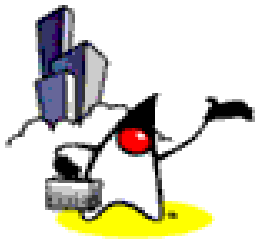
Servlet Basics



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

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)

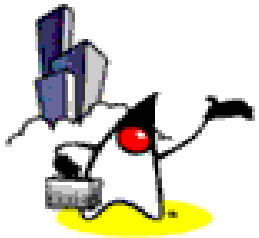
Topics

- 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

Advanced Topics:

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

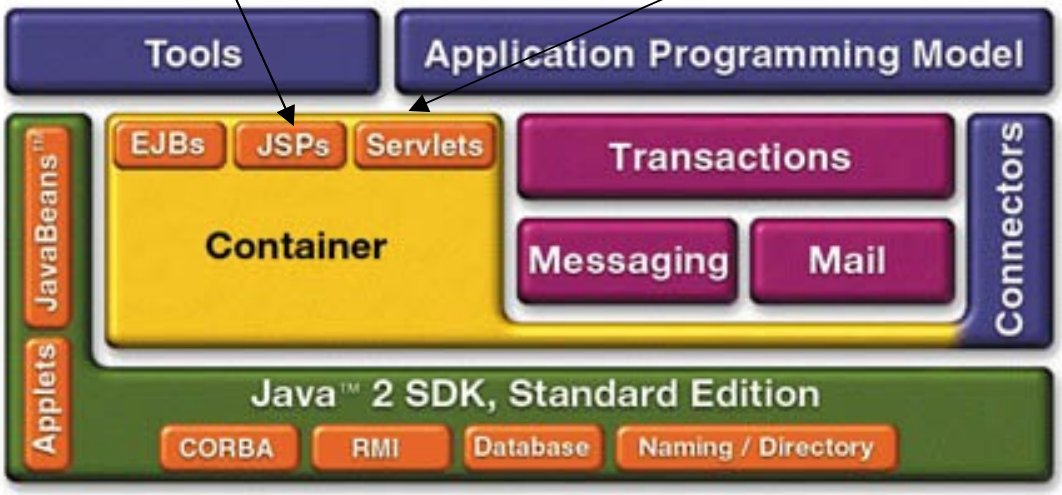
Servlet in a Big Picture of J2EE



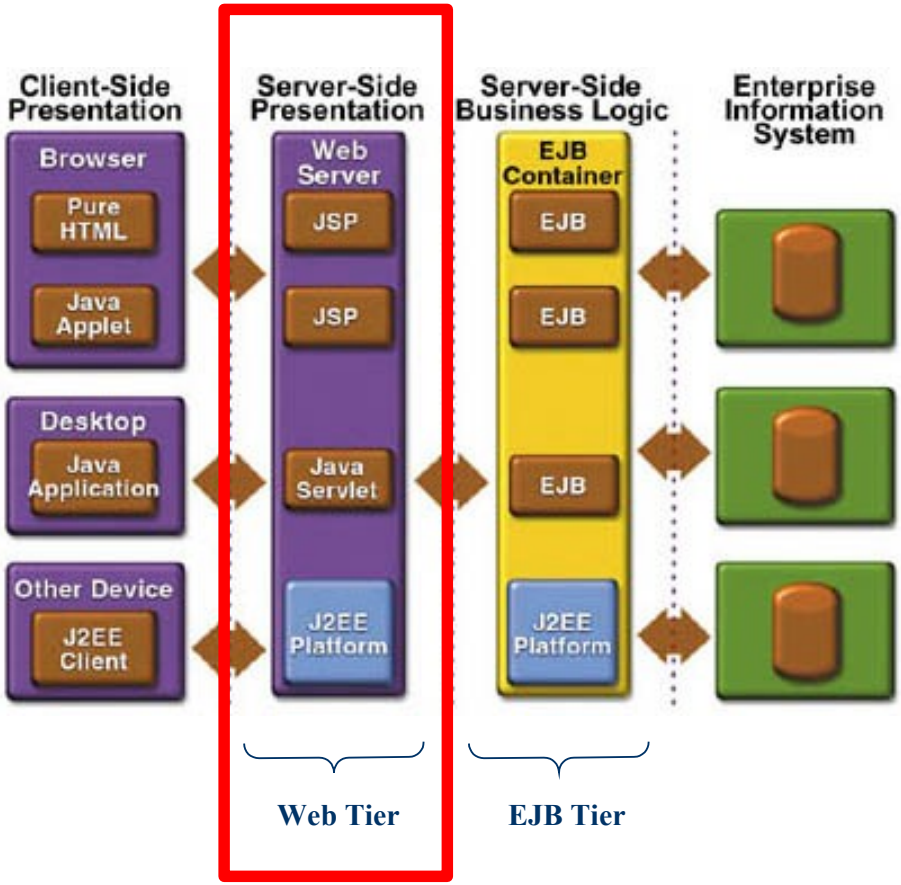
J2EE 1.2 Architecture

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**.



Where are Servlet and JSP?



