

SERVLETS I

CS 434

T. Asma Alsaleh and T. Lamees Alhazzaa

Java Servlets: Study Materials

2

- Chapter 19 in Core Web Programming, 2nd edition.
 - ▣ Or free online version of 2nd edition (chapters 1 - 9):
<http://pdf.coreservlets.com/>
- Chapter 36 in Internet & World Wide Web How to Program, 3rd edition.

Outlines

3

- Overview of servlets.
- Servlets life cycle.
- Handling the client request:
 - ▣ Form data.
 - ▣ HTTP request headers.
- Generating the server response:
 - ▣ HTTP status codes.
 - ▣ HTTP response headers.
- Handling cookies.
- Session tracking.

Overview of Servlets

Why Build Web Pages dynamically?

5

- The Web page is based on data submitted by the user.
 - ▣ E.g., results page from search engines.

- The Web page is derived from data that changes frequently.
 - ▣ E.g., a weather report or news headlines page.

- The Web page uses information from databases or other server-side sources.
 - ▣ E.g., an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale.

What is a java servlet ?

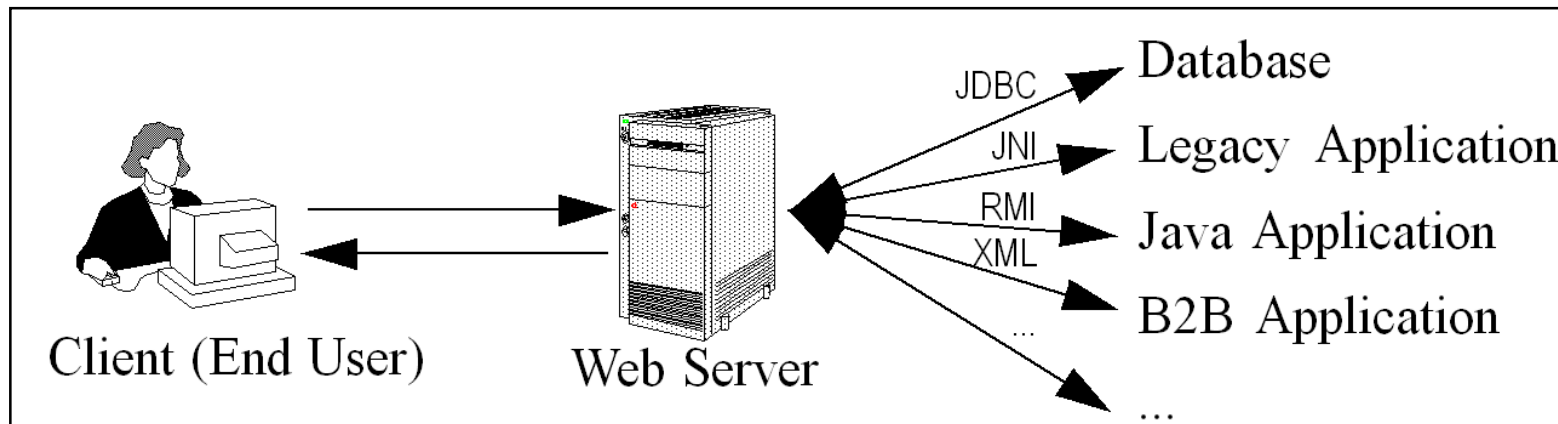
6

- A means for generating dynamic Web content.
- Defined as a Java class.
- A type of server-side program that can:
 1. Accept the input of a form.
 2. Do something useful with that input.
 3. Provide a customized, relevant response to the user who submitted the input.

What is a java servlet ?

7

- Java programs that run on a web server.
- Acting as a middle layer between a request coming from a web browser or other http client and databases or applications on the http server.



A servlet's job

8

1. Read data sent by client
 - ▣ Explicit (form data).
 - ▣ Implicit (request headers).
2. Generate the results
3. Send the data back to client
 - ▣ Explicit (HTML).
 - ▣ Implicit (status codes and response headers).

