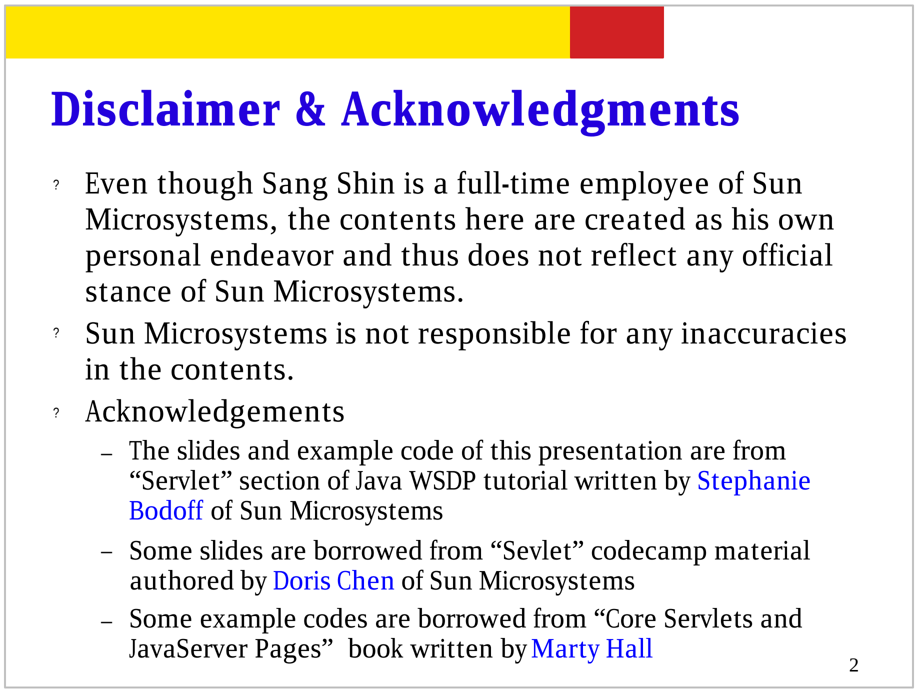
Servlet Basics

1

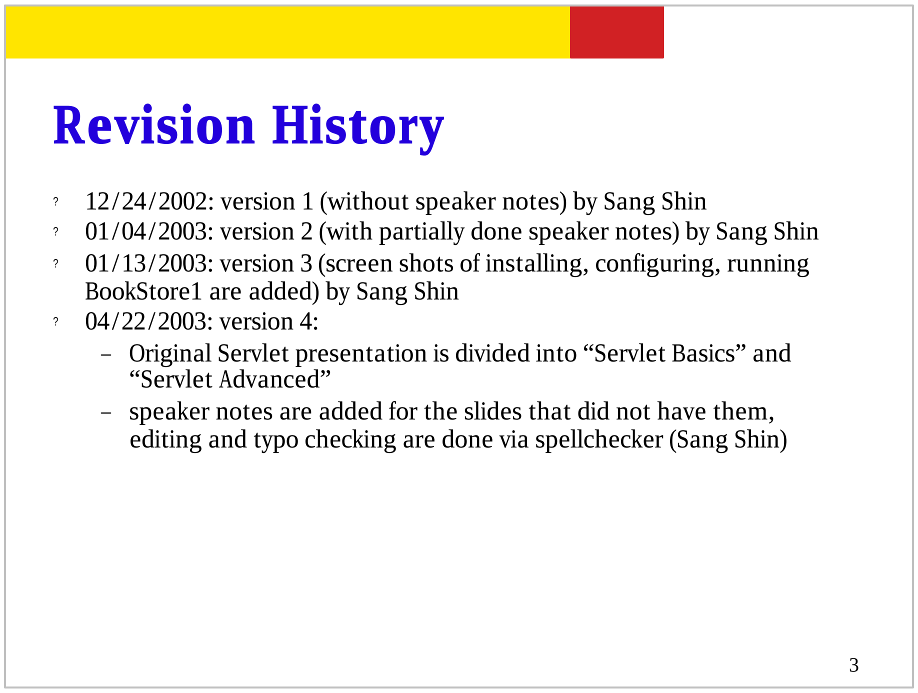
In this session, I will talk about basic concepts of a Servlet. And during the next class, we will talk about more advanced topics of a Servlet.



Disclaimer & Acknowledgments

- Even though Sang Shin is a full-time employee of Sun Microsystems, the contents here are created as his own personal endeavor and thus does not reflect any official stance of Sun Microsystems.
- Sun Microsystems is not responsible for any inaccuracies in the contents.
- Acknowledgements
  – The slides and example code of this presentation are from "Servlet" section of Java WSDP tutorial written by Stephanie Bodoff of Sun Microsystems
  – Some slides are borrowed from "Sevlet" codecamp material authored by Doris Chen of Sun Microsystems
  – Some example codes are borrowed from "Core Servlets and JavaServer Pages" book written by Marty Hall

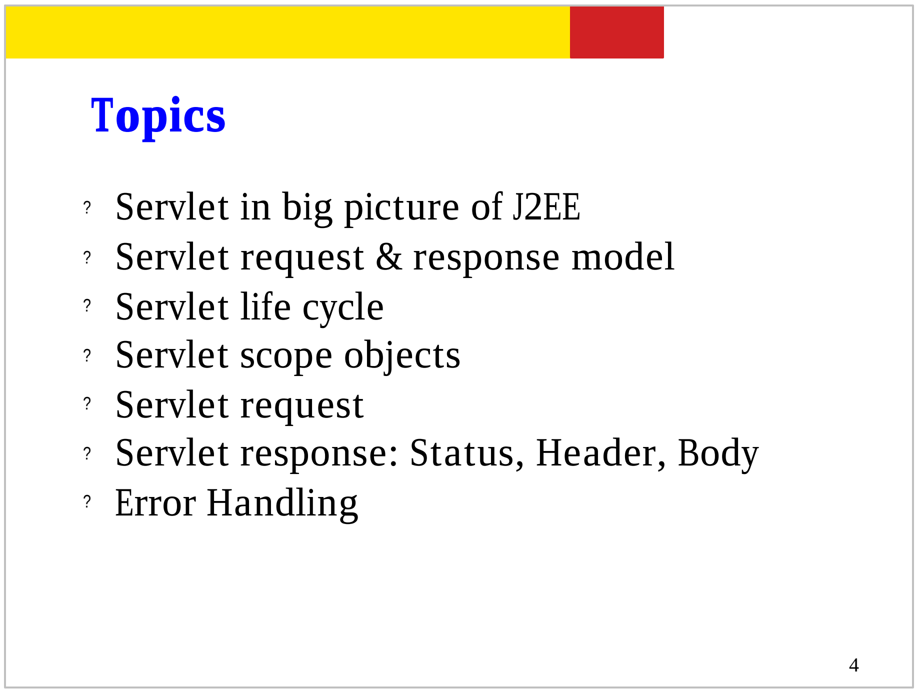
2



Revision History

- 12/24/2002: version 1 (without speaker notes) by Sang Shin
- 01/04/2003: version 2 (with partially done speaker notes) by Sang Shin
- 01/13/2003: version 3 (screen shots of installing, configuring, running BookStore1 are added) by Sang Shin
- 04/22/2003: version 4:
  - Original Servlet presentation is divided into “Servlet Basics” and “Servlet Advanced”
  - speaker notes are added for the slides that did not have them, editing and typo checking are done via spellchecker (Sang Shin)

3



Topics

- Servlet in big picture of J2EE
- Servlet request & response model
- Servlet life cycle
- Servlet scope objects
- Servlet request
- Servlet response: Status, Header, Body
- Error Handling

4

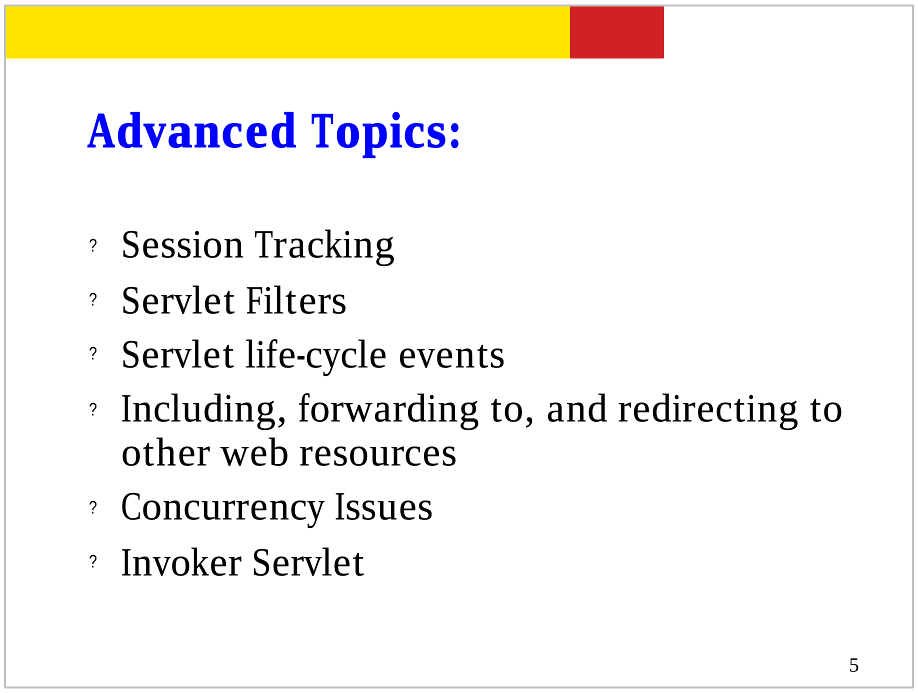
So what are we going to talk about in this session? First, we will take a look at Servlet from the standpoint of J2EE architecture, that is, what role Servlet plays in a multi-tier web-based application. We will also compare Servlet against JSP.

Next we will take a look at the “request and response” model of Servlet. Servlet is basically a web technology in which HTTP request is being received and handled and then proper HTTP response is being created and then returned to the client.

Then we will look into Servlet life-cycle, that is, how an instance of Servlet gets created to serve incoming HTTP requests. We will then look into so called “scope objects” which are system objects that can be used to store system and application specific information.

We will then take a look into the internal structure of the servlet request and servlet response, especially HTTP request and HTTP response. We will then take a look at how error handling is done.

The advanced servlet topics such as session tracking and servlet filtering will be dealt with in advanced servlet session later on.



Advanced Topics:

- Session Tracking
- Servlet Filters
- Servlet life-cycle events
- Including, forwarding to, and redirecting to other web resources
- Concurrency Issues
- Invoker Servlet

5

These are advanced topics which will be dealt with in advanced servlet session.

Servlet in a Big Picture of J2EE

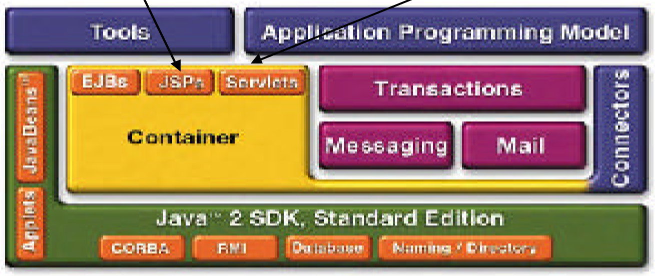
6

Now let's take a look at where Servlet fits in in the big picture of J2EE.

J2EE 1.2 Architecture

An extensible Web technology that uses template data, custom elements, scripting languages, and server-side Java objects to **return dynamic content to a client**. Typically the template data is HTML or XML elements. The client is often a **Web browser**

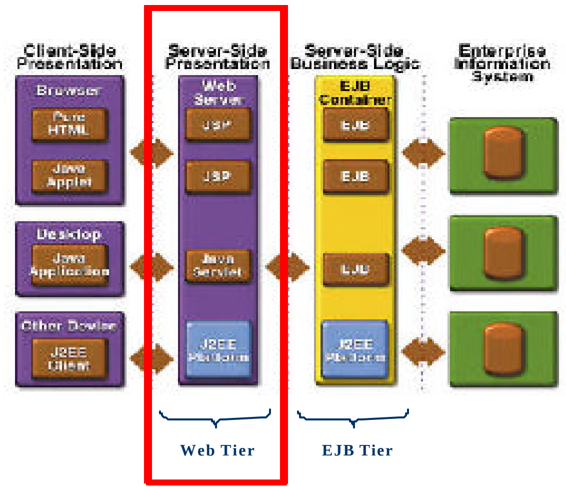
Java Servlet A Java program that extends the functionality of a Web server, generating dynamic content and interacting with Web clients using a **request-response paradigm**



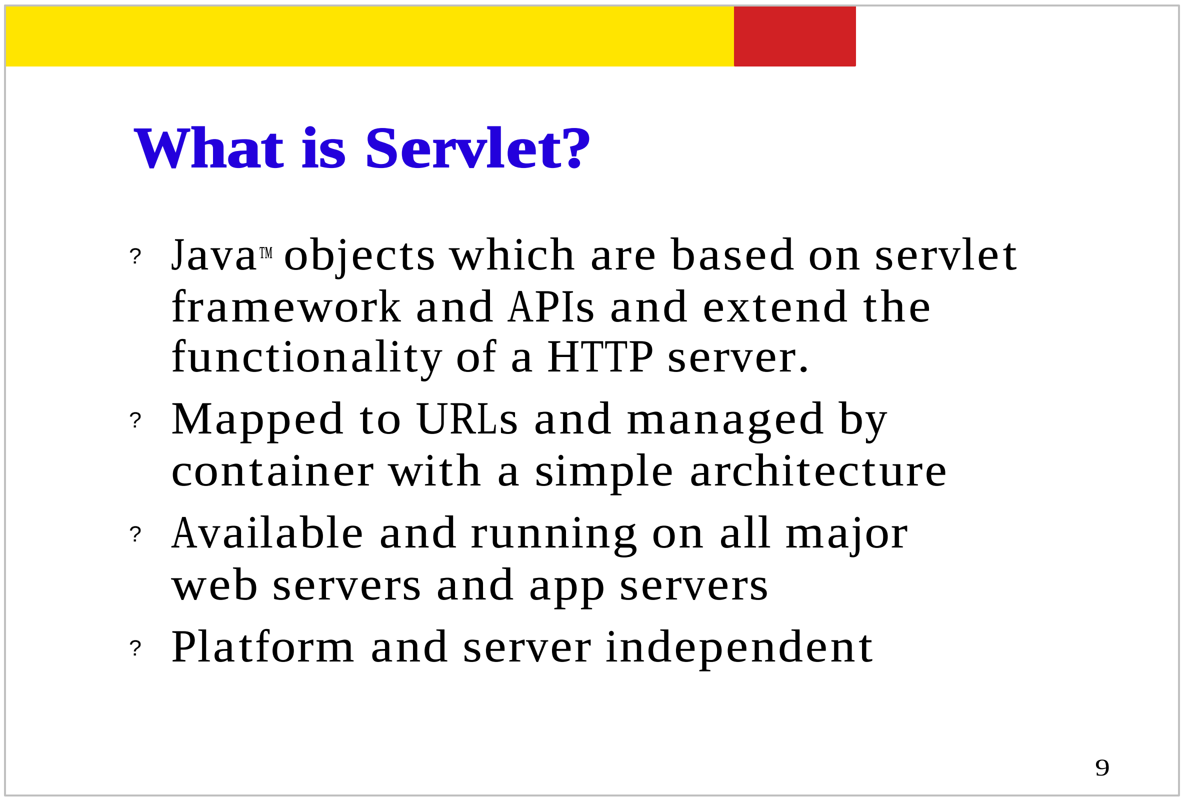
7

This picture describes the roles that Servlet and JSP play in the J2EE architecture. (Please read the text above.)

Where are Servlet and JSP?



This is the same picture we've seen in the “J2EE overview” session. As you can see, the Servlet and JSP are web tier components that are running within a web-tier container. The role that web components play are basically (1) receiving client requests that are coming in the form of HTTP requests and then (2) perform dynamic contents generation or performing business logic by themselves or delegating it to the EJB tier components, and then (3) return responses to the clients.



What is Servlet?

? Java™ objects which are based on servlet framework and APIs and extend the functionality of a HTTP server.

? Mapped to URLs and managed by container with a simple architecture

? Available and running on all major web servers and app servers

? Platform and server independent

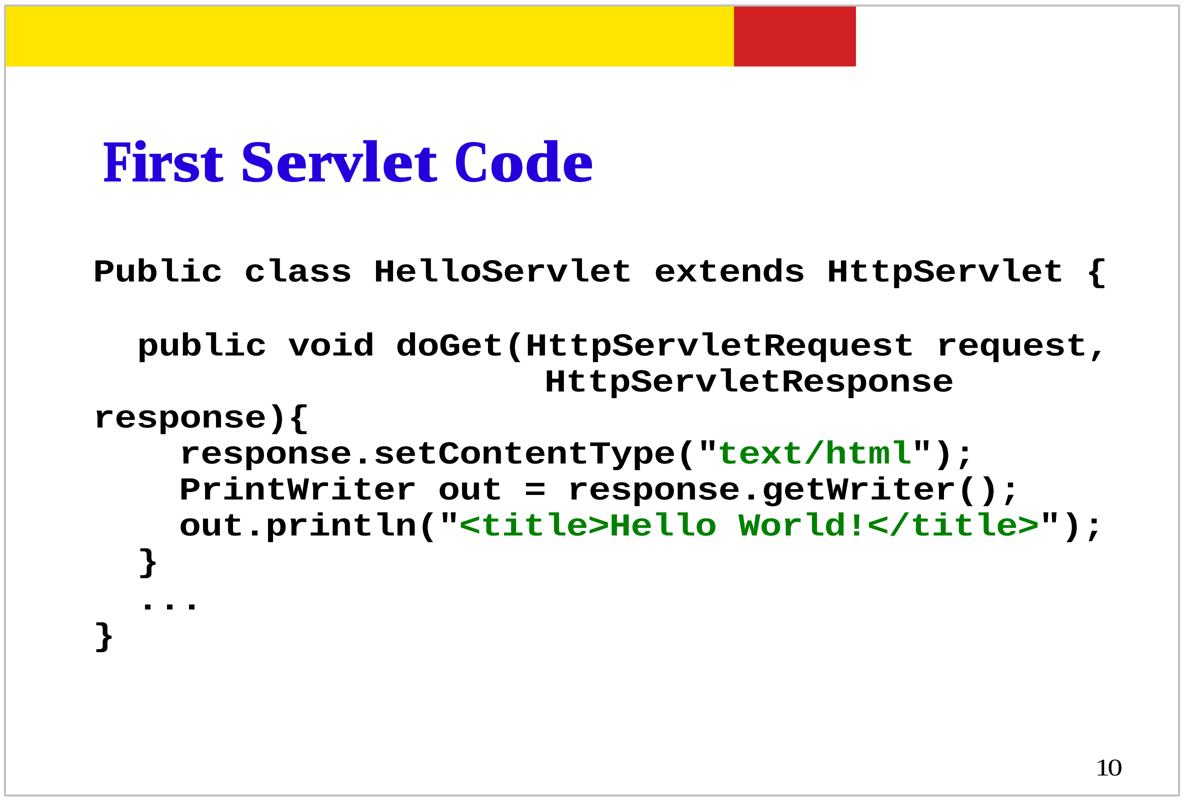
9

So what is a Servlet? A technical description of Servlet is it is a Java object that is based on Servlet framework and extends the functionality of a web server basically being responsible for creating dynamic contents.

A Servlet is mapped to a corresponding URL and its life-cycle is managed by the container. The URL is the address to which a client send HTTP request.

Servlet technology is available and running on all major web servers and app servers.

Since it is based on Java, it is platform and server independent.



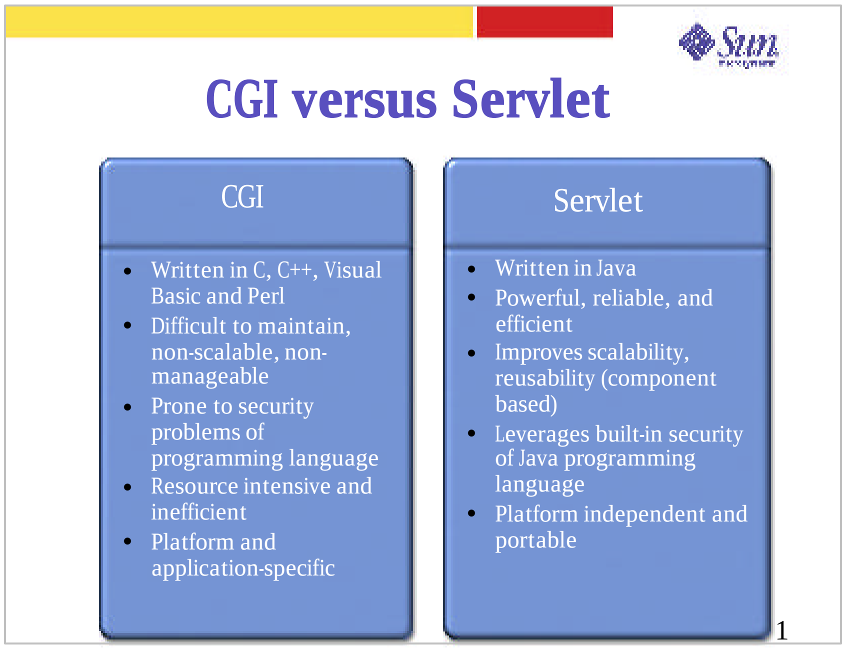
First Servlet Code

```
Public class HelloServlet extends HttpServlet {

  public void doGet(HttpServletRequest request,
                             HttpServletResponse
response){
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>Hello World!</title>");
  }
  ...
}
```

10

So this is an example of very simple Servlet code. As you can see, a Servlet is a Java code. Here a Servlet is named as HelloServlet and it extends a Java interface called HttpServlet. Inside the code, there are a few predefined methods you want to override, for example, here doGet() method gets called with HttpServletRequest object and HttpServletResponse object as parameters. The HttpServletRequest is a java object that is created by the container and captures an incoming HTTP request in an object form.

Now in this example, the handling of the request is very simple - just send back “Hello World!” message. That is, the browser, once it receives the HTTP response message that contains the “Hello World” message, will display “Hello World!” message on the screen as a result of accessing this Servlet.

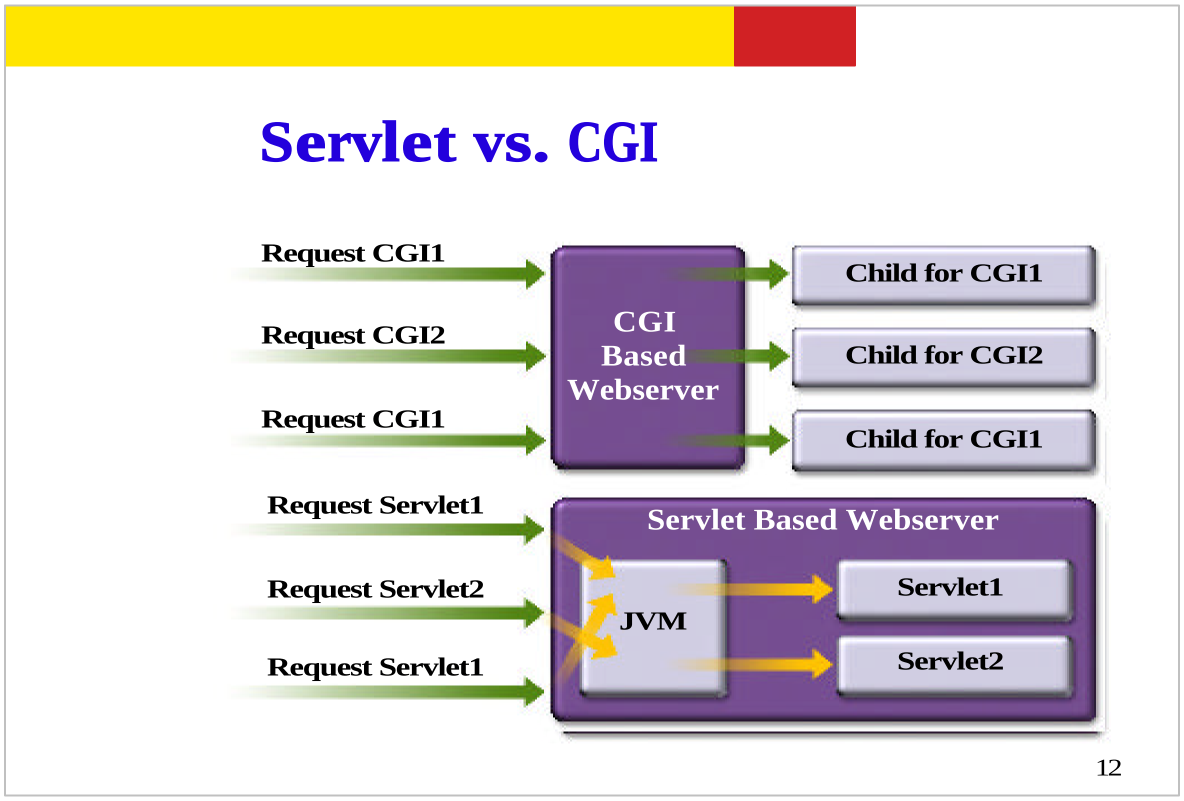


CGI versus Servlet

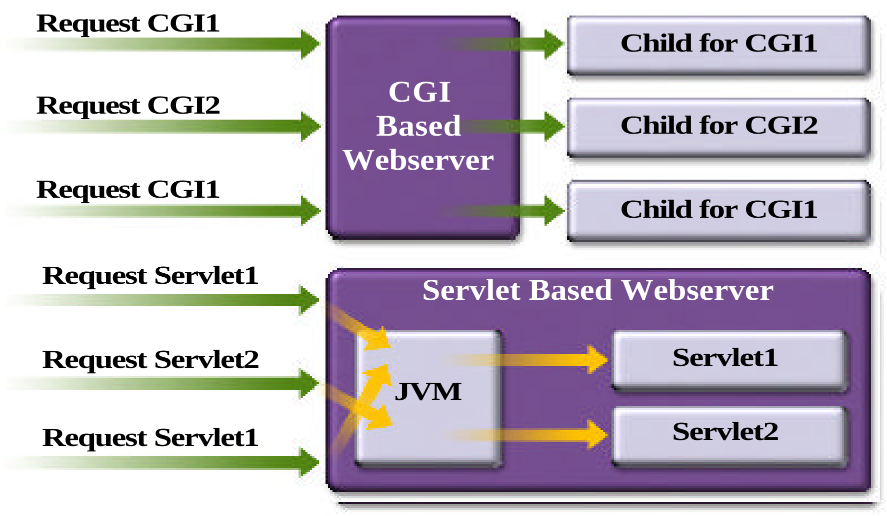
CGI	Servlet
• Written in C, C++, Visual Basic and Perl • Difficult to maintain, non-scalable, non-manageable • Prone to security problems of programming language • Resource intensive and inefficient • Platform and application-specific	• Written in Java • Powerful, reliable, and efficient • Improves scalability, reusability (component based) • Leverages built-in security of Java programming language • Platform independent and portable

1

(read the slide)



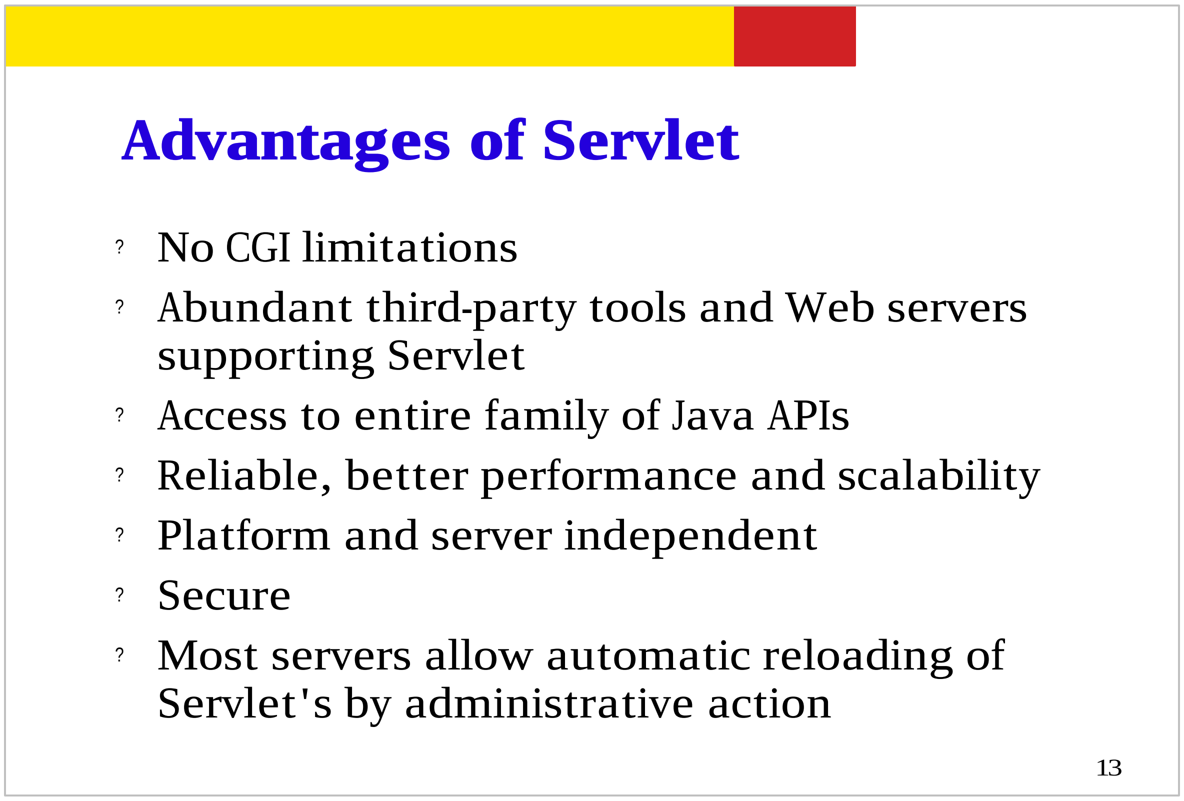
Servlet vs. CGI



12

This picture shows difference between CGI and servlet-based model. In CGI, for every HTTP request, a new process has to be created while in servlet model, it is the thread that gets created in the same Java VM (Virtual Machine) and that thread can stay there for servicing other requests.

Also in CGI, every time a new request comes, the program image of CGI has to be loaded in memory, which results in many redundant load of the same program. In the case of Servlet, a single class is loaded for serving many requests. This results in efficient memory usage.

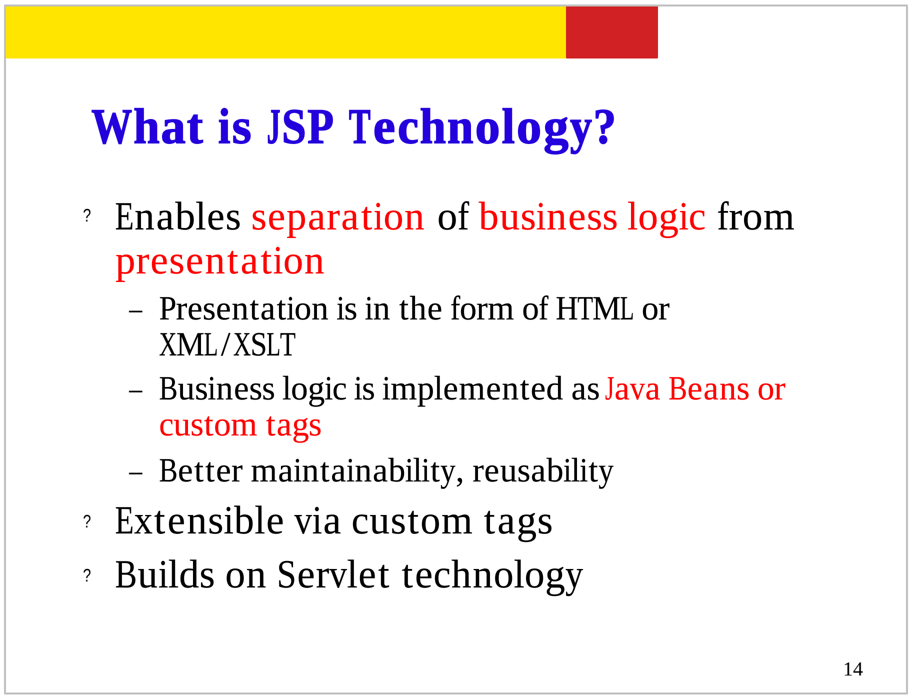


Advantages of Servlet

- No CGI limitations
- Abundant third-party tools and Web servers supporting Servlet
- Access to entire family of Java APIs
- Reliable, better performance and scalability
- Platform and server independent
- Secure
- Most servers allow automatic reloading of Servlet's by administrative action

13

So just to reiterate the advantages of Servlet, it does not have the limitations of CGI. There are abundant third-party tools and Web servers that support Servlet. Again because Servlet is Java class, it can access all the Java APIs available. It provides more reliable, better performing, and scalable web-tier technology. Because it is based in Java technology, it is platform and server independent. It provides more secure platform then CGI. Finally most servers allow automatic reloading of servlets when they are modified.



What is JSP Technology?

- Enables separation of business logic from presentation
  - Presentation is in the form of HTML or XML/XSLT
  - Business logic is implemented as Java Beans or custom tags
  - Better maintainability, reusability
- Extensible via custom tags
- Builds on Servlet technology

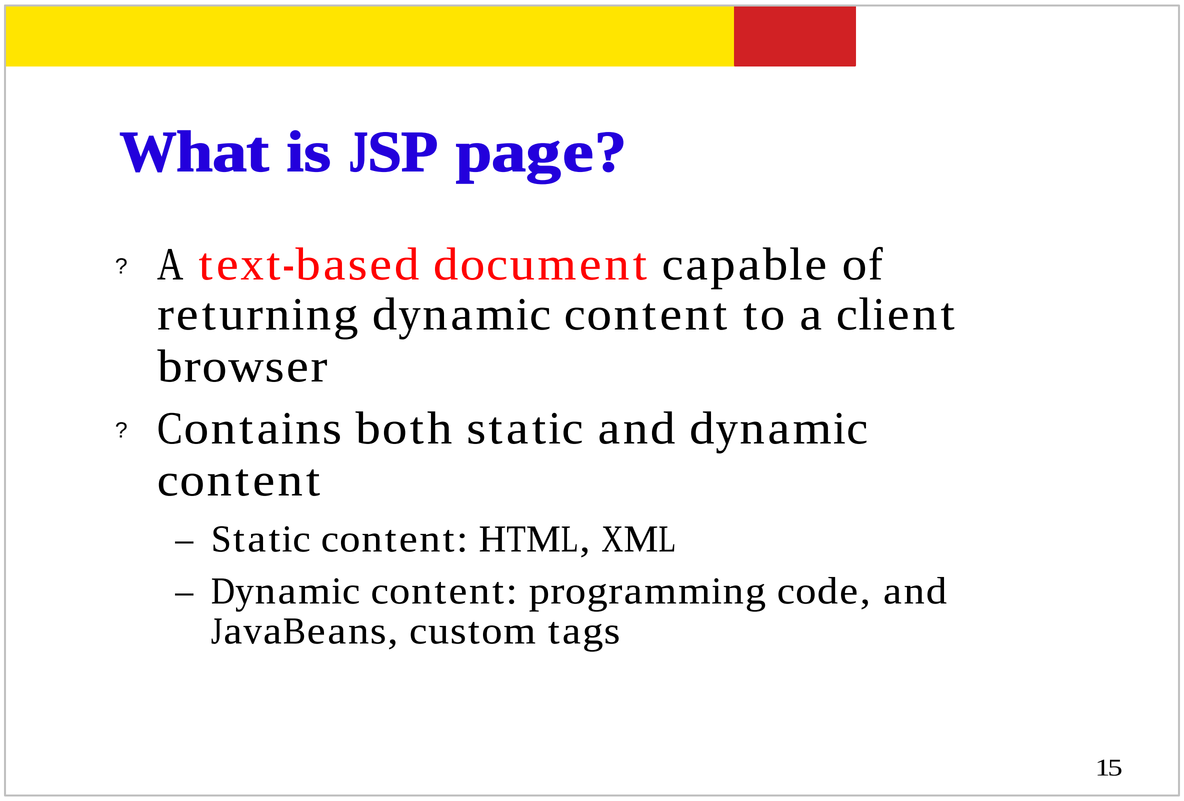
14

JSP, Java Server Pages, was introduced as a follow-on technology to the Servlet. Even though the Servlet solves many problems associated with CGI for dynamic contents generation, it has one downside. The downside is that, under Servlet, the presentation, typically HTML pages, has to be generated as part of the servlet Java code, for example, using printf statement. What this means is that whenever you have to make some change to the presentation, the Java code has to be changed and then recompiled, redeployed. This in turn result in maintenance problem of your applications. Also it makes web-page prototyping effort rather a difficult task.

JSP is designed to address of this shortcoming of the Servlet while maintaining all the benefits of Servlet. That is, it provides a clear separation between the presentation and business logic code. That is, the presentation will be designed by Web page designers in the form of either HTML or XML or JSP page while the business logic will be implemented by Java programmers either in the form of Java Beans or custom tags. This separation will result in a better maintainability of both presentation pages and business code. And because the business logic is encapsulated into Java beans or custom tags, it increased reusability of the code as well.

I mentioned about custom tags. Custom tags are basically specialized Java beans which encapsulate the application-specific business logic. The functionality of enterprise applications can be extended by building more custom tags.

Finally, JSP technology is built over servlet. In fact, JSP pages when deployed get converted into servlet first. Because it is built over servlet, it maintains all the benefits of servlet. For example, all the ready-to-use objects in a servlet such as session objects can be also available to JSP page designers and custom tag developers.

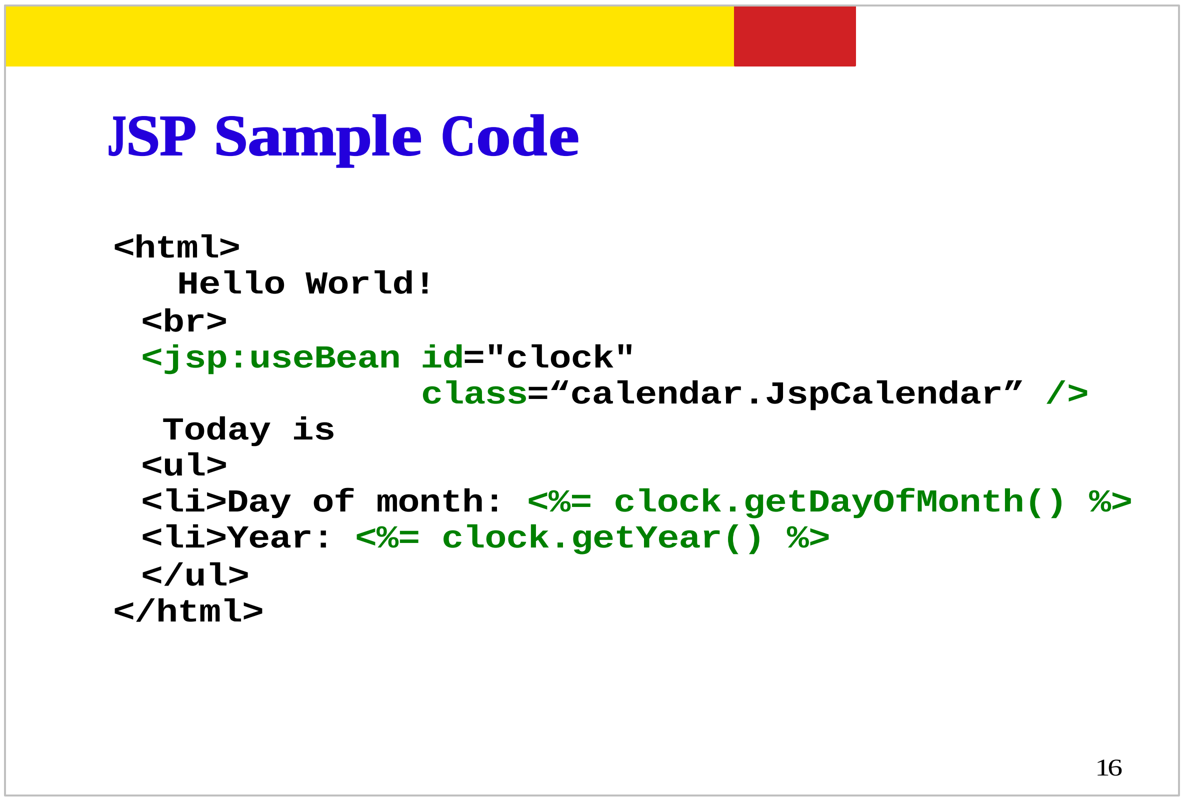


What is JSP page?

- A text-based document capable of returning dynamic content to a client browser
- Contains both static and dynamic content
  - Static content: HTML, XML
  - Dynamic content: programming code, and JavaBeans, custom tags

15

So by using JSP technology, you create JSP page. A JSP page is basically a text-based document in which both static contents and logic for dynamic contents generation are present. The static content is basically in the form of HTML or XML while the dynamic contents generation can be done via embedded programming code called scriplets or JavaBeans or custom tags. We will talk about this in detail in the JSP session.