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

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>");
  }
  ...
}
```

CGI versus Servlet

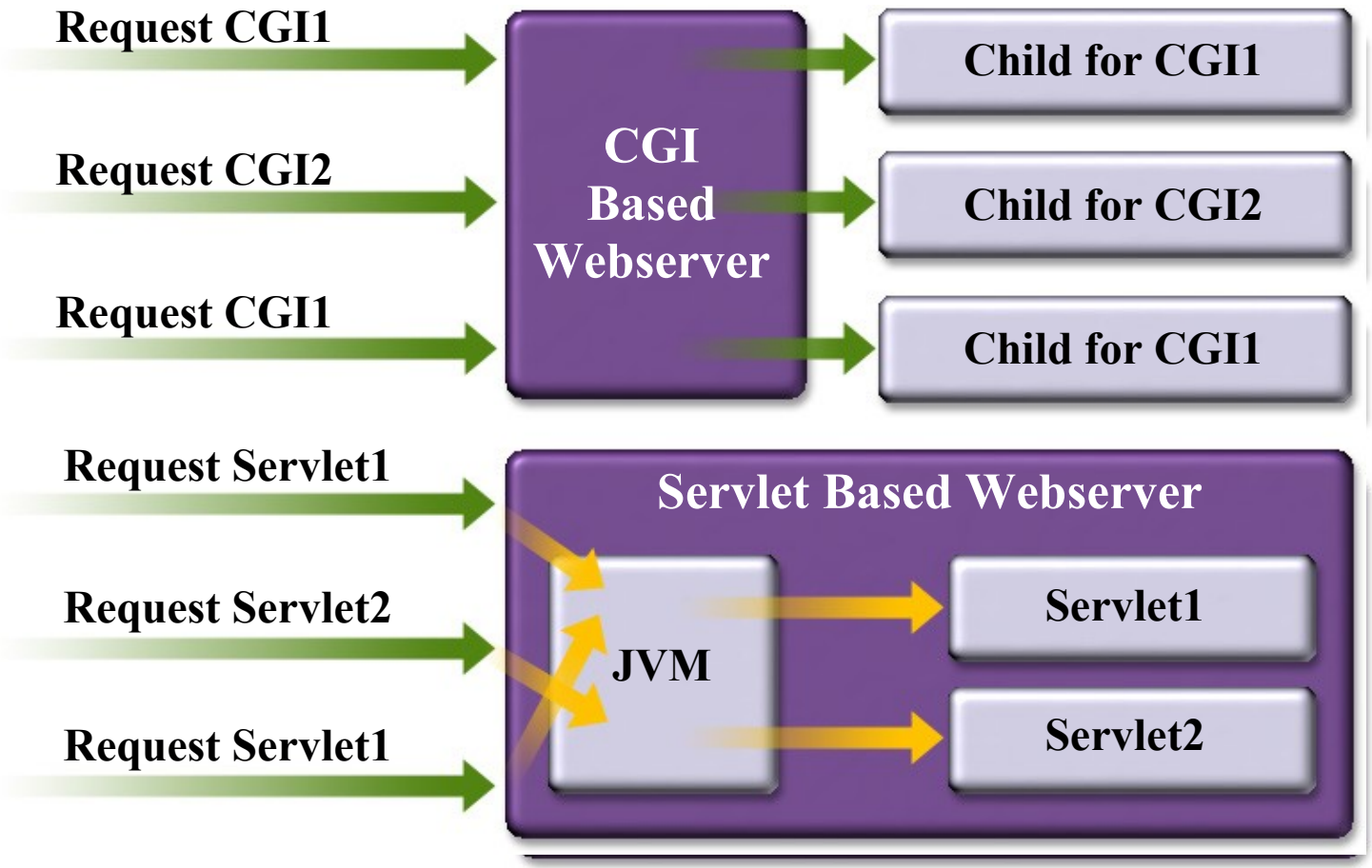
CGI

- 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

Servlet

- Written in Java
- Powerful, reliable, and efficient
- Improves scalability, reusability (component based)
- Leverages built-in security of Java programming language
- Platform independent and portable

Servlet vs. CGI



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

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

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

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>
```

Servlets and JSP - Comparison

Servlets

- HTML code in Java
- Any form of Data
- Not easy to author a web page

JSP

- Java-like code in HTML
- Structured Text
- Very easy to author a web page
- Code is compiled into a servlet

