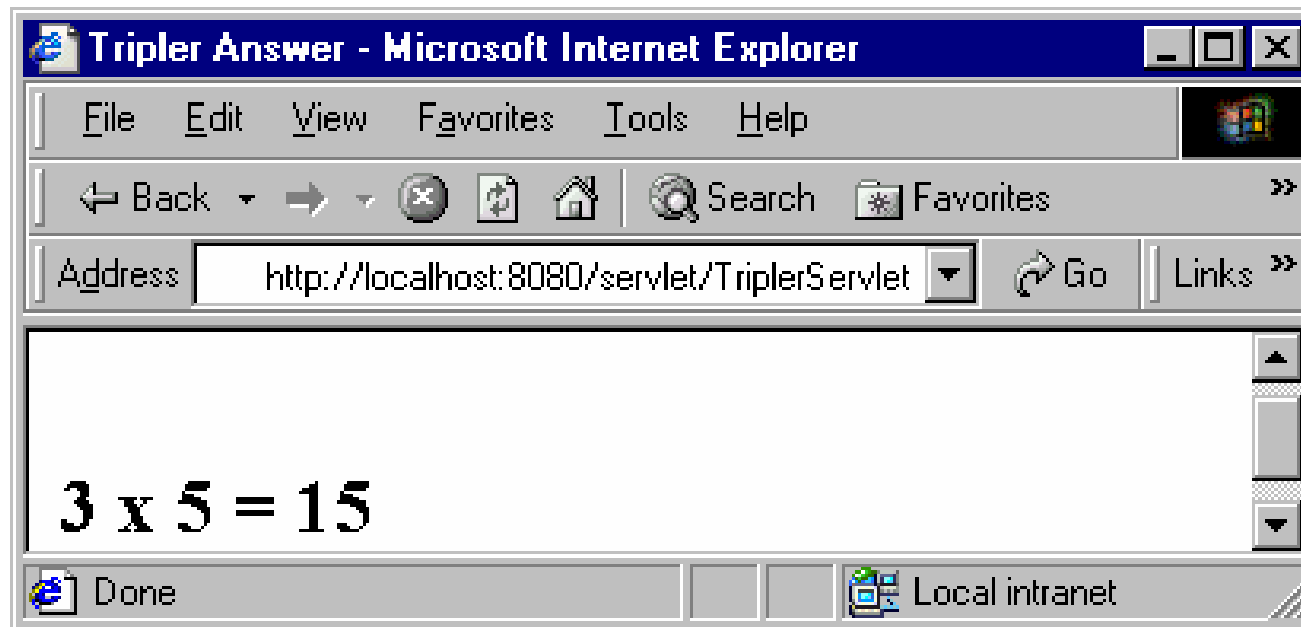
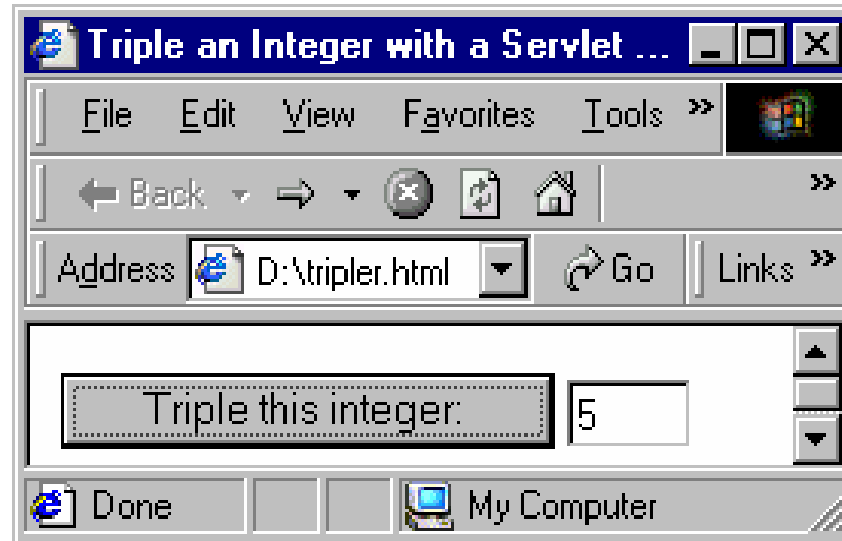