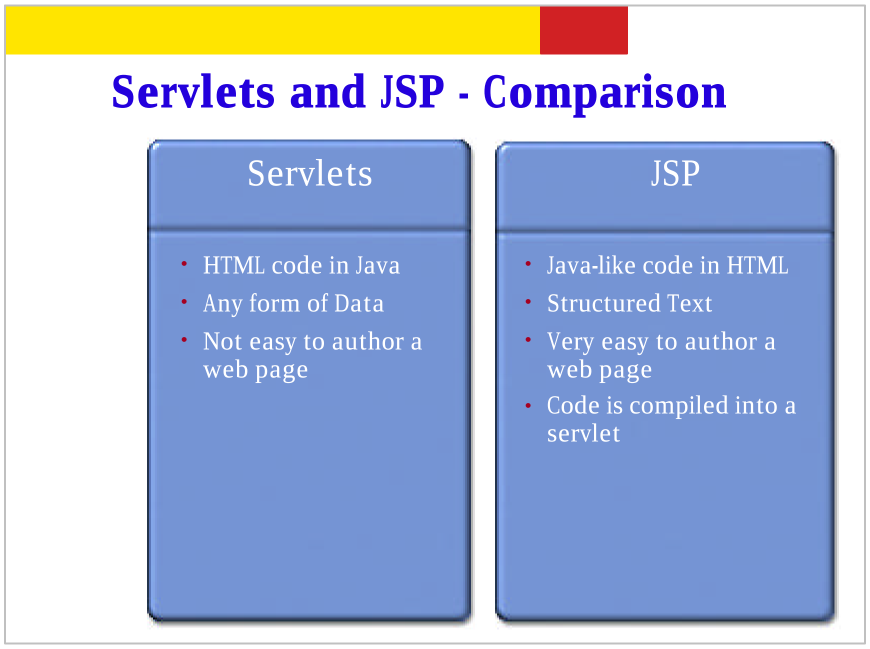


JSP Sample Code

```
<html>
   Hello World!
  <br>
  <jsp:useBean id="clock"
               class=“calendar.JspCalendar” />
   Today is
  <ul>
  <li>Day of month: <%= clock.getDayOfMonth() %>
  <li>Year: <%= clock.getYear() %>
  </ul>
</html>
```

16

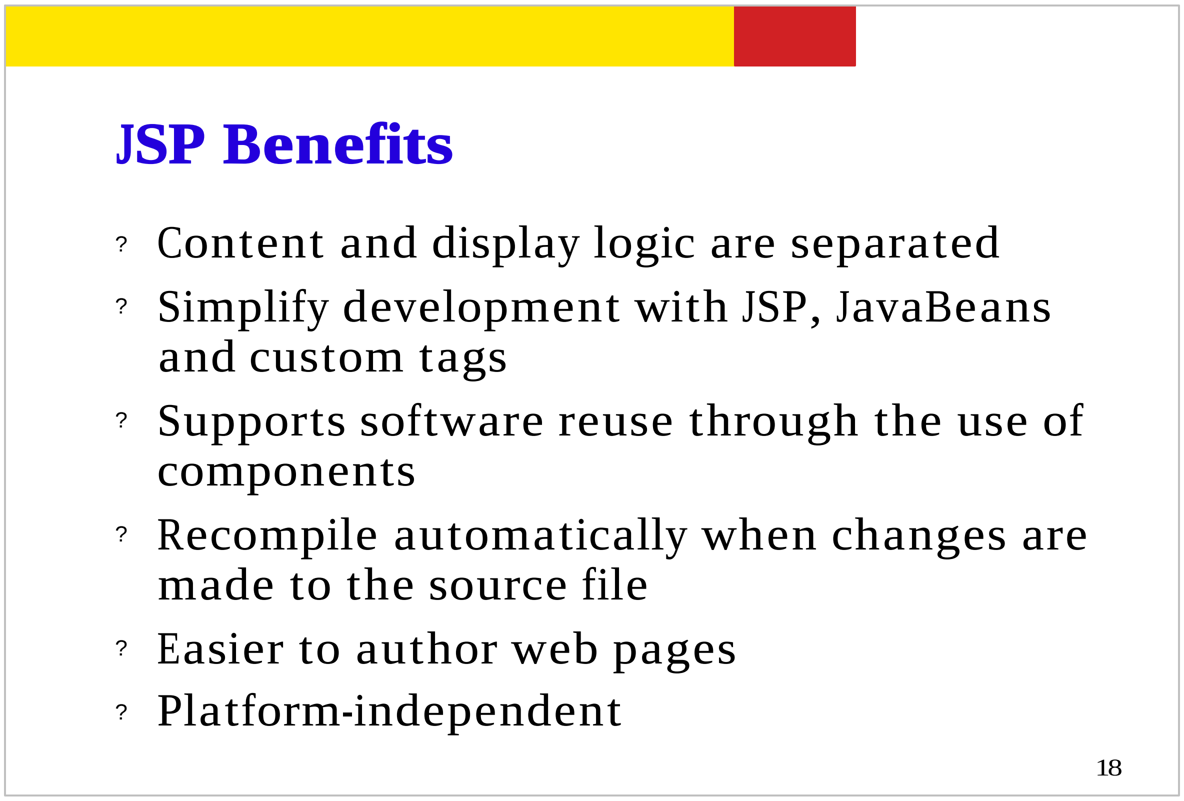
So this is a very simple JSP page. Here it contains various HTML tags as static content and displays date and year as dynamic content by using JavaBeans.



Servlets and JSP - Comparison

Servlets	JSP
• HTML code in Java • Any form of Data • Not easy to author a web page	• Java-like code in HTML • Structured Text • Very easy to author a web page • Code is compiled into a servlet

So this is a comparison of servlet and JSP. In servlet, HTML page is coded in Java while in JSP it is the reverse, that is, the Java code can be inserted in HTML like page. In servlet, any type of data can be handled while in JSP, the type of data is mainly text data. In servlet, it is not really easy to author the webpage while in JSP, it is really easy to author a webpage.



JSP Benefits

? Content and display logic are separated

? Simplify development with JSP, JavaBeans and custom tags

? Supports software reuse through the use of components

? Recompile automatically when changes are made to the source file

? Easier to author web pages

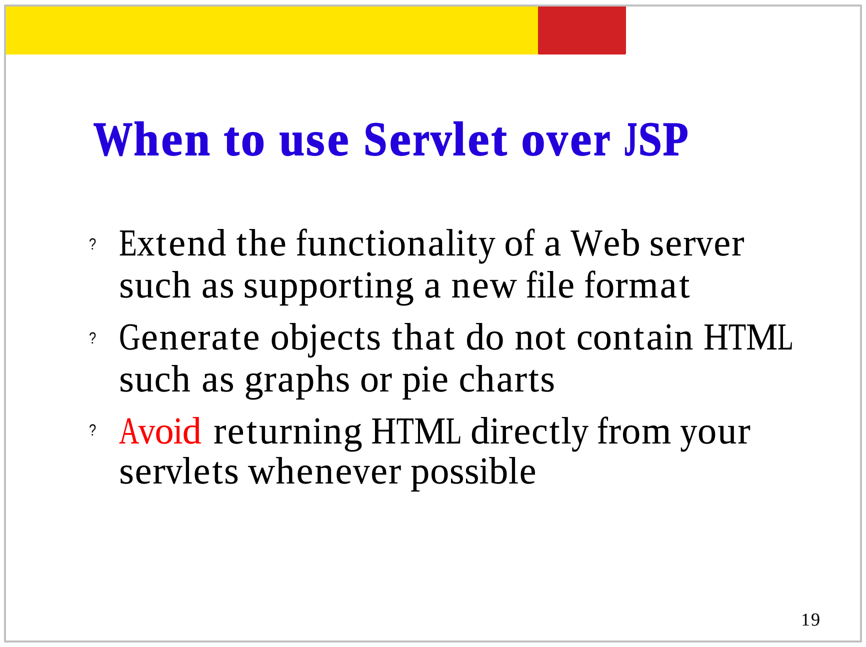
? Platform-independent

18

So the primary benefit of using JSP is that under JSP the content/business-logic and display logic are separated. This separation simplifies the development of Web application. As we will talk about in JSP session, the business logic is captured in the form of Java Beans and custom tags, which can be reused.

Another benefit of using JSP over servlet is that JSP pages are automatically compiled and deployed by the container whenever changes are made to them. So it is just a matter of putting the newly changed JSP pages to the proper directory.

Of course, because JSP pages look very similar to HTML pages, it is a lot easier for Web page designers to work with JSP pages instead of servlet. And just like servlet, JSP technology is platform independent.

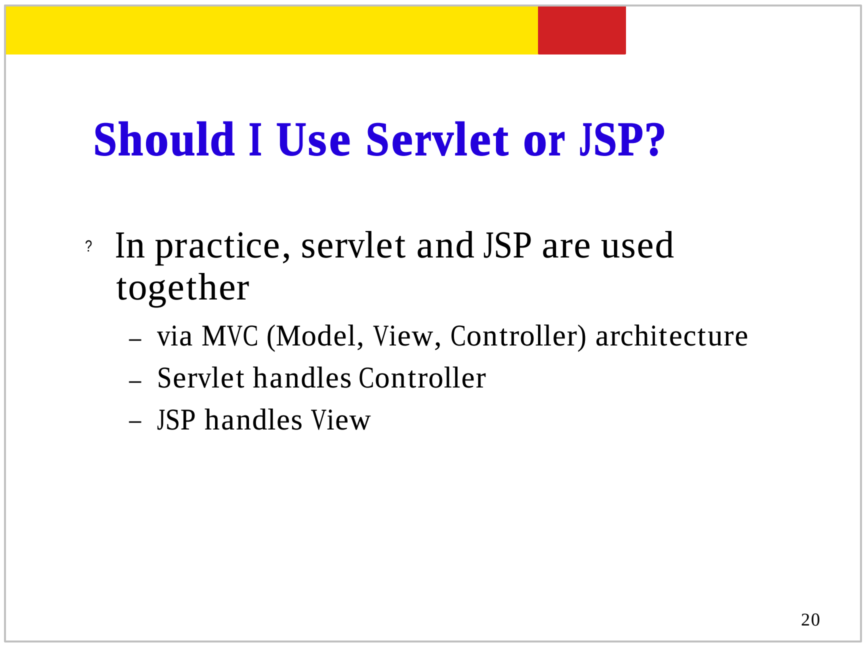


When to use Servlet over JSP

- Extend the functionality of a Web server such as supporting a new file format
- Generate objects that do not contain HTML such as graphs or pie charts
- Avoid returning HTML directly from your servlets whenever possible

19

So in most cases, you want to think about using JSP for Web tier applications. But there are cases where you want to use Servlet over JSP. (please read the slide)



Should I Use Servlet or JSP?

- In practice, servlet and JSP are used together
  - via MVC (Model, View, Controller) architecture
  - Servlet handles Controller
  - JSP handles View

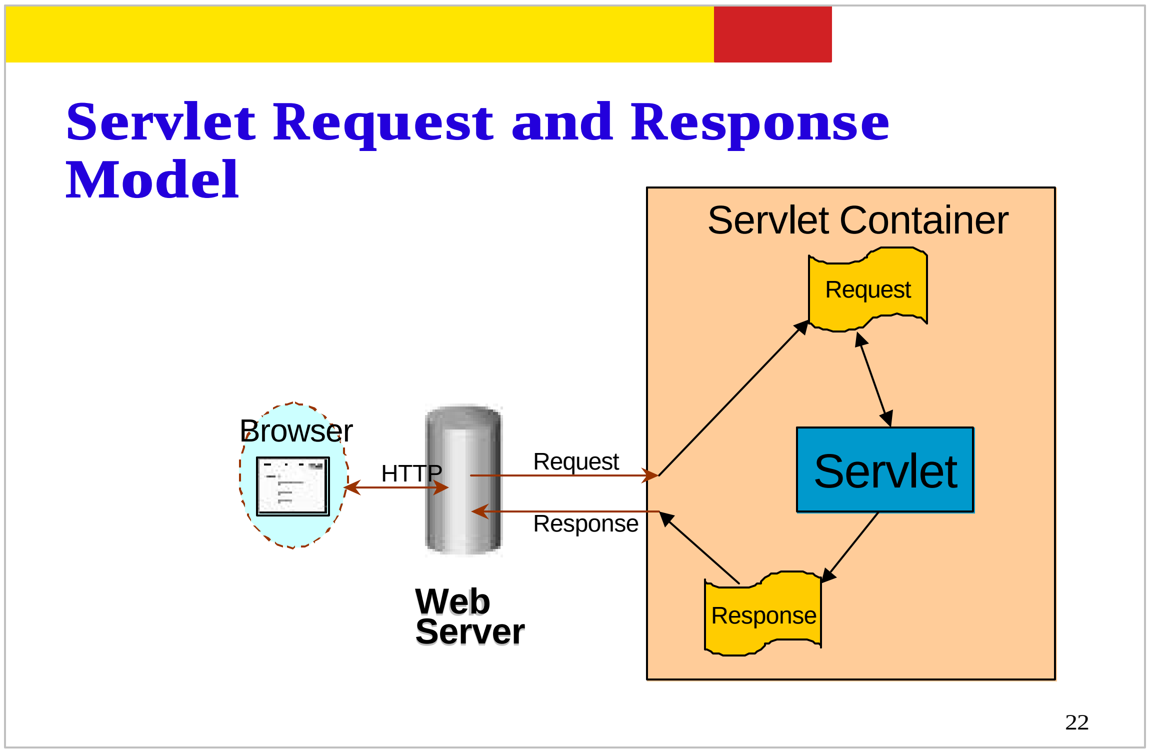
20

Now I hope I did not give you an impression that you have to use either servlet or JSP. In practice, servlet and JSP are used together levering the strength of each other. Servlet is good for controlling functionality while JSP is good for handling presentation logic. So they are used in what is called MVC architecture in which servlet is used as a controller while JSP is used for handling the view.

Servlet Request & Response Model

21

Now let's talk about servlet request and response model.

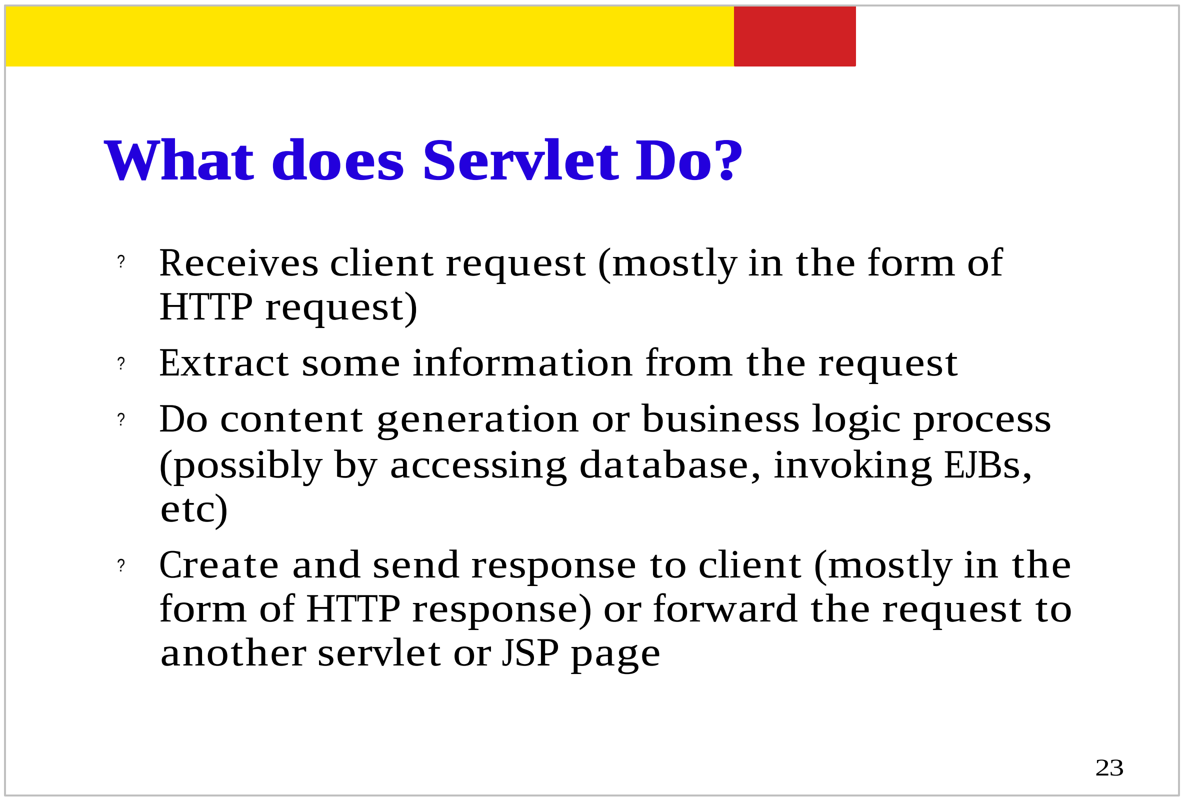


Servlet Request and Response Model

This picture shows “servlet request and response” model.

There are three different players in this picture: browser, web server, and servlet container. In many cases, a web server and a servlet container are running in a same machine even in a single virtual machine. So they are not really distinguished in many cases. Anyway, the role of the web server is to receive HTTP request and then passes it to the web container or servlet container which then creates Java objects that represent “HTTP request” and a “session” and then dispatches the request to the servlet by invoking service() method defined in the servlet.

And once the servlet handles the request, it creates a HTTP response, which is then sent to the client through the web server.



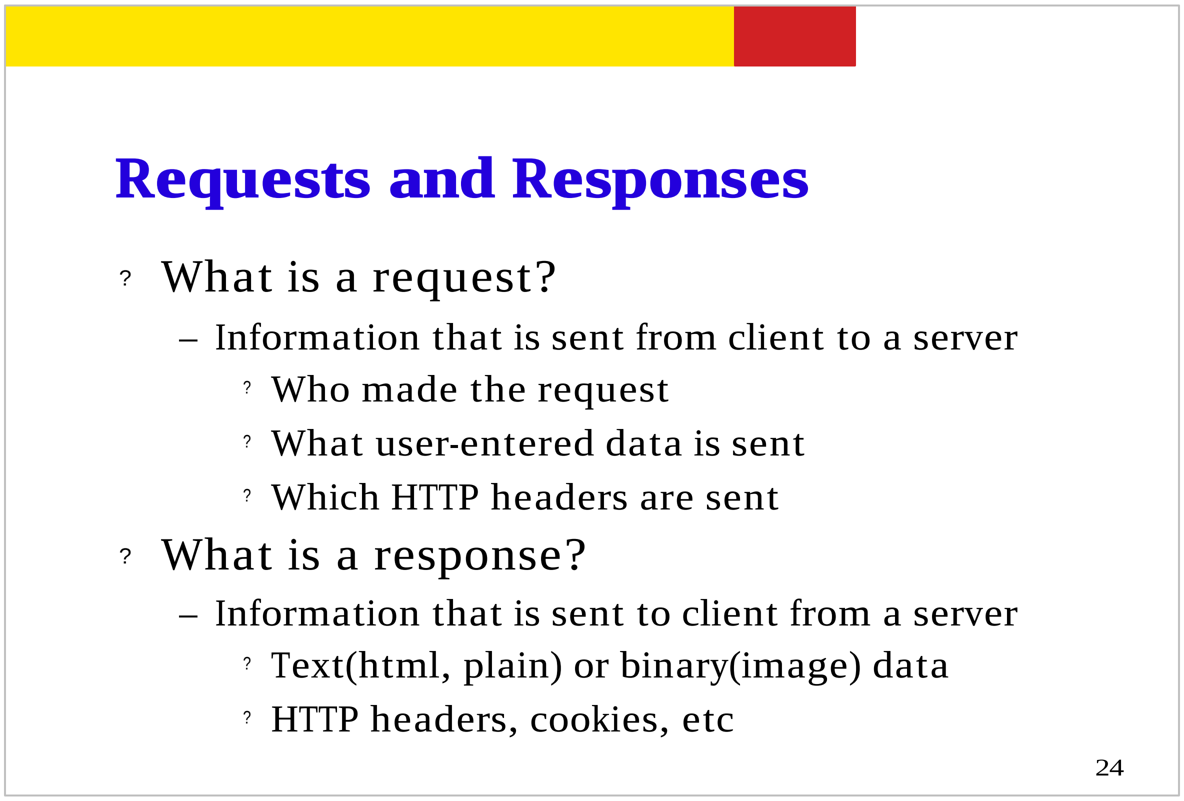
What does Servlet Do?

- Receives client request (mostly in the form of HTTP request)
- Extract some information from the request
- Do content generation or business logic process (possibly by accessing database, invoking EJBs, etc)
- Create and send response to client (mostly in the form of HTTP response) or forward the request to another servlet or JSP page

23

So this slide just repeats what I just said in the previous slides. A servlet container receives client requests typically in the form of HTTP requests. A HTTP request is then abstracted as Java object called HTTPServletRequest by the container, which is then passed to a particular servlet. (By the way, the container knows which servlet is to receive the request since the deployment descriptor, web.xml, specifies which servlet is mapped to which URL and the client request should contain the URL.)

The servlet then extracts any user-entered information from the HTTP request and then performs some contents generation or business logic processing. And the business logic processing might involve accessing database or invoking EJB components in the EJB tier. Once contents generation and business logic processing are done, then the servlet will create a HTTP response and send it back to the client or it can forward the request to another servlet or JSP page.



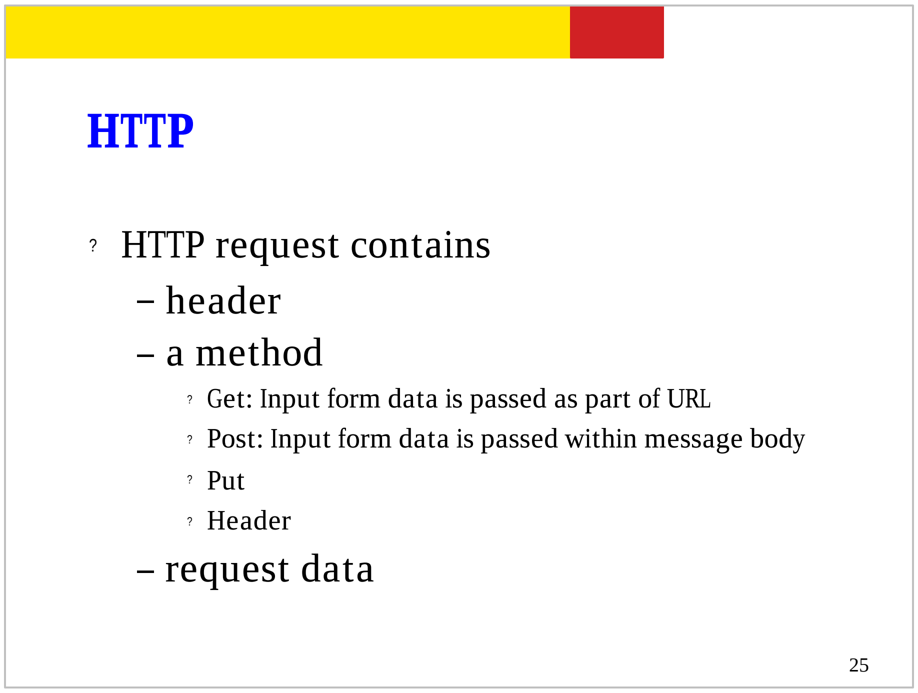
Requests and Responses

- What is a request?
  - Information that is sent from client to a server
    - Who made the request
    - What user-entered data is sent
    - Which HTTP headers are sent
- What is a response?
  - Information that is sent to client from a server
    - Text(html, plain) or binary(image) data
    - HTTP headers, cookies, etc

24

So what are servlet request and servlet response? A request contains information that is sent from the client to a web server while response contains information that is sent from the server to the client. For example, the request contains information such as (1) who made the request and (2) user-entered data and (3) HTTP header entries.

And the response contains some static text such as HTML text or binary data such as images and it also contains HTTP response header entries and cookies, which are used to maintain session state.

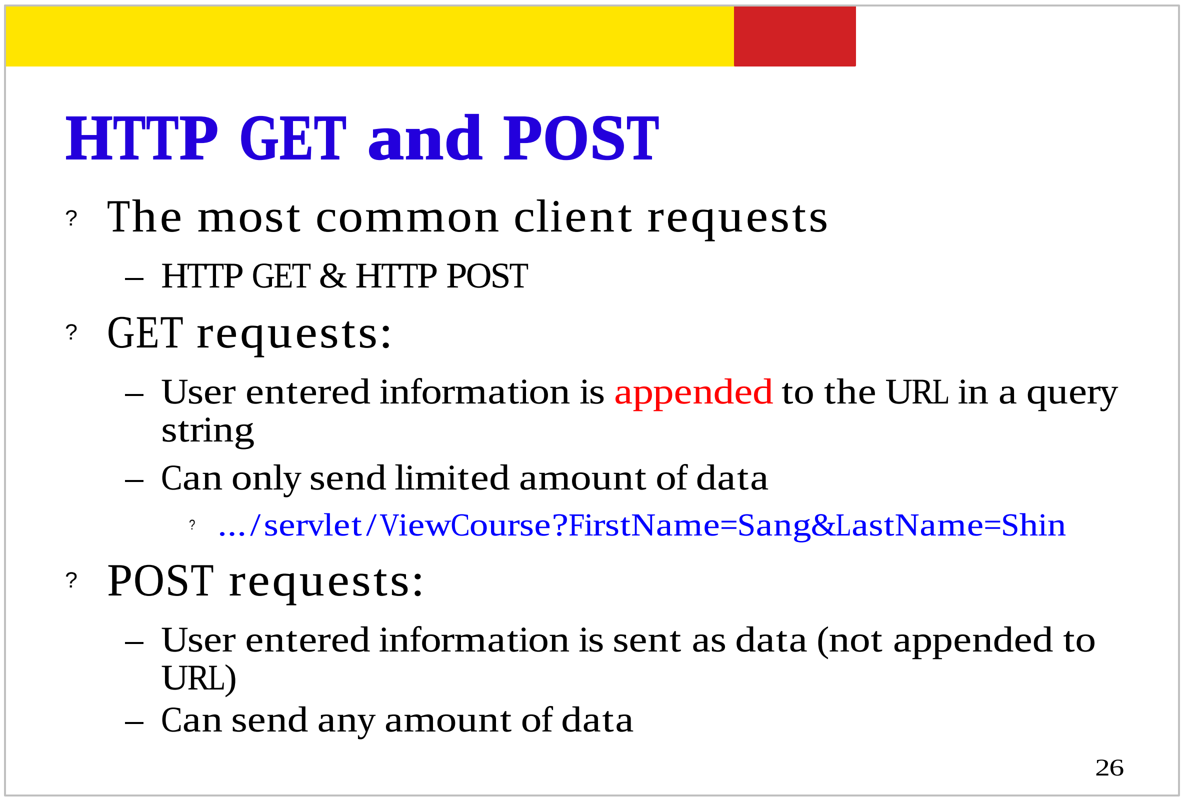


HTTP

- HTTP request contains
  - header
  - a method
    - Get: Input form data is passed as part of URL
    - Post: Input form data is passed within message body
    - Put
    - Header
  - request data

25

.



HTTP GET and POST

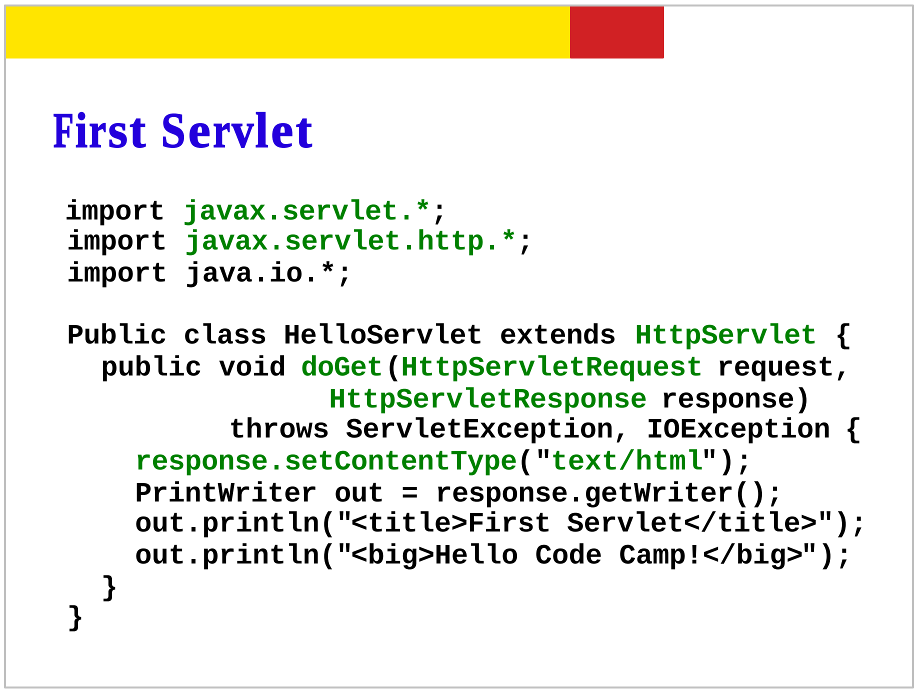
- The most common client requests
  - HTTP GET & HTTP POST
- GET requests:
  - User entered information is appended to the URL in a query string
  - Can only send limited amount of data
    - .../servlet/ViewCourse?FirstName=Sang&LastName=Shin
- POST requests:
  - User entered information is sent as data (not appended to URL)
  - Can send any amount of data

26

The most common form of client requests are HTTP GET and HTTP POST requests.

In the HTTP GET request, the user entered information is appended to the URL as a query string. One caveat of the HTTP GET request is that only limited amount of data can be sent because of the limited space at the end of the URL.

In the HTTP POST request, on the other hand, the user entered information is sent as data. And since it is sent as data, there is no limitation to the amount of data you can send in using HTTP POST.