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

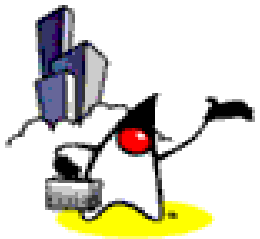
When to use Servlet over JSP

- 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

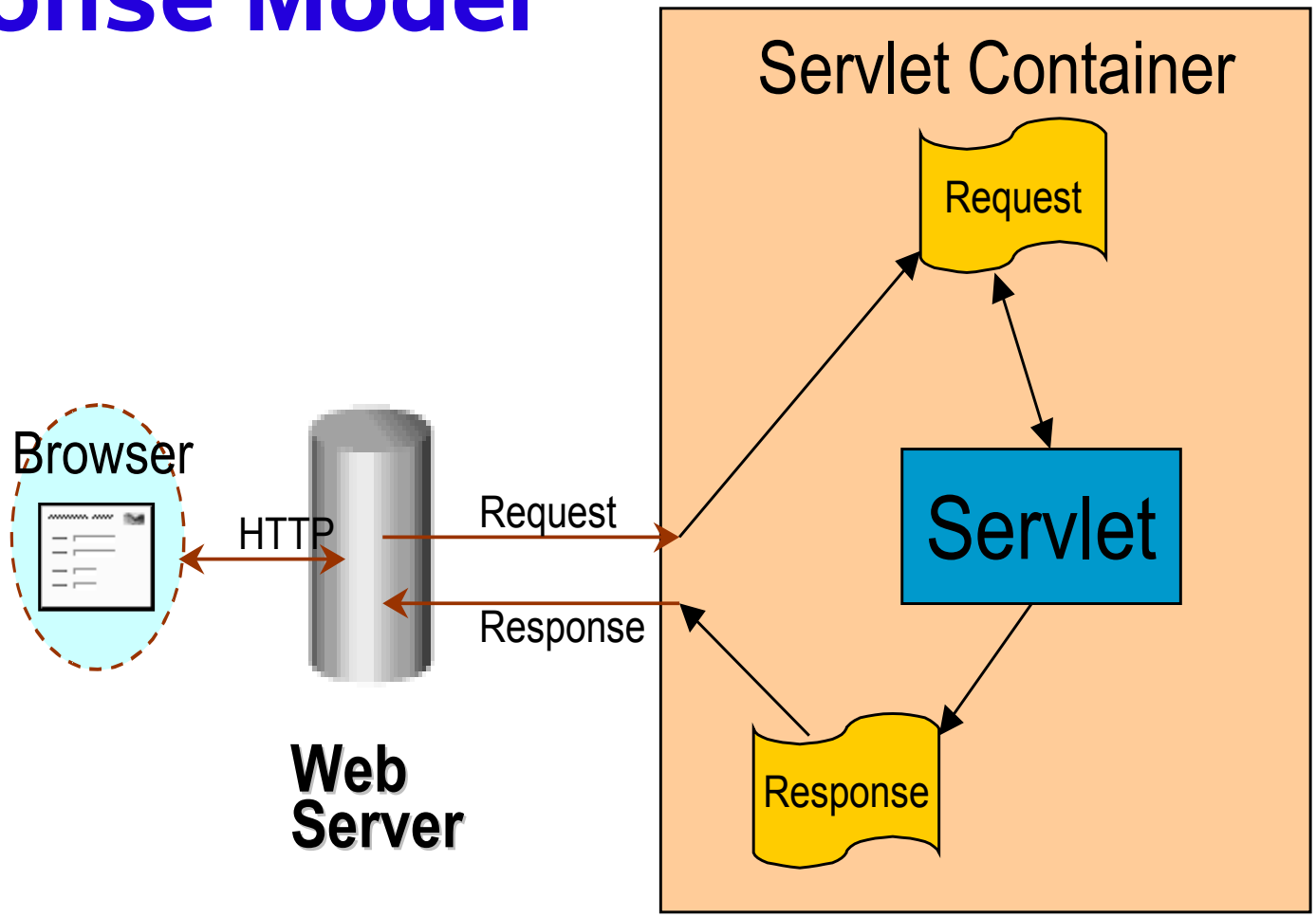
Should I Use Servlet or JSP?

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

Servlet Request & Response Model



Servlet Request and Response Model