The Advantages of Servlets Over “CGI”

9

1. Efficient
 - ▣ Threads instead of OS processes, one servlet copy.
2. Convenient
 - ▣ Lots of high-level utilities.
3. Powerful
 - ▣ Sharing data, pooling, persistence.
4. Portable
 - ▣ Run on virtually all operating systems and servers.
5. Inexpensive
 - ▣ There are plenty of free and low-cost servers.

GET & POST requests

10

- GET requests are generated by browsers when:
 - User types in a URL on the address line,
 - Follows a link from a Web page, or
 - Submits an HTML form that either does not specify a METHOD or specifies METHOD="GET".
- With GET request User data is sent as part of the request URL.
- POST requests made by browsers when:
 - Someone submits an HTML form that specifies METHOD="POST".

Basic Servlet Structure

11

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

public class SomeServlet extends HttpServlet
{
    // Override methods that handle get & post requests

    public void doGet(HttpServletRequest request, HttpServletResponse response )
        throws ServletException, java.io.IOException
    {
        ...
    }

    public void doPost(HttpServletRequest request, HttpServletResponse response )
        throws ServletException, java.io.IOException
    {
        // usually call doGet method
        doGet( request, response );
    }
}
```

Import the required java packages

Your servlet name

Override method **doGet** to provide custom get request processing.

Override method **doPost** to provide custom post request processing.

Basic Servlet Structure

12

- No main(), instead doGet or doPost or both.
- You have to import classes in:
 - ▣ java.io (for PrintWriter, etc.).
 - ▣ javax.servlet (for HttpServlet, etc.).
 - ▣ javax.servlet.http (for HttpServletRequest and HttpServletResponse).

Basic Servlet Structure

13

- Servlet container (servlet engine).
 - ▣ Server that executes a servlet.
 - ▣ E.g. Tomcat.

- Servlet instances are created (invoked) by servlet container automatically when requested.
 - ▣ E.g. by POST and GET requests.

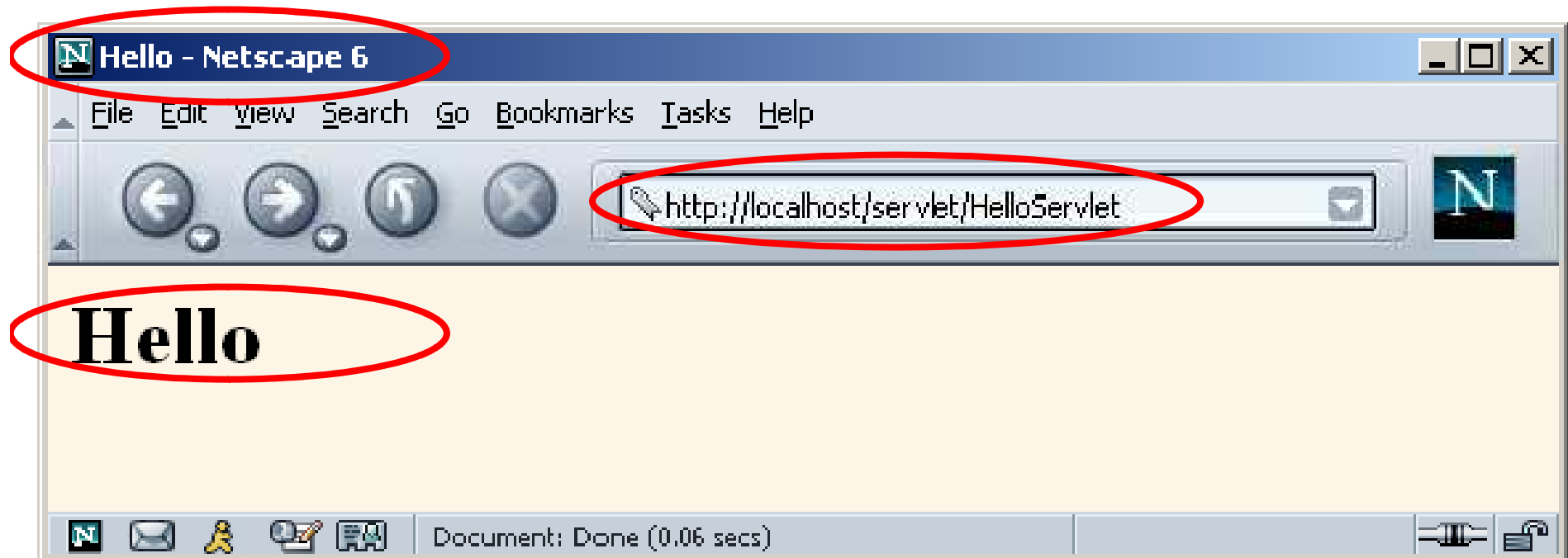
Basic Servlet Structure

14

- ❑ To be a servlet, a class should extend HttpServlet and override doGet or doPost (or both).
- ❑ These methods take two arguments:
 - ❑ The **HttpServletRequest** has methods that let you find out about incoming information such as FORM data, HTTP request headers.
 - ❑ The **HttpServletResponse** has methods that lets you specify the HTTP response line (200, 404, etc.), response headers (Content-Type, etc.), and lets you obtain a PrintWriter used to send output back to the client.

Example I: Generate HTML

15



Example I: Generate HTML

16

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

public class HelloServlet extends HttpServlet
{
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        String docType = "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \"+
            \"Transitional//EN\">\n";

        out.println( docType +
            "<HTML>\n" +
            "<HEAD><TITLE>Hello</TITLE></HEAD>\n"+
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1>Hello</H1>\n" +
            "</BODY></HTML>" );
    }
}
```