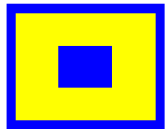
A Servlet for Tripling Integers

- ◆ The client enters an integer in a text field, and then clicks on button “Triple this integer”
- ◆ The Web browser sends a POST request to the specified Web server with the integer
- ◆ Web server invokes the servlet to triple the integer, and passes back the response from the servlet



Running Triple Servlet





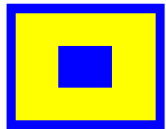
Triple.html

```
<html>
<head>
<title>Triple an Integer with a Servlet</title>
</head>
<body>
<form method="post"
      action="http://localhost:8080/servlet/TriplerServlet">
  <input type="submit" value="Triple this integer: ">
  <input type="text" name="number" value="0" size="4">
</form>
</body>
</html>
```

TriplerServlet.java

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

public class TriplerServlet extends HttpServlet {
    public void doPost(HttpServletRequest request,
                       HttpServletResponse response)
        throws ServletException, IOException {
        String value = request.getParameter("number");
        int n = Integer.parseInt(value);
        response.setContentType("text/html");
        PrintWriter output = response.getWriter();
```

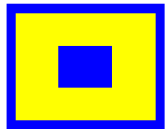


TriplerServlet.java ...

```
output.println("<html>");
output.println("<title>Tripler Answer</title>");
output.println("<br><br><h2>");
output.println("3 x " + n + " = " + 3*n);
output.println("</h2>\n</html>");
output.close();
}
}
```

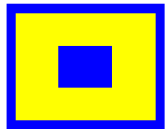
HttpSession

- ◆ In addition to using cookies to maintain session data on client machines, servlets also support HttpSession