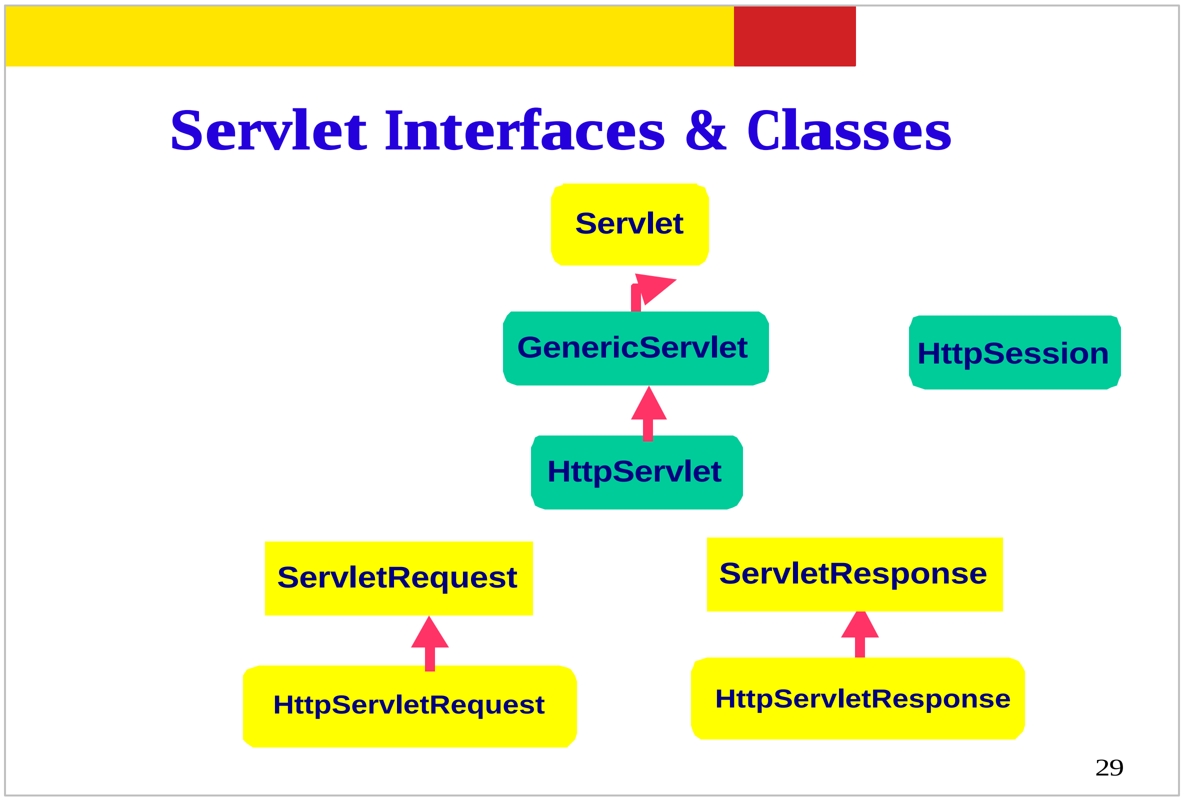
First Servlet

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

Public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
           throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello Code Camp!</big>");
  }
}
```

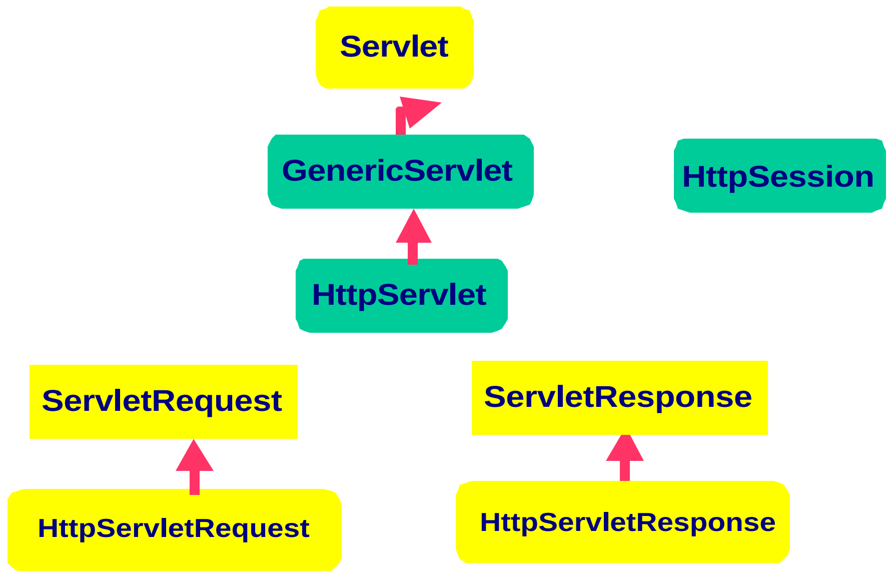
So in this simple servlet program, the request comes in in the form of HTTPServetRequest object and response is created in the form of HttpServletResponse object. By the way, as was mentioned before, these objects are created by the container.

Interfaces & Classes of Servlet

Now let's talk about Servlet interfaces and classes.



Servlet Interfaces & Classes

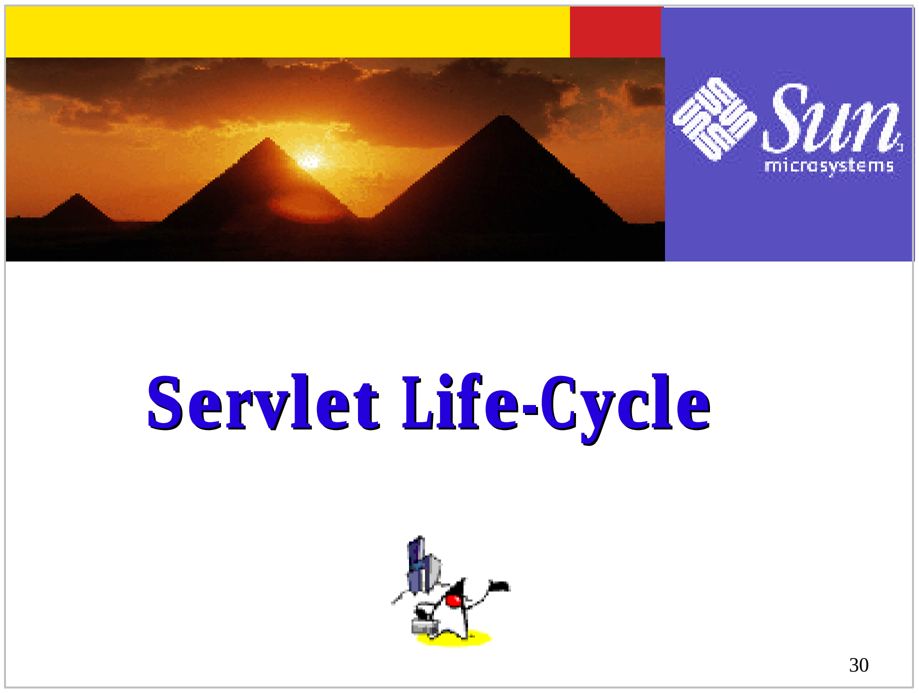


29

This picture shows important servlet interfaces and classes. The ones in yellow color are Java interfaces while the ones in green color are Java classes.

When you create your own servlet, you either extend GenericServlet class or more likely HTTPServlet class. In terms of request and response objects, the container creates objects of HttpServletRequest and HttpServletResponse types for you and pass them to service() method of the HTTPServlet class.

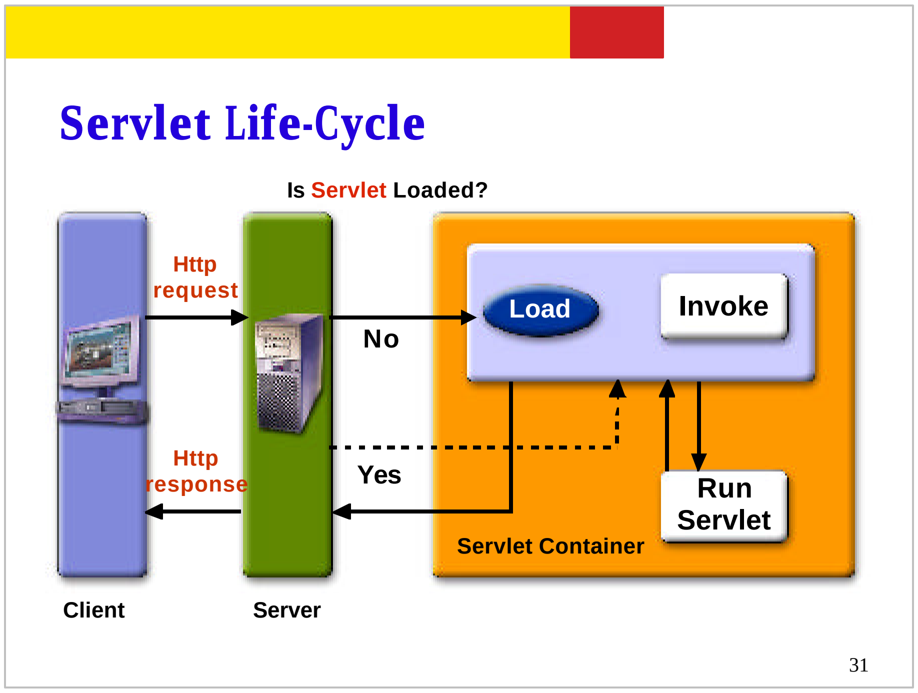
Another important class is HTTPSession class which contains session-wide state information. And we will talk about these in rather detail later on.



Servlet Life-Cycle

30

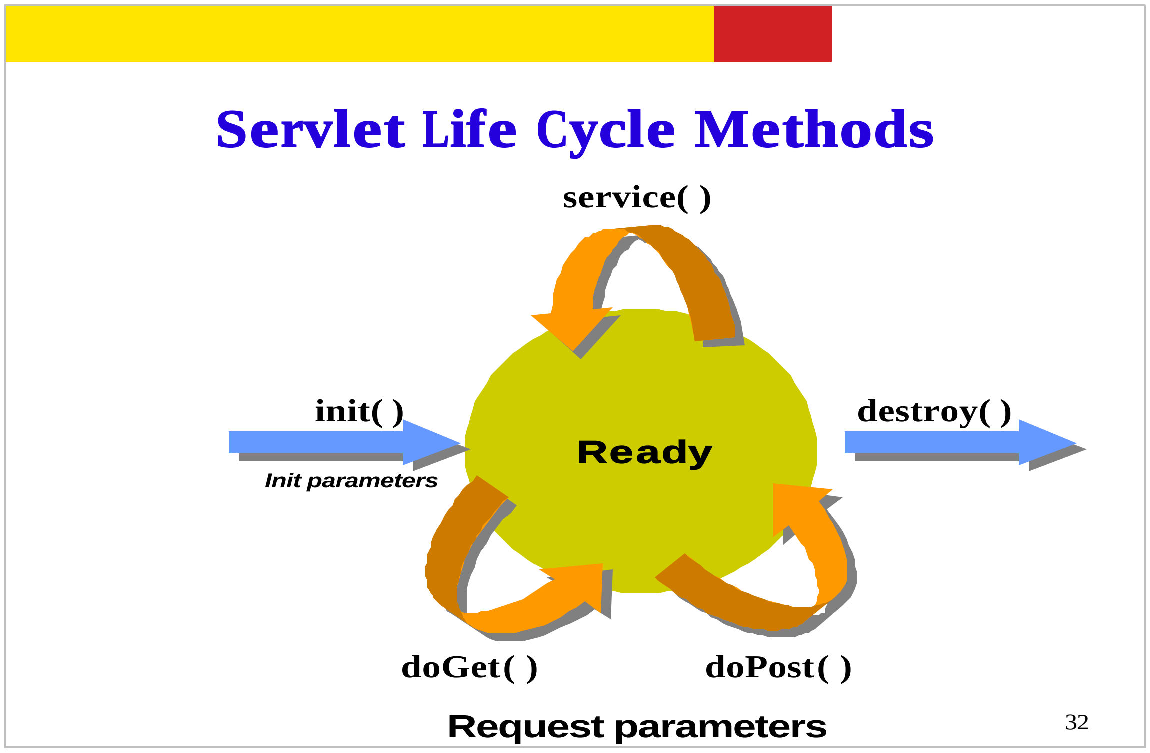
Now let's talk about servlet life-cycle. Understanding servlet life cycle is necessary for you in order to write well-functioning servlet code.



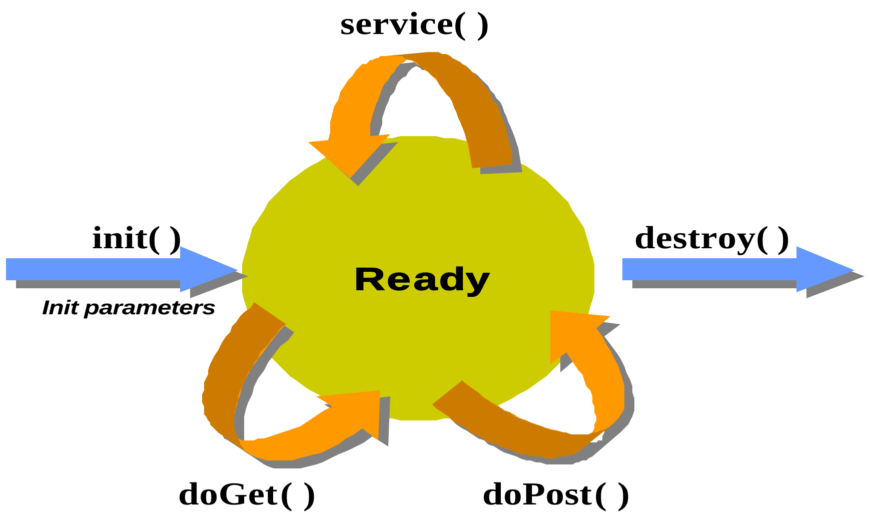
Servlet Life-Cycle

The life cycle of a servlet is controlled by servlet-container in which the servlet has been deployed. When a HTTP request is mapped to a servlet, the container performs the following steps.

# If an instance of the servlet does not exist, the Web container
    # Loads the servlet class.
    # Creates an instance of the servlet class.
    # Initializes the servlet instance by calling the init() method .
# Invokes the service method, passing HttpServletRequest and HttpServletResponse objects as parameters.



Servlet Life Cycle Methods

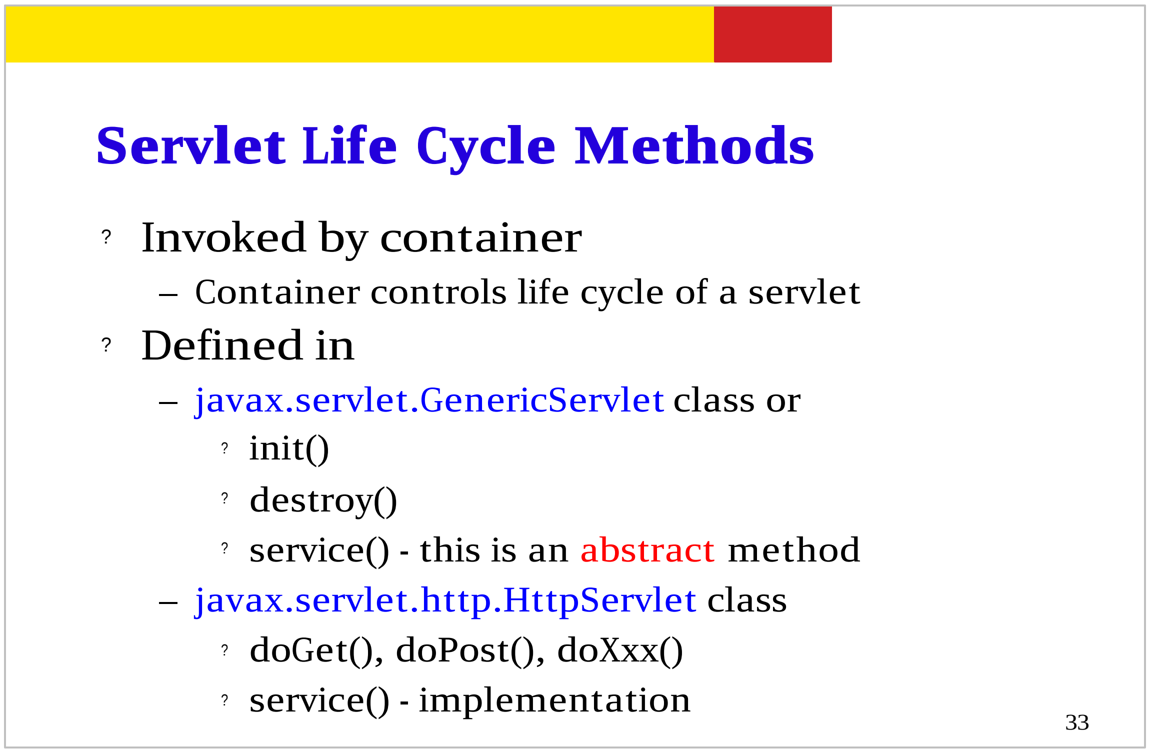


Request parameters

32

The init() method gets called once when a servlet instance is created for the first time. And then service() method gets called every time there comes a new request. Now service() method in turn calls doGet() or doPost() methods for incoming HTTP requests.

And finally when the servlet instance gets removed, the destroy() method gets called. So init() and destroy() methods get called only once while service(), doGet(), and doPost() methods are called a number of times depending on how many HTTP requests are received.



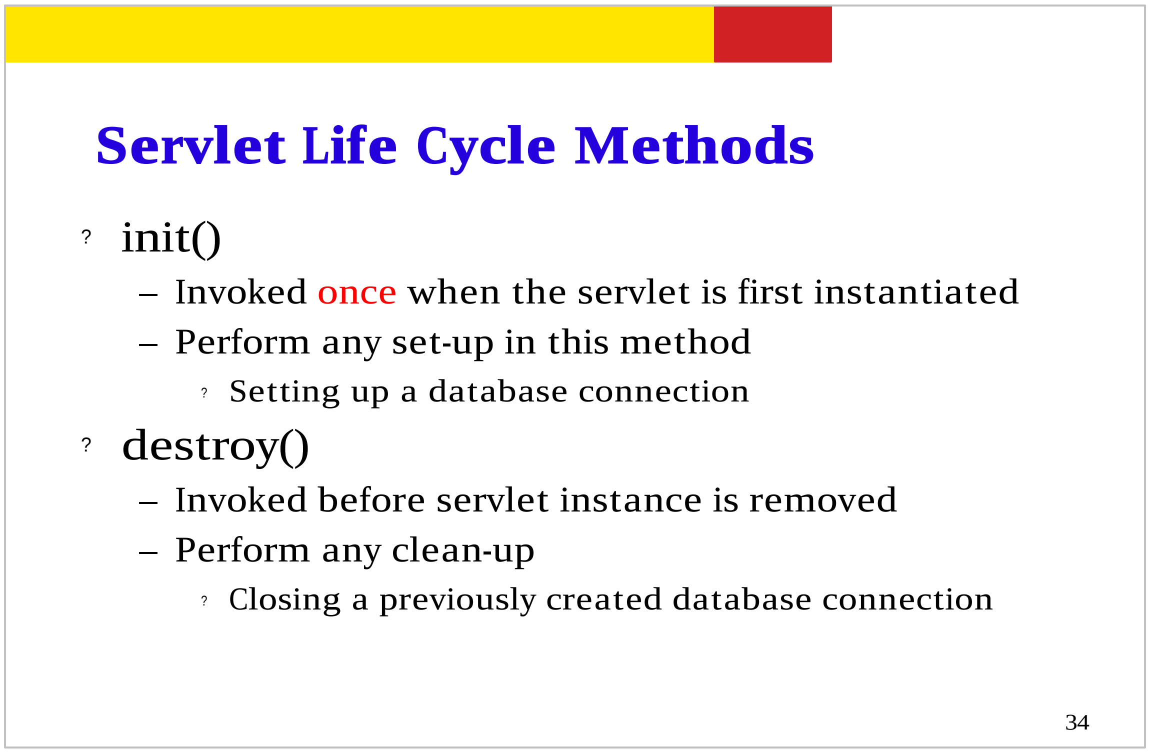
Servlet Life Cycle Methods

- Invoked by container
  - Container controls life cycle of a servlet
- Defined in
  - javax.servlet.GenericServlet class or
    - init()
    - destroy()
    - service() - this is an abstract method
  - javax.servlet.http.HttpServlet class
    - doGet(), doPost(), doXxx()
    - service() - implementation

33

Please note that it is the web-container that invokes these servlet life cycle methods. In other words, it is the container who knows when to call these life cycle methods. That is the reason why we say the life cycle of servlets are controlled by the container.

Now the init(), destroy() and service() methods are defined in the GenericServlet class. Please note that service() method is an abstract method and GenericServlet class is thus abstract class. What this means is that the service() method of the GenericServlet class has to be implemented by a subclass. And the subclass is HTTPServlet class. So HTTPServlet class implemented service() method and in its implementation. It also checks which HTTP command it receives and then delegate the call to a corresponding doXXX() calls, mostly to doGet() and doPost() methods.

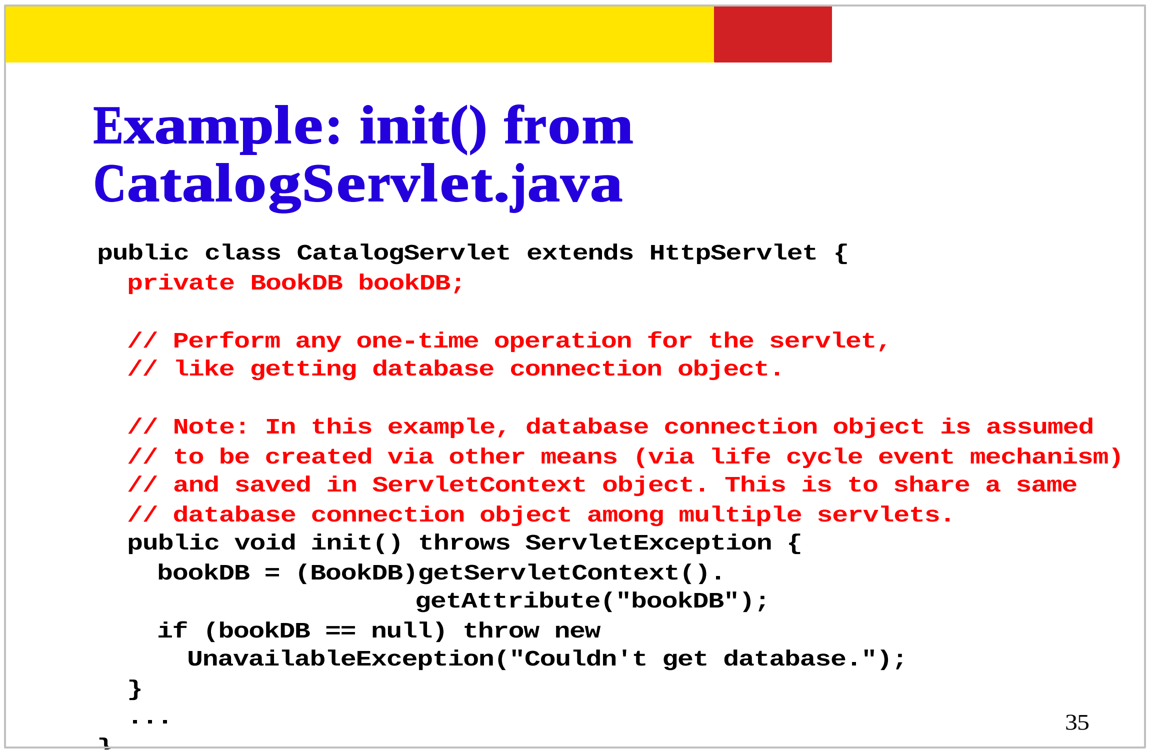


Servlet Life Cycle Methods

- init()
  - Invoked once when the servlet is first instantiated
  - Perform any set-up in this method
    - Setting up a database connection
- destroy()
  - Invoked before servlet instance is removed
  - Perform any clean-up
    - Closing a previously created database connection

34

Now let's talk about init() and destroy() methods one more time. As was mentioned before, init() and destroy() methods are called only once, init() at the time service instance is created while destroy() gets called at the time servlet instance gets removed. And init() can be used to perform some set up operation such as setting up a database connection and destroy() method is used to perform any clean up, for example, removing a previously created database connection.



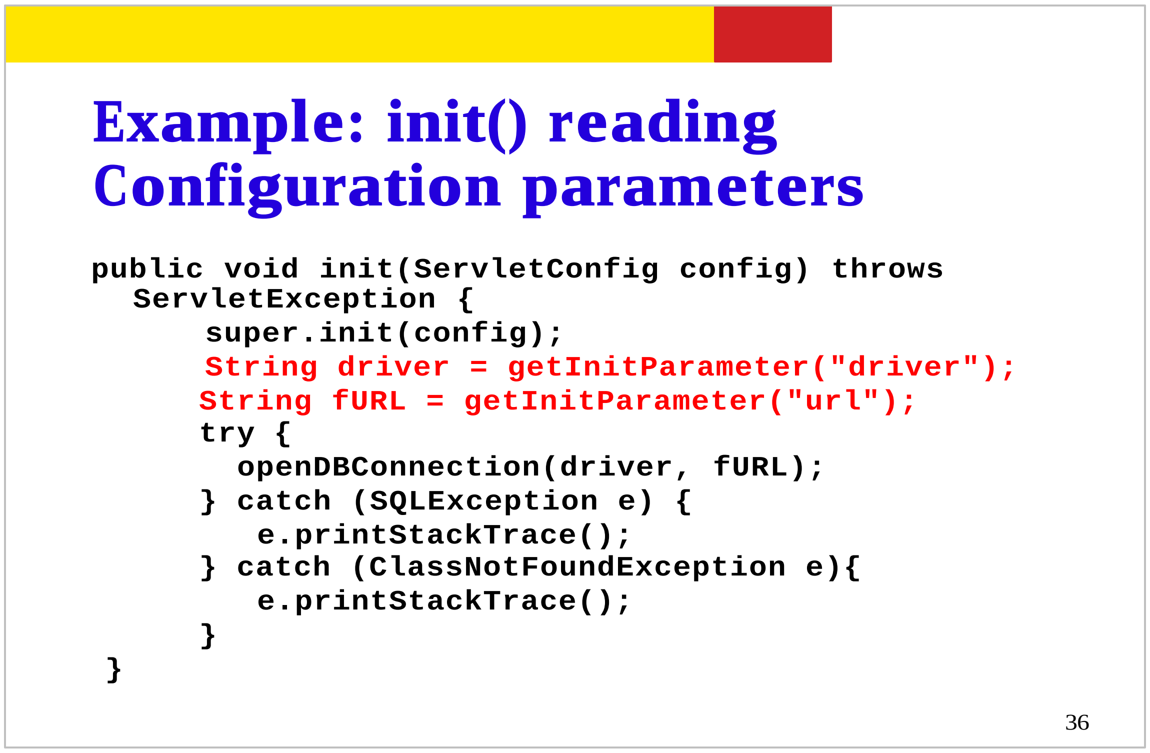
Example: init() from CatalogServlet.java

```
public class CatalogServlet extends HttpServlet {
  private BookDB bookDB;

  // Perform any one-time operation for the servlet,
  // like getting database connection object.

  // Note: In this example, database connection object is assumed
  // to be created via other means (via life cycle event mechanism)
  // and saved in ServletContext object. This is to share a same
  // database connection object among multiple servlets.
  public void init() throws ServletException {
    bookDB = (BookDB)getServletContext().
                      getAttribute("bookDB");
    if (bookDB == null) throw new
      UnavailableException("Couldn't get database.");
  }
  ...
}
```

35

This is an example servlet code that shows how init() method is used to perform any one-time operations such as getting a database connection object.

This example is from the CatalogServlet.java code of Duke's bookstore web application in Java WSDP. Here it is assumed that the database connection is created through another means (via life cycle event mechanism) and saved in ServletContext object before init() method of this servlet code is invoked. This is to share a common database connection among multiple servlets instead of each servlet creating its own database connection. So here in this code, it just gets the reference to the database connection object.

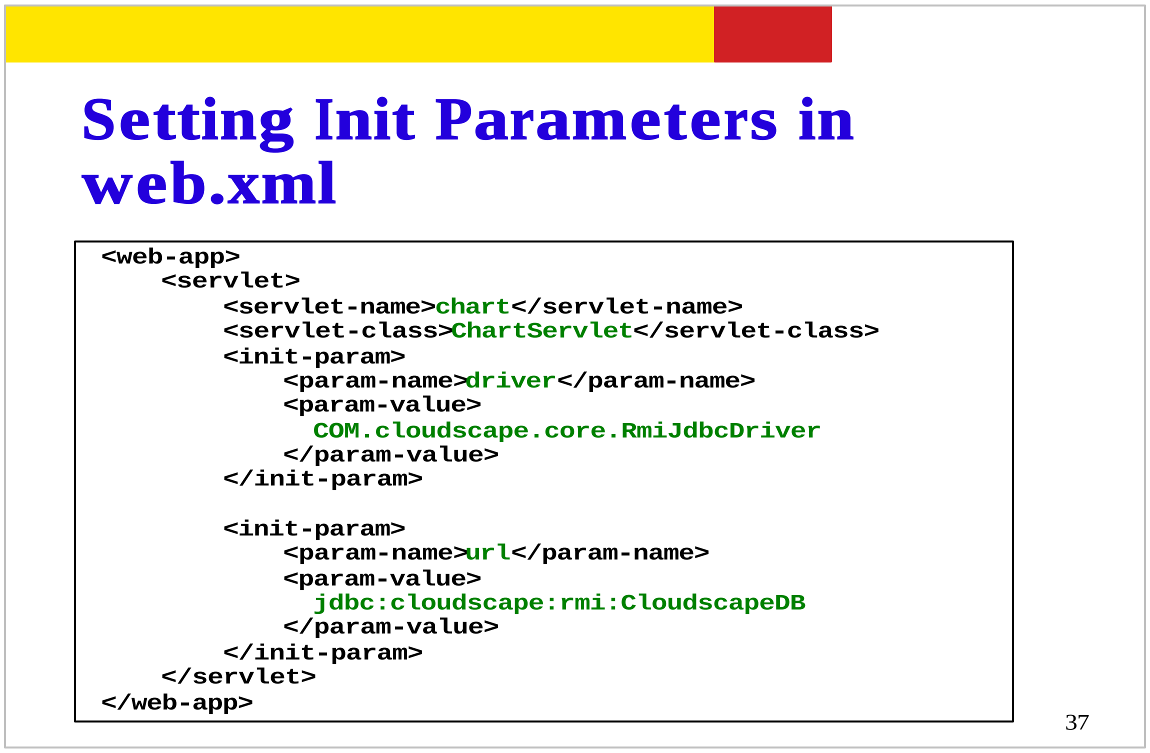


Example: init() reading Configuration parameters

```
public void init(ServletConfig config) throws
  ServletException {
      super.init(config);
      String driver = getInitParameter("driver");
     String fURL = getInitParameter("url");
     try {
       openDBConnection(driver, fURL);
     } catch (SQLException e) {
        e.printStackTrace();
     } catch (ClassNotFoundException e){
        e.printStackTrace();
     }
 }
```

36

In this example, you are actually setting up your own database connection by reading init parameter values. The init parameter values are configured in the web.xml deployment descriptor. Please don't get confused about init parameters with user-entered parameters that come from the browser. The parameters that come from the browser can be accessed via getParameter() method while init parameters from the web.xml deployment descriptor file can be obtained via getInitParameter() method.

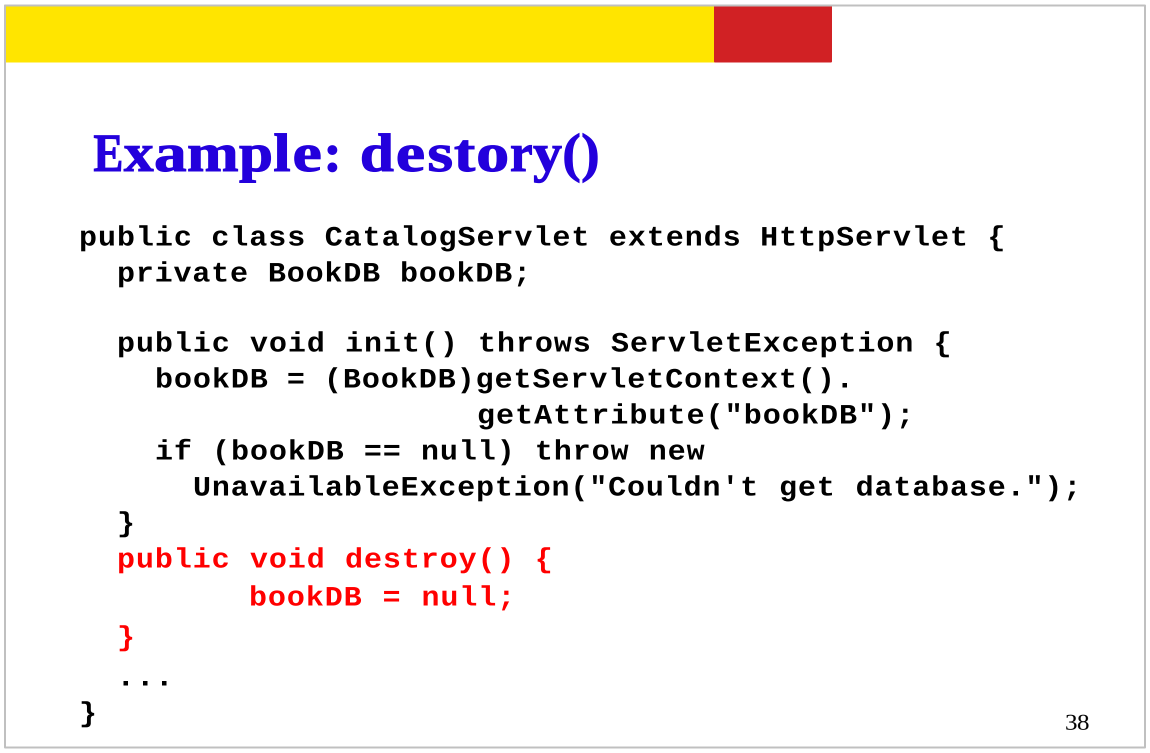


Setting Init Parameters in web.xml

```
<web-app>
    <servlet>
        <servlet-name>chart</servlet-name>
        <servlet-class>ChartServlet</servlet-class>
        <init-param>
            <param-name>driver</param-name>
            <param-value>
              COM.cloudscape.core.RmiJdbcDriver
            </param-value>
        </init-param>

        <init-param>
            <param-name>url</param-name>
            <param-value>
              jdbc:cloudscape:rmi:CloudscapeDB
            </param-value>
        </init-param>
    </servlet>
</web-app>
```

37

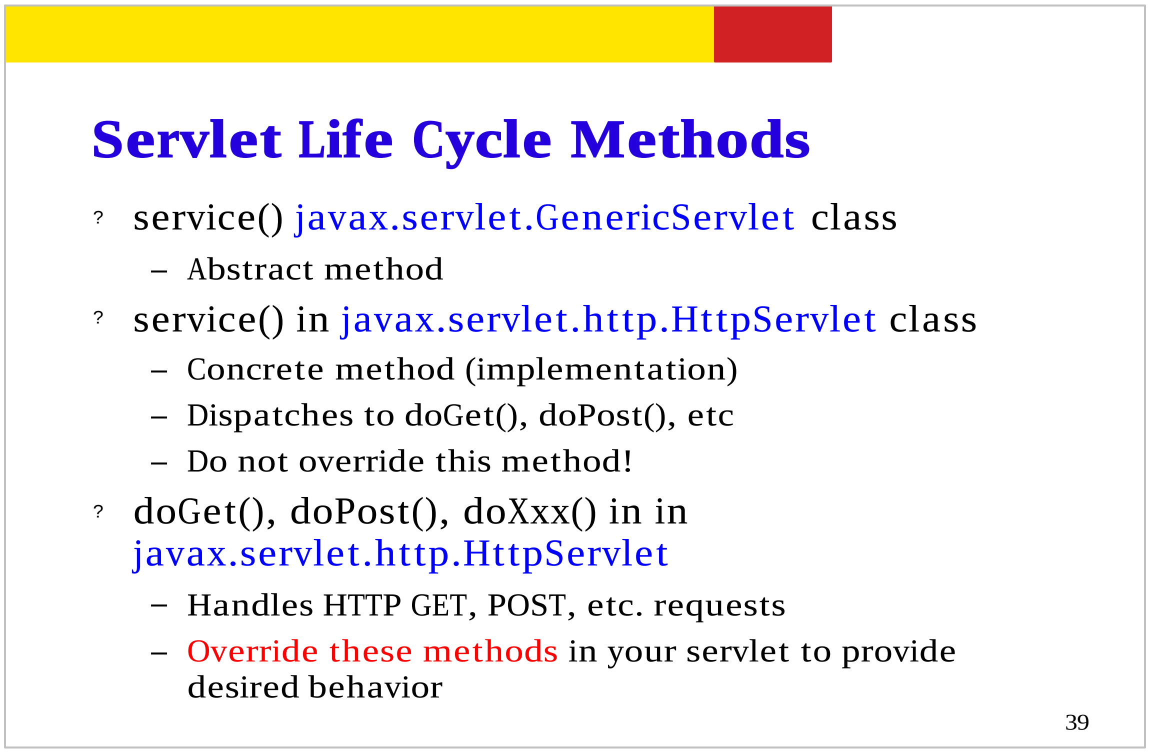
So this is an example of web.xml deployment descriptor in which init parameters are configured for the example code in the previous slide.



Example: destory()

```
public class CatalogServlet extends HttpServlet {
  private BookDB bookDB;

  public void init() throws ServletException {
    bookDB = (BookDB)getServletContext().
                     getAttribute("bookDB");
    if (bookDB == null) throw new
      UnavailableException("Couldn't get database.");
  }
  public void destroy() {
         bookDB = null;
  }
  ...
}
```

This is destroy example code again from CatalogServlet code. Here destroy() method nulling the local variable that contains the reference to database connection. (Again in this example, since a common database connection is used by multiple servlets, you don't want to close the database connection.)



Servlet Life Cycle Methods

? service() javax.servlet.GenericServlet class
  - Abstract method

? service() in javax.servlet.http.HttpServlet class
  - Concrete method (implementation)
  - Dispatches to doGet(), doPost(), etc
  - Do not override this method!

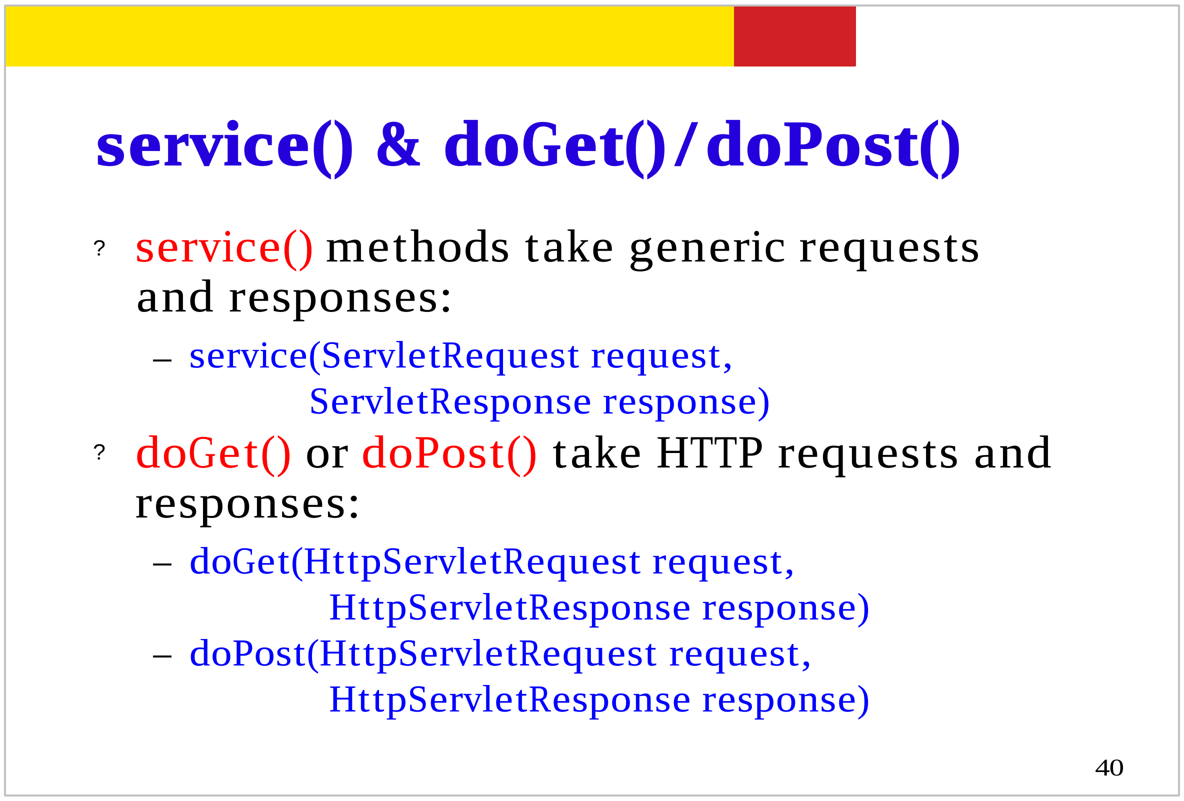
? doGet(), doPost(), doXxx() in in javax.servlet.http.HttpServlet
  - Handles HTTP GET, POST, etc. requests
  - Override these methods in your servlet to provide desired behavior

39

Now let's talk about service() and doGet(), doPost() methods. These are the methods into which you put your business logic or dynamic contents generation logic.

As was mentioned, the service() method is an abstract method in GenericServlet class which is then implemented in a subclass. And HTTPServlet is a subclass that is already provided for the developers. The service() method implementation of the HTTPServlet class then dispatches the call to doXXX() methods depending on the HTTP request type.

As a servlet developer, you want to override the doXXX() methods to implement a desired behavior of your servlet.

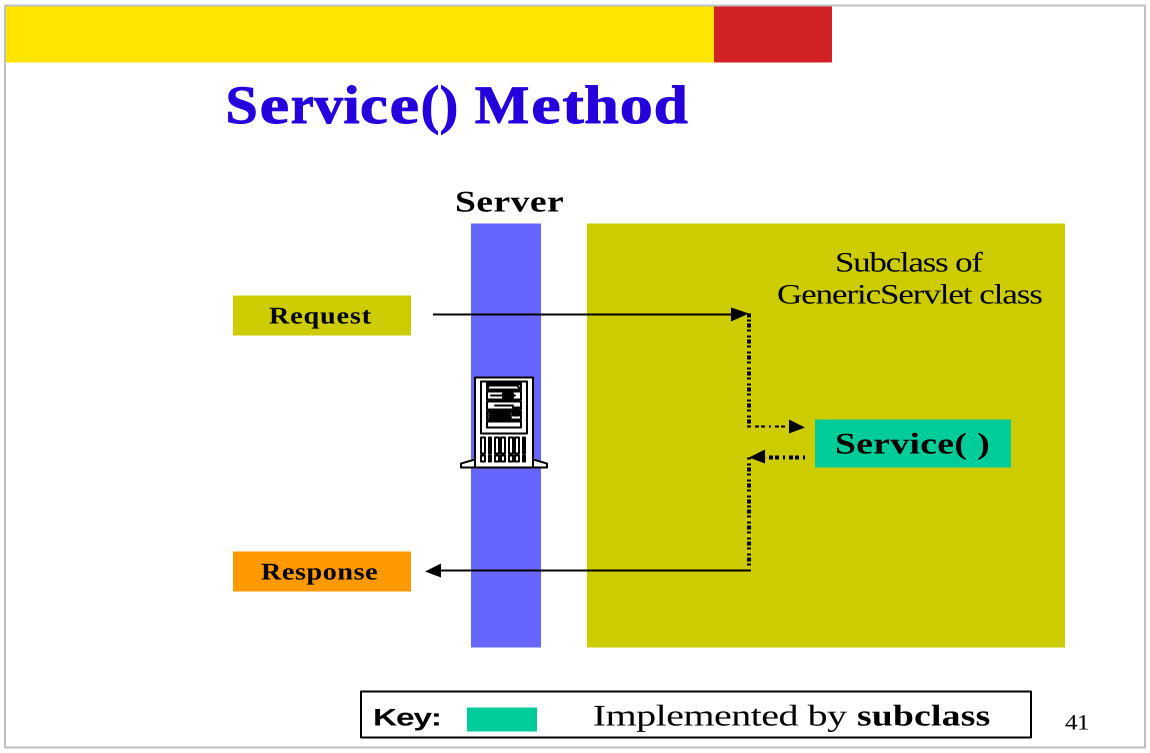


service() & doGet()/doPost()

- service() methods take generic requests and responses:
  - service(ServletRequest request, ServletResponse response)
- doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponse response)

40

In this slide, the method signatures of service(), doGet() and doPost() methods are shown. Basically they receive HttpServletRequest and HttpServletResponse objects as input parameters.



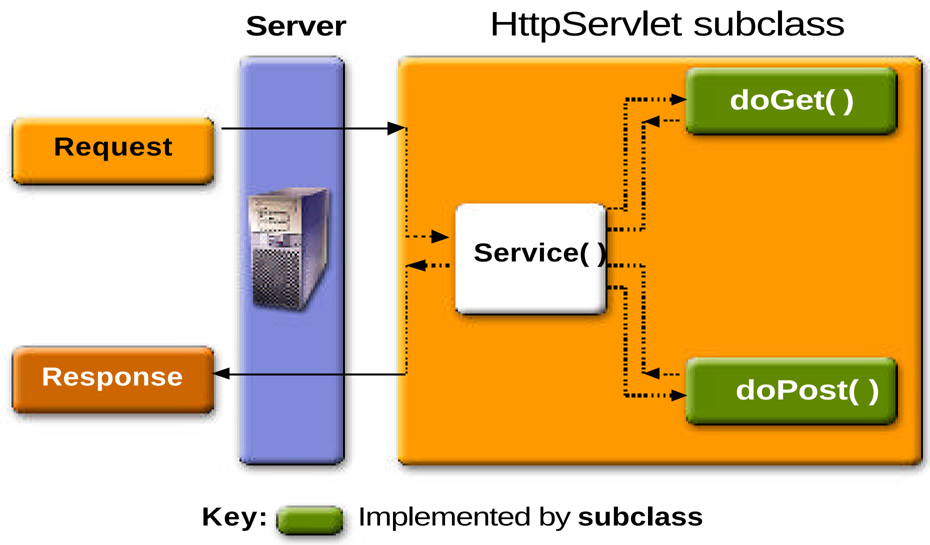
Service() Method

Key: 🟩 Implemented by **subclass**

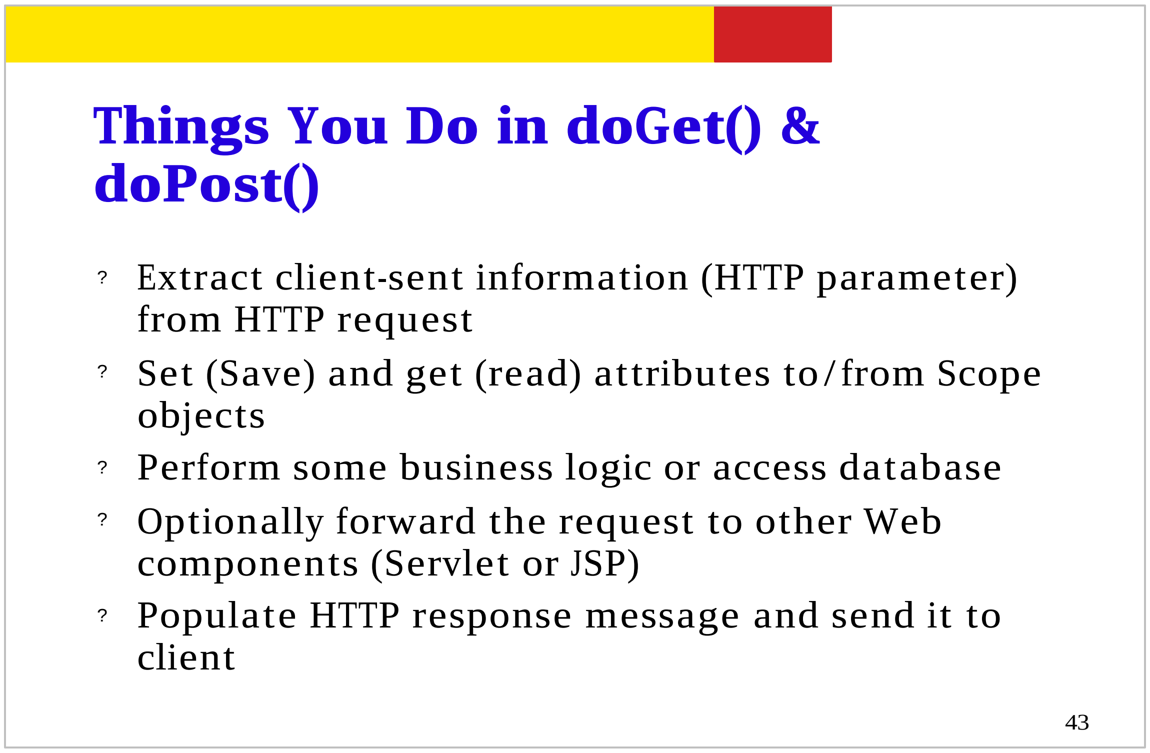
41

This picture shows how service() method of a subclass of GenericServlet class is invoked.

doGet() and doPost() Methods



As was mentioned, doGet() and doPost() methods of HTTPServlet class are invoked from concrete implementation of service() method in the HTTPServlet class.



Things You Do in doGet() & doPost()

- Extract client-sent information (HTTP parameter) from HTTP request
- Set (Save) and get (read) attributes to/from Scope objects
- Perform some business logic or access database
- Optionally forward the request to other Web components (Servlet or JSP)
- Populate HTTP response message and send it to client

43

So what are the things you want to do in doGet() and doPost() methods? Several things.

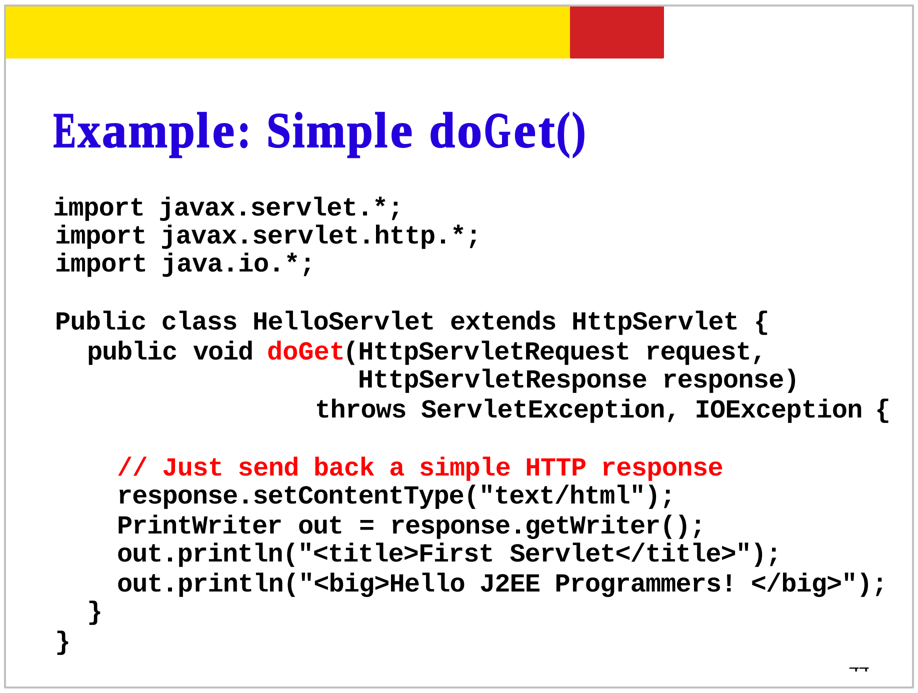
First, you can extract client sent information such as user-entered parameter values that were sent as query string.

Second, you can set and get attributes to and from scope objects.

Third, you perform some business logic or access the database.

Fourth, you can optionally include or forward your requests to other web components.

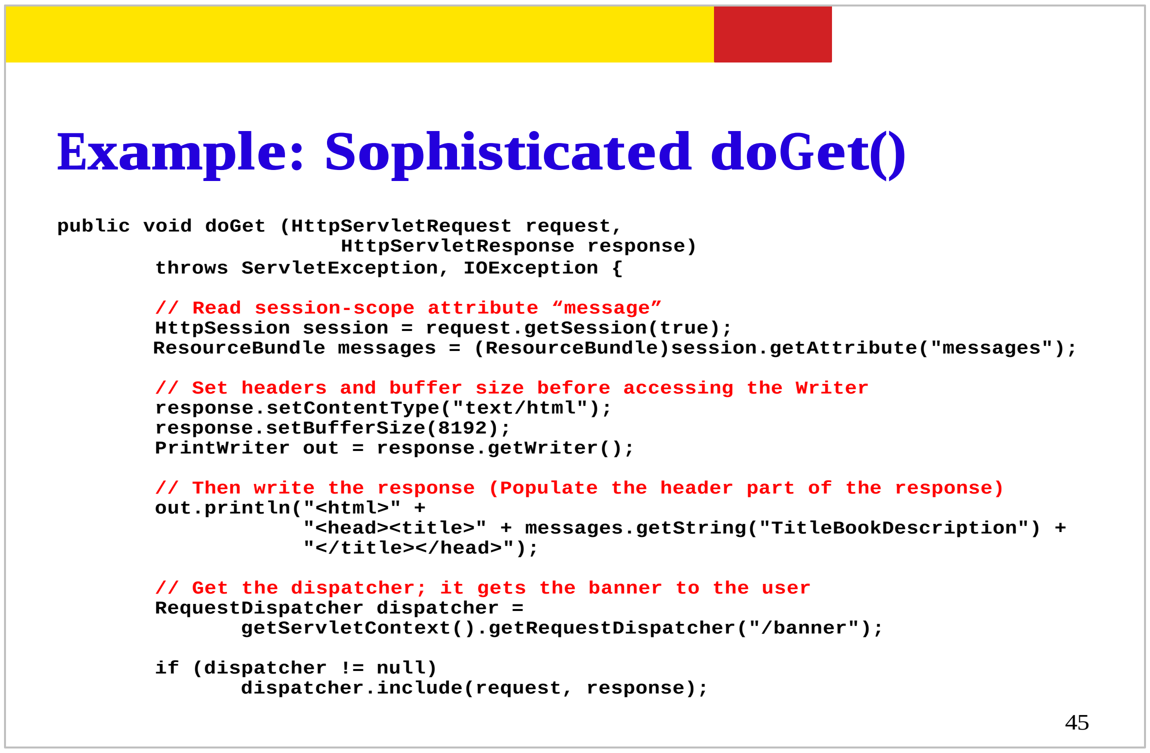
Finally,you can populate HTTP response message and then send it to client.



Example: Simple doGet()

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

Public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                 throws ServletException, IOException {

    // Just send back a simple HTTP response
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello J2EE Programmers! </big>");
  }
}
```

This is a very simple example code of doGet() method. In this example, a simple HTTP response message is created and then sent back to client.