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

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

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

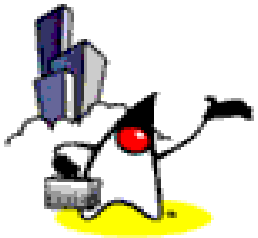
HTTP GET and POST

- 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

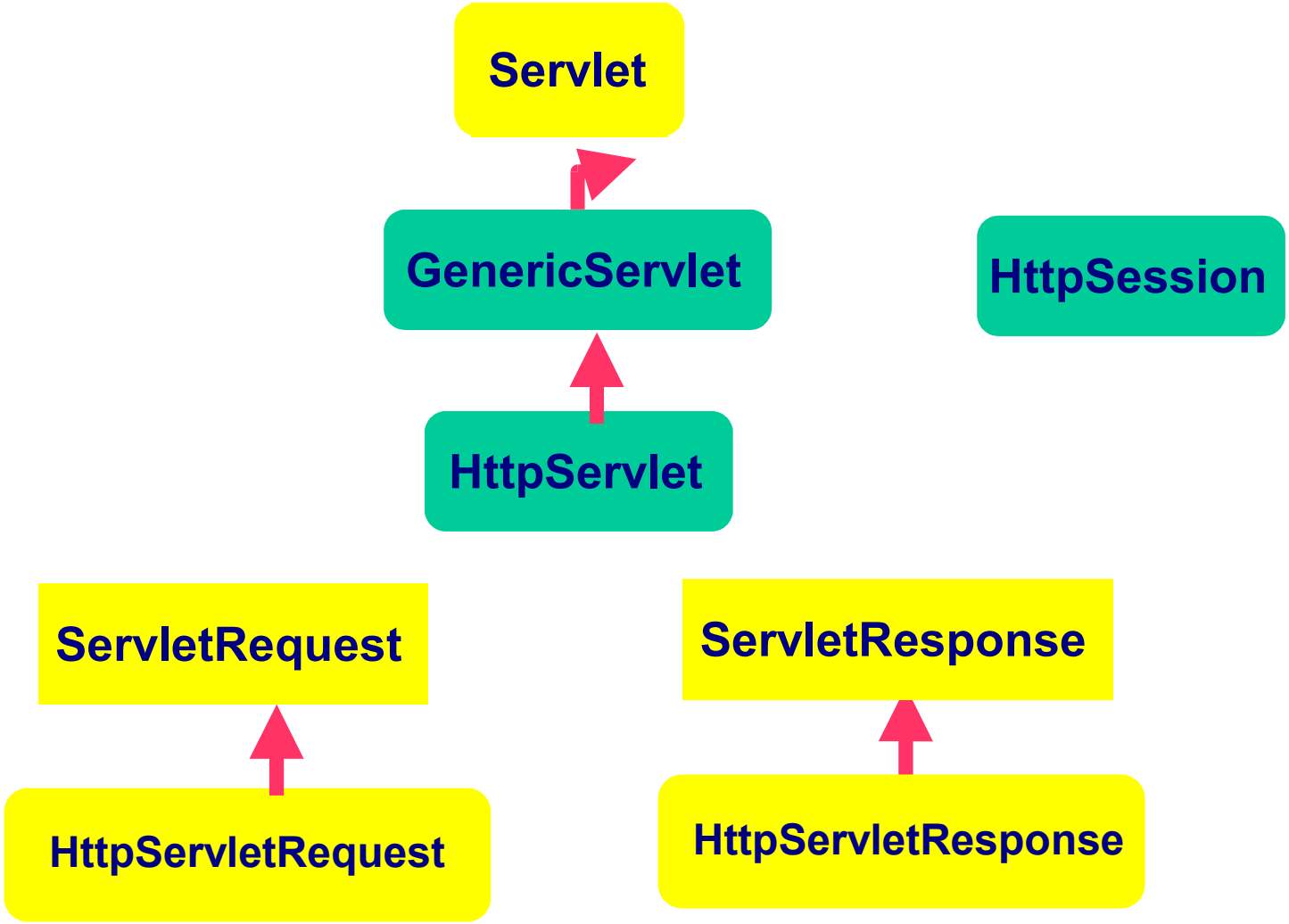
First Servlet

```
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>");
  }
}
```