Uses the response object's **getWriter** method to obtain a reference to the **PrintWriter** object that enables the servlet to send content to the client.

Uses the response object's **setContentType** method to specify the content type of the data to be sent as the response to the client

Create the XHTML document by writing strings with the **out** object's **println** method.

Example I: Generate HTML

17

- You should set the Content-Type response header via `setContentType` method.
 - ▣ Note that you need to set response headers *before* actually returning any of the content via the `PrintWriter`.

Using Utility file

18

- It is a bit cumbersome to generate HTML with `println` statements
- The real solution is to use Java Server Pages (JSP), which will be discussed later
- There are two parts of the Web page that are unlikely to change
 - ▣ (DOCTYPE and HEAD)
- We could benefit from incorporating them into a simple utility file

Simple Utility file

19

```
public class ServletUtilities {

    public static final String DOCTYPE =
        "<!DOCTYPE HTML PUBLIC \"-//W3C//DTD HTML 4.0 \" +
        \"Transitional//EN\">";

    public static String headWithTitle(String title)
    {
        return(DOCTYPE + "\n" +
            "<HTML>\n" +
            "<HEAD><TITLE>" + title +
            "</TITLE></HEAD>\n");
    }
    ...
}
```

Use utility file in servlet

20

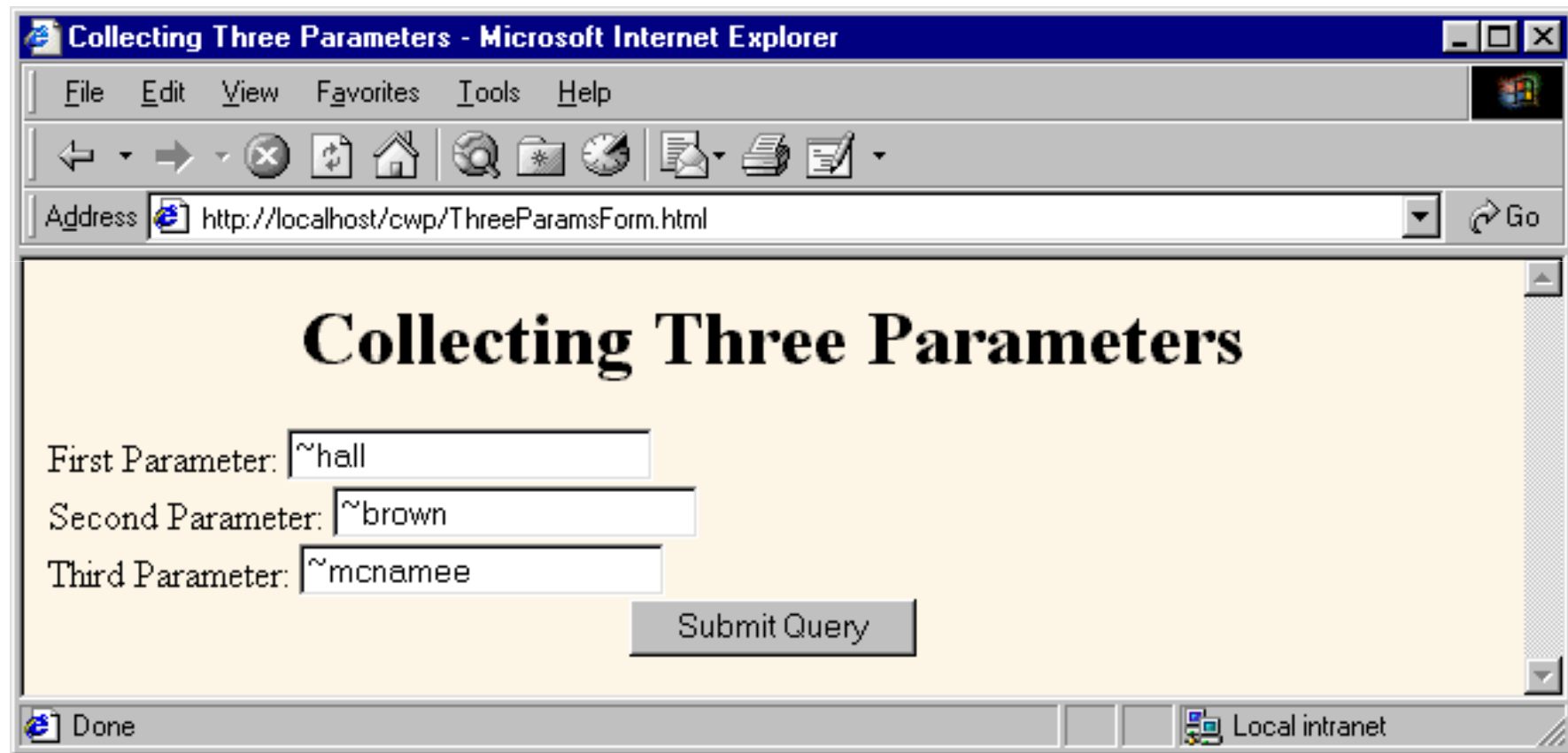
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class HelloServlet3 extends HttpServlet {
    public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException
    {
        response.setContentType("text/html");
        PrintWriter out = response.getWriter();
        String title = "Hello (3)";
        out.println(ServletUtilities.headWithTitle(title)+
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1>" + title + "</H1>\n" +
            "</BODY></HTML>");