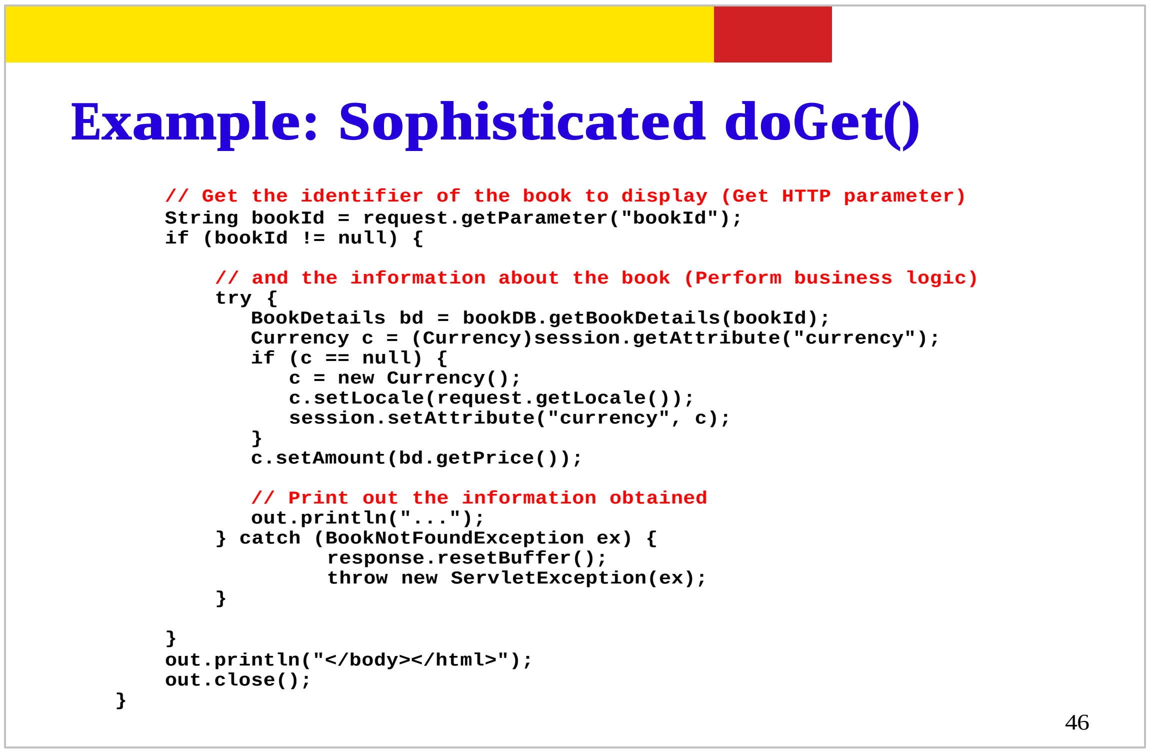


Example: Sophisticated doGet()

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

        // Read session-scope attribute “message”
        HttpSession session = request.getSession(true);
        ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

        // Set headers and buffer size before accessing the Writer
        response.setContentType("text/html");
        response.setBufferSize(8192);
        PrintWriter out = response.getWriter();

        // Then write the response (Populate the header part of the response)
        out.println("<html>" +
                    "<head><title>" + messages.getString("TitleBookDescription") +
                    "</title></head>");

        // Get the dispatcher; it gets the banner to the user
        RequestDispatcher dispatcher =
               getServletContext().getRequestDispatcher("/banner");

        if (dispatcher != null)
               dispatcher.include(request, response);
```

45

And this is a bit more sophisticated example of the doGet() method. As you can see, within doGet() method, you can (1) read values of attributes maintained in the scope objects (we will talk about scope objects later on), (2) set the properties of the Writer object such as buffer size, (3) write actual response via Writer object,(4) get a Dispatcher object and include the output from another web component.



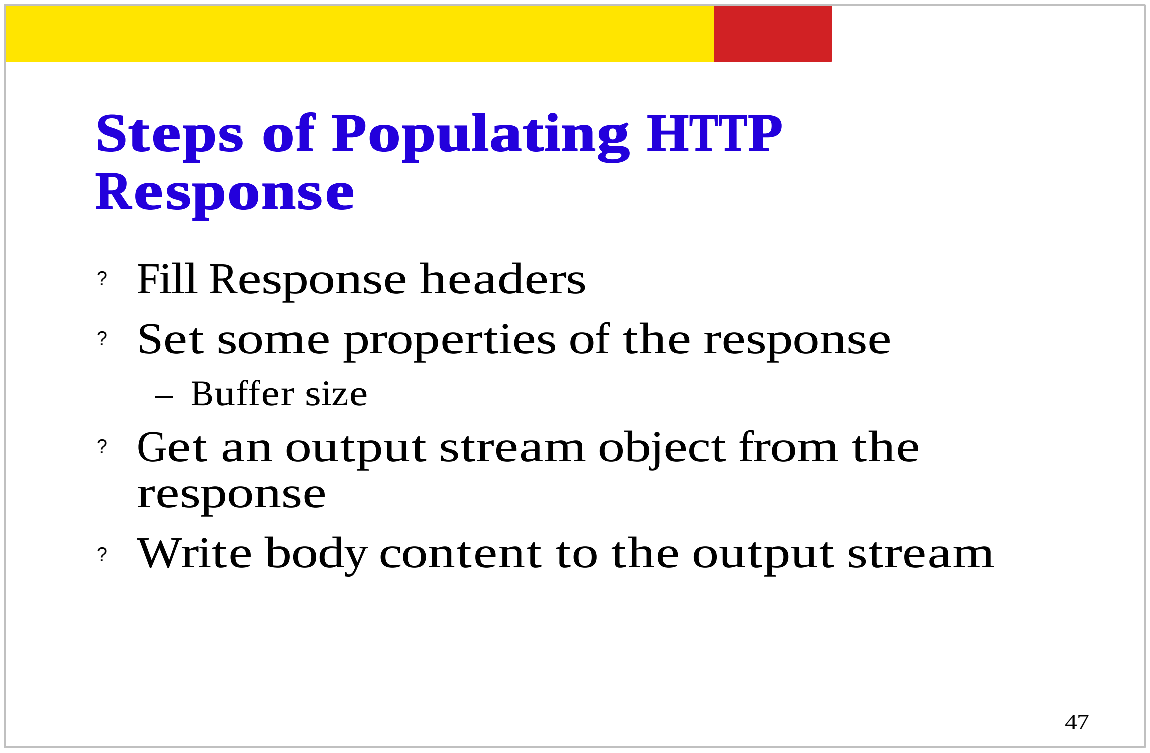
Example: Sophisticated doGet()

```
        // Get the identifier of the book to display (Get HTTP parameter)
        String bookId = request.getParameter("bookId");
        if (bookId != null) {

            // and the information about the book (Perform business logic)
            try {
               BookDetails bd = bookDB.getBookDetails(bookId);
               Currency c = (Currency)session.getAttribute("currency");
               if (c == null) {
                  c = new Currency();
                  c.setLocale(request.getLocale());
                  session.setAttribute("currency", c);
               }
               c.setAmount(bd.getPrice());

               // Print out the information obtained
               out.println("...");
            } catch (BookNotFoundException ex) {
                     response.resetBuffer();
                     throw new ServletException(ex);
            }

        }
        out.println("</body></html>");
        out.close();
    }
```

46

This is a continuation of the previous slide.

Here in this part of the example code, you get the value of user entered parameter value that is called “bookId”. Using the value of bookId parameter, the example code then gets detailed information on the book such as price. Finally the code creates HTTP response message that contains the just retrieved information on the book.



Steps of Populating HTTP Response

? Fill Response headers

? Set some properties of the response

  – Buffer size

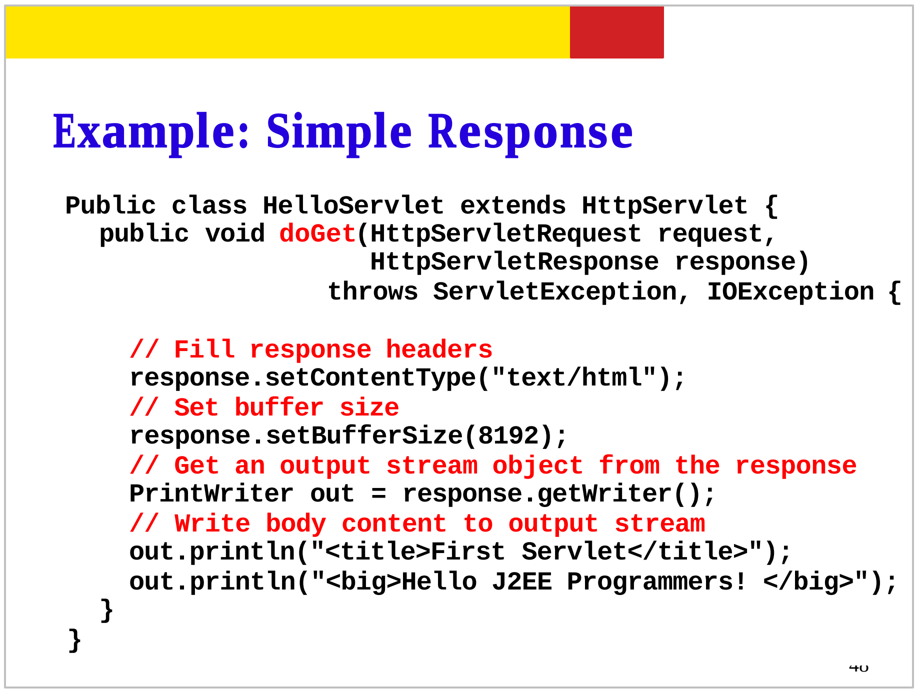
? Get an output stream object from the response

? Write body content to the output stream

47

So far, we have seen several example codes in which the HTTP response message is created and then sent back to the client. So what are the typical steps you follow when creating a HTTP response?

First you fill some HTTP response headers such as content type. Second, you set some properties such as buffer size. Third, you get an output stream object from the response object and then write body contents to the output stream.



Example: Simple Response

```
Public class HelloServlet extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                 throws ServletException, IOException {

    // Fill response headers
    response.setContentType("text/html");
    // Set buffer size
    response.setBufferSize(8192);
    // Get an output stream object from the response
    PrintWriter out = response.getWriter();
    // Write body content to output stream
    out.println("<title>First Servlet</title>");
    out.println("<big>Hello J2EE Programmers! </big>");
  }
}
```

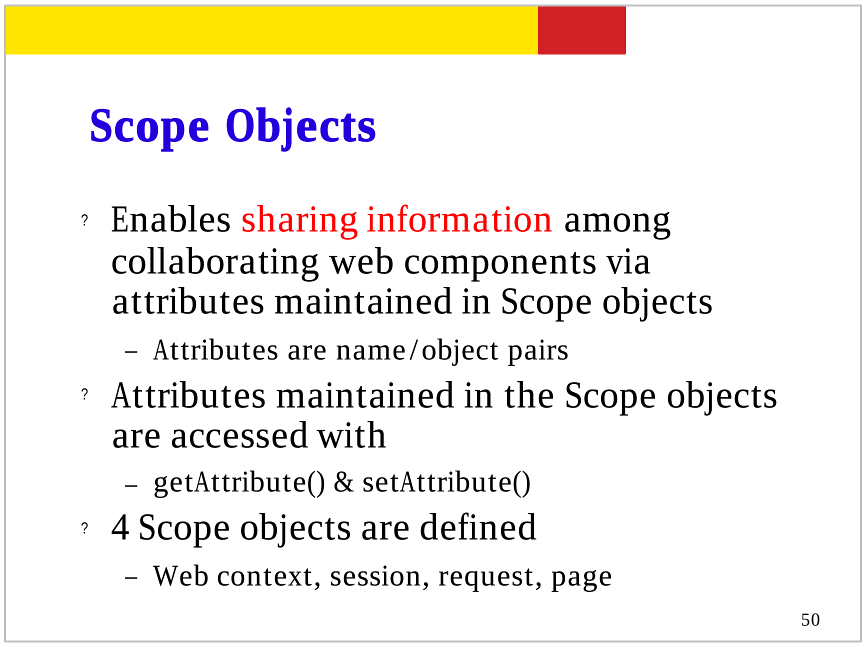
48

This is a simple servlet example code in which some properties of a response object are set. First you set the content type as text/html and then set the buffer size. Setting a buffer size means that the output stream will buffer the data up to buffer size before sending it to client. And then retrieve an output stream (PrintWriter object in this example) and then write the body content to the output stream.

Scope Objects

49

We have seen some scope objects already such as Session object or ServletContext objects. Now let's spend some time talking about these scope objects because you will use scope objects in your servlet code to maintain application or servlet wide state information.



Scope Objects

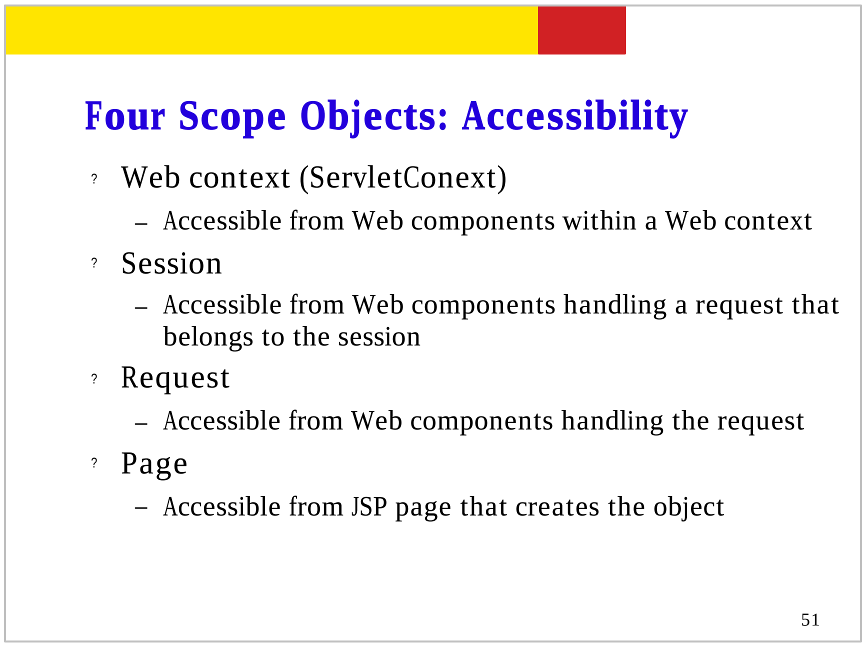
- Enables sharing information among collaborating web components via attributes maintained in Scope objects
  - Attributes are name/object pairs
- Attributes maintained in the Scope objects are accessed with
  - getAttribute() & setAttribute()
- 4 Scope objects are defined
  - Web context, session, request, page

50

So what is a scope object? Scope objects enables sharing information among collaborating web components, that is, servlets and JSP components, via what is called attributes. And these attributes are basically name/object pairs that are maintained in the scope objects.

And in your code, you get the value of an attribute via getAttribute() method and set the value of an attribute via setAttribute() method.

Now in servlet architecture, there are 4 different types of scope objects depending on the scope they cover.



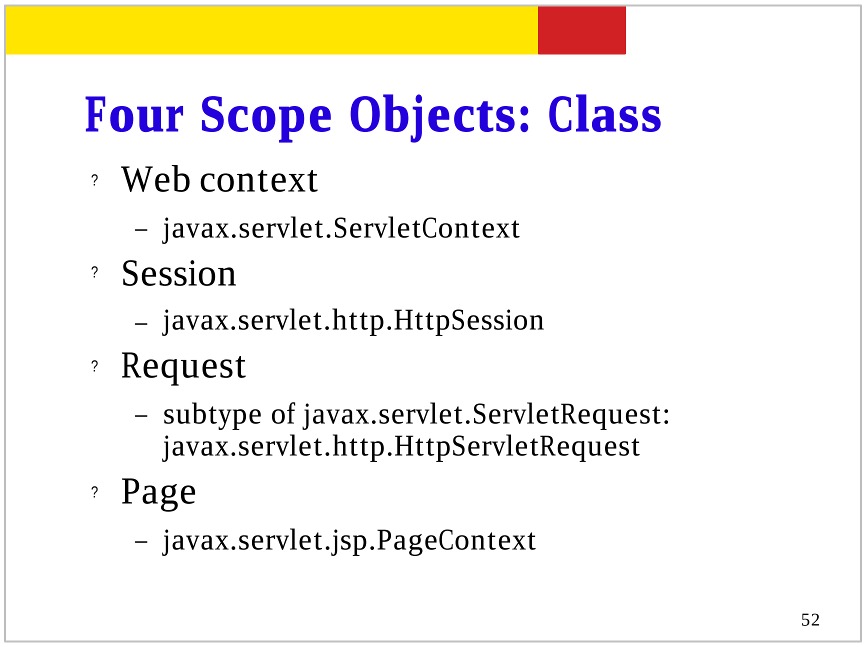
Four Scope Objects: Accessibility

- Web context (ServletConext)
  - Accessible from Web components within a Web context
- Session
  - Accessible from Web components handling a request that belongs to the session
- Request
  - Accessible from Web components handling the request
- Page
  - Accessible from JSP page that creates the object

51

The four scopes objects are (1) web context scope object, (2) session scope object, (3) request object, and (4) page object.

Each of these four scope objects have difference scope of accessibility. For example, web context scope object has a scope of web application, that is, a web context object is shared by all web components within a single web application. And a session object is shared by web components that share a same session. Request object is shared by web components that handle the same request. And page object is an object used within a JSP page.



Four Scope Objects: Class

- Web context
  - javax.servlet.ServletContext
- Session
  - javax.servlet.http.HttpSession
- Request
  - subtype of javax.servlet.ServletRequest: javax.servlet.http.HttpServletRequest
- Page
  - javax.servlet.jsp.PageContext

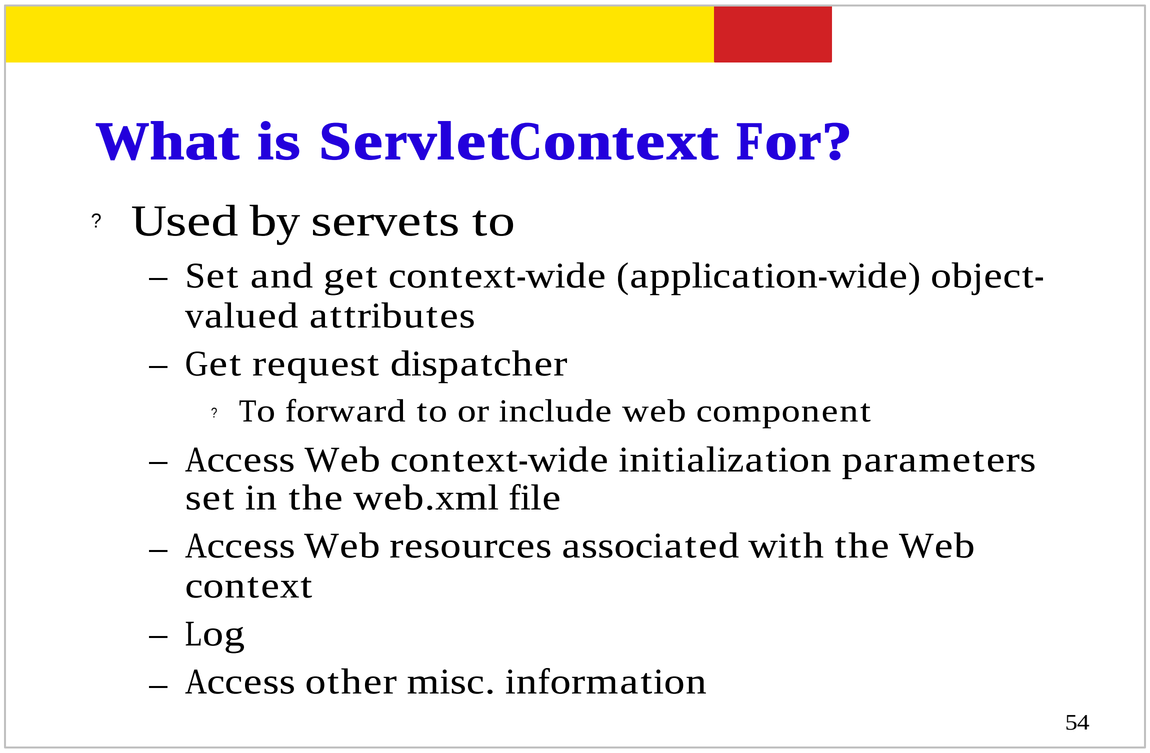
52

This slide lists the classes that represent these four scope objects.

Web Context (ServletContext)

53

Now let's talk about each of the scope objects and see how they get used in your servlet code. First, let's talk about web context scope object. By the way, a web context object is represented by ServletContext object.



What is ServletContext For?

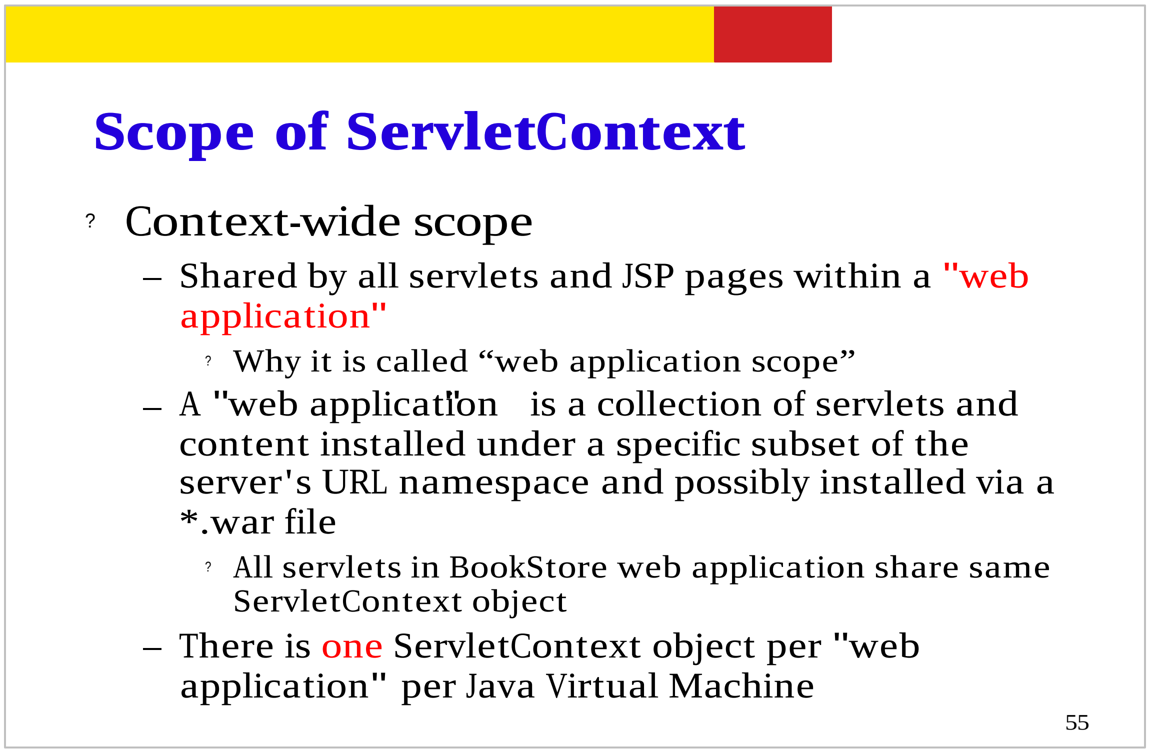
- Used by servets to
  - Set and get context-wide (application-wide) object-valued attributes
  - Get request dispatcher
    - To forward to or include web component
  - Access Web context-wide initialization parameters set in the web.xml file
  - Access Web resources associated with the Web context
  - Log
  - Access other misc. information

54

So what is ServletContext object for? It is used by servlets to set and get context-wide object-value attributes.

You can also get request dispatcher object from the ServletContext object. And you use a request dispatcher object in order to forward a HTTP request to another web component or include the output of another web component.

You can also access context-wide initialization parameters that were set in the web.xml deployment descriptor. You can also access web resources associated within the web context. You can also access logger object and other misc. information.



Scope of ServletContext

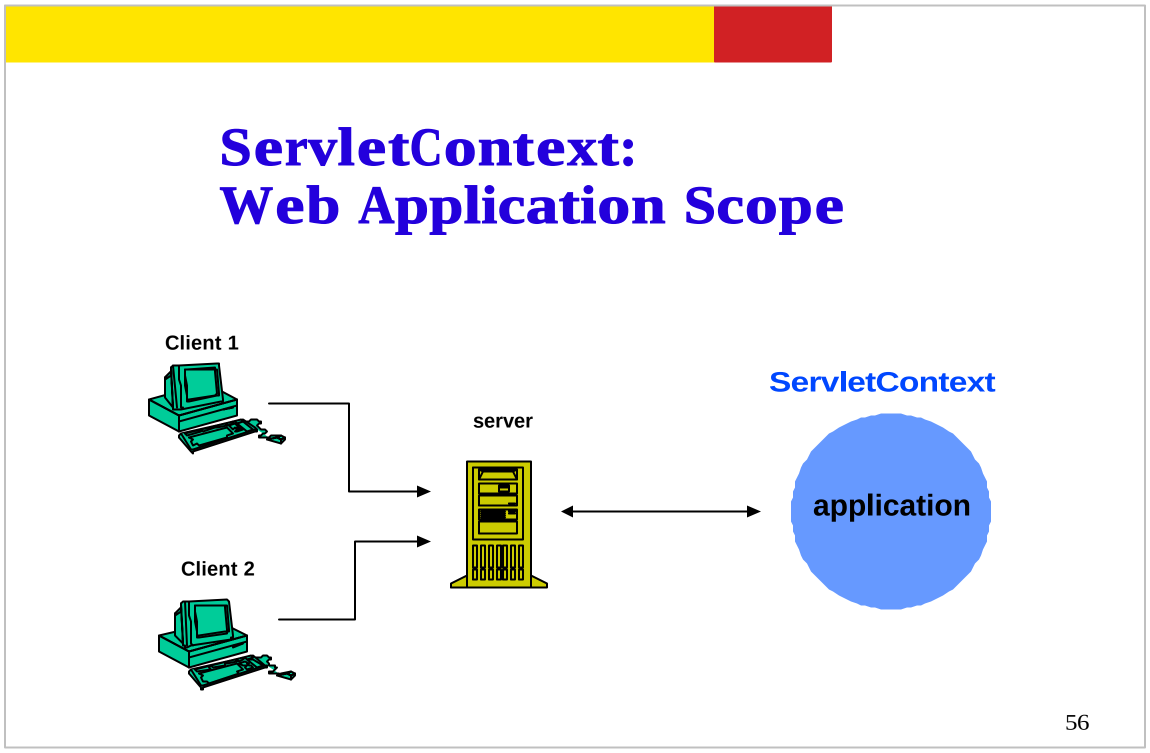
- Context-wide scope
  - Shared by all servlets and JSP pages within a "web application"
    - Why it is called “web application scope”
  - A "web application" is a collection of servlets and content installed under a specific subset of the server's URL namespace and possibly installed via a *.war file
    - All servlets in BookStore web application share same ServletContext object
  - There is one ServletContext object per "web application" per Java Virtual Machine

55

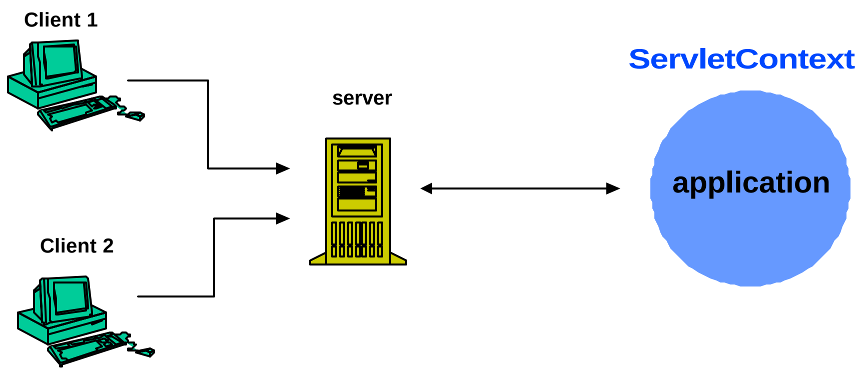
So I mentioned “context-wide” scope several times already. What does that mean? Context-wide scope is shared by all servlets and JSP pages within a single Web application. This is why Context-wide scope is called web application scope.

Now a Web application is a collection of multiple servlets and contents and typically they all share a subset of the URL namespace. And they all belong to a single *.WAR file. For example, all the servlets in the BookStore web application in Java WSDP share the same ServletContext object.

Finally please note that there is one ServletContext object per a web application.

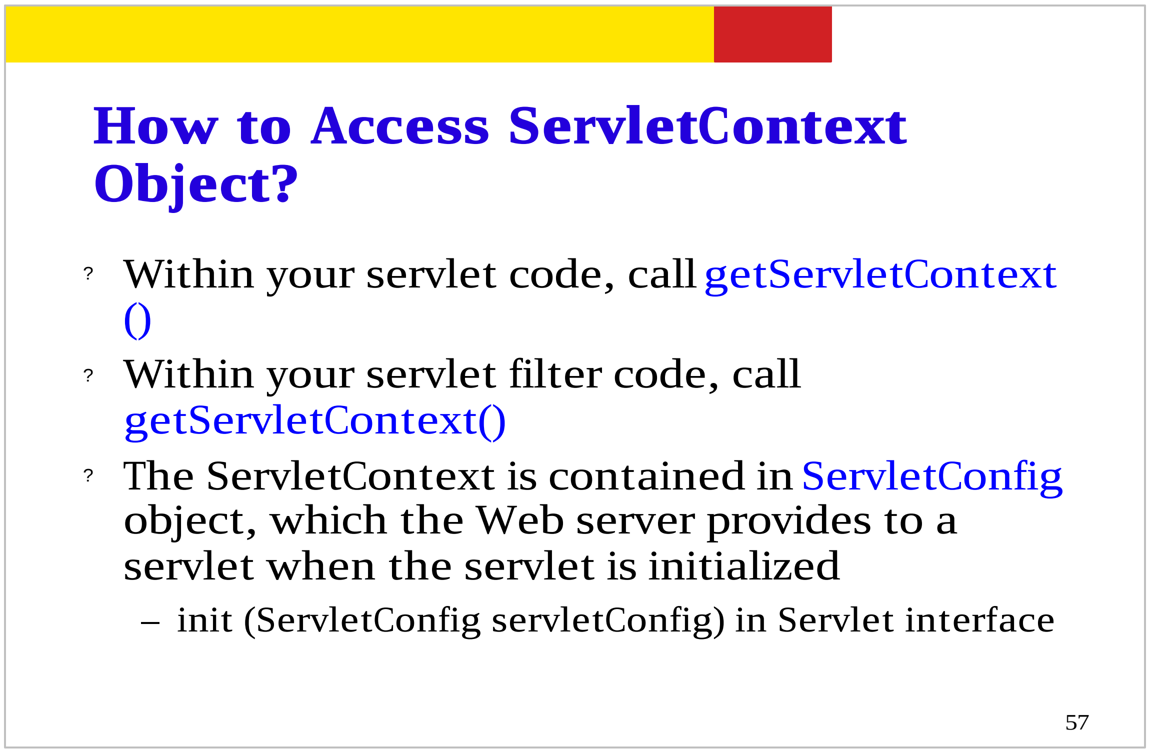


ServletContext: Web Application Scope



56

This picture shows relationship between multiple clients that access the same Web application. The instances of servlet classes that belong to a single Web application will share the same ServletContext object regardless of who the client is.



How to Access ServletContext Object?

? Within your servlet code, call getServletContext ()

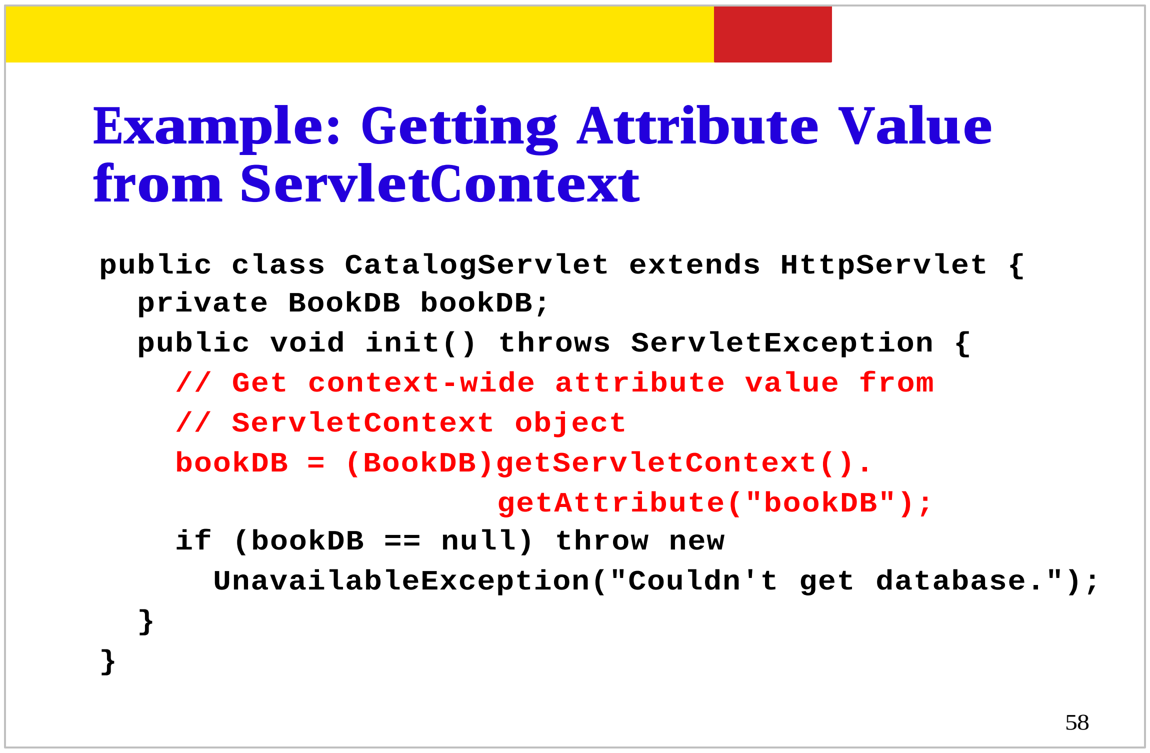
? Within your servlet filter code, call getServletContext()

? The ServletContext is contained in ServletConfig object, which the Web server provides to a servlet when the servlet is initialized

– init (ServletConfig servletConfig) in Servlet interface

57

So how do you access ServletContext object? Within your servlet and servlet filter code, you can call getServletContext() method.
Also if you have ServletConfig object, you can also retrieve the ServletContext object from it.

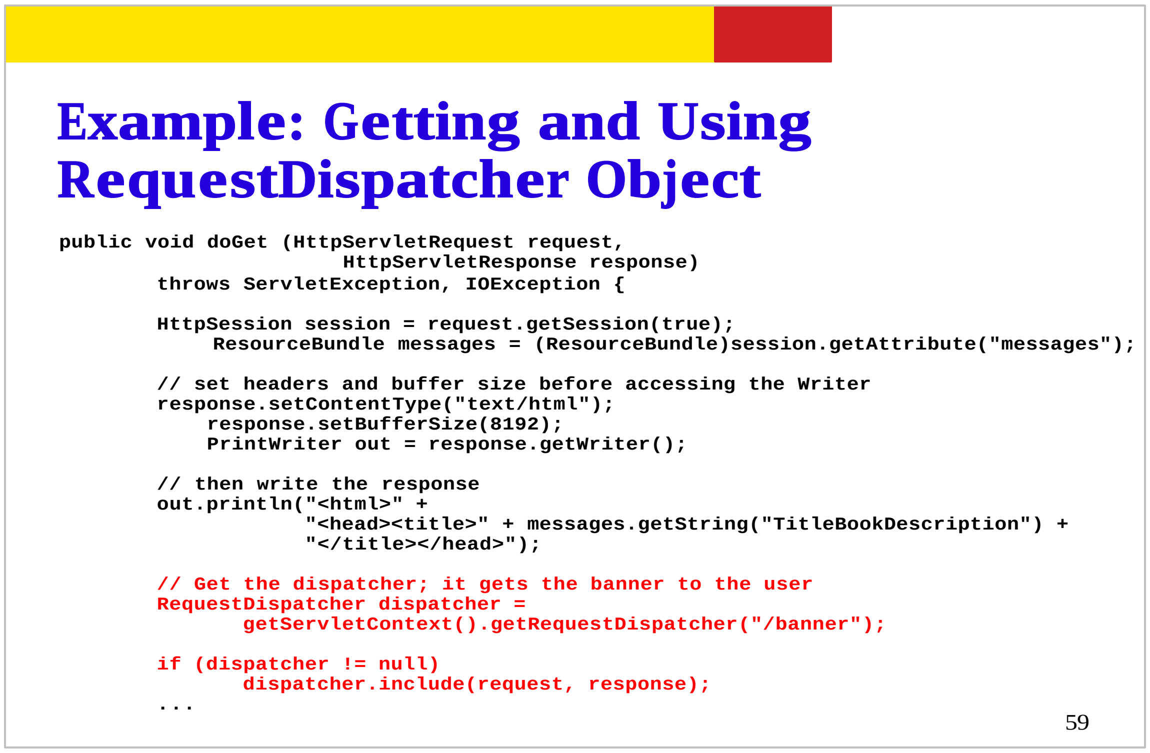


Example: Getting Attribute Value from ServletContext

```
public class CatalogServlet extends HttpServlet {
  private BookDB bookDB;
  public void init() throws ServletException {
    // Get context-wide attribute value from
    // ServletContext object
    bookDB = (BookDB)getServletContext().
                     getAttribute("bookDB");
    if (bookDB == null) throw new
      UnavailableException("Couldn't get database.");
  }
}
```

58

This is an servlet example code in which the value of a context-wide attribute called bookDB is retrieved in init() method of the servlet class.



Example: Getting and Using RequestDispatcher Object

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

        HttpSession session = request.getSession(true);
             ResourceBundle messages = (ResourceBundle)session.getAttribute("messages");

        // set headers and buffer size before accessing the Writer
        response.setContentType("text/html");
            response.setBufferSize(8192);
            PrintWriter out = response.getWriter();

        // then write the response
        out.println("<html>" +
                    "<head><title>" + messages.getString("TitleBookDescription") +
                    "</title></head>");

        // Get the dispatcher; it gets the banner to the user
        RequestDispatcher dispatcher =
               getServletContext().getRequestDispatcher("/banner");

        if (dispatcher != null)
               dispatcher.include(request, response);
        ...
```

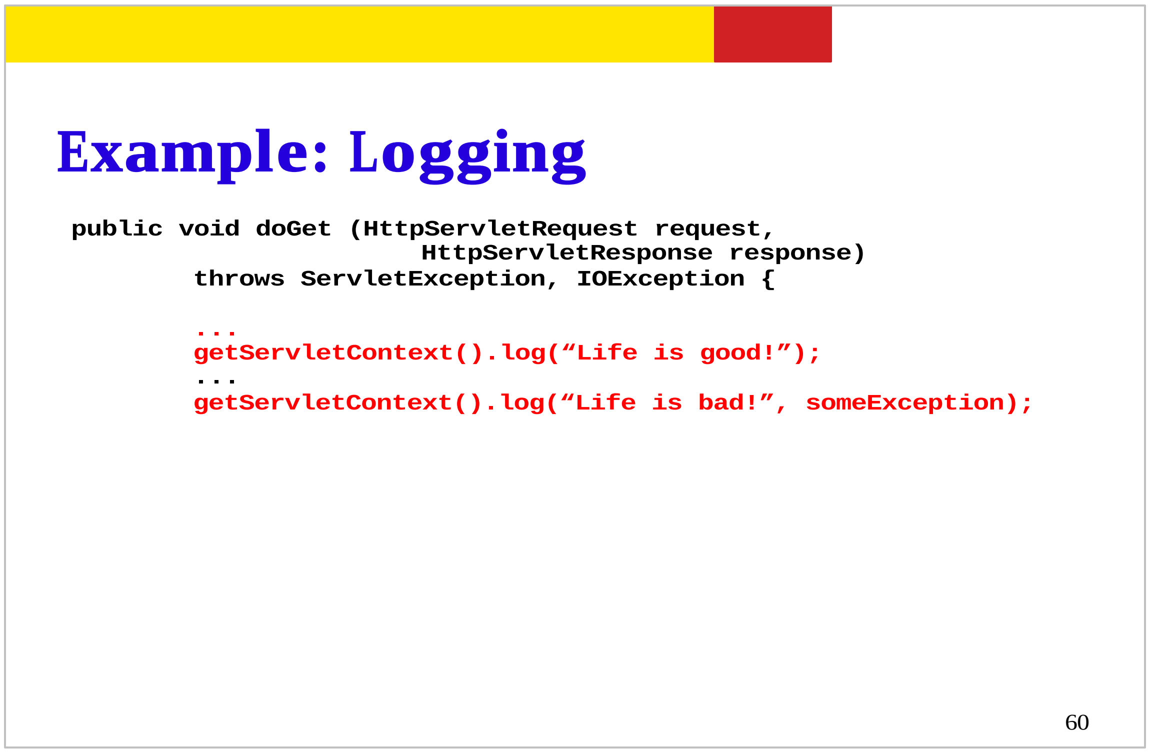
59

This is an example code of doGet() method in which RequestDispatcher object for another web component is retrieved from ServletContext object. Once you have RequestDispatcher object for another web component, you can dispatch (either forwarding or including) the request to that web component using the dispatcher. Please note that HTTP request and response are passed as parameters.   A few more details are:

- /banner is servlet name of the BannerServlet.
- You will find BannerServlet.java under <jwasp_install>/tutorial/examples/web/bookstore1/src
- web.xml file for bookstore1 application can be found under <jwasp_install>/tutorial/examples/web/bookstore1/web/WEB-INF and it has a mapping between servlet name and actual servlet class as following

```
  <servlet>
     <servlet-name>banner</servlet-name>
     <display-name>banner</display-name>
     <description>no description</description>
     <servlet-class>BannerServlet</servlet-class>
   </servlet>
```

-RequestDispatcher object is a wrapper for a particular web resource such as servlet
-RequestDispatcher object can be obtained by calling getRequestDispatcher(<servlet-name-to-forward-or-include>) method of ServletContext.