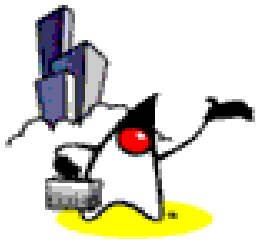
Interfaces & Classes of Servlet



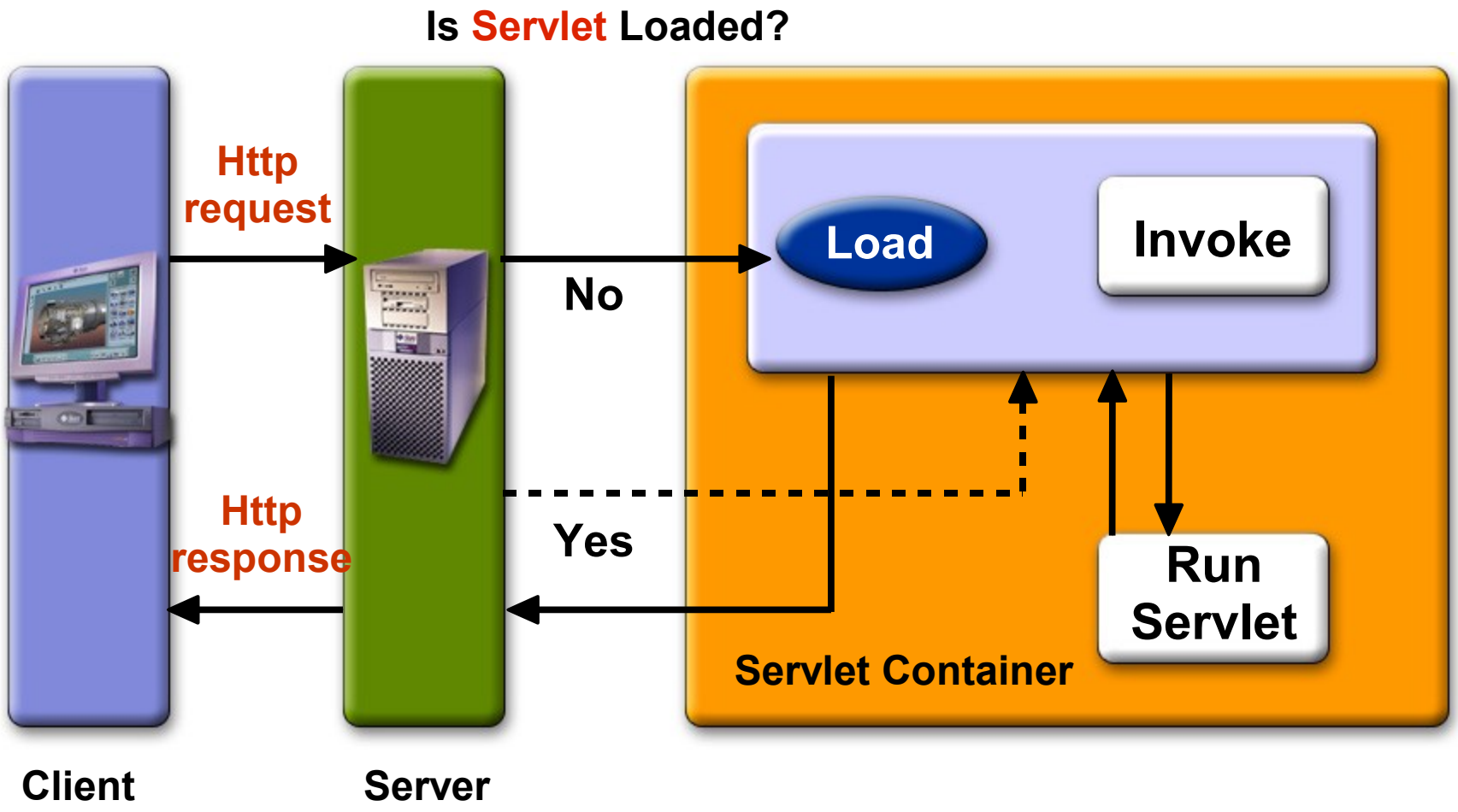
Servlet Interfaces & Classes



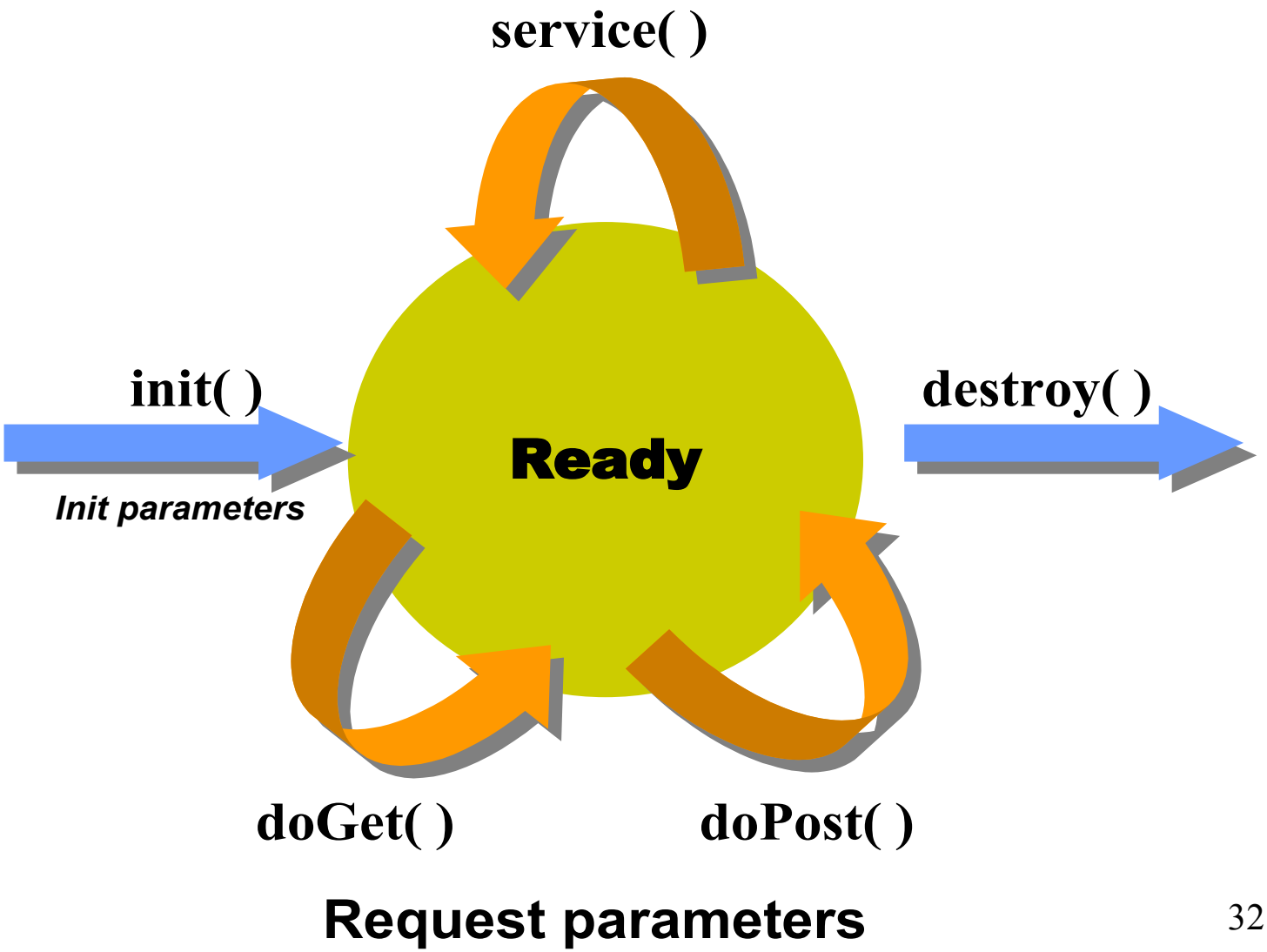
Servlet Life-Cycle



Servlet Life-Cycle



Servlet Life Cycle Methods



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

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

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.");
  }
  ...
}
```

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();
        }
 }
```

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>
```

Example: destroy()