    }
}
```

Handling the Client Request Form Data

Request Form Data: Example I

22



The screenshot shows a Microsoft Internet Explorer window titled "Collecting Three Parameters - Microsoft Internet Explorer". The address bar displays "http://localhost/cwp/ThreeParamsForm.html". The main content area has a yellow background and contains the title "Collecting Three Parameters" in a large, bold, black serif font. Below the title are three text input fields, each preceded by a label: "First Parameter:" with the value "~hall", "Second Parameter:" with the value "~brown", and "Third Parameter:" with the value "~mcnamee". A "Submit Query" button is located below the third input field. The status bar at the bottom shows "Done" and "Local intranet".

Collecting Three Parameters

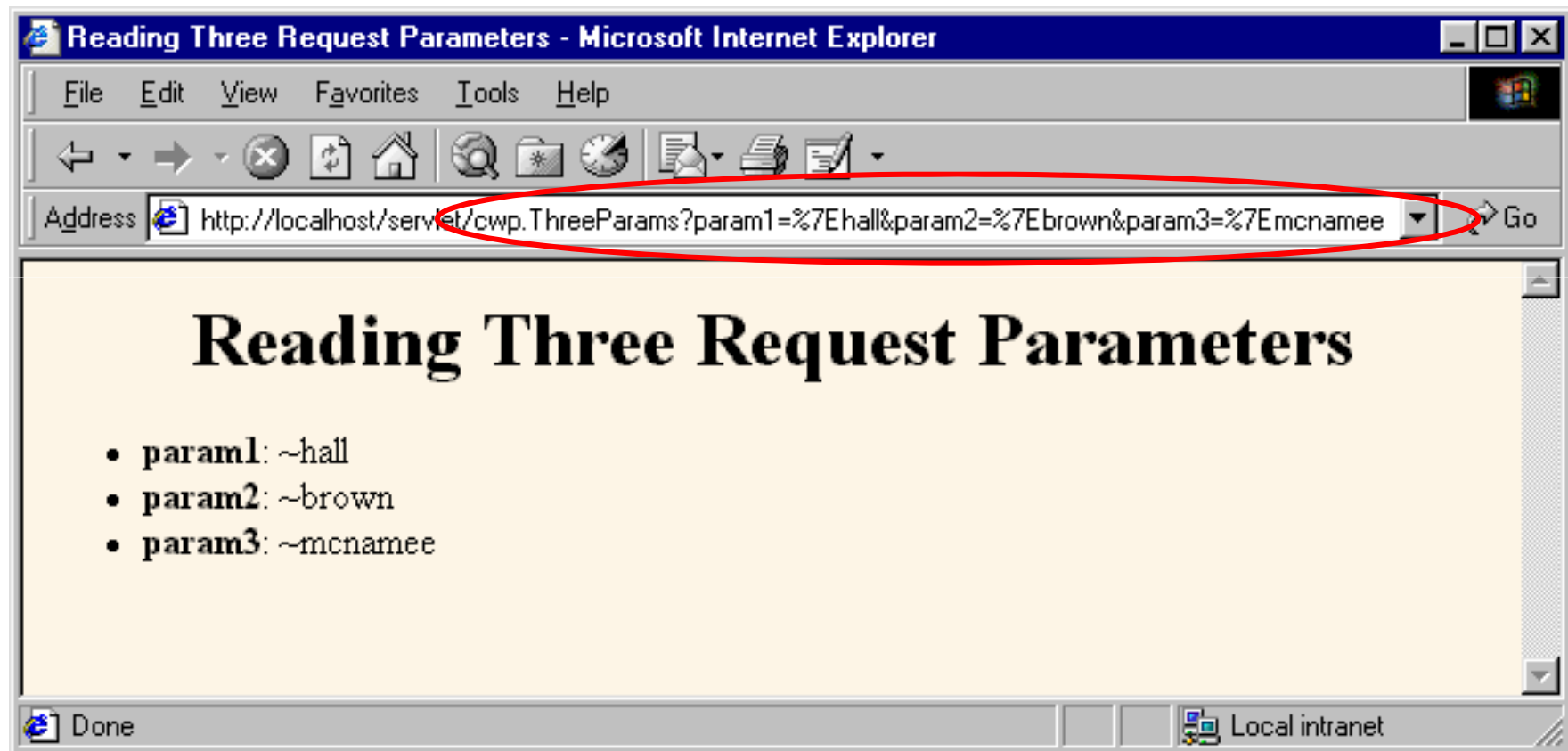
First Parameter:

Second Parameter:

Third Parameter:

Request Form Data: Example I

23



Request Form Data: Example I

24