Example: Logging

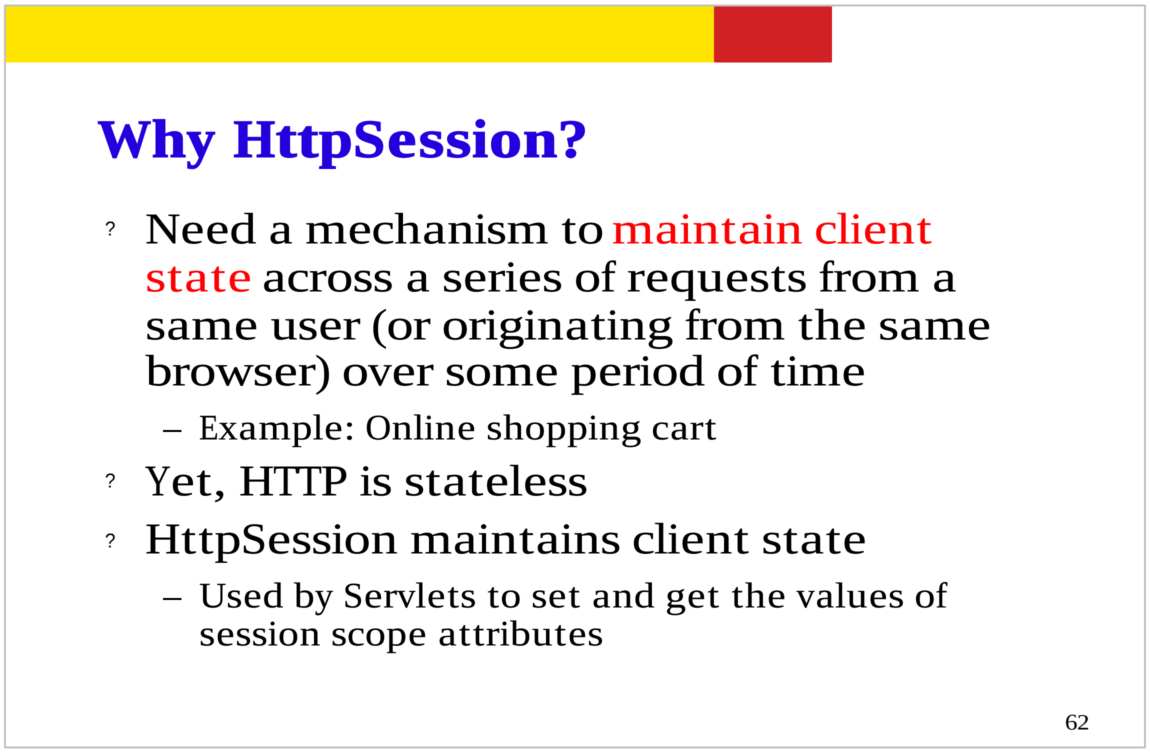
```
public void doGet (HttpServletRequest request,
                      HttpServletResponse response)
        throws ServletException, IOException {

        ...
        getServletContext().log(“Life is good!”);
        ...
        getServletContext().log(“Life is bad!”, someException);
```

This example code shows how to use log() method of the ServletContext object. The name and type of the log file is container specific.

Session (HttpSession)

We will talk more on HTTPSession later in “Session Tracking”

Now let's talk about Session scope object. Understanding Session scope object is important since Session object is the most frequently used scope object in your code.



Why HttpSession?

? Need a mechanism to maintain client state across a series of requests from a same user (or originating from the same browser) over some period of time
  – Example: Online shopping cart

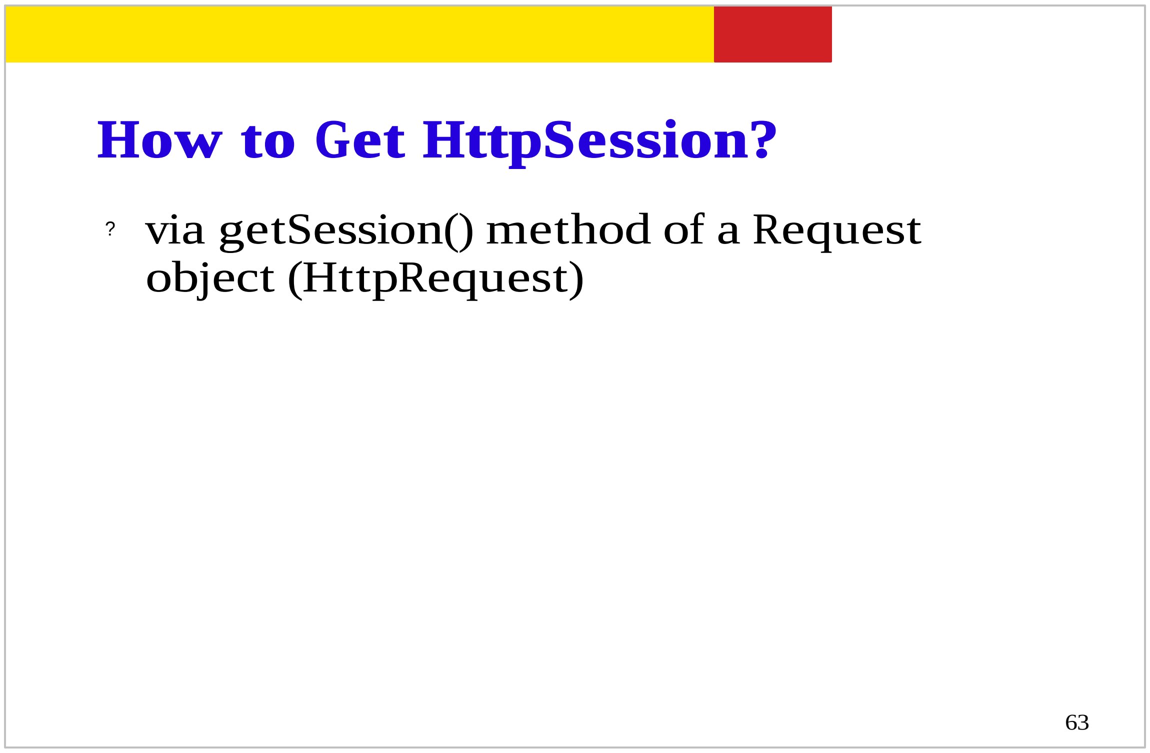
? Yet, HTTP is stateless

? HttpSession maintains client state
  – Used by Servlets to set and get the values of session scope attributes

62

So why do we need something like HTTPSession object?

First we need a mechanism to maintain client state information across a series of HTTP requests from a same user over a period of time. And good example is online shopping cart servlet. In online shopping cart servlet, the items a user has put in his/her shopping cart has to be preserved. Yet, HTTP protocol is stateless. HTTPSession object can maintain the client state in the form of attributes and the attributes remain in session scope.

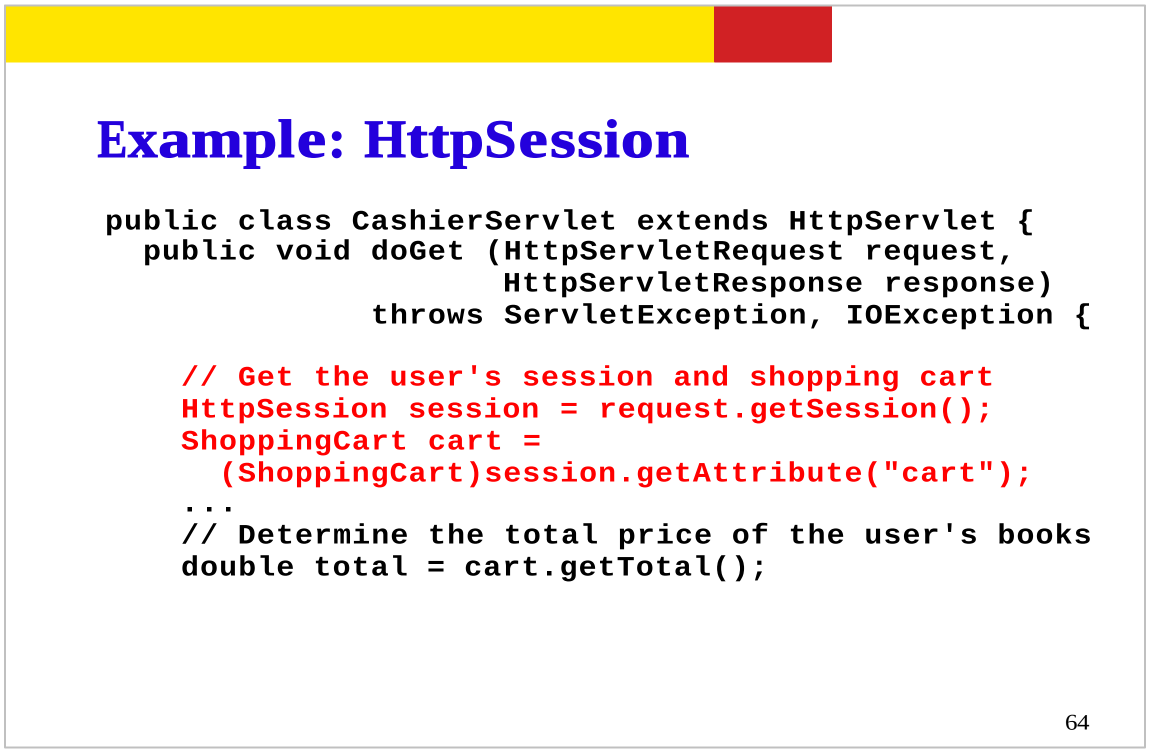


How to Get HttpSession?

- via getSession() method of a Request object (HttpRequest)

63

So how do you get HTTPSession object in your Servlet code? HTTPRequest object that is passed to your servlet code as an input parameter of service() or doXXX() methods has a method called getSession() method.

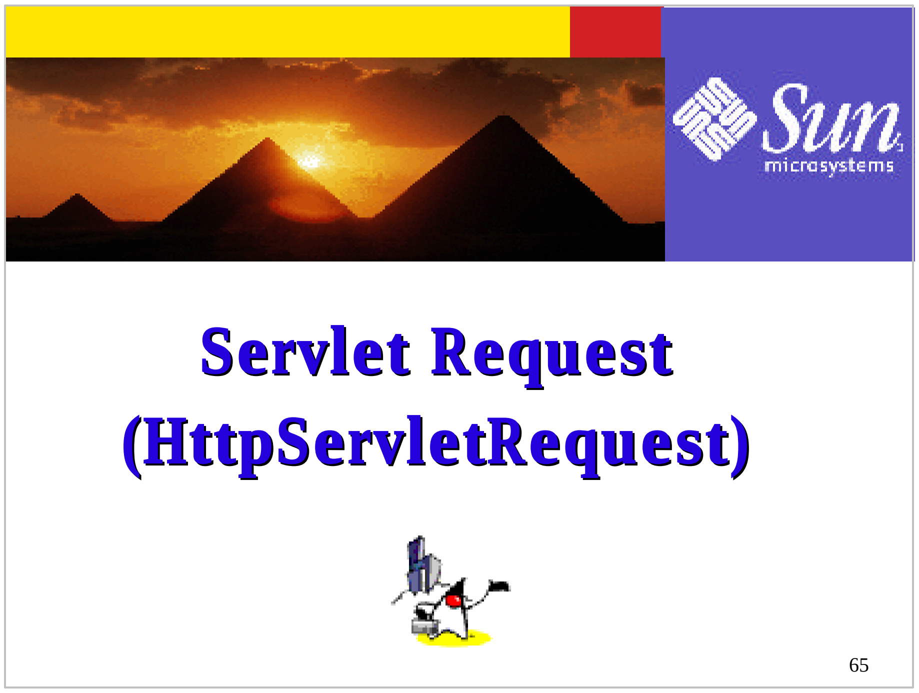


Example: HttpSession

```
public class CashierServlet extends HttpServlet {
  public void doGet (HttpServletRequest request,
                    HttpServletResponse response)
             throws ServletException, IOException {

    // Get the user's session and shopping cart
    HttpSession session = request.getSession();
    ShoppingCart cart =
      (ShoppingCart)session.getAttribute("cart");
    ...
    // Determine the total price of the user's books
    double total = cart.getTotal();
```

64

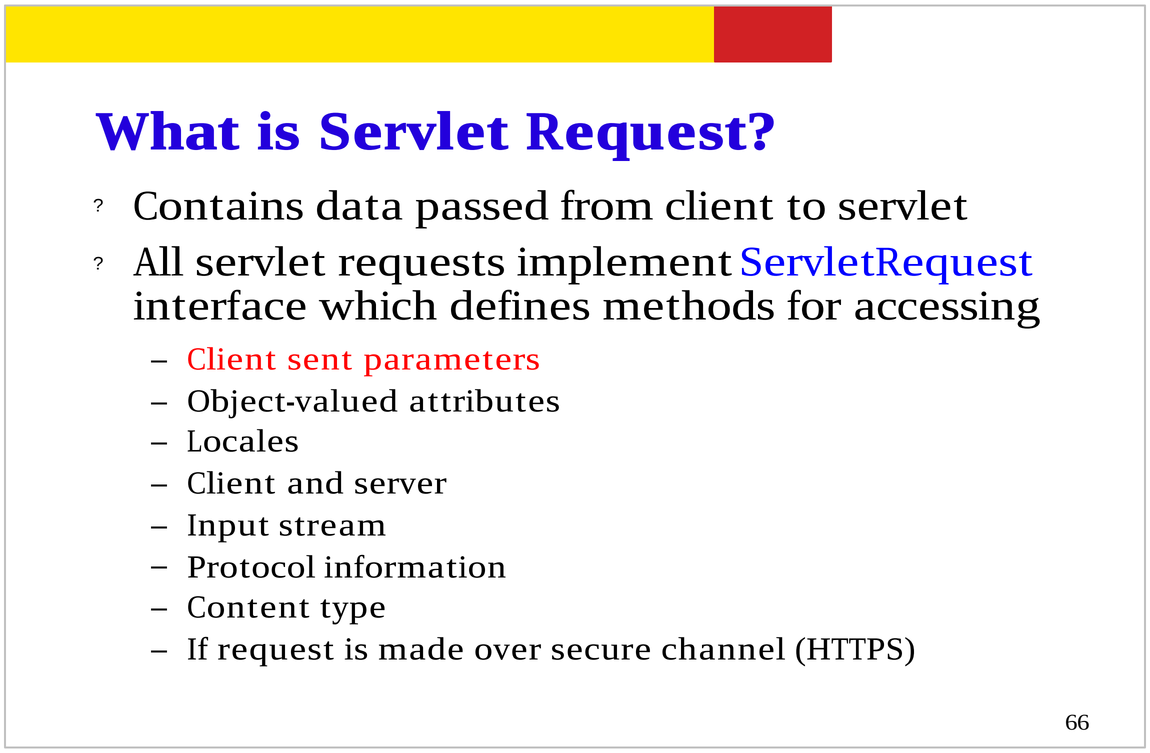
This is an example servlet code in which getting HTTPSession object is demonstrated.



Servlet Request (HttpServletRequest)

65

Now it is time to talk about the internal structure of client request, specifically HTTP request, which is represented as HttpServletRequest object. Please do remember it is the responsibility of the container to create the HttpServletRequest object from the incoming HTTP request.



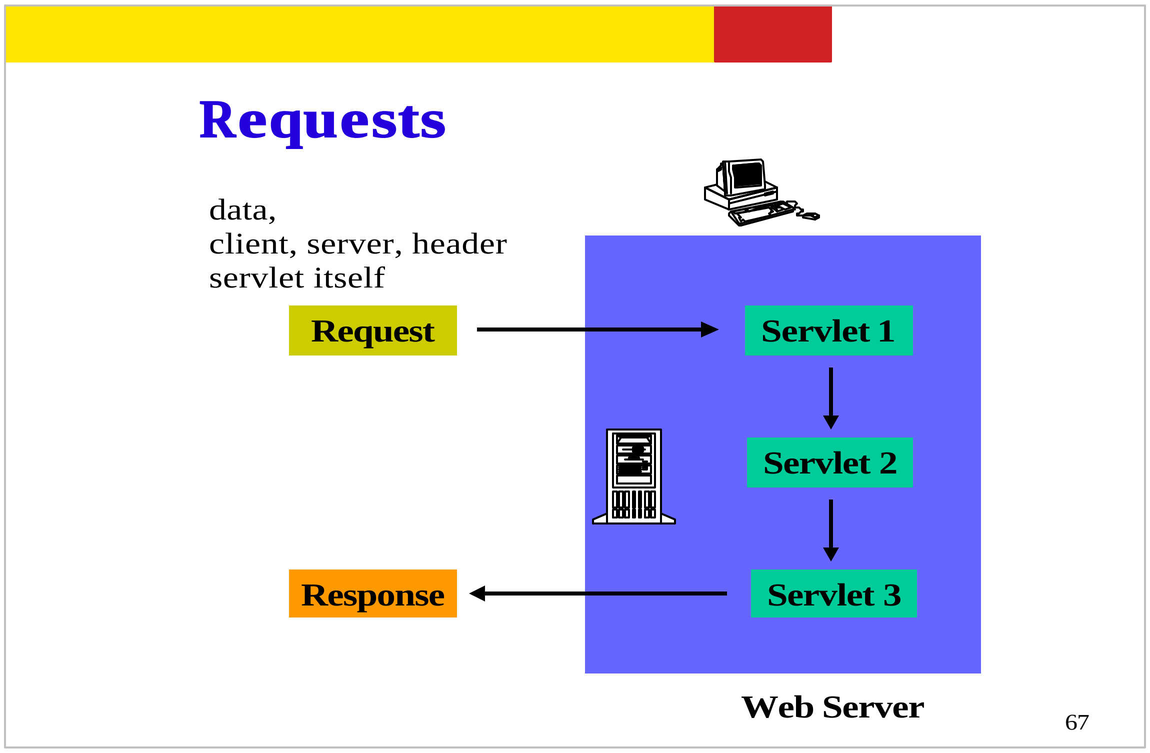
What is Servlet Request?

- Contains data passed from client to servlet
- All servlet requests implement ServletRequest interface which defines methods for accessing
  - Client sent parameters
  - Object-valued attributes
  - Locales
  - Client and server
  - Input stream
  - Protocol information
  - Content type
  - If request is made over secure channel (HTTPS)

66

So what is servlet request? A servlet request contains data passed from client (HTTP browser) to a servlet.

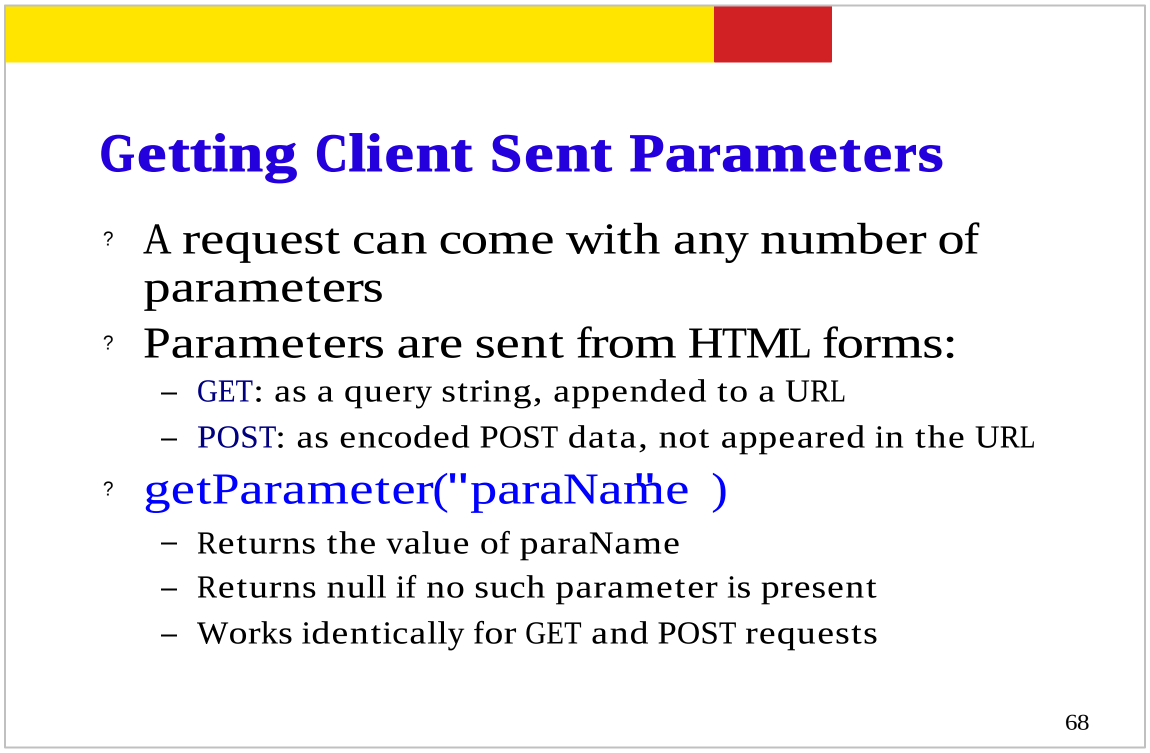
All servlet requests implement ServletRequest interface, which contains methods for access various information.



Requests

data,
client, server, header
servlet itself

This picture shows a simplified view on how client request is received by a chain of servlets and then a response message is returned to the client.



Getting Client Sent Parameters

? A request can come with any number of parameters

? Parameters are sent from HTML forms:

– GET: as a query string, appended to a URL

– POST: as encoded POST data, not appeared in the URL

? getParameter("paraName")

– Returns the value of paraName

– Returns null if no such parameter is present

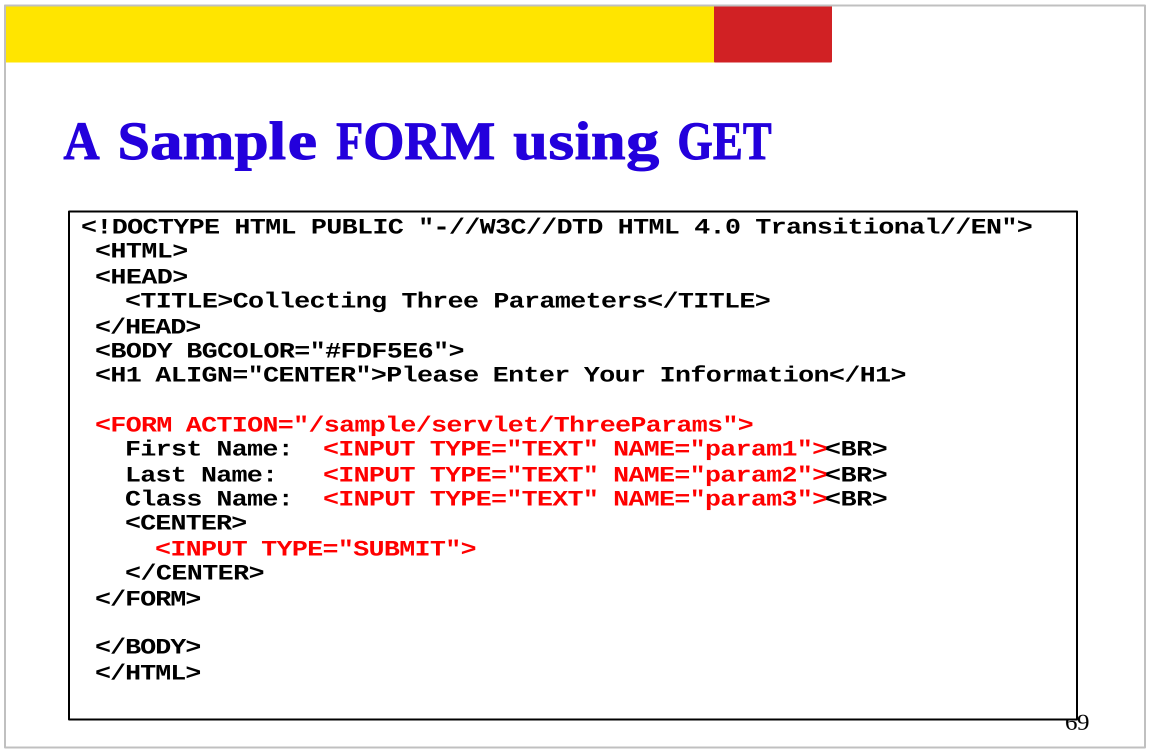
– Works identically for GET and POST requests

68

How do you get parameters client sent parameters? (I sometimes call these parameters as "user-entered" parameters as well.)

Please note that a request can come with any number of client sent parameters. Now parameters are sent from client in two different forms: HTTP GET and HTTP POST. When a parameter is sent via HTTP GET message, the name and value pairs of the parameters are sent as appendix to the the URL. When it is sent via HTTP POST message, the name and value pairs of the parameters are sent as user data.

Now ServletRequest interface has getParameter() method which can be used in your code to get the value of the parameter. This method works the same for both HTTP GET and POST requests.



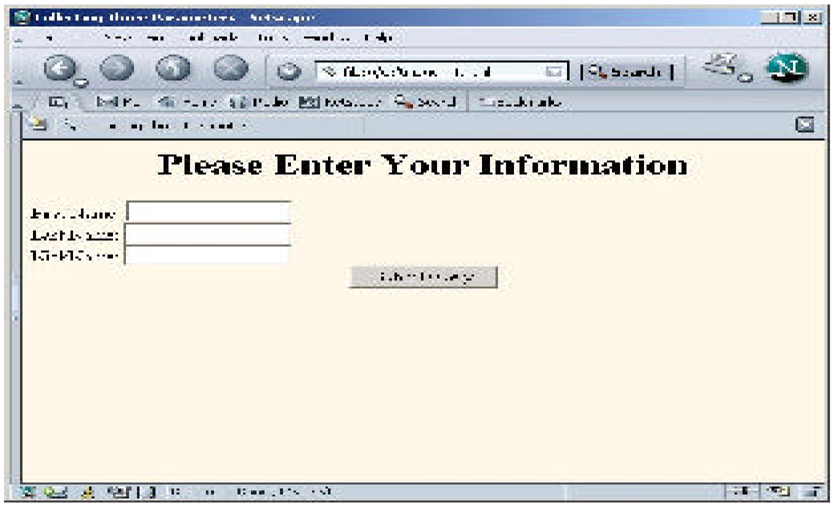
A Sample FORM using GET

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
 <HTML>
 <HEAD>
   <TITLE>Collecting Three Parameters</TITLE>
 </HEAD>
 <BODY BGCOLOR="#FDF5E6">
 <H1 ALIGN="CENTER">Please Enter Your Information</H1>

 <FORM ACTION="/sample/servlet/ThreeParams">
   First Name:  <INPUT TYPE="TEXT" NAME="param1"><BR>
   Last Name:   <INPUT TYPE="TEXT" NAME="param2"><BR>
   Class Name:  <INPUT TYPE="TEXT" NAME="param3"><BR>
   <CENTER>
     <INPUT TYPE="SUBMIT">
   </CENTER>
 </FORM>

 </BODY>
 </HTML>
```

69

This is an example HTML page in which values of three parameters are collected as From values and then transported to the servlet through HTTP GET message. We will see actual HTML page in the following slide.

A Sample FORM using GET



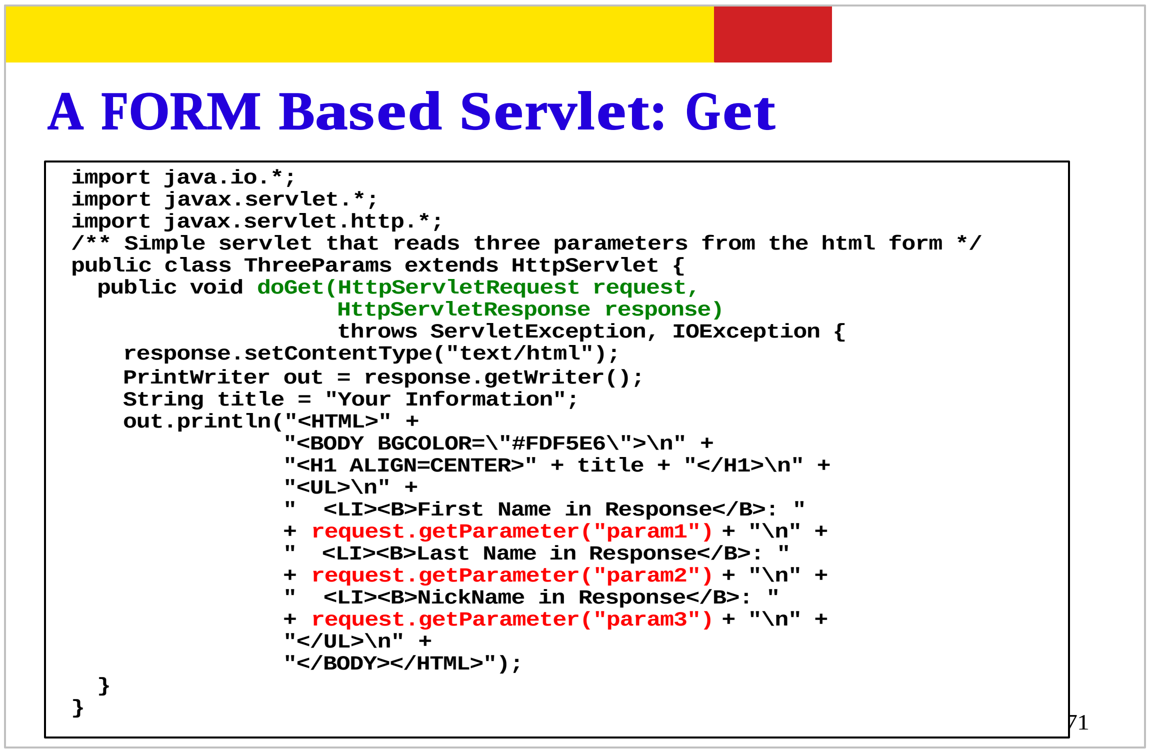
Please Enter Your Information

First Name:
Last Name:
Middle Name:

Submit Query

70

So this is the actual display of the previous HTML page.

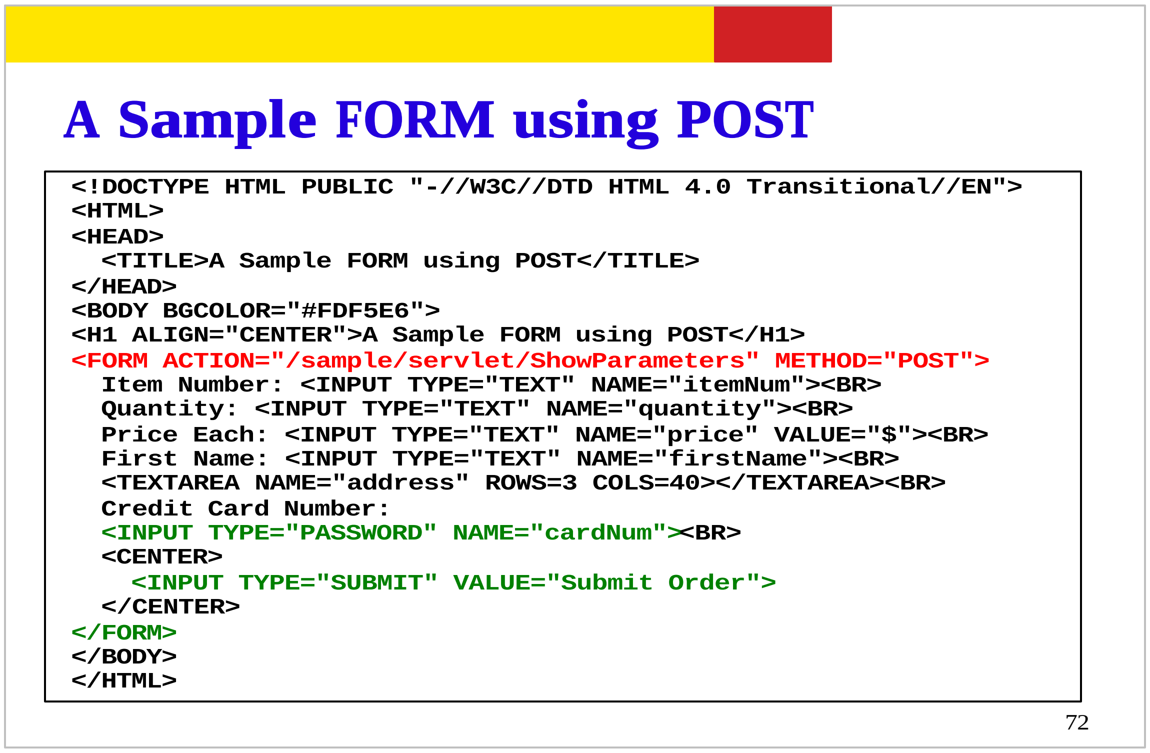


A FORM Based Servlet: Get

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
/** Simple servlet that reads three parameters from the html form */
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 = "Your Information";
    out.println("<HTML>" +
                "<BODY BGCOLOR=\"#FDF5E6\">\n" +
                "<H1 ALIGN=CENTER>" + title + "</H1>\n" +
                "<UL>\n" +
                "  <LI><B>First Name in Response</B>: "
                + request.getParameter("param1") + "\n" +
                "  <LI><B>Last Name in Response</B>: "
                + request.getParameter("param2") + "\n" +
                "  <LI><B>NickName in Response</B>: "
                + request.getParameter("param3") + "\n" +
                "</UL>\n" +
                "</BODY></HTML>");
  }
}
```

71

This example code shows how to extract the values of the three parameters that have been entered by an end-user.



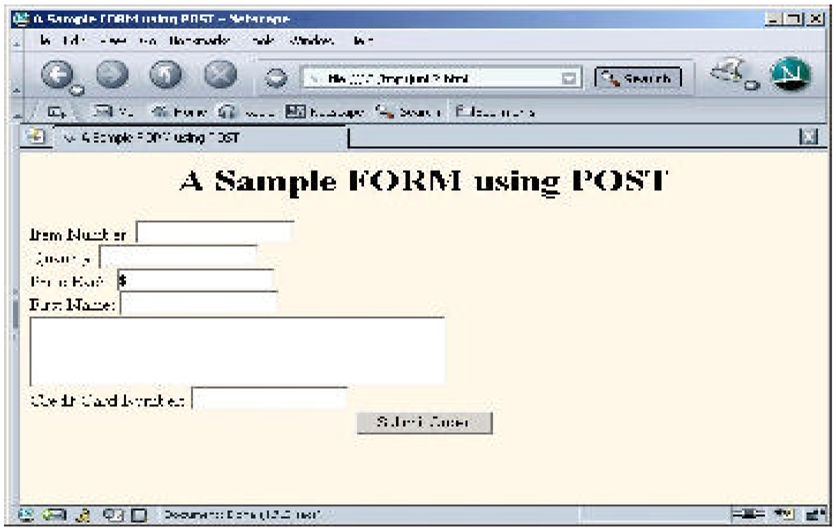
A Sample FORM using POST

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
  <TITLE>A Sample FORM using POST</TITLE>
</HEAD>
<BODY BGCOLOR="#FDF5E6">
<H1 ALIGN="CENTER">A Sample FORM using POST</H1>
<FORM ACTION="/sample/servlet/ShowParameters" METHOD="POST">
  Item Number: <INPUT TYPE="TEXT" NAME="itemNum"><BR>
  Quantity: <INPUT TYPE="TEXT" NAME="quantity"><BR>
  Price Each: <INPUT TYPE="TEXT" NAME="price" VALUE="$"><BR>
  First Name: <INPUT TYPE="TEXT" NAME="firstName"><BR>
  <TEXTAREA NAME="address" ROWS=3 COLS=40></TEXTAREA><BR>
  Credit Card Number:
  <INPUT TYPE="PASSWORD" NAME="cardNum"><BR>
  <CENTER>
    <INPUT TYPE="SUBMIT" VALUE="Submit Order">
  </CENTER>
</FORM>
</BODY>
</HTML>
```

72

This is the example HTML page in which values of input parameters are collected as HTML Form values, which are then transcribed to the servlet as HTTP POST message.

A Sample FORM using POST



A Sample FORM using POST

Item Number:

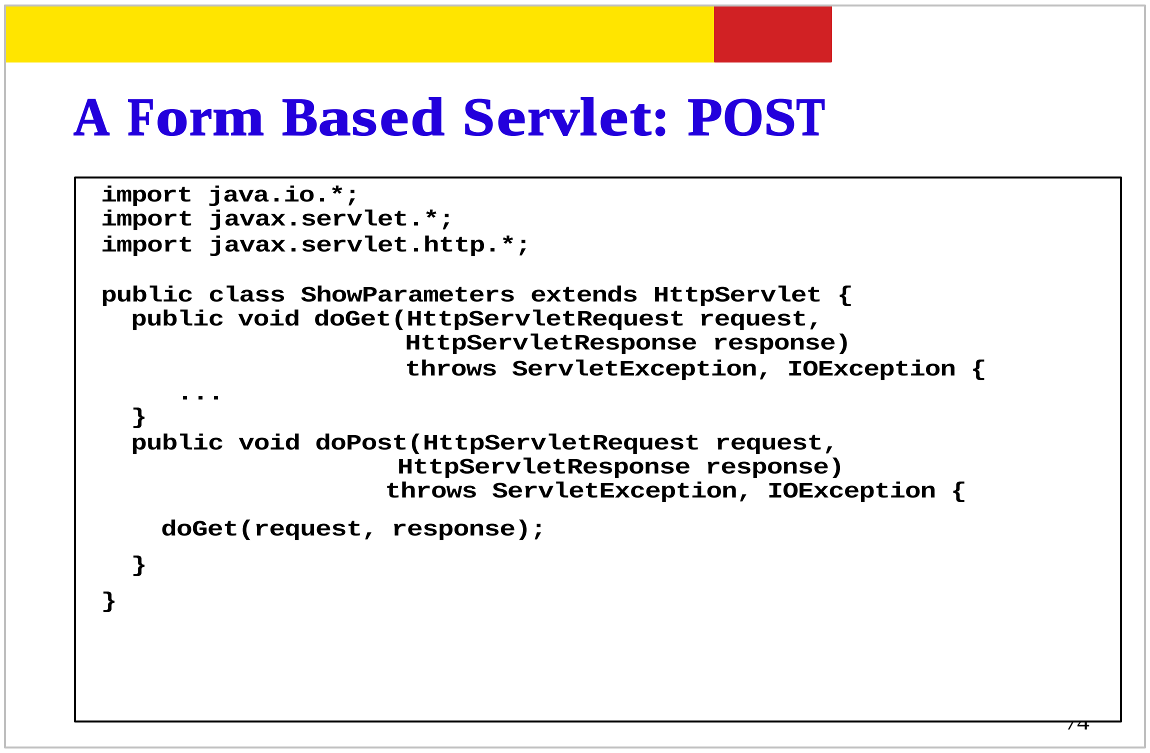
Quantity:

Price Each: $

First Name:

Credit Card Number:

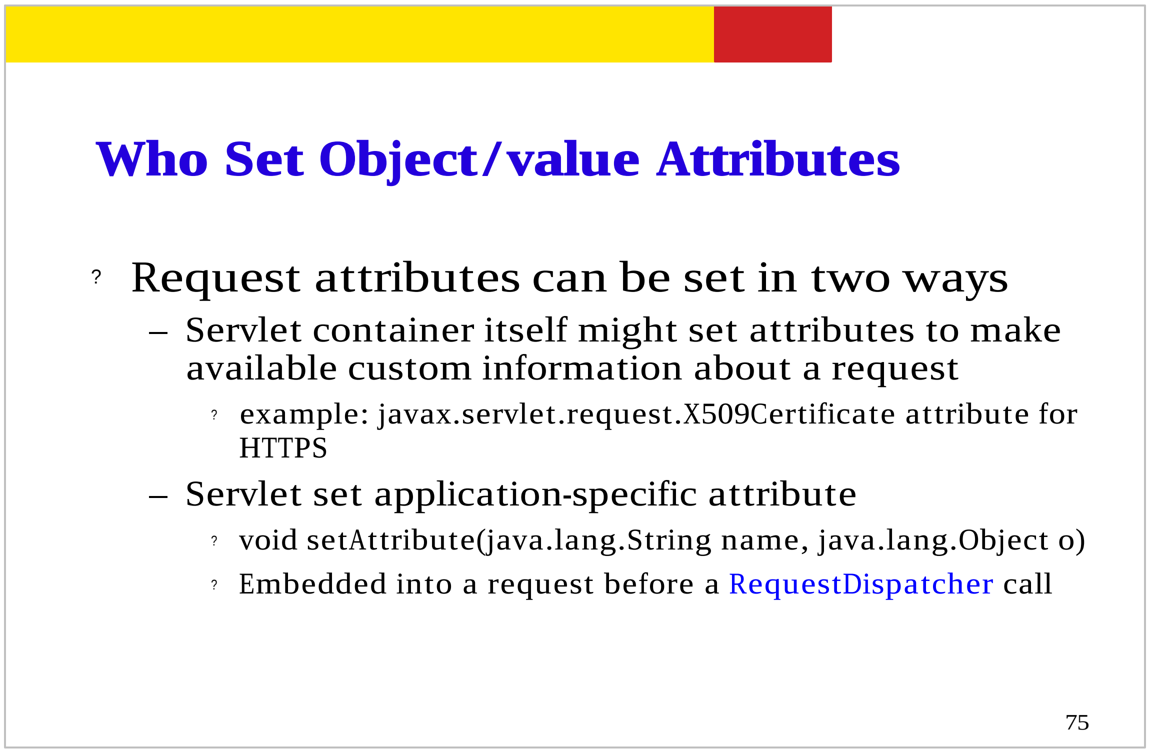
This is display of the HTML page of the previous slide.



A Form Based Servlet: POST

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

public class ShowParameters extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
     ...
  }
  public void doPost(HttpServletRequest request,
                    HttpServletResponse response)
                   throws ServletException, IOException {
    doGet(request, response);
  }
}
```

This is doPost() example code.