```
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;
  }
  ...
}
```

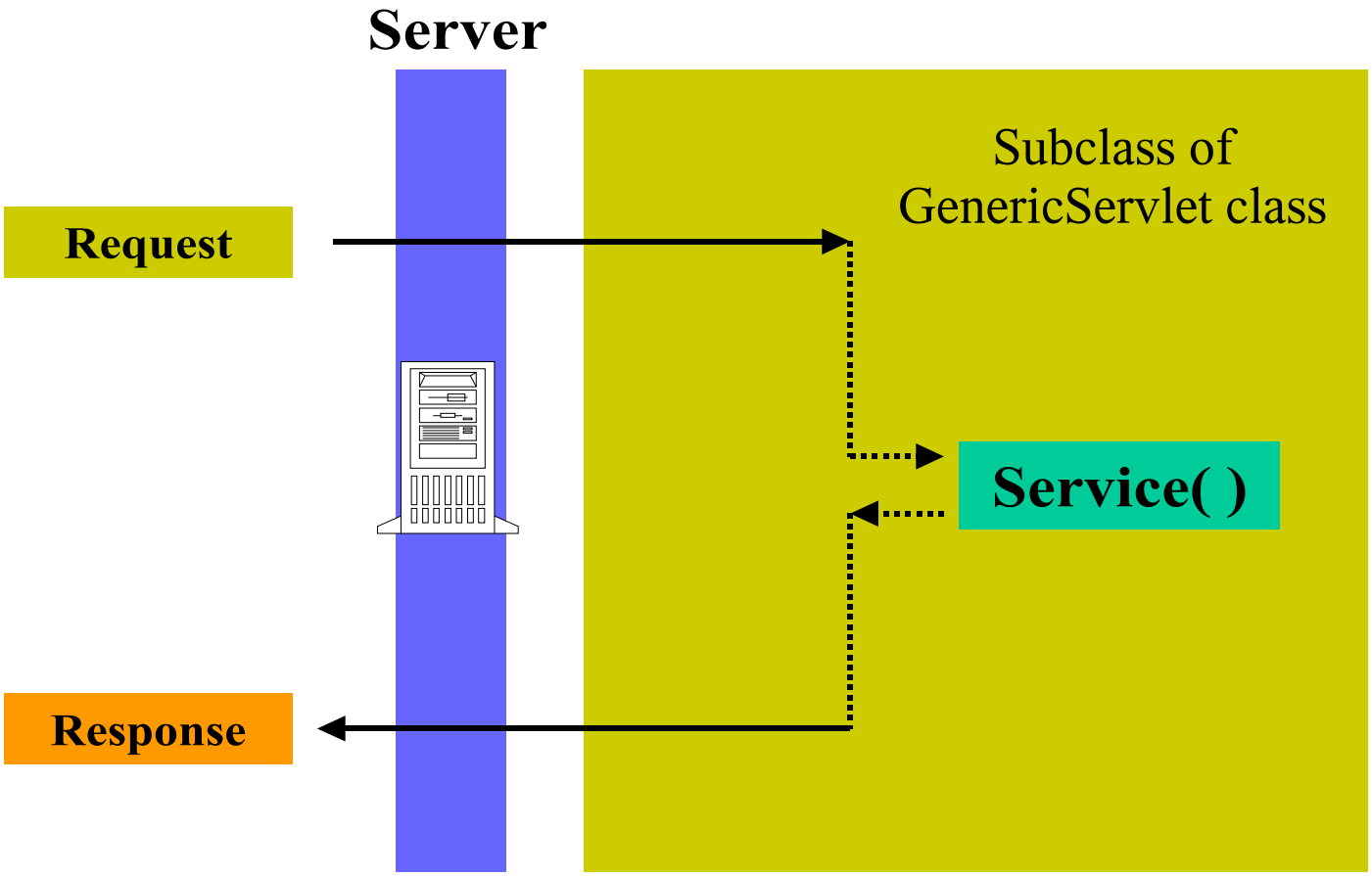
Servlet Life Cycle Methods

- 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

service() & doGet()/doPost()

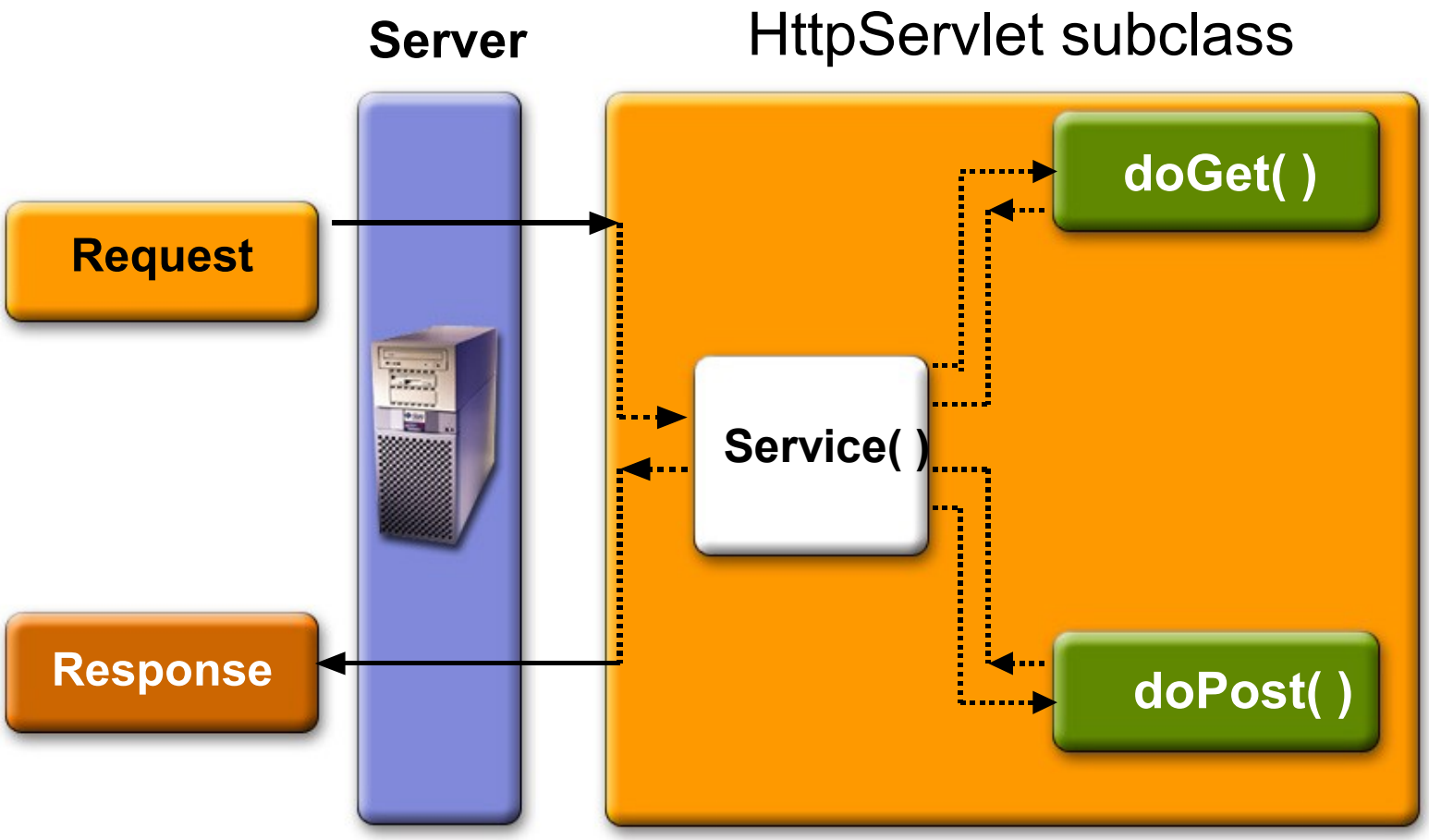
- service() methods take generic requests and responses:
  - service(ServletRequest request, ServletResponse response)
- doGet() or doPost() take HTTP requests and responses:
  - doGet(HttpServletRequest request, HttpServletResponse response)
  - doPost(HttpServletRequest request, HttpServletResponse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

Service() Method



Key: Implemented by **subclass**

doGet() and doPost() Methods



Key: Implemented by **subclass**

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

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>");
  }
}
```