```
<form action="/servlet/cwp.ThreeParams" method="get">
```

```
    First Parameter:  <input type="text" name="param1"> <br />
```

```
    Second Parameter: <input type="text" name="param2"> <br />
```

```
    Third Parameter:  <input type="text" name="param3"> <br />
```

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

```
</form>
```

Request Form Data: Example I

25

```
public class ThreeParams extends HttpServlet {

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

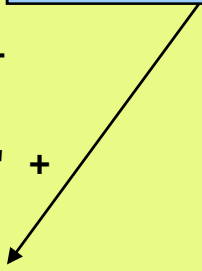
        response.setContentType("text/html");

        PrintWriter out = response.getWriter();

        String title = "Reading Three Request Parameters";

        out.println(ServletUtilities.headWithTitle(title) +
            "<BODY BGCOLOR=\"#FDF5E6\">\n" +
            "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
            "<UL>\n" +
            "    <LI><B>param1</B>: "
            + request.getParameter("param1") + "\n" +
            "    <LI><B>param2</B>: "
            + request.getParameter("param2") + "\n" +
            "    <LI><B>param3</B>: "
            + request.getParameter("param3") + "\n" +
            "</UL>\n" +
            "</BODY></HTML>"); }}
```

Obtains the parameters from the request.



Reading Form Data

26

- Form data can be
 - ▣ Attached to the end of the URL after a question mark for GET requests.
 - ▣ Sent to the server on a separate line for POST requests.
- You simply call the **getParameter** method of the **HttpServletRequest**, supplying the case-sensitive parameter name as an argument.
 - ▣ ("Param1") <> ("param1").

Reading Form Data:

Methods for reading form data

27

Method	Description
String getParameter(String name)	•Obtains the value of a parameter sent to the servlet as part of a get or post request. The name argument represents the parameter name. •An empty String is returned if the parameter exists but has no value. •null is returned if there was no such parameter.
Enumeration getParameterNames()	•Returns the names of all the parameters sent to the servlet as part of a request. •Returns Enumeration of request params (i.e. a full list).
String[] getParameterValues(String name)	•For a parameter with multiple values, this method returns an array of strings containing the values for a specified servlet parameter. •Returns a one-element array if param not repeated. •Returns null if no such parameter is in query.

Reading Form Data

28

- Examples:

- String p1 = **request.getParameter** ("p1");

- String p2 = **request.getParameter** ("p2");

- String p[] = **request.getParameterValues** ("p1");

- All parameter values are String type

- String** p1 = request.getParameter("p1");

- What if numbers are needed?

- ▣ Integer.parseInt("123")

- ▣ Double.parseDouble("123.5")

Request Form Data: Example II

(self-study)

29

- ❑ Do not count on `getParameterNames` returning the names in any particular order.
- ❑ See chapter 4, section 4.4 in the on-line version of Core Web Programming (Example: Reading All Parameters).

Filtering Query Data

30

- It is not safe to insert the request parameters directly into the page being generated.
- Why??
 - ▣ the request parameters might contain HTML characters such as `<`, `>`, `"`, and `&` that could cause some of the subsequent tags to be interpreted incorrectly.
- A safer approach
 - ▣ Filter out the HTML-specific characters before inserting the values into the page
- Add a static `filter` method in the `ServletUtilities` class

ServerUtilites

31

```
public class ServletUtilities {  
    public static String filter(String input)  
    {  
        if (!hasSpecialChars(input)) {  
            return(input);  
        }  
        StringBuffer filtered = new StringBuffer(input.length());  
        char c;  
        for(int i=0; i<input.length(); i++)  
        {  
            c = input.charAt(i);  
            switch(c) {  
                case '<': filtered.append("&lt;"); break;  
                case '>': filtered.append("&gt;"); break;  
                case '"': filtered.append("&quot;"); break;  
                case '&': filtered.append("&amp;"); break;  
                default: filtered.append(c);  
            }  
        }  
        return(filtered.toString());  
    }  
}
```

Use StringBuffer not
String If you so string
concatenation repeatedly

Checking for Missing and Malformed Data

32

- Using JavaScript for client-side validation does not remove the need for also doing this type of checking on the server.

- E.g. user can:
 - ▣ Disable JavaScript.
 - ▣ Manually edit URLs contain GET data.

Checking for Missing and Malformed Data

33

□ Missing

- ▣ Field missing in form (i.e. no element in the form with this name)
 - `getParameter` returns null
- ▣ Field blank when form submitted (e.g. text field is empty)
 - `getParameter` returns an empty string (or possibly a string with whitespace in it: use `trim()`)
- ▣ Must check for null before checking for empty string:

```
String param = request.getParameter("someName");

if ((param == null) || (param.trim().equals("")))
{
    doSomethingForMissingValues(...);
}
else
{
    doSomethingWithParameter(param);
}
```

Checking for Missing and Malformed Data

34

❑ Malformed

- ❑ Value is a nonempty string in the wrong format.
- ❑ E.g. a text field must have a numeric value, exactly 7 characters, single letters.
- ❑ E.g. Should be a number:

```
private int getAge(String ageString)
{
    try
    {
        return (Integer.parseInt(ageString));
    }
    catch
    {
        return (defaultAge);
    }
}
```

Other Methods of HttpServlet Class

35

Method	Description
doDelete	Called in response to an HTTP delete request. Such a request is normally used to delete a file from a server. This may not be available on some servers, because of its inherent security risks (e.g., the client could delete a file that is critical to the execution of the server or an application).
doHead	Called in response to an HTTP head request. Such a request is normally used when the client only wants the headers of a response, such as the content type and content length of the response.
doOptions	Called in response to an HTTP options request. This returns information to the client indicating the HTTP options supported by the server, such as the version of HTTP (1.0 or 1.1) and the request methods the server supports.
doPut	Called in response to an HTTP put request. Such a request is normally used to store a file on the server. This may not be available on some servers, because of its inherent security risks (e.g., the client could place an executable application on the server, which, if executed, could damage the server—perhaps by deleting critical files or occupying resources).
doTrace	Called in response to an HTTP trace request. Such a request is normally used for debugging. The implementation of this method automatically returns an HTML document to the client containing the request header information (data sent by the browser as part of the request).

Some Methods of HttpServletRequest interface

36

Method	Description
<code>String getParameter(String name)</code>	Obtains the value of a parameter sent to the servlet as part of a get or post request. The name argument represents the parameter name.
<code>Enumeration getParameterNames()</code>	Returns the names of all the parameters sent to the servlet as part of a post request.
<code>String[] getParameterValues(String name)</code>	For a parameter with multiple values, this method returns an array of strings containing the values for a specified servlet parameter.
<code>Cookie[] getCookies()</code>	Returns an array of Cookie objects stored on the client by the server. Cookie objects can be used to uniquely identify clients to the servlet.
<code>HttpSession getSession(boolean create)</code>	Returns an HttpSession object associated with the client's current browsing session. This method can create an HttpSession object (true argument) if one does not already exist for the client. HttpSession objects are used in similar ways to Cookies for uniquely identifying clients.

Some Methods of HttpServletResponse interface

37

Method	Description
<code>void addCookie(Cookie cookie)</code>	Used to add a Cookie to the header of the response to the client. The Cookie 's maximum age and whether Cookies are enabled on the client determine if Cookies are stored on the client.
<code>ServletOutputStream getOutputStream()</code>	Obtains a byte-based output stream for sending binary data to the client.
<code>PrintWriter getWriter()</code>	Obtains a character-based output stream for sending text data to the client.
<code>void setContentType(String type)</code>	Specifies the MIME type of the response to the browser. The MIME type helps the browser determine how to display the data (or possibly what other application to execute to process the data). For example, MIME type "text/html" indicates that the response is an HTML document, so the browser displays the HTML page.

Other HTTP Methods

38

HEAD	like GET but just returns header info. Useful for checking last-modified info for caching purposes, size of doc before downloading, ...
PUT	place documents on the server (not widely supported)
DELETE	remove docs from server (not widely supported)
TRACE	returns to client exact contents of its request (used for debugging)
CONNECT	
OPTIONS	asks server which methods it supports

References

39

- Material for this lecture was obtained from:
 - ▣ "Internet & World Wide Web How to Program"; Deitel, Deitel & Nieto; Prentice Hall, 3rd edition.
 - ▣ The core web programming, by Marty Hall and Larry Brown, A Sun Microsystems Press/Prentice Hall PTR Book.