Who Set Object/value Attributes

- Request attributes can be set in two ways
  - Servlet container itself might set attributes to make available custom information about a request
    - example: javax.servlet.request.X509Certificate attribute for HTTPS
  - Servlet set application-specific attribute
    - void setAttribute(java.lang.String name, java.lang.Object o)
    - Embedded into a request before a RequestDispatcher call

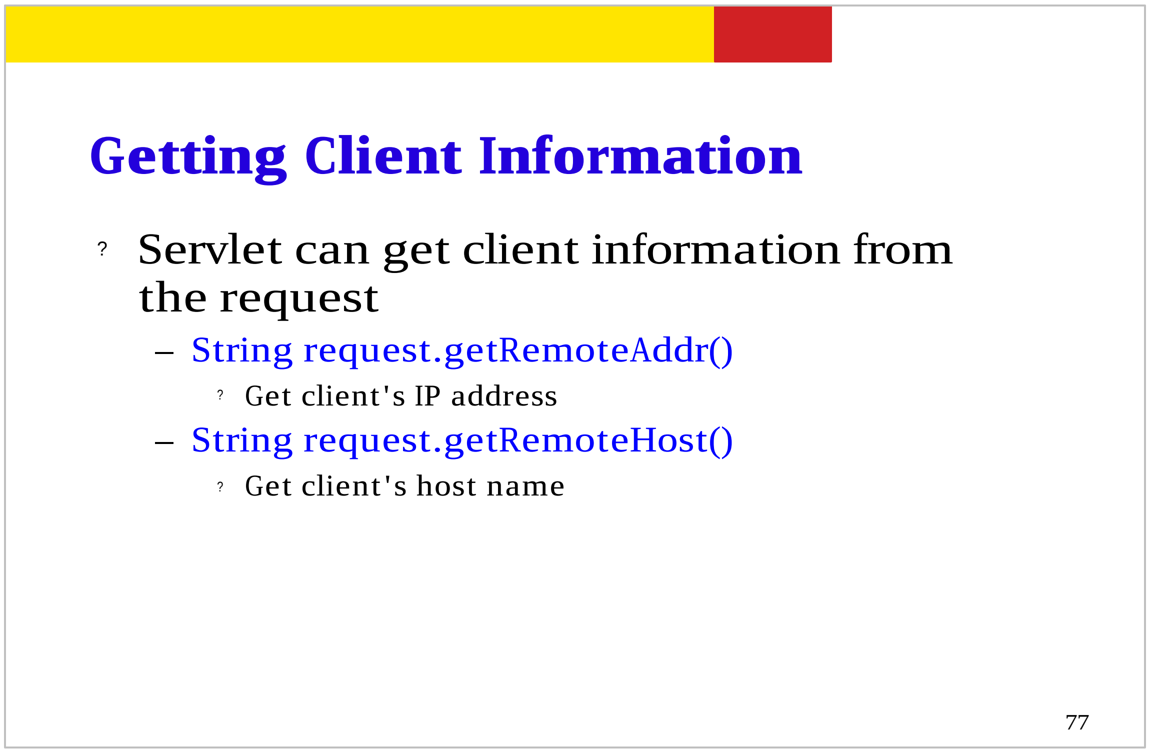
75

Now let's talk about who set the attributes? By the way, the attributes are set as object/value pairs. There are two different ways that the attributes can be set. First, there are certain set of attributes that are set by the container, for example, X509Certificate attribute is set by the container when the container receives HTTPS request. Second, your servlet code can set the custom attributes.

Getting Locale Information

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

      HttpSession session = request.getSession();
      ResourceBundle messages =
          (ResourceBundle)session.getAttribute
  ("messages");

      if (messages == null) {
         Locale locale=request.getLocale();
         messages = ResourceBundle.getBundle(
                "messages.BookstoreMessages", locale);
         session.setAttribute("messages", messages);
      }
```

This is an example code in which messages attribute is retrieved and set with ResourceBundle object.

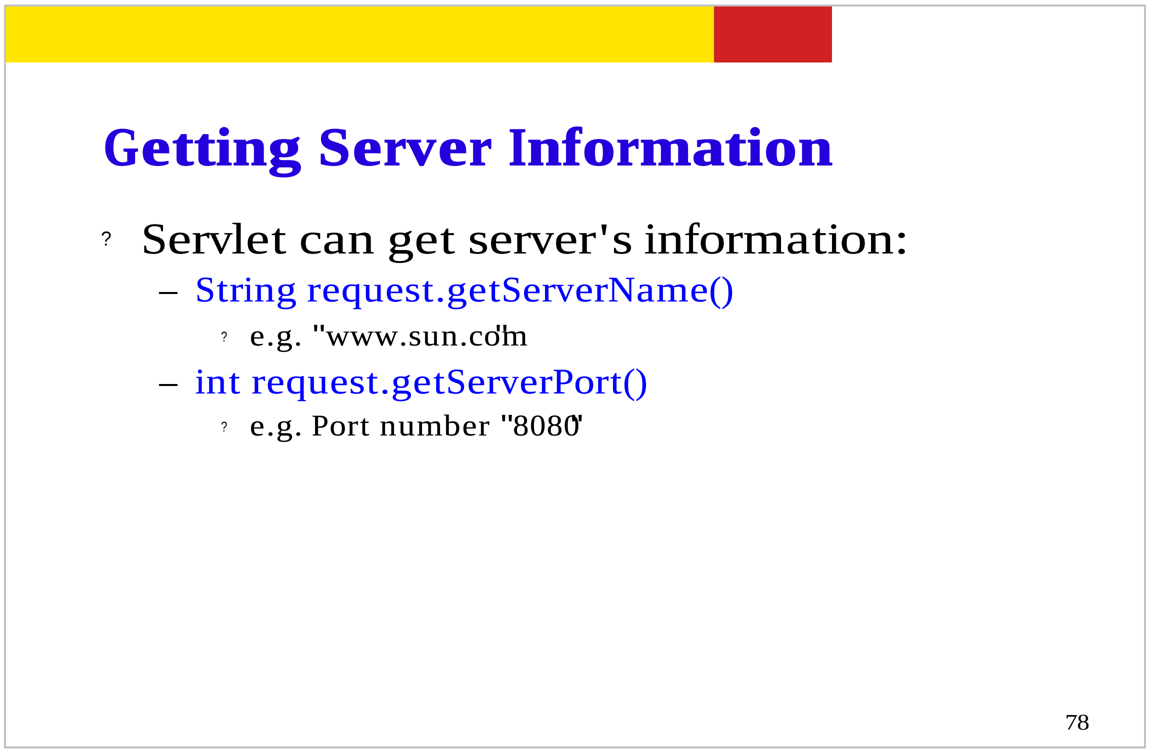


Getting Client Information

- Servlet can get client information from the request
  - String request.getRemoteAddr()
    - Get client's IP address
  - String request.getRemoteHost()
    - Get client's host name

77

In your servlet code, you can also retrieve client specific information such as client IP address or hostname.



Getting Server Information

? Servlet can get server's information:

– String request.getServerName()

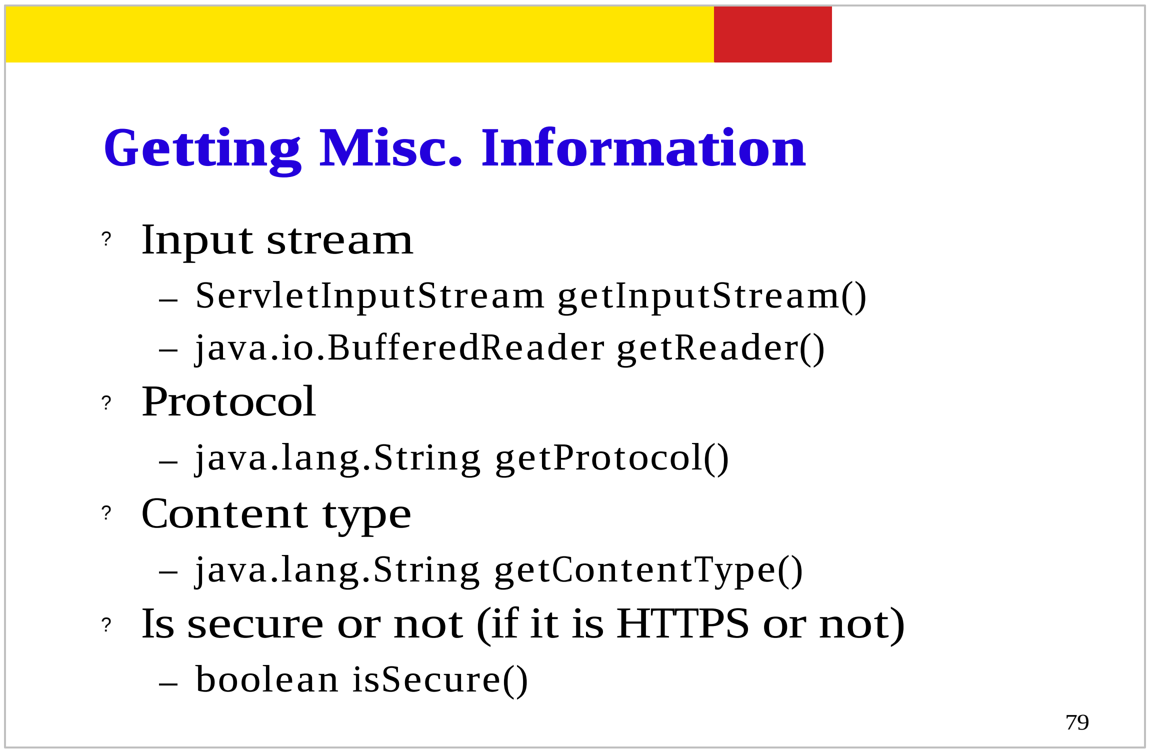
? e.g. "www.sun.com"

– int request.getServerPort()

? e.g. Port number "8080"

78

In your servlet code, you can also retrieve server specific information such as server's DNS-based hostname and port number.



Getting Misc. Information

? Input stream

– ServletInputStream getInputStream()

– java.io.BufferedReader getReader()

? Protocol

– java.lang.String getProtocol()

? Content type

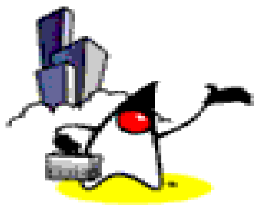
– java.lang.String getContentType()

? Is secure or not (if it is HTTPS or not)

– boolean isSecure()

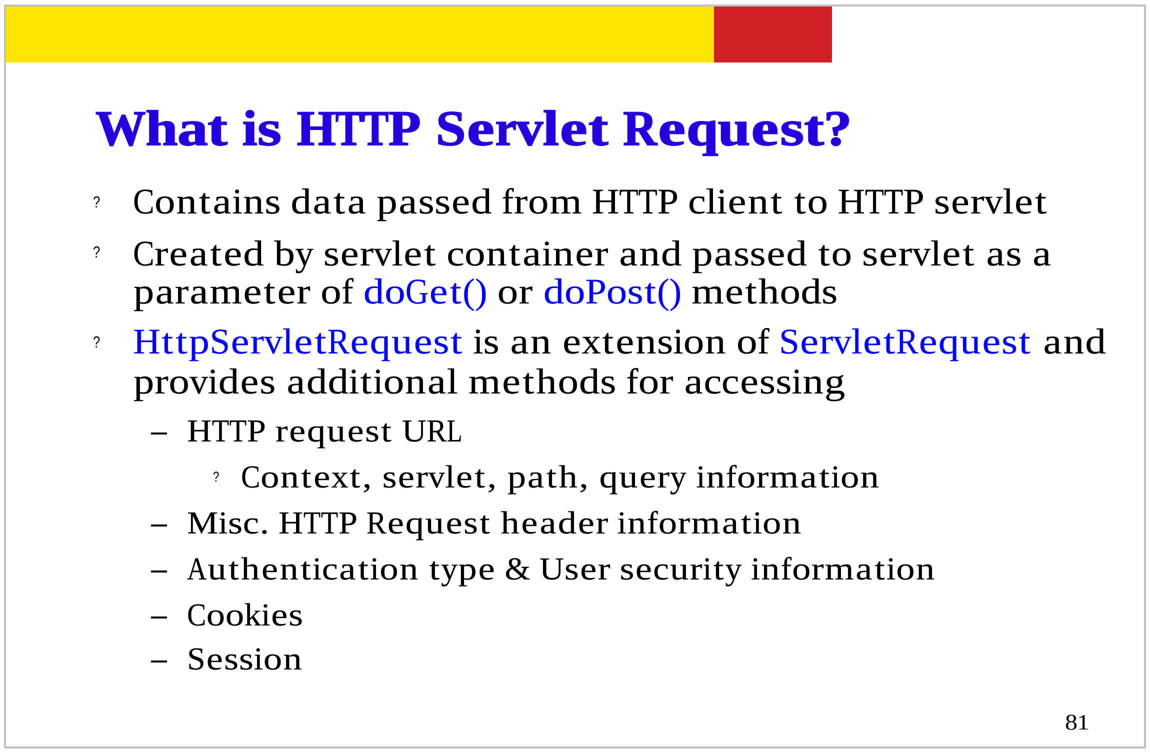
79

In your servlet code, other misc. information can be also retrieved. Examples are input stream object, protocol, content type, and security information.



HTTPServletRequest

Now let's talk about HTTPServletRequest.



What is HTTP Servlet Request?

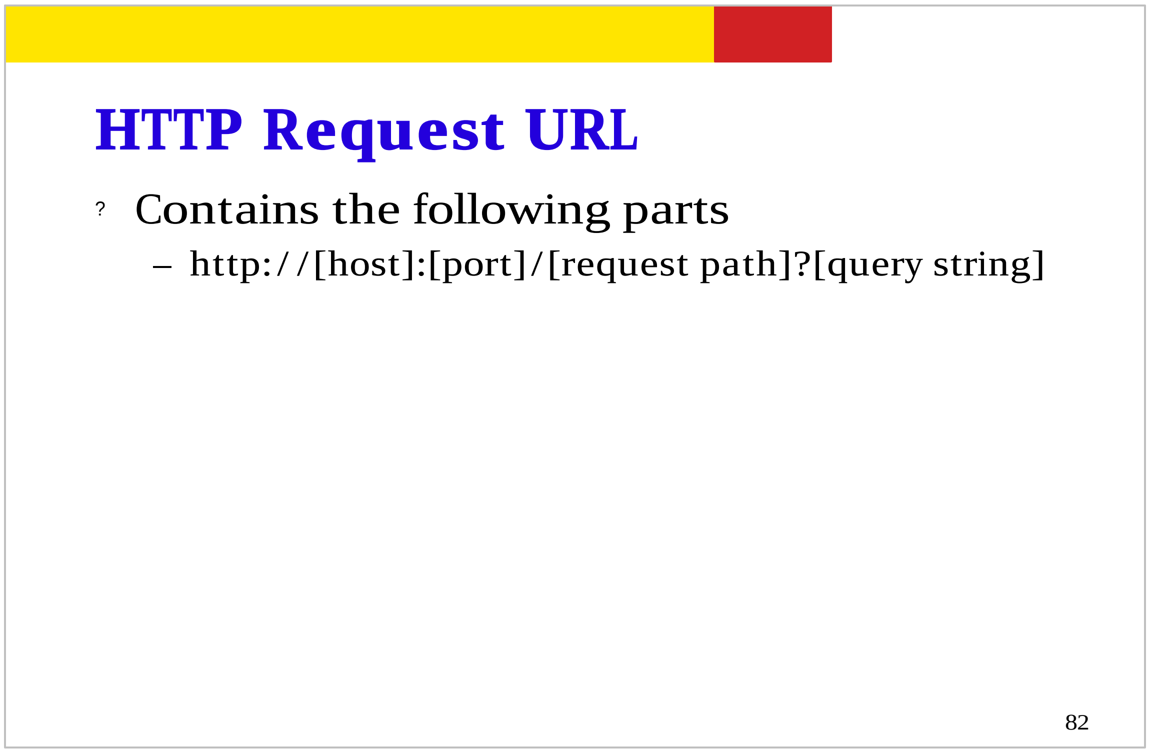
- Contains data passed from HTTP client to HTTP servlet
- Created by servlet container and passed to servlet as a parameter of doGet() or doPost() methods
- HttpServletRequest is an extension of ServletRequest and provides additional methods for accessing
  - HTTP request URL
    - Context, servlet, path, query information
  - Misc. HTTP Request header information
  - Authentication type & User security information
  - Cookies
  - Session

81

What is HTTP Servlet Request? It contains data passed from HTTP client to HTTP servlet.

The HTTP Servlet request is represented by HTTPServletRequest object which is created by the container and then passed to the servlet as a parameter of doGet() or doPost() methods.

HTTPServletRequest type is an extension of ServletRequest Java interface type and provides the additional methods for accessing the information mentioned above, for example, HTTP request URL, HTTP header entries, authentication and user security information, and session information, and so on.

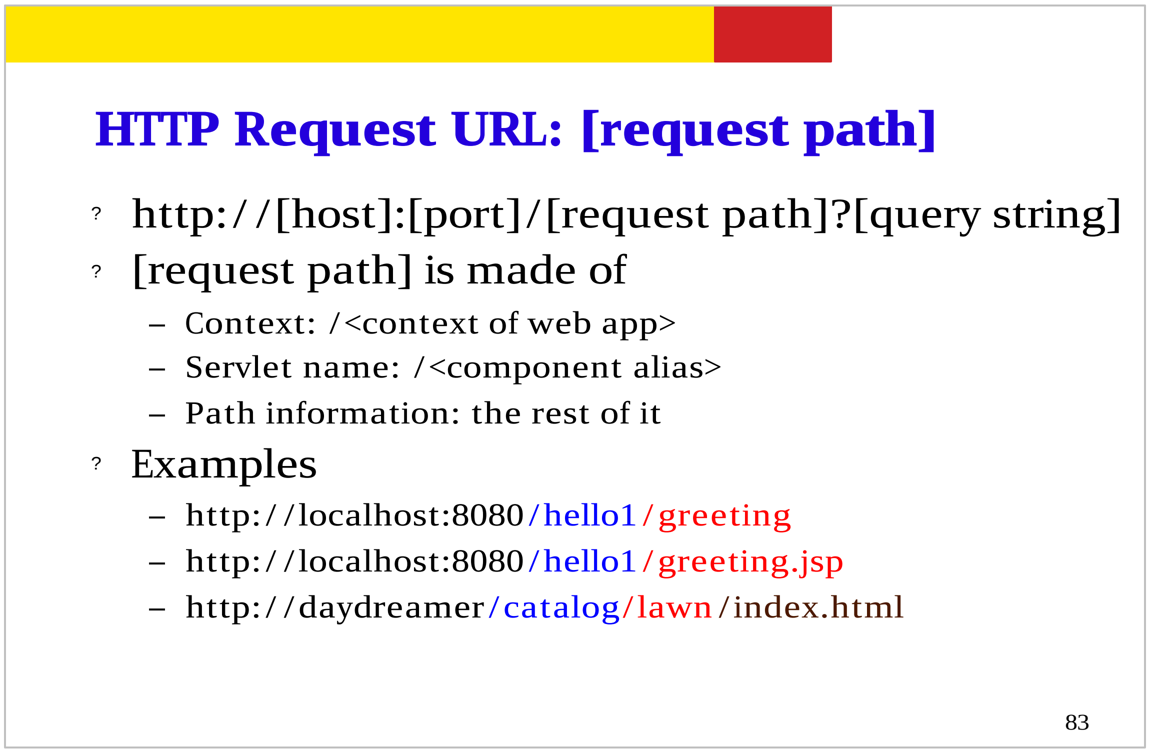


HTTP Request URL

- Contains the following parts
  - http://[host]:[port]/[request path]?[query string]

82

Now let's talk about HTTP request URL first. HTTP request URL information is passed as part of HTTP request. And the URL is made of hostname, port number, request path and query string. Now let's talk about them in a bit more detail.



HTTP Request URL: [request path]

- http://[host]:[port]/[request path]?[query string]
- [request path] is made of
  - Context: /<context of web app>
  - Servlet name: /<component alias>
  - Path information: the rest of it
- Examples
  - http://localhost:8080/hello1/greeting
  - http://localhost:8080/hello1/greeting.jsp
  - http://daydreamer/catalog/lawn/index.html

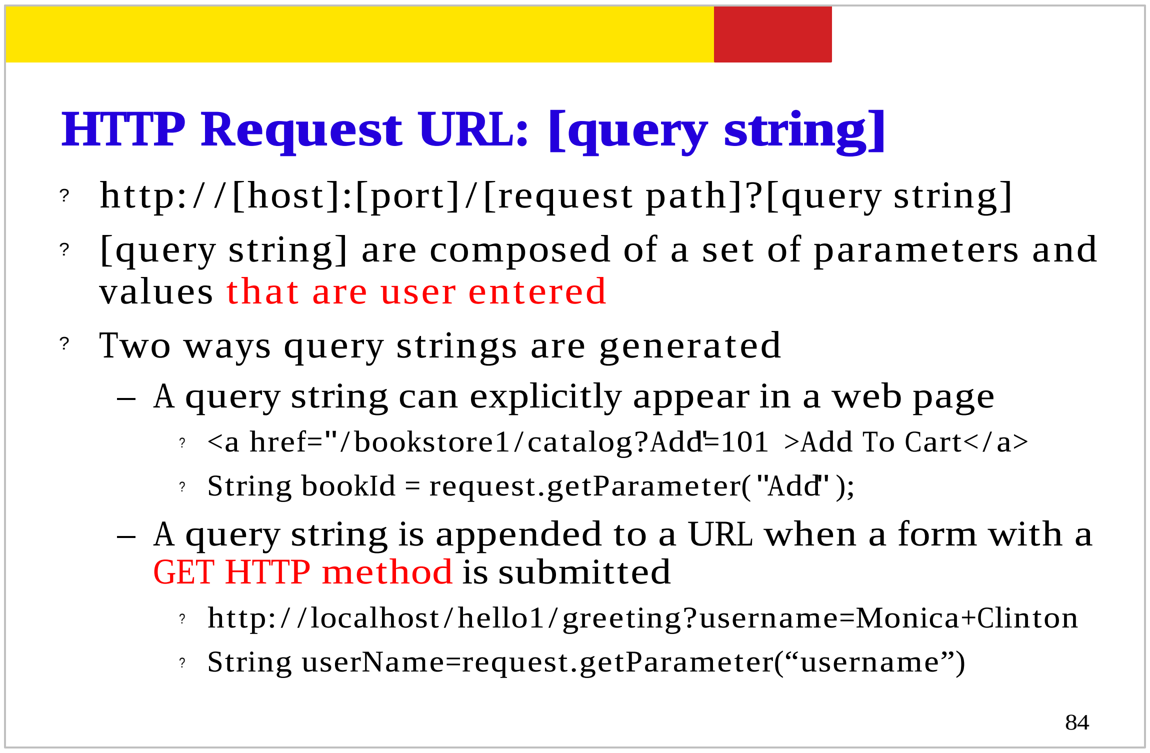
83

Now let's talk about request path first. The request path is made of three things: context path, servlet path, and path information.

The context is the context of the web application that is defined by the deployer of the web application. As we talked about in the web application architecture presentation, every web application has a corresponding context. For those of you who forgot what a context is, it is basically a given name to a root directory of your web application. The servlet name is the alias of your web component, which is again defined in the web.xml deployment descriptor.

The path information is the rest of the URL. So examples for the hello1 web application, a user will either specify http://localhost:8080/hello1/greeting for servlet or http://localhost:8080/hello1/greeting.jsp for JSP page. In these cases, the /hello1 is the context and /greeting or /greeting.jsp are servlet path. And since there is nothing after that, the path information is null.

Now one thing you need to remember is that the request path to servlet/JSP alias mapping is specified in the web.xml file. This is how web container knows which servlet to pass the HTTP request to.



HTTP Request URL: [query string]

? http://[host]:[port]/[request path]?[query string]

? [query string] are composed of a set of parameters and values that are user entered

? Two ways query strings are generated

– A query string can explicitly appear in a web page

? <a href="/bookstore1/catalog?Add"=101 >Add To Cart</a>

? String bookId = request.getParameter("Add");

– A query string is appended to a URL when a form with a GET HTTP method is submitted

? http://localhost/hello1/greeting?username=Monica+Clinton

? String userName=request.getParameter(“username”)

84

Now let's talk about the query string. The query string is composed of a set of parameters and values. These parameters and values are basically user entered data that are being passed from the browser to the servlet as an extension to the URL right after ? mark.

And there are two different ways in which query strings are generated.

First, a query string can be explicitly set in a web page. So when a user opens a web page or selects a component that has a HTTP reference and if that reference contains parameter and value pair, that will generate a query string.

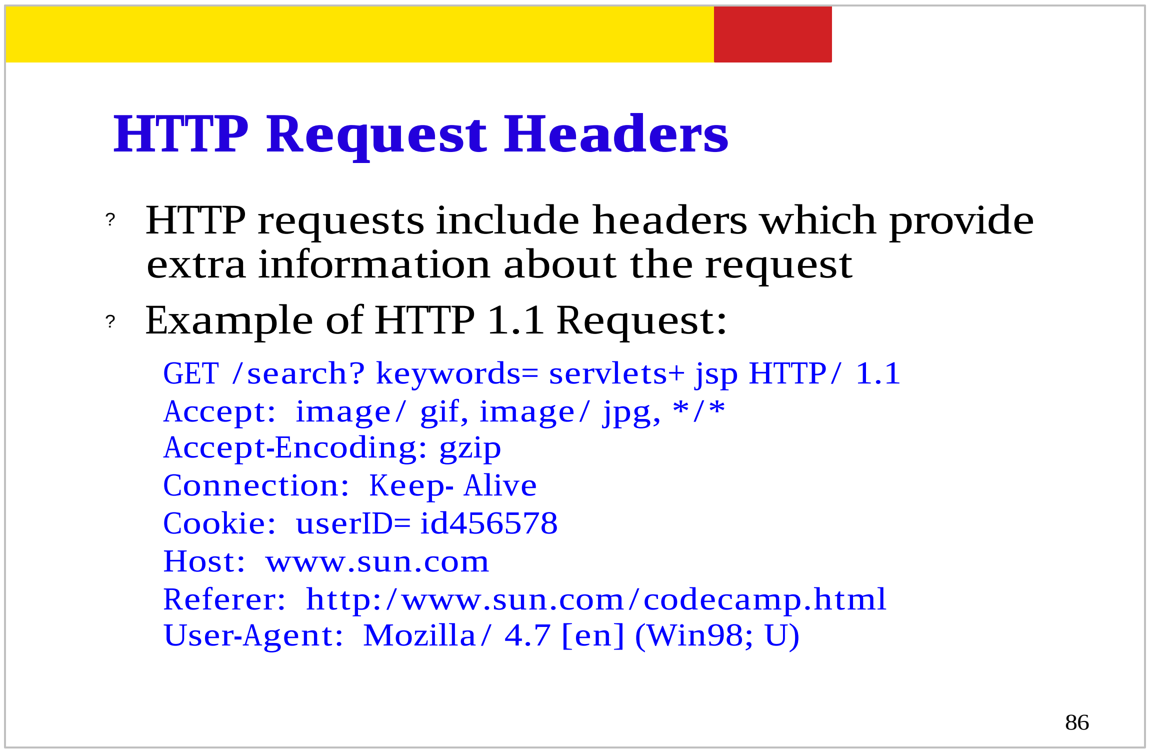
A more pervasive form in which query string is generated is a user entered some data as HTML form value.

Now how do you get the values of these parameters? You use getParameter() method of the HttpServletRequest object.

Context, Path, Query, Parameter Methods

- String getContextPath()
- String getQueryString()
- String getPathInfo()
- String getPathTranslated()

This is the list of methods that are provided as part of HttpServletRequest The getContextPath() method is used to get context information. The getQueryString() method is to get the query string. The getPathInfo() method is used to get the path information. The getPathTranslated() method is to get the path information in real path form.



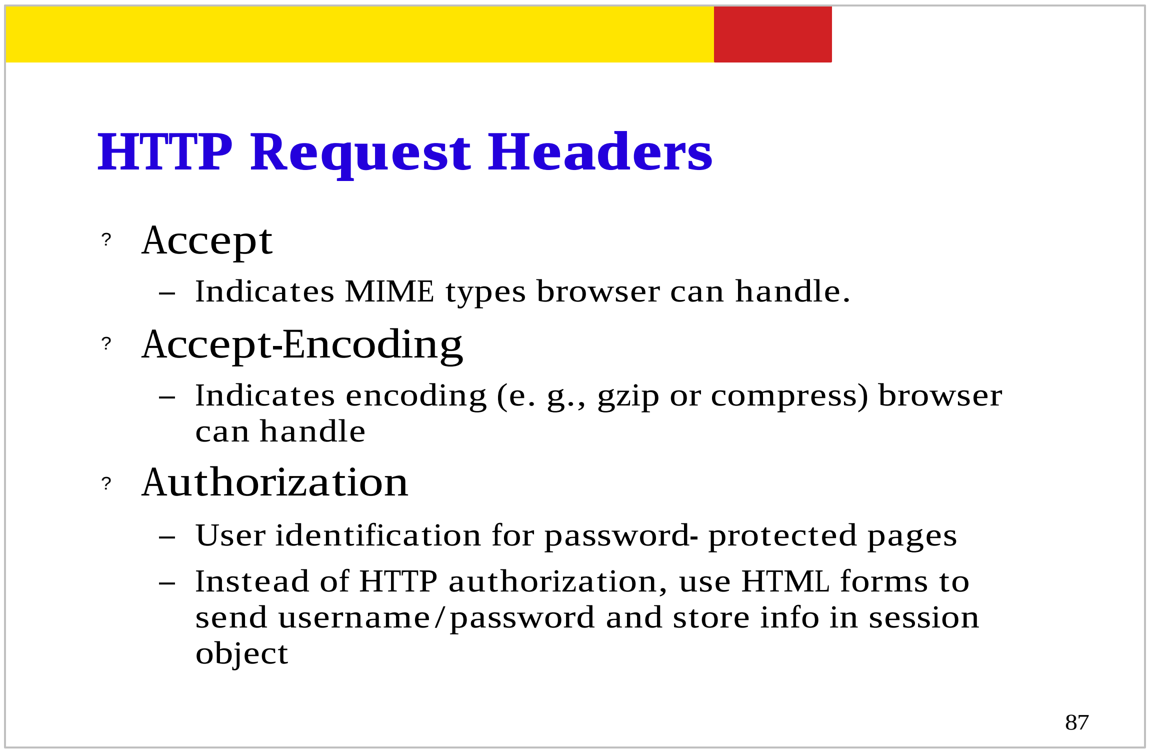
HTTP Request Headers

- HTTP requests include headers which provide extra information about the request
- Example of HTTP 1.1 Request:

```
GET /search? keywords= servlets+ jsp HTTP/ 1.1
Accept:  image/ gif, image/ jpg, */*
Accept-Encoding: gzip
Connection:  Keep- Alive
Cookie:  userID= id456578
Host:  www.sun.com
Referer:  http:/www.sun.com/codecamp.html
User-Agent:  Mozilla/ 4.7 [en] (Win98; U)
```

86

How let's talk about HTTP request header structure. Every HTTP request has a header structure which is used to convey extra information about the request.



HTTP Request Headers

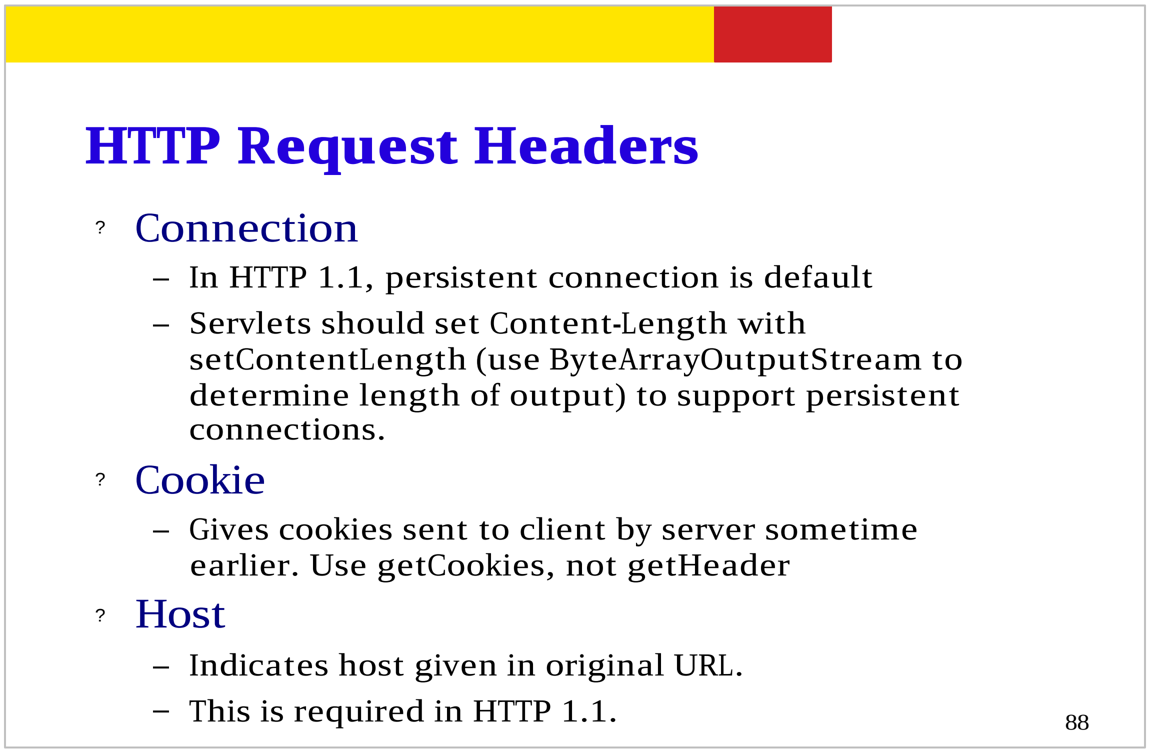
? Accept
- Indicates MIME types browser can handle.

? Accept-Encoding
- Indicates encoding (e. g., gzip or compress) browser can handle

? Authorization
- User identification for password- protected pages
- Instead of HTTP authorization, use HTML forms to send username/password and store info in session object

87

(read the slide)

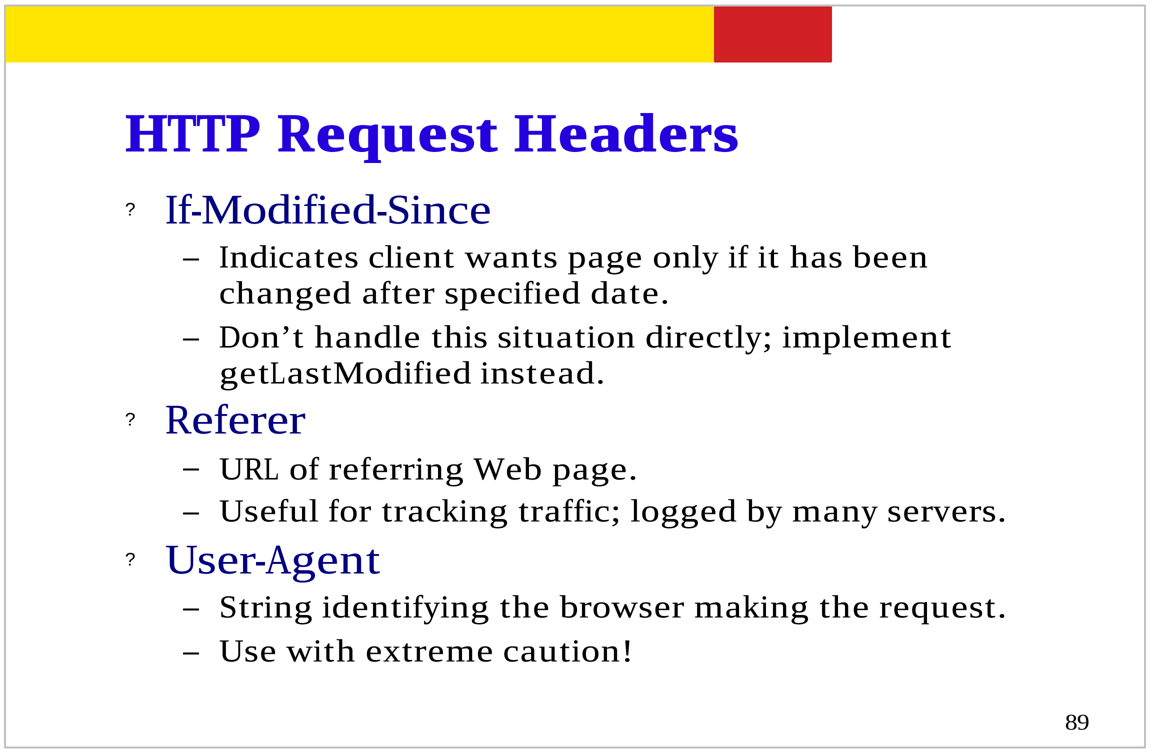


HTTP Request Headers

- Connection
  - In HTTP 1.1, persistent connection is default
  - Servlets should set Content-Length with setContentLength (use ByteArrayOutputStream to determine length of output) to support persistent connections.
- Cookie
  - Gives cookies sent to client by server sometime earlier. Use getCookies, not getHeader
- Host
  - Indicates host given in original URL.
  - This is required in HTTP 1.1.

88

(read the slide)

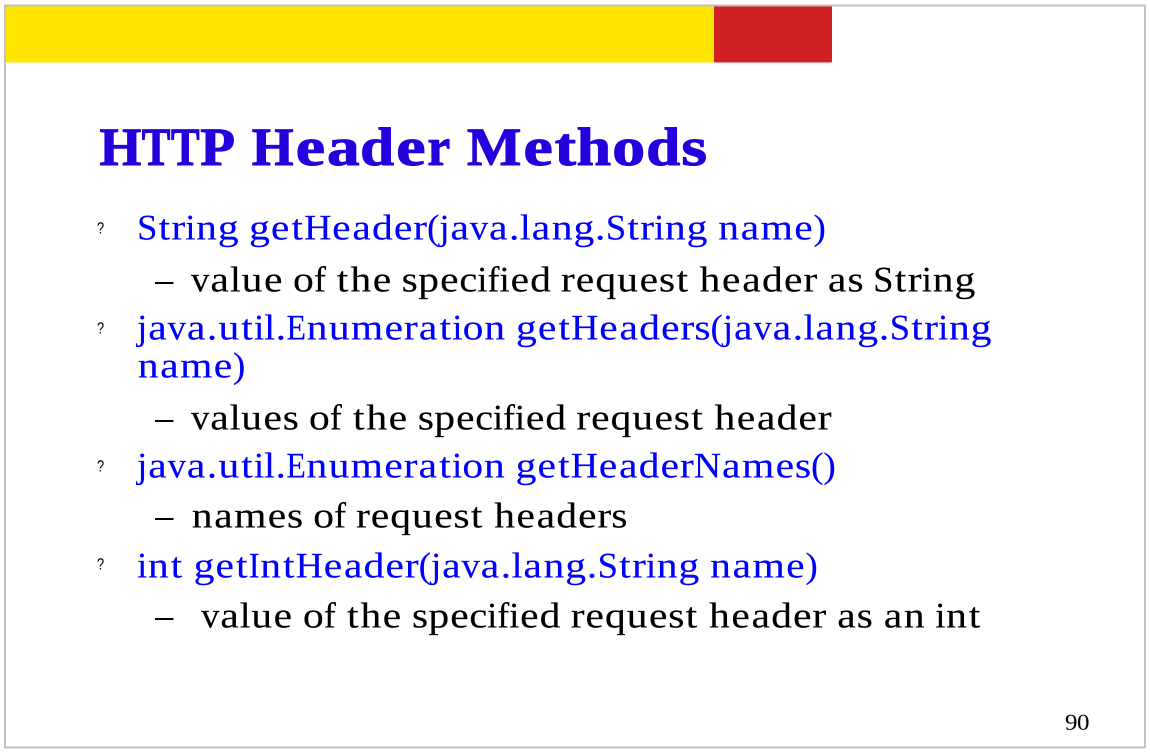


HTTP Request Headers

- If-Modified-Since
  - Indicates client wants page only if it has been changed after specified date.
  - Don’t handle this situation directly; implement getLastModified instead.
- Referer
  - URL of referring Web page.
  - Useful for tracking traffic; logged by many servers.
- User-Agent
  - String identifying the browser making the request.
  - Use with extreme caution!

89

(read the slide)



HTTP Header Methods

? String getHeader(java.lang.String name)
   – value of the specified request header as String

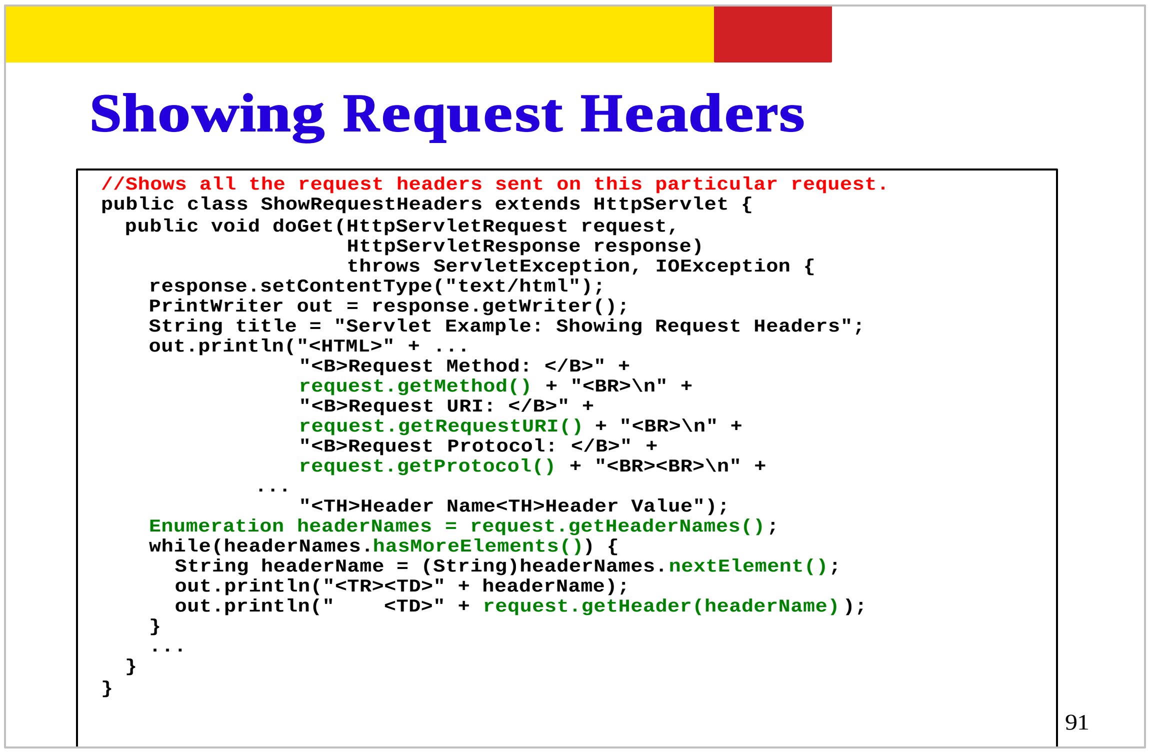
? java.util.Enumeration getHeaders(java.lang.String name)
   – values of the specified request header

? java.util.Enumeration getHeaderNames()
   – names of request headers

? int getIntHeader(java.lang.String name)
   – value of the specified request header as an int

90

These are set of HTTP header methods.



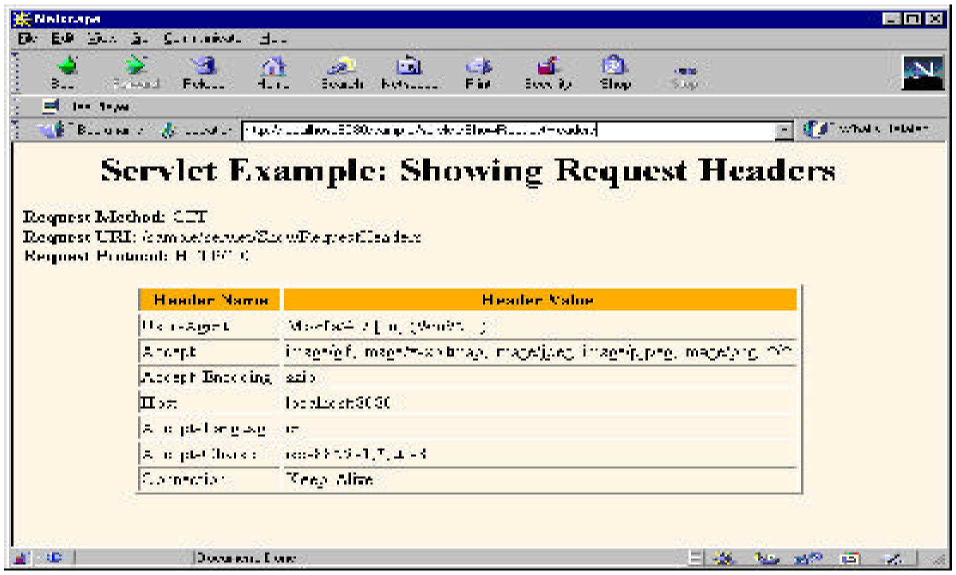
Showing Request Headers