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);
```

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();
}
```

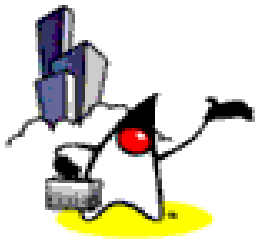
Steps of Populating HTTP Response

- 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

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>");
  }
}
```

Scope Objects



Scope Objects

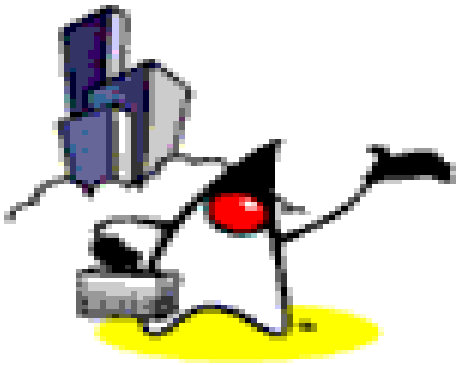
- Enables sharing information among collaborating web components via attributes maintained in Scope objects
  - Attributes are name/object pairs
- Attributes maintained in the Scope objects are accessed with
  - getAttribute() & setAttribute()
- 4 Scope objects are defined
  - Web context, session, request,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

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

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



Web Context (ServletContext)

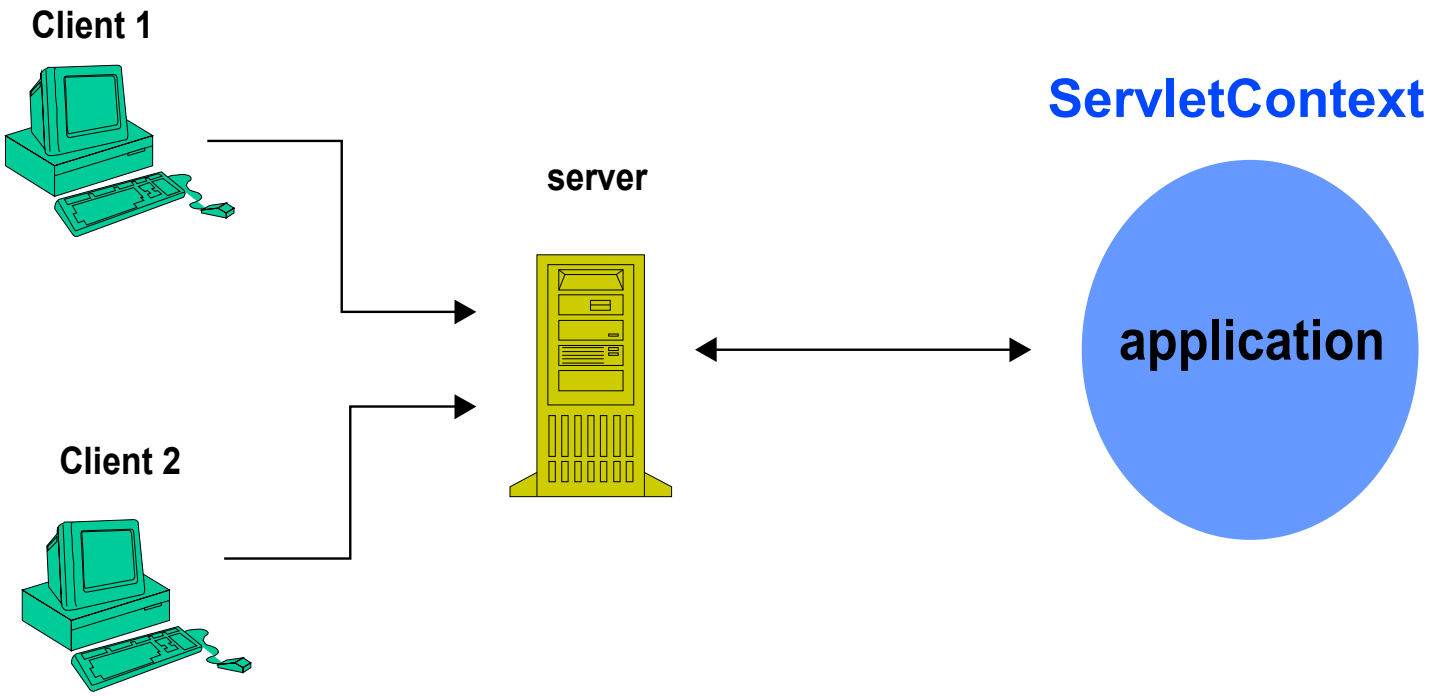
What is ServletContext For?

- 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

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

ServletContext: Web Application Scope



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

Example: Getting Attribute Value from ServletContext

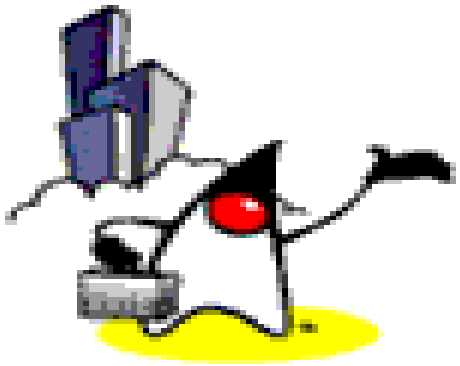
```
public class CatalogServlet extends HttpServlet {
  private BookDB bookDB;
  public void init() throws ServletException {
    // Get context-wide attribute value from
    // ServletContext object
    bookDB = (BookDB)getServletContext().
                     getAttribute("bookDB");
    if (bookDB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