- ◆ Each client session can be associated with a unique HttpSession on the Web server, which is basically a hash table of name/value pairs



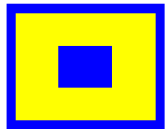
Observations

- ◆ CGI and servlet are both simple extensions to Web servers to provide dynamic Web pages
- ◆ CGI and servlet are not suitable to build peer-to-peer distributed systems
- ◆ Since CGI and servlet invocations go through Web server, efficiency is low



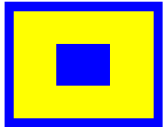
Remote Method Invocations (RMI)

- ◆ Remote Method Invocation (RMI)
 - Allows remote method calls
 - Objects in different programs can communicate
 - Method calls appear as normal
 - Based on Remote Procedure Calls (RPC)
 - Developed in 1980's
 - Allows procedural program (like C) to call function on another computer
 - Performs networking and data marshalling (packaging arguments and return values)
 - Not compatible with objects
 - Interface Definition Language required - describe functions
 - RMI is Java's implementation of RPC

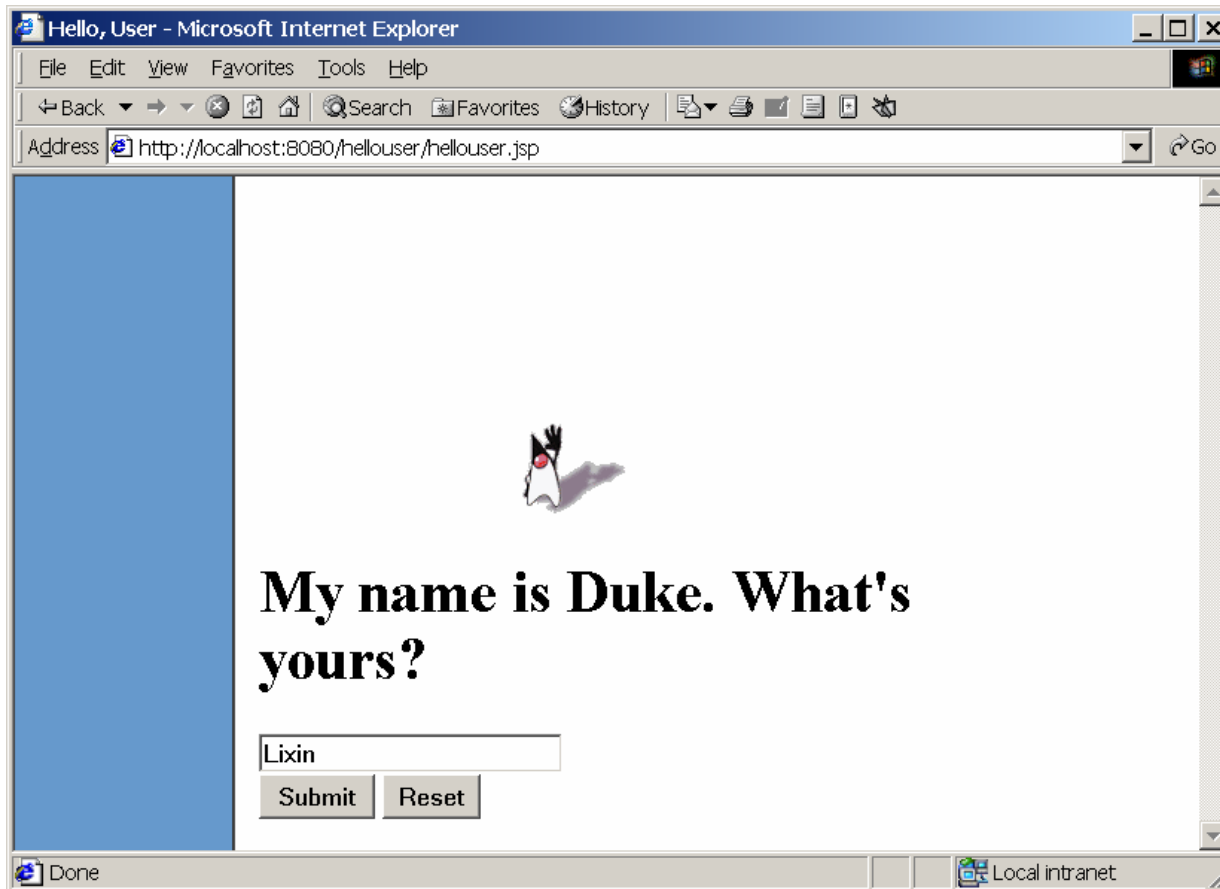


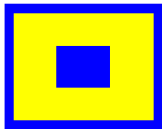
Java Server Pages

- ◆ Java Server Pages (JSP) is based on servlets, but greatly simplifies the development process
- ◆ A JSP page is an HTML file, with embedded scripts for filling in dynamic data
- ◆ Upon first client invocation, a JSP file is compiled on the Web server into a servlet
- ◆ Debugging for JSP scripts is harder
- ◆ While JSP mainly uses Java as script language, Active Server Pages (ASP) of Microsoft uses VBScript, VB, and C#

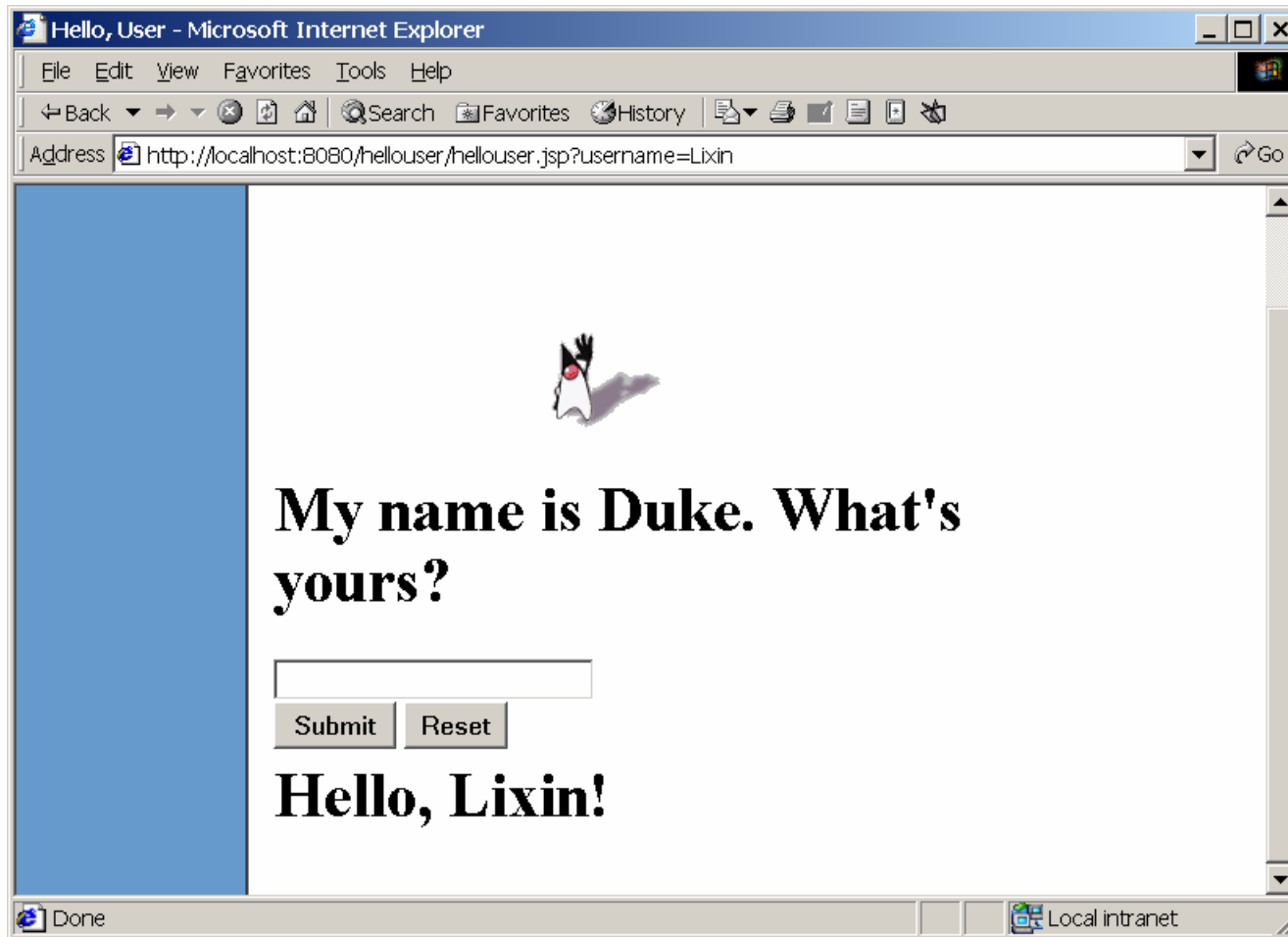


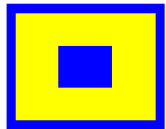
A Dynamic Hello World JSP





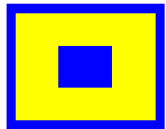
Response Page





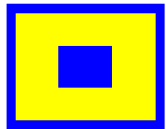
hellouser.jsp