```
//Shows all the request headers sent on this particular request.
public class ShowRequestHeaders extends HttpServlet {
  public void doGet(HttpServletRequest request,
                    HttpServletResponse response)
                    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "Servlet Example: Showing Request Headers";
    out.println("<HTML>" + ...
                "<B>Request Method: </B>" +
                request.getMethod() + "<BR>\n" +
                "<B>Request URI: </B>" +
                request.getRequestURI() + "<BR>\n" +
                "<B>Request Protocol: </B>" +
                request.getProtocol() + "<BR><BR>\n" +
             ...
                "<TH>Header Name<TH>Header Value");
    Enumeration headerNames = request.getHeaderNames();
    while(headerNames.hasMoreElements()) {
      String headerName = (String)headerNames.nextElement();
      out.println("<TR><TD>" + headerName);
      out.println("    <TD>" + request.getHeader(headerName));
    }
    ...
  }
}
```

91

This is an example servlet code retrieving HTTP header information and then display them.

Request Headers Sample



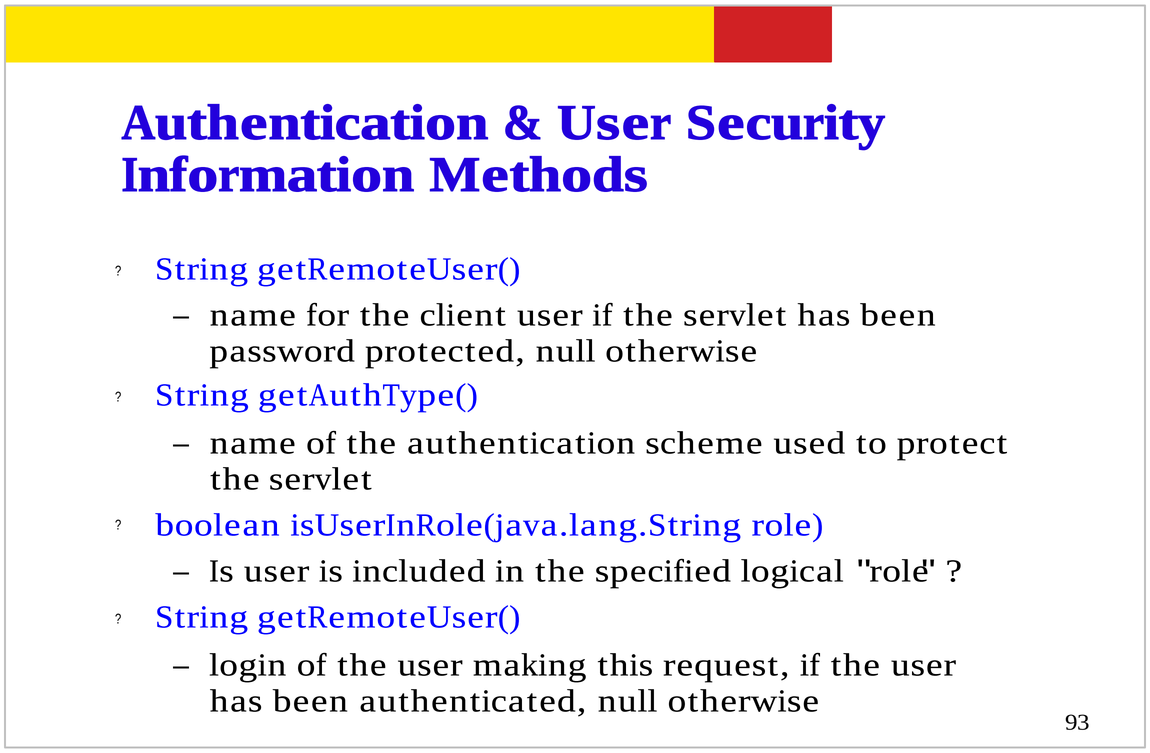
Servlet Example: Showing Request Headers

Request Method: GET
Request URI: /sample/servlet/ShowRequestHeaders
Request Protocol: HTTP/1.0

Header Name	Header Value
User-Agent	Mozilla/4.7 [en] (Win98; I)
Accept	image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding	gzip
Host	localhost:8080
Accept-Language	en
Accept-Charset	iso-8859-1,*,utf-8
Connection	Keep-Alive

92

This is the output of the previous code.

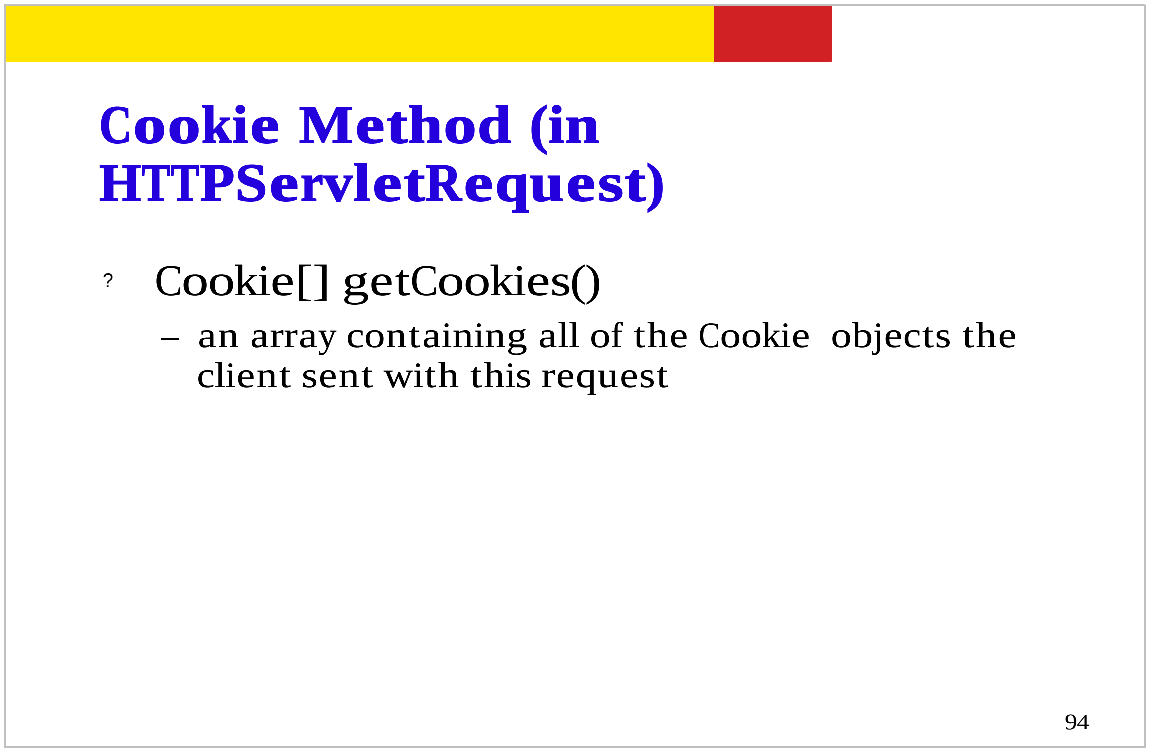


Authentication & User Security Information Methods

- String getRemoteUser()
  – name for the client user if the servlet has been password protected, null otherwise
- String getAuthType()
  – name of the authentication scheme used to protect the servlet
- boolean isUserInRole(java.lang.String role)
  – Is user is included in the specified logical "role" ?
- String getRemoteUser()
  – login of the user making this request, if the user has been authenticated, null otherwise

93

This slide has a set of methods that are related to authentication and user security information.



Cookie Method (in HTTPServletRequest)

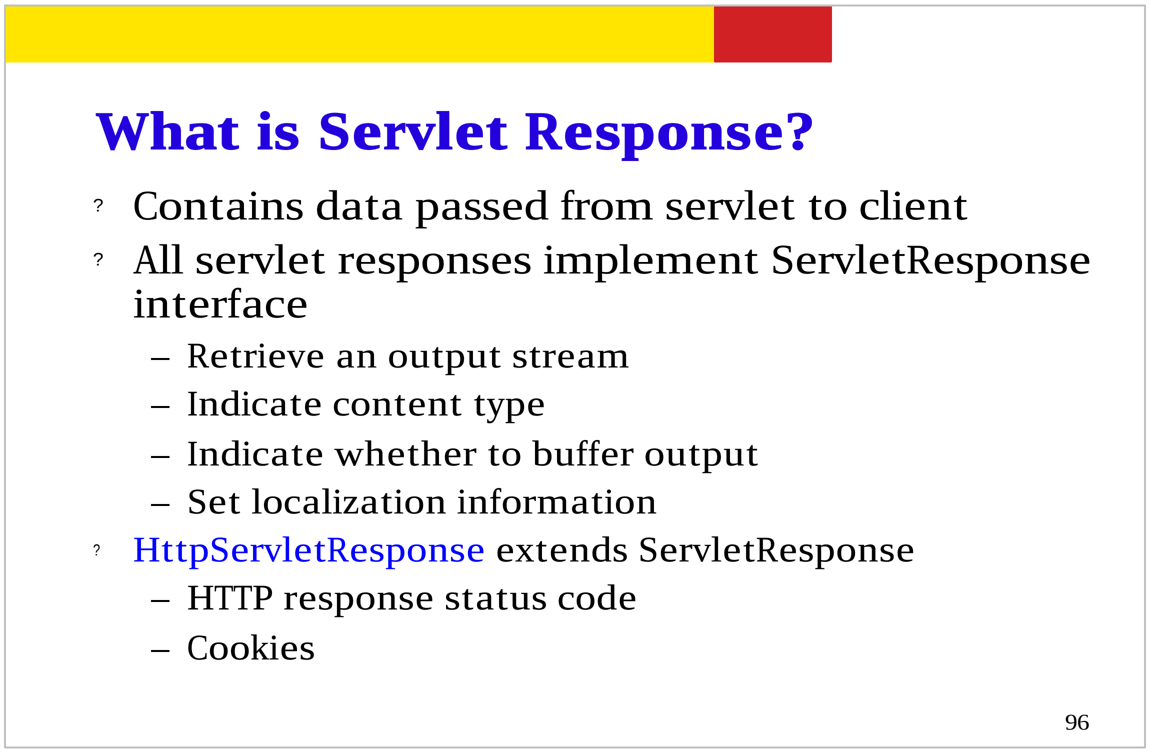
- Cookie[] getCookies()
  - an array containing all of the Cookie objects the client sent with this request

94

getCookies() method returns an array of Cookie objects the client sent with the HTTP request.

Servlet Response (HttpServletResponse)

So far we talked about Servlet request in some detail. Now let's talk about Servlet response.



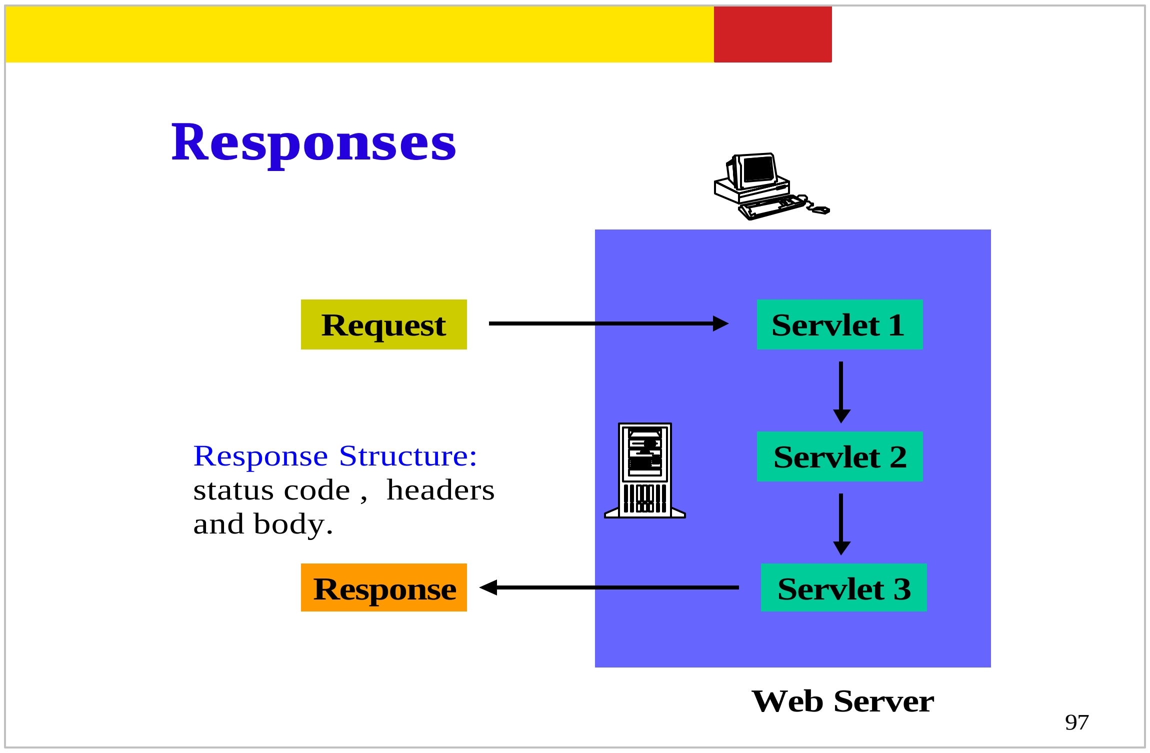
What is Servlet Response?

- Contains data passed from servlet to client
- All servlet responses implement ServletResponse interface
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information
- HttpServletResponse extends ServletResponse
  - HTTP response status code
  - Cookies

96

Servlet response contains data that is supplied by the servlet or container.
All servlet responses implement ServletResponse Java interface, which contains methods for retrieving an output stream, indicating content type, indicating whether to buffer output or not, for setting localization information.

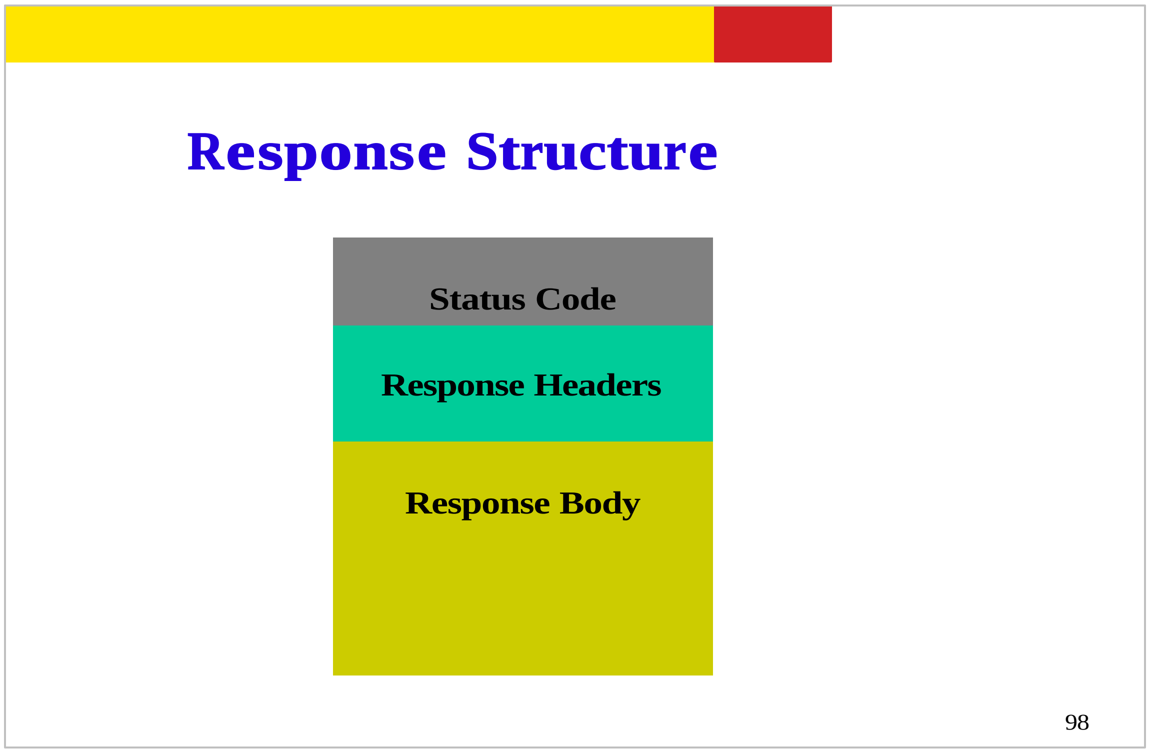
Now HttpServletResponse is Java interface that extends ServletResponse. The HttpServletResponse interface contains methods for setting HTTP response status code and cookies.



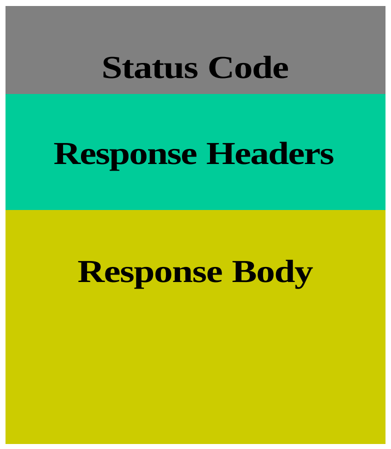
Responses

Response Structure:
status code , headers
and body.

This picture shows the response message that is generated at the server. The response message contains HTTP status code, headers, and body.

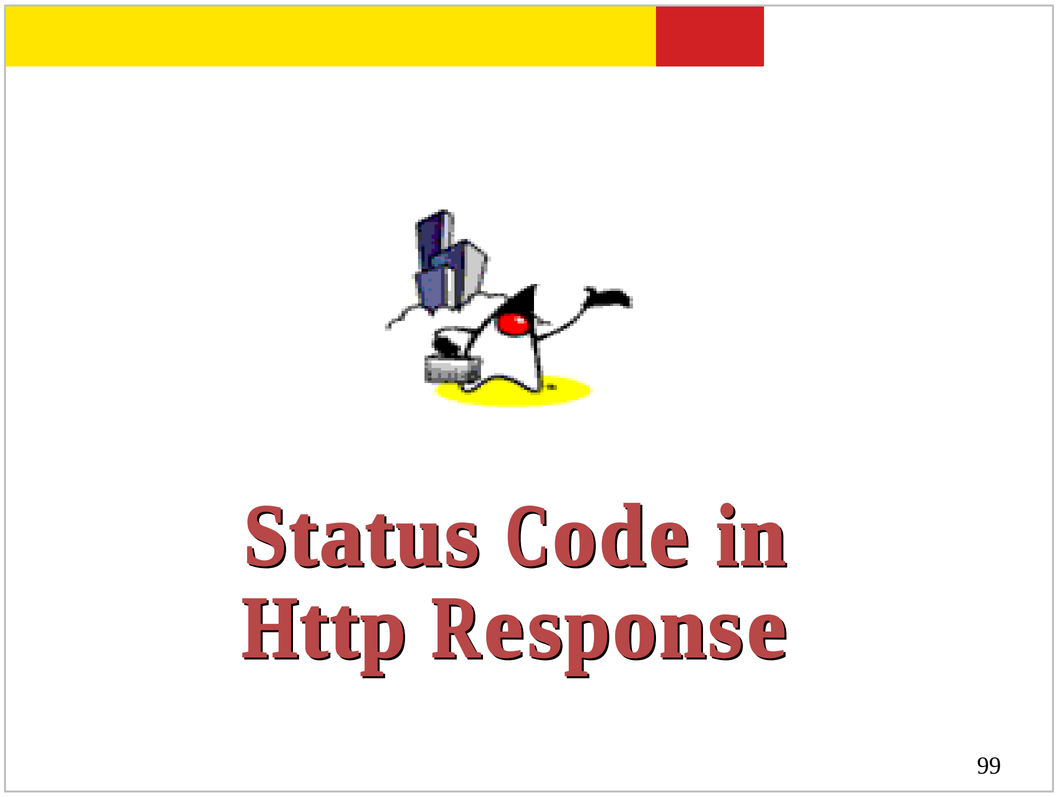


Response Structure

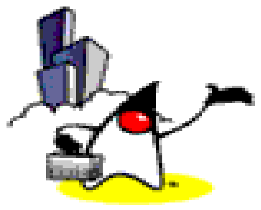


98

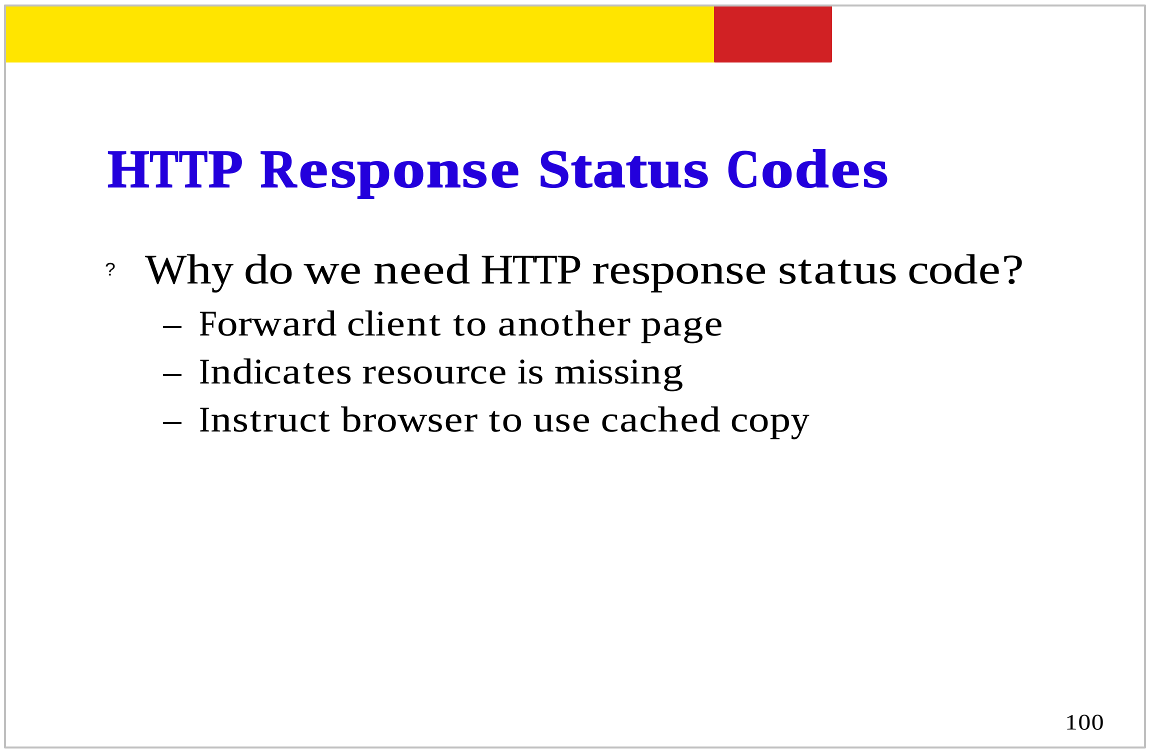
This picture just shows one more time how a HTTP response message is made of - it is made of status code, response headers, and response body.



Status Code in Http Response



Now let's talk a little bit on status code in HTTP response message.

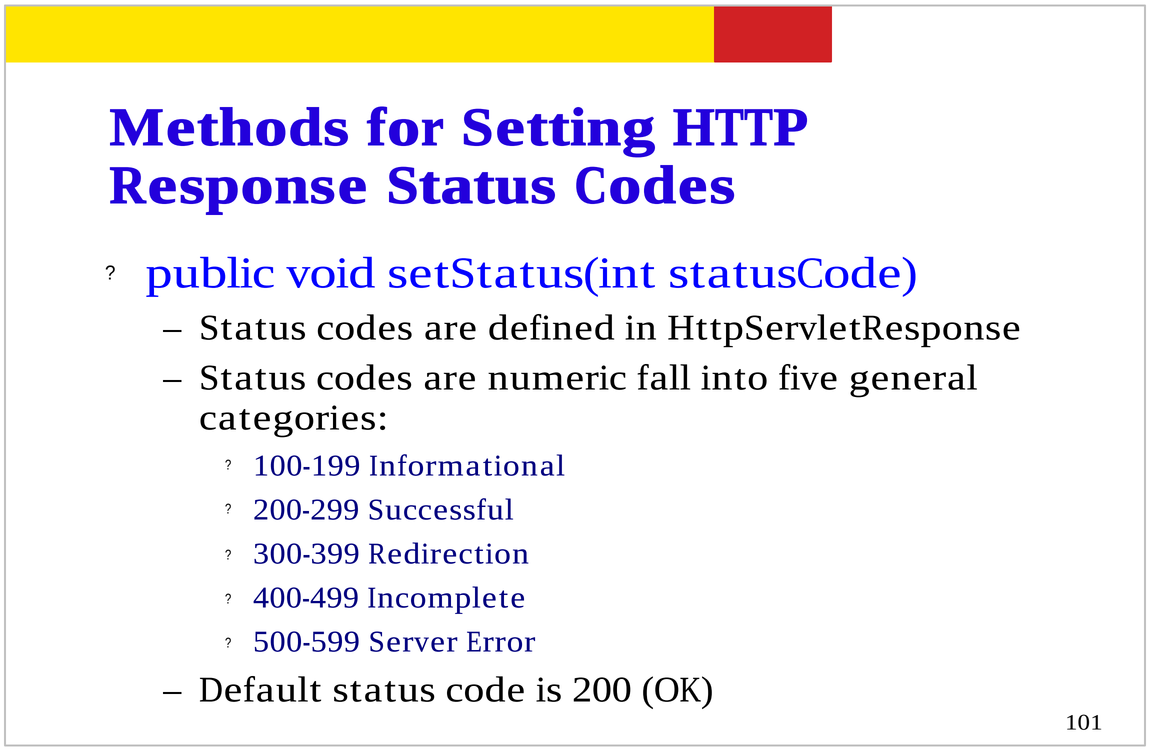


HTTP Response Status Codes

- Why do we need HTTP response status code?
  - Forward client to another page
  - Indicates resource is missing
  - Instruct browser to use cached copy

100

So how does HTTP response status code get used? First, it could be used as an instruction to the browser to forward the client to another page. Second it could be used to indicate resource is missing. Third, it could be used to instruct the browser to use cached copy of data.

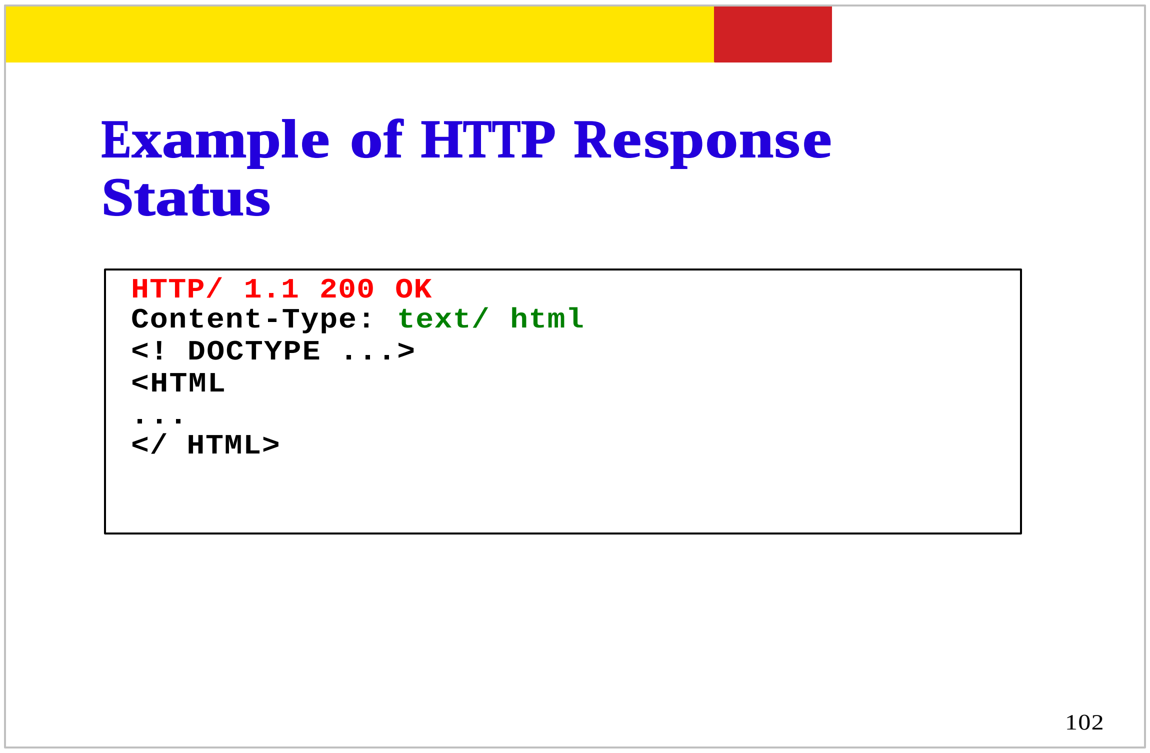


Methods for Setting HTTP Response Status Codes

- public void setStatus(int statusCode)
  - Status codes are defined in HttpServletResponse
  - Status codes are numeric fall into five general categories:
    - 100-199 Informational
    - 200-299 Successful
    - 300-399 Redirection
    - 400-499 Incomplete
    - 500-599 Server Error
  - Default status code is 200 (OK)

101

You can use setStatus() method of HttpServletResponse class. The HTTP status codes fall into five general categories mentioned above. (read the status code in the slide) while the default status code is 200.

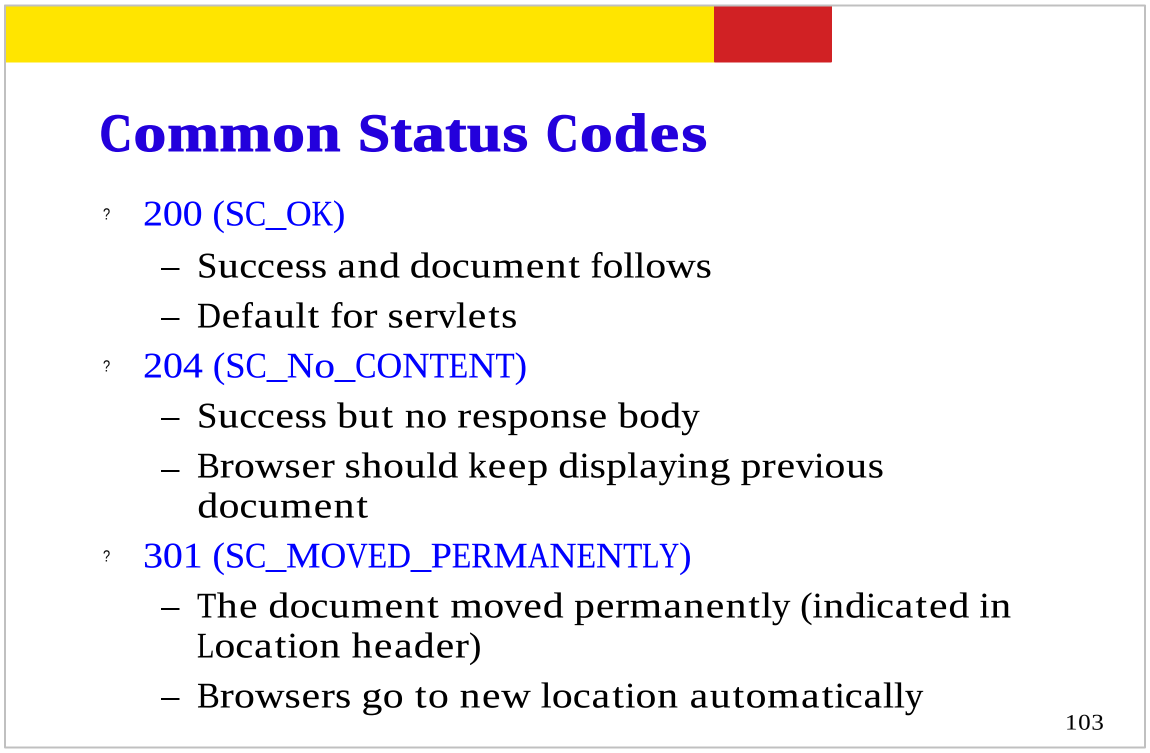


Example of HTTP Response Status

```
HTTP/ 1.1 200 OK
Content-Type: text/ html
<! DOCTYPE ...>
<HTML
...
</ HTML>
```

102

This is HTTP response example that contains the default OK status.

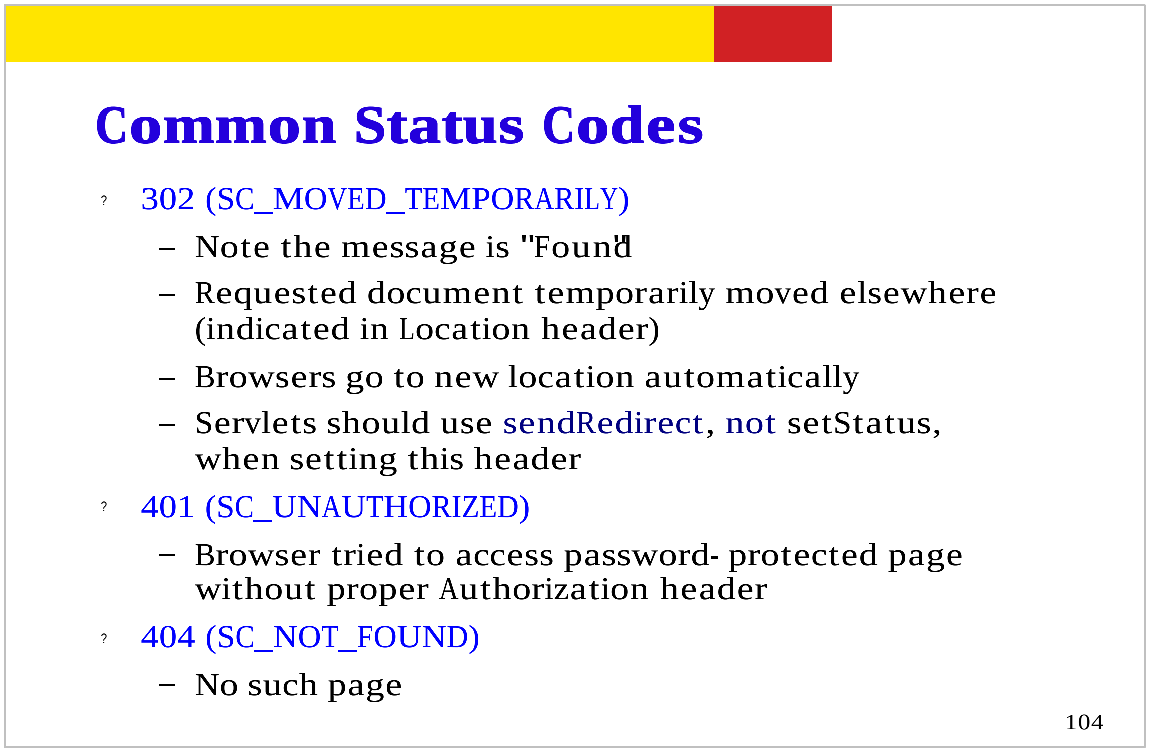


Common Status Codes

- 200 (SC_OK)
  - Success and document follows
  - Default for servlets
- 204 (SC_No_CONTENT)
  - Success but no response body
  - Browser should keep displaying previous document
- 301 (SC_MOVED_PERMANENTLY)
  - The document moved permanently (indicated in Location header)
  - Browsers go to new location automatically

103

This slide and following slide show the common status code your servlet code can set.

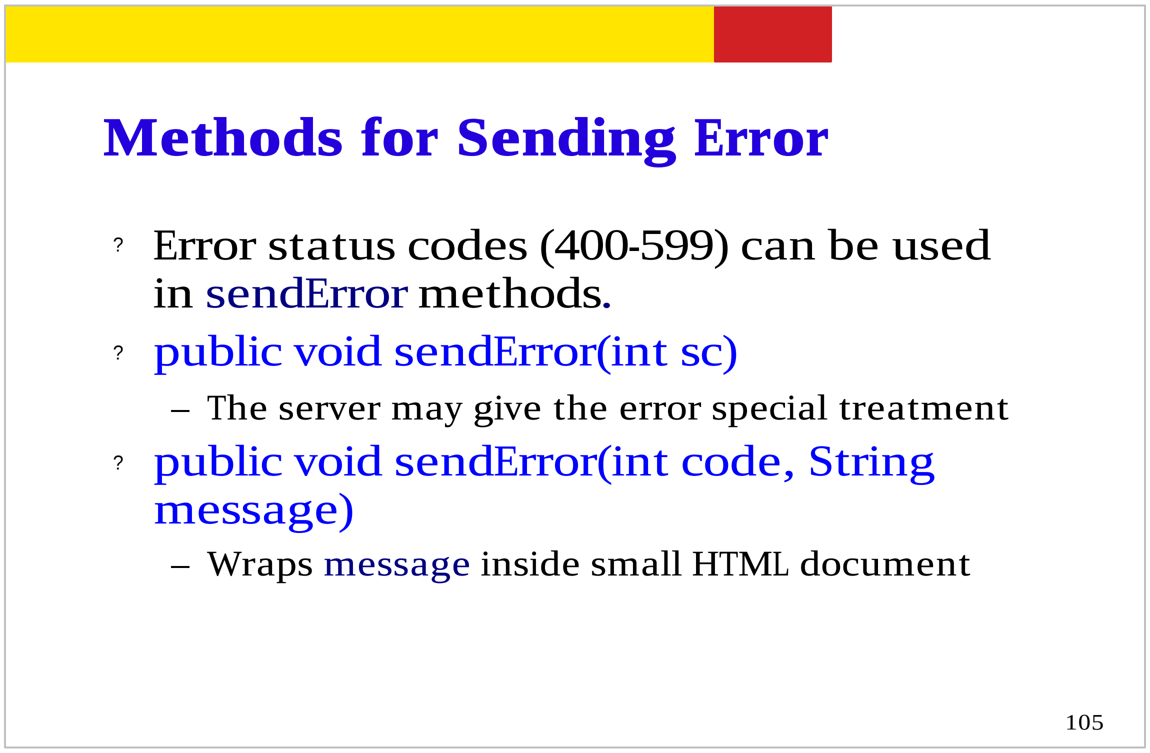


Common Status Codes

- 302 (SC_MOVED_TEMPORARILY)
  - Note the message is "Found"
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use sendRedirect, not setStatus, when setting this header
- 401 (SC_UNAUTHORIZED)
  - Browser tried to access password- protected page without proper Authorization header
- 404 (SC_NOT_FOUND)
  - No such page

104

This is the list of common status code your servlet code can set. Please note that status 401 indicates that a user tried to access password-protected page without proper authentication header and status 404 is the result of accessing a URL which is not present.

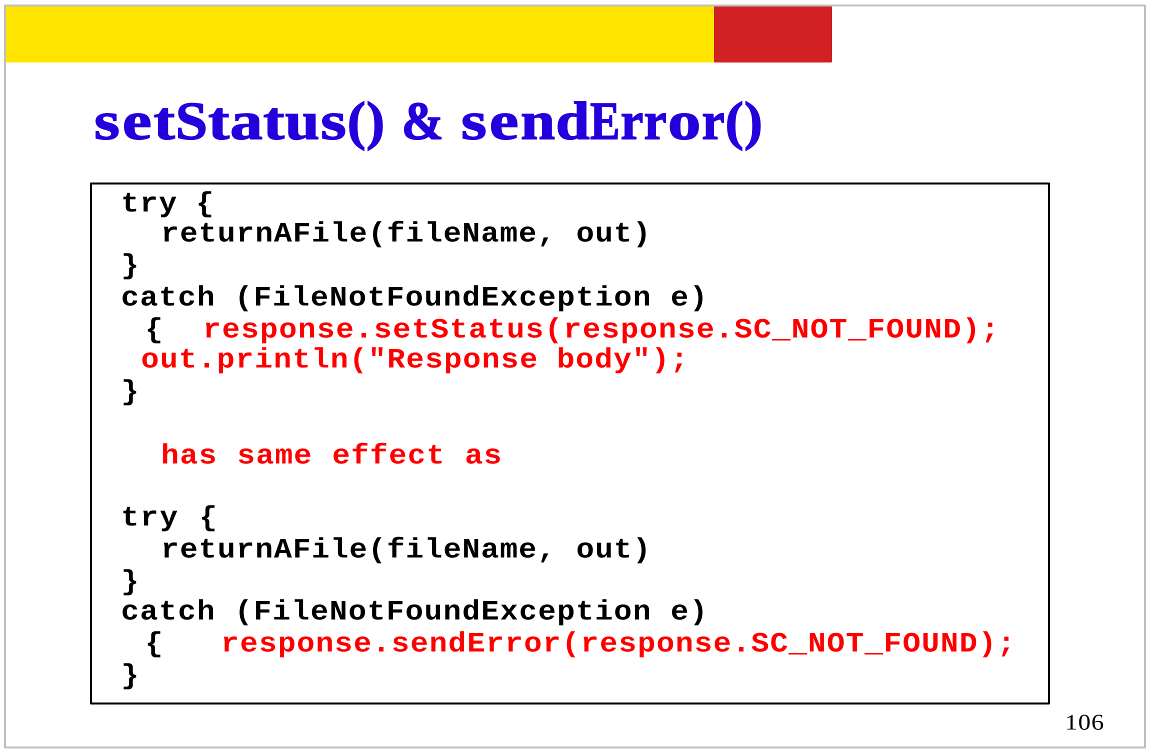


Methods for Sending Error

- Error status codes (400-599) can be used in sendError methods.
- public void sendError(int sc)
  – The server may give the error special treatment
- public void sendError(int code, String message)
  – Wraps message inside small HTML document

105

Instead of using setStatus() method and then writing status message out to the output stream, you can accomplish both with a single method called sendError (). We will see an example code in the following slide.

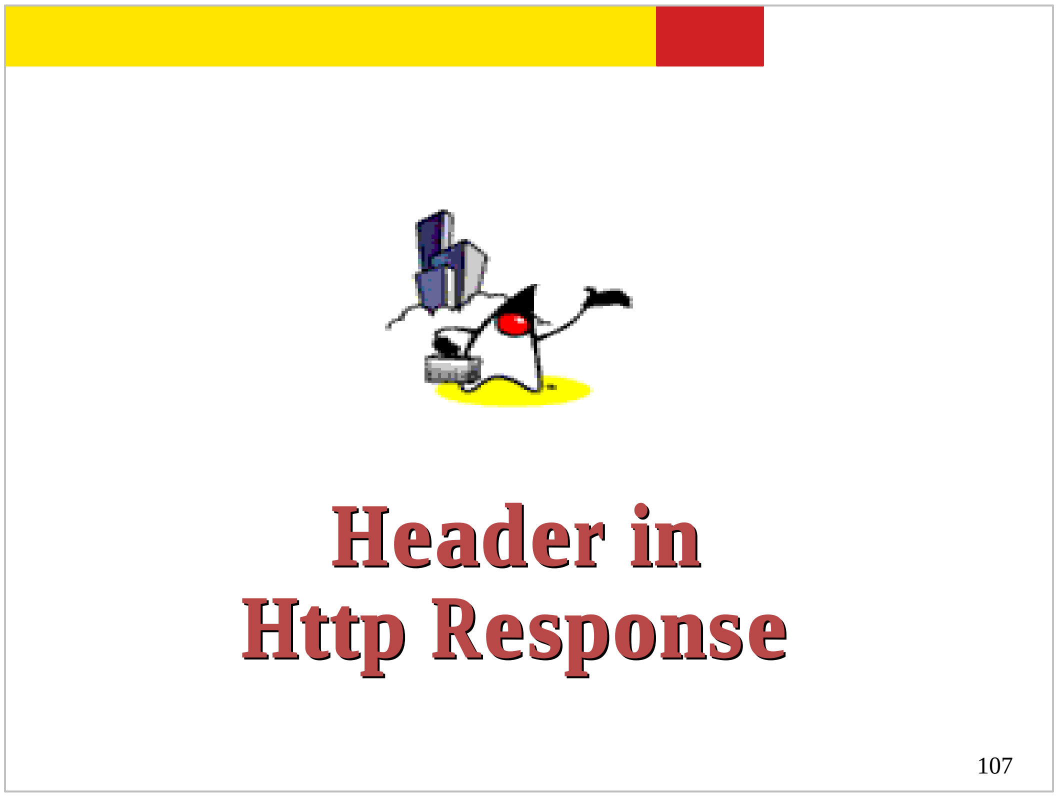


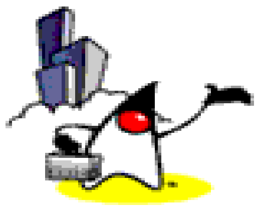
setStatus() & sendError()

```
try {
   returnAFile(fileName, out)
}
catch (FileNotFoundException e)
  {  response.setStatus(response.SC_NOT_FOUND);
 out.println("Response body");
}

   has same effect as

try {
   returnAFile(fileName, out)
}
catch (FileNotFoundException e)
  {   response.sendError(response.SC_NOT_FOUND);
}
```

106

This slide compares the usage of setStatus() method and sendError() method as mentioned in the previous slide.

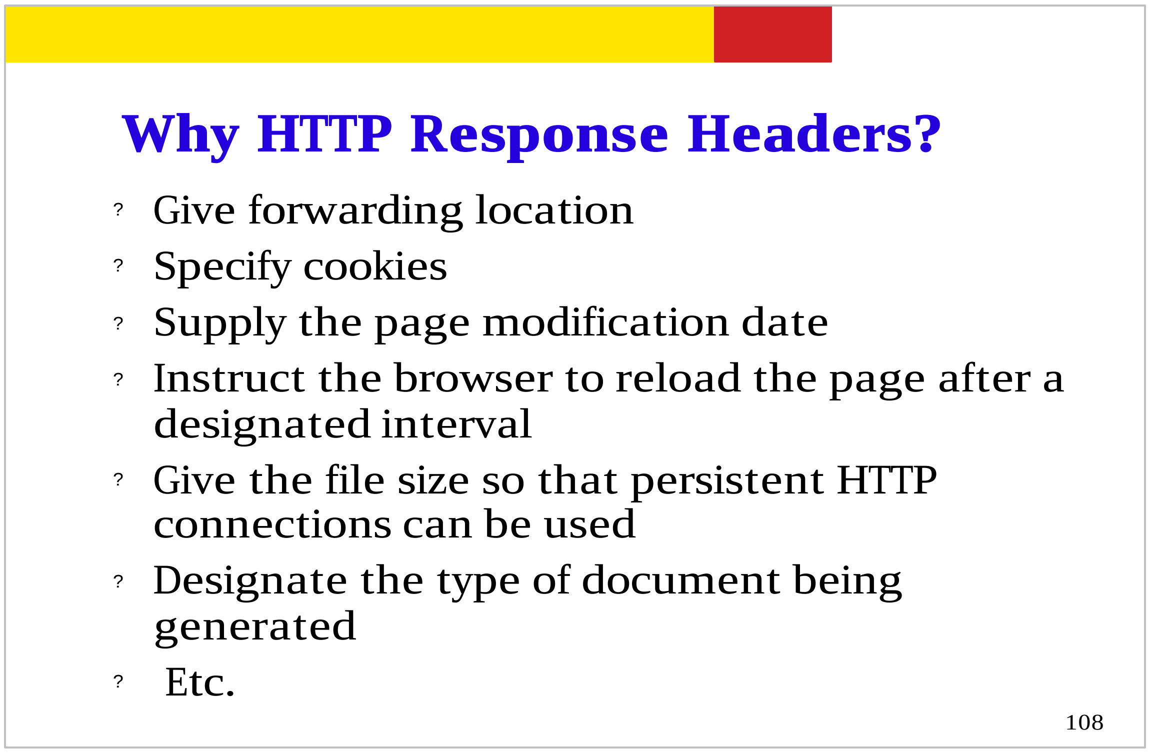




Header in Http Response

107

Now let's take a look at the header structure of HTTP response.



Why HTTP Response Headers?

- Give forwarding location
- Specify cookies
- Supply the page modification date
- Instruct the browser to reload the page after a designated interval
- Give the file size so that persistent HTTP connections can be used
- Designate the type of document being generated
- Etc.

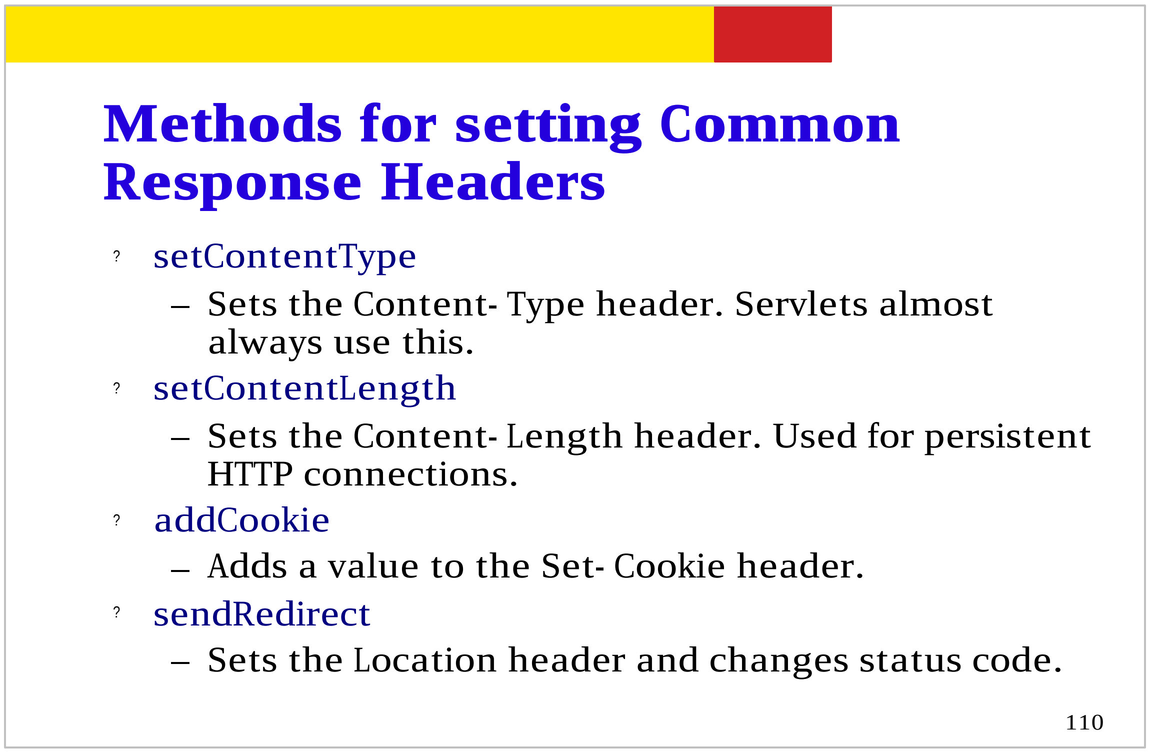
108

HTTP response header contains information that can be used for several things mentioned in the slide. (Read the slide.)

Methods for Setting Arbitrary Response Headers

- public void setHeader(String headerName, String headerValue)
  - Sets an arbitrary header.
- public void setDateHeader(String name, long millisecs)
  - Converts milliseconds since 1970 to a date string in GMT format
- public void setIntHeader(String name, int headerValue)
  - Prevents need to convert int to String before calling setHeader
- addHeader, addDateHeader, addIntHeader
  - Adds new occurrence of header instead of replacing.

109

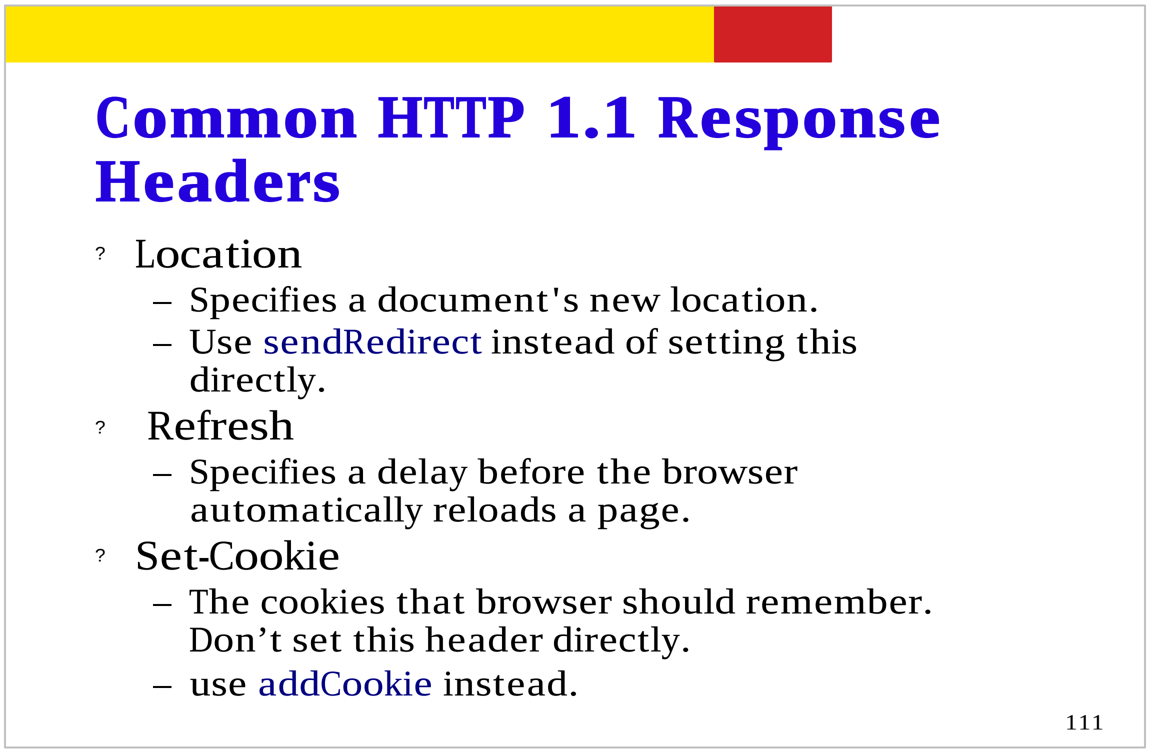


Methods for setting Common Response Headers

- setContentType
  – Sets the Content- Type header. Servlets almost always use this.
- setContentLength
  – Sets the Content- Length header. Used for persistent HTTP connections.
- addCookie
  – Adds a value to the Set- Cookie header.
- sendRedirect
  – Sets the Location header and changes status code.

110

This is the list of method that are used for setting common response headers.
(read the slide.)

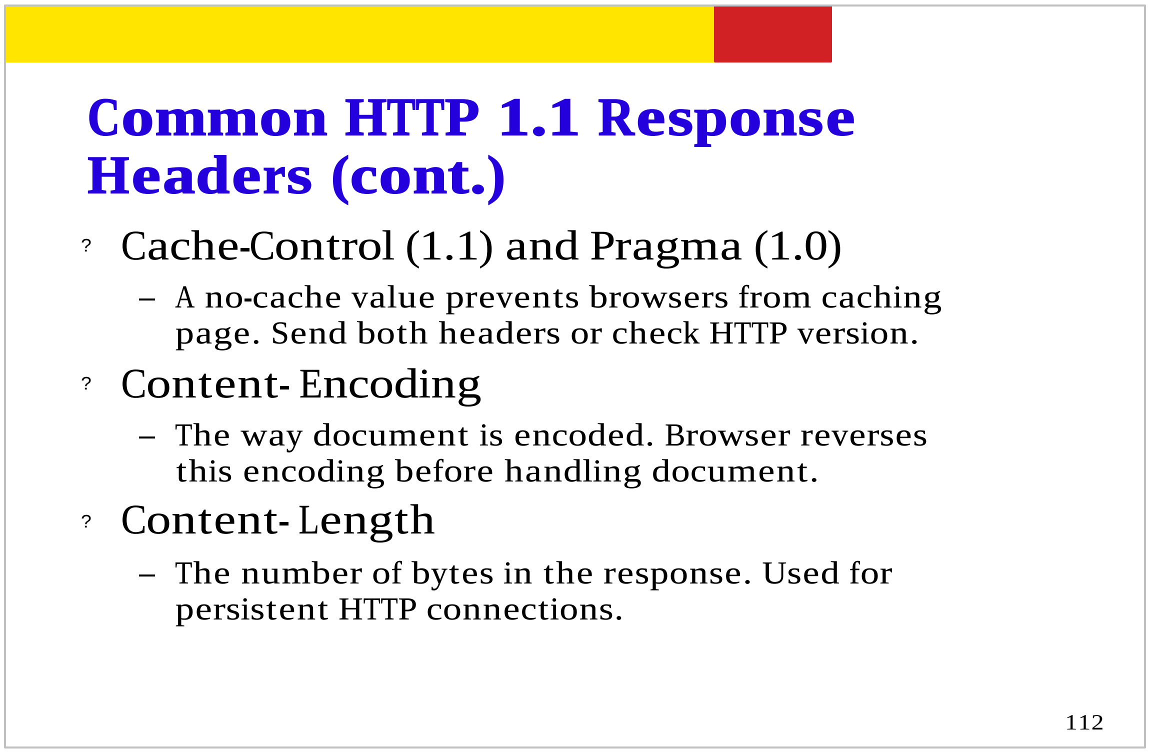


Common HTTP 1.1 Response Headers

- Location
  - Specifies a document's new location.
  - Use sendRedirect instead of setting this directly.
- Refresh
  - Specifies a delay before the browser automatically reloads a page.
- Set-Cookie
  - The cookies that browser should remember. Don’t set this header directly.
  - use addCookie instead.

111

These are the common HTTP response header fields. (read the slide.)

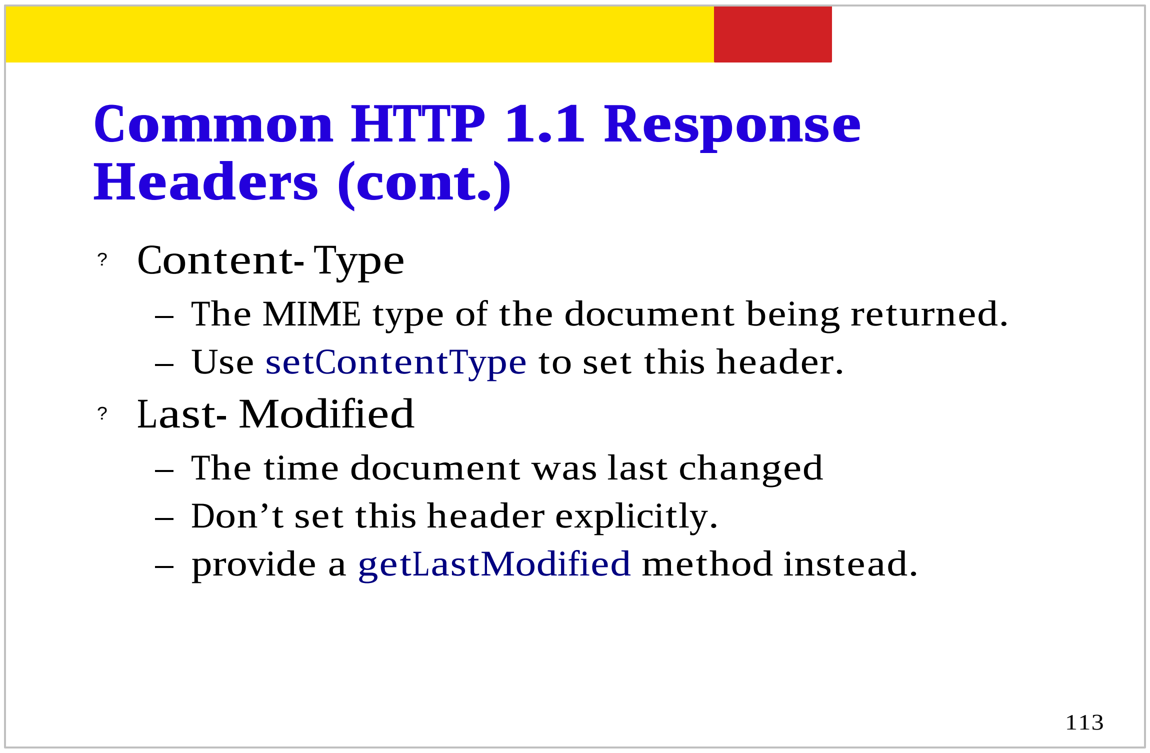


Common HTTP 1.1 Response Headers (cont.)

- Cache-Control (1.1) and Pragma (1.0)
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- Content- Encoding
  - The way document is encoded. Browser reverses this encoding before handling document.
- Content- Length
  - The number of bytes in the response. Used for persistent HTTP connections.

112

This is the continuation of HTTP response header fields. (read the slide.)

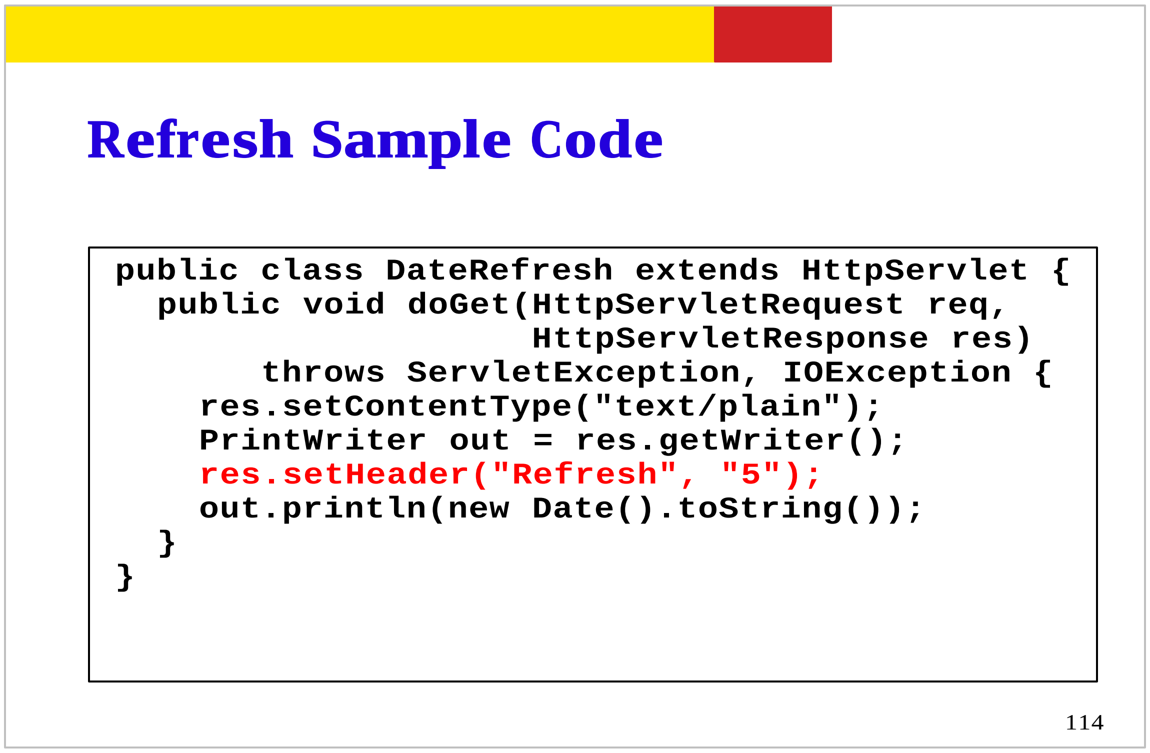


Common HTTP 1.1 Response Headers (cont.)

- Content-Type
  - The MIME type of the document being returned.
  - Use setContentType to set this header.
- Last-Modified
  - The time document was last changed
  - Don’t set this header explicitly.
  - provide a getLastModified method instead.

113

This is continuation of HTTP response header fields. (read the slide.)

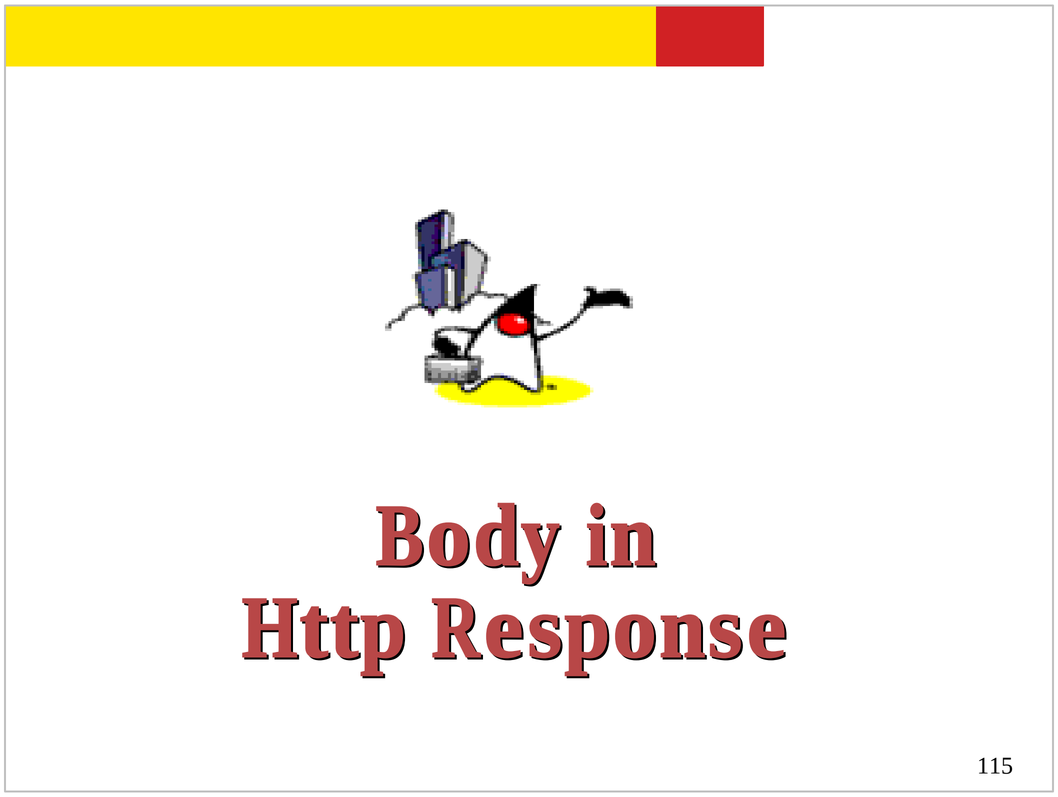


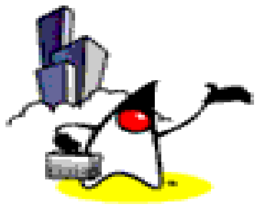
Refresh Sample Code

```
public class DateRefresh extends HttpServlet {
  public void doGet(HttpServletRequest req,
                    HttpServletResponse res)
       throws ServletException, IOException {
    res.setContentType("text/plain");
    PrintWriter out = res.getWriter();
    res.setHeader("Refresh", "5");
    out.println(new Date().toString());
  }
}
```

114

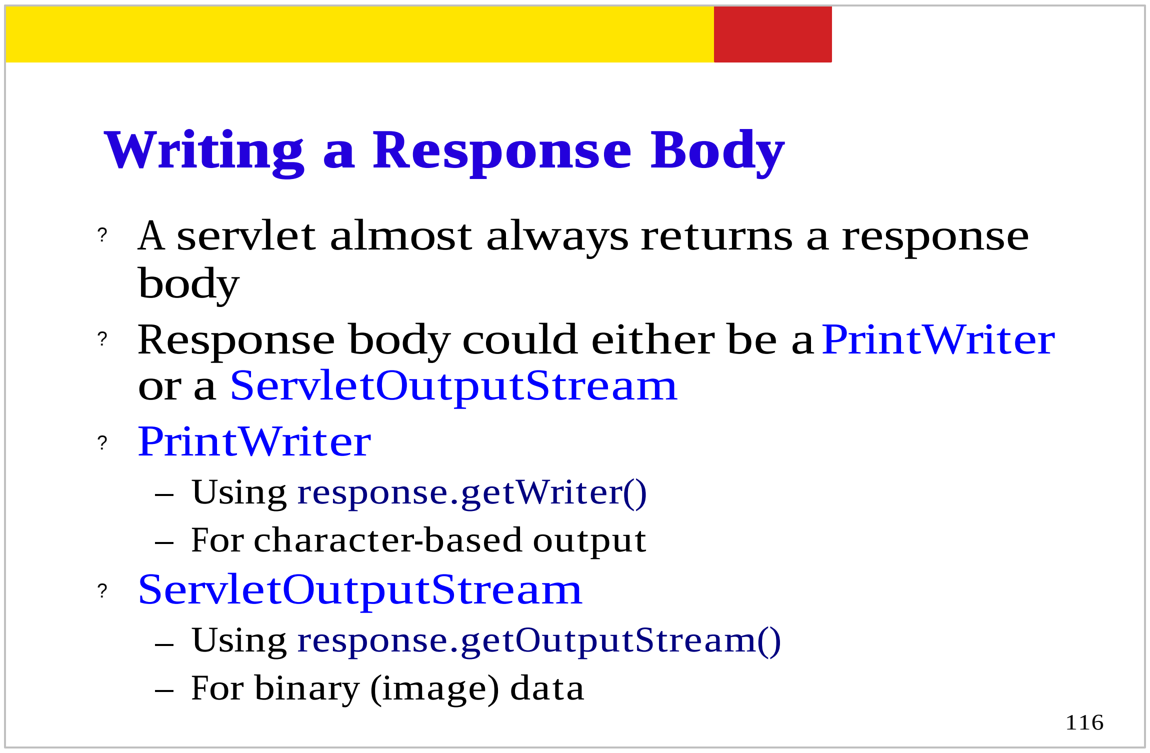
This is an example code in which you can instruct browser to refresh the page every 5 seconds.





Body in Http Response

We just looked into HTTP response header structure. Now let's take a look at the HTTP response body structure.



Writing a Response Body

? A servlet almost always returns a response body

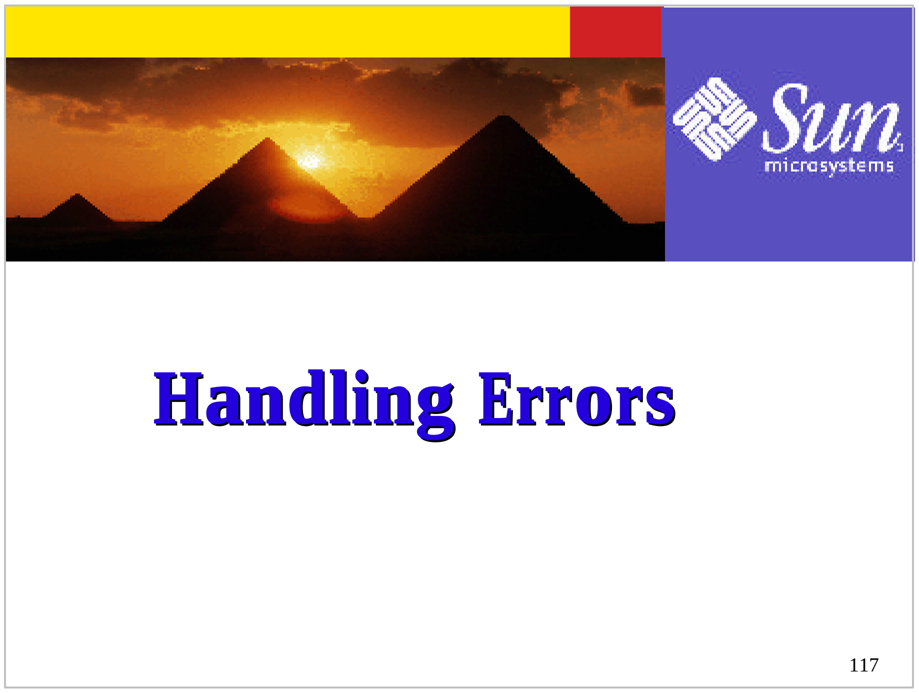
? Response body could either be a PrintWriter or a ServletOutputStream

? PrintWriter
  – Using response.getWriter()
  – For character-based output

? ServletOutputStream
  – Using response.getOutputStream()
  – For binary (image) data

116

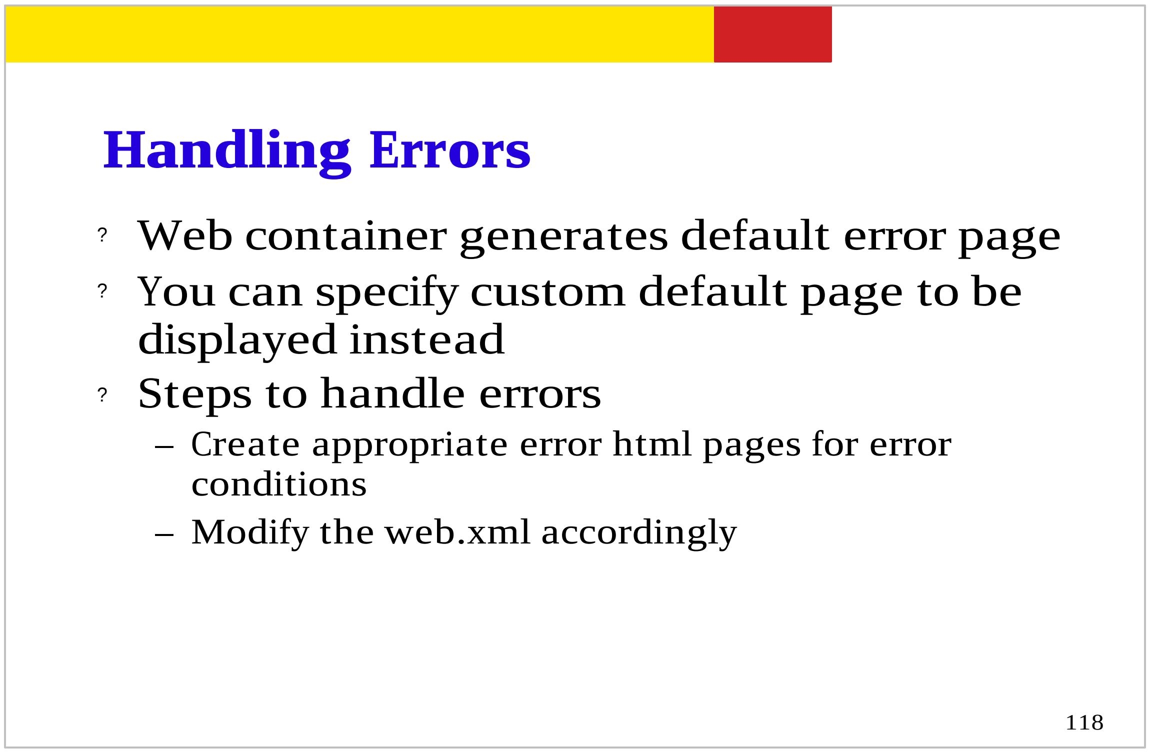
A response body could be the type pf either PrintWriter or ServletOutputStream. The former object can be retrieved via getWriter() method while the latter is obtained via getOutputStream() method. The difference between the two is that the former is used for character based output while the latter is used for sending out binary data.



Handling Errors

117

Now let's talk about how we can handle errors in your servlet code.



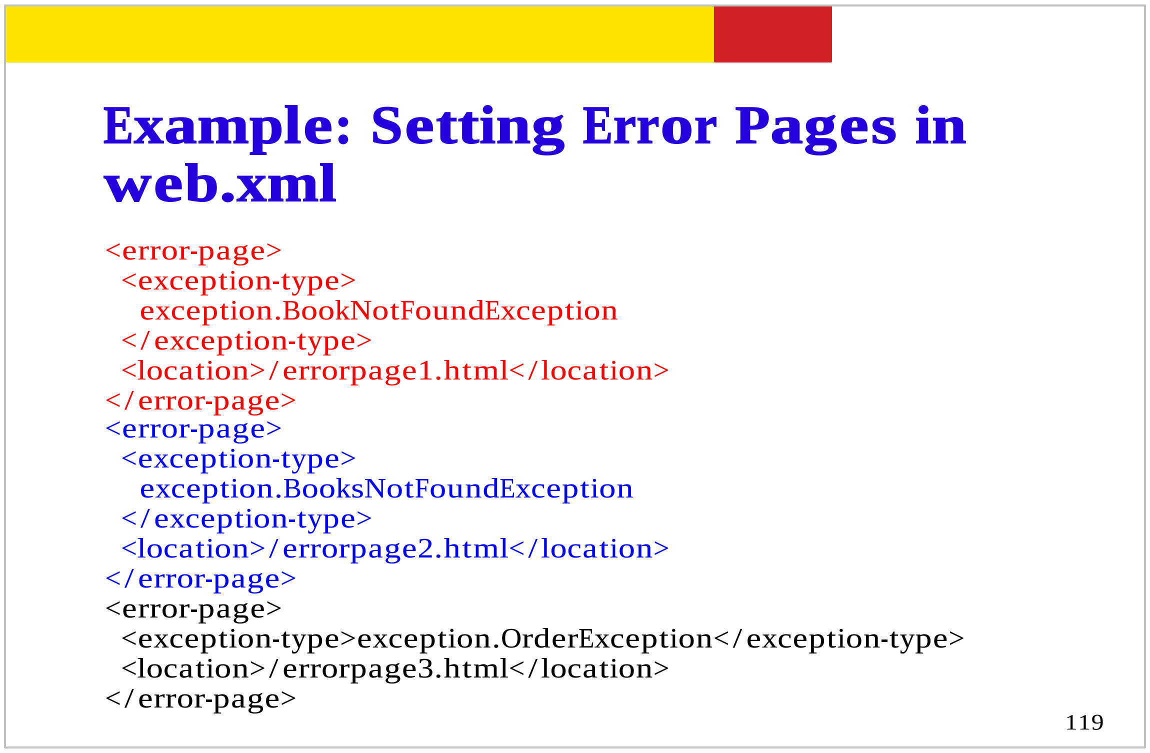
Handling Errors

- Web container generates default error page
- You can specify custom default page to be displayed instead
- Steps to handle errors
  - Create appropriate error html pages for error conditions
  - Modify the web.xml accordingly

118

Please note that web container generates a default error page unless you specify a custom error page to be displayed.

The steps you take in order to set a custom error page is first (1) create appropriate error page for error conditions (2) specify what error pages are to be displayed for what error conditions in web.xml deployment descriptor.



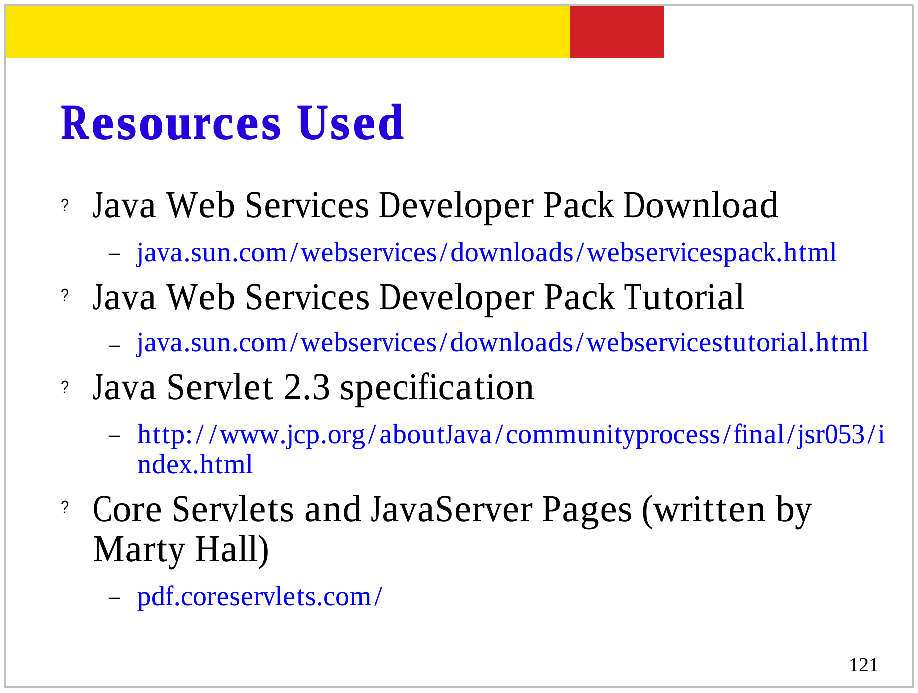
Example: Setting Error Pages in web.xml

```
<error-page>
  <exception-type>
    exception.BookNotFoundException
  </exception-type>
  <location>/errorpage1.html</location>
</error-page>
<error-page>
  <exception-type>
    exception.BooksNotFoundException
  </exception-type>
  <location>/errorpage2.html</location>
</error-page>
<error-page>
  <exception-type>exception.OrderException</exception-type>
  <location>/errorpage3.html</location>
</error-page>
```

119

This is an example of web.xml deployment descriptor in which 3 error conditions (Exceptions) are mapped into 3 custom error pages.

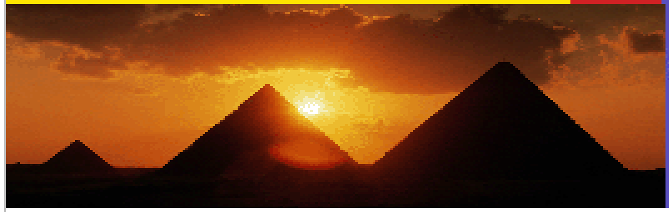
Resources



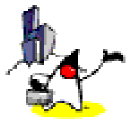
Resources Used

- Java Web Services Developer Pack Download
  - java.sun.com/webservices/downloads/webservicespack.html
- Java Web Services Developer Pack Tutorial
  - java.sun.com/webservices/downloads/webservicestutorial.html
- Java Servlet 2.3 specification
  - http://www.jcp.org/aboutJava/communityprocess/final/jsr053/index.html
- Core Servlets and JavaServer Pages (written by Marty Hall)
  - pdf.coreservlets.com/

121



Passion!



122