```
<%@ page import="hello.NameHandler" %>
<jsp:useBean id="mybean" scope="page" class="hello.NameHandler" />
<jsp:setProperty name="mybean" property="*" />
<html>
<head><title>Hello, User</title></head>
<body bgcolor="#ffffff" background="background.gif">
<%@ include file="dukebanner.html" %>
<table border="0" width="700">
<tr>
<td width="150"> &nbsp; </td>
<td width="550">
```



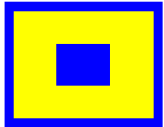
hellouser.jsp ...

```
<h1>My name is Duke. What's yours?</h1>
</td></tr>
<tr>
<td width="150" &nbsp; </td>
<td width="550">
<form method="get">
<input type="text" name="username" size="25"><br>
<input type="submit" value="Submit">
<input type="reset" value="Reset">
</td></tr>
</form>
</table>
```



hellouser.jsp ...

```
<%  
    if ( request.getParameter("username") != null ) {  
%>  
<%@ include file="response.jsp" %>  
<%  
    }  
%>  
</body>  
</html>
```



response.jsp

```
<table border="0" width="700">
```

```
<tr>
```

```
<td width="150"> &nbsp; </td>
```

```
<td width="550">
```

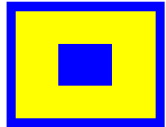
```
<h1>Hello, <jsp:getProperty name="mybean" property="username" />!
```

```
</h1>
```

```
</td>
```

```
</tr>
```

```
</table>
```



NameHandler.java

```
package hello;

public class NameHandler {
    private String username;
    public NameHandler() {
        username = null;
    }
    public void setUsername( String name ) {
        username = name;
    }
    public String getUsername() {
        return username;
    }
}
```