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 =
               session.getServletContext().getRequestDispatcher("/banner");

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

Example: Logging

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

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



Session (HttpSession)

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

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

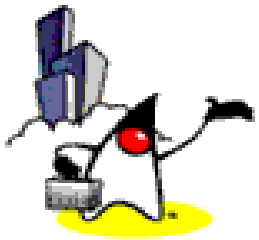
How to Get HttpSession?

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

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();
```

Servlet Request (HttpServletRequest)

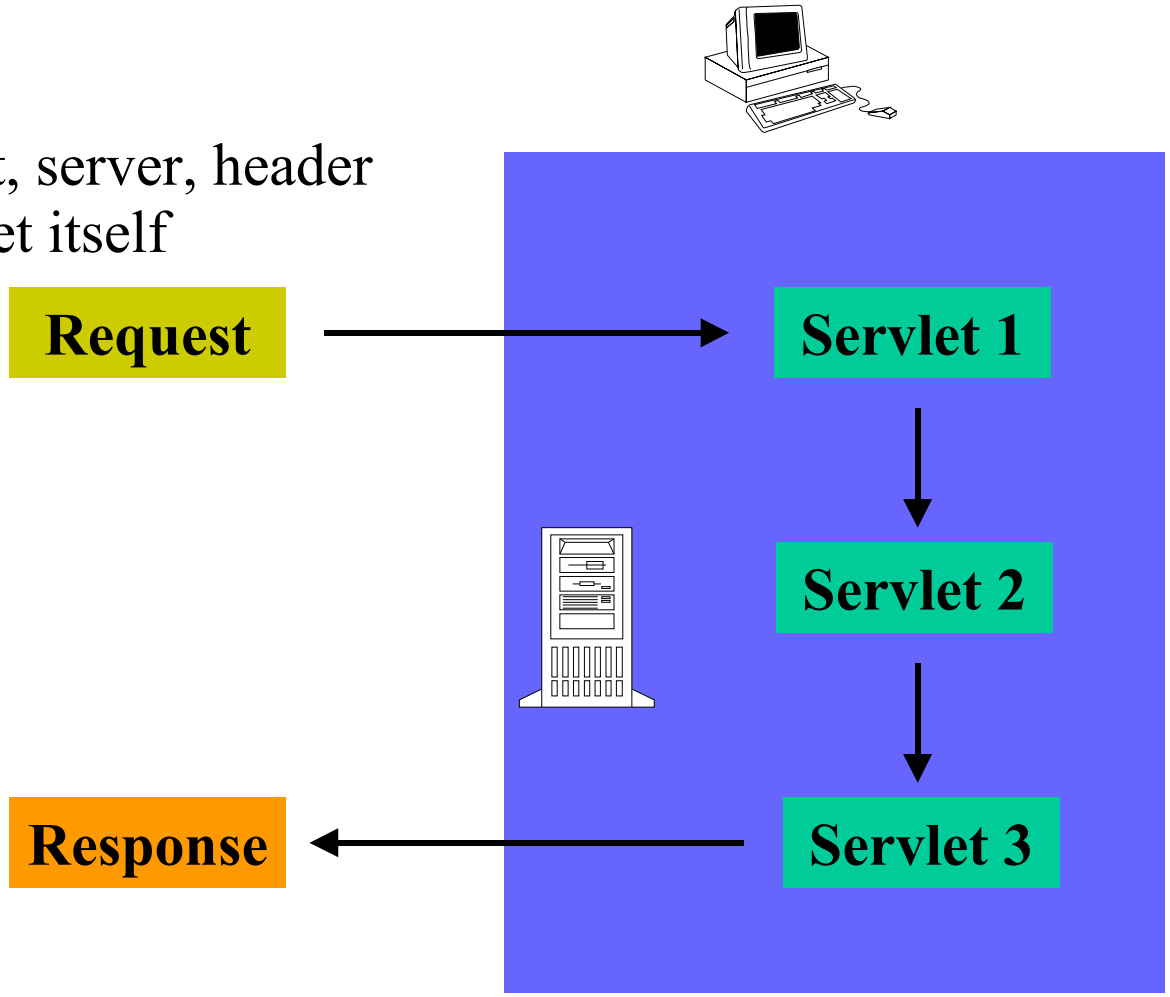


What is Servlet Request?

- 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)

Requests

data,
client, server, header
servlet itself



Web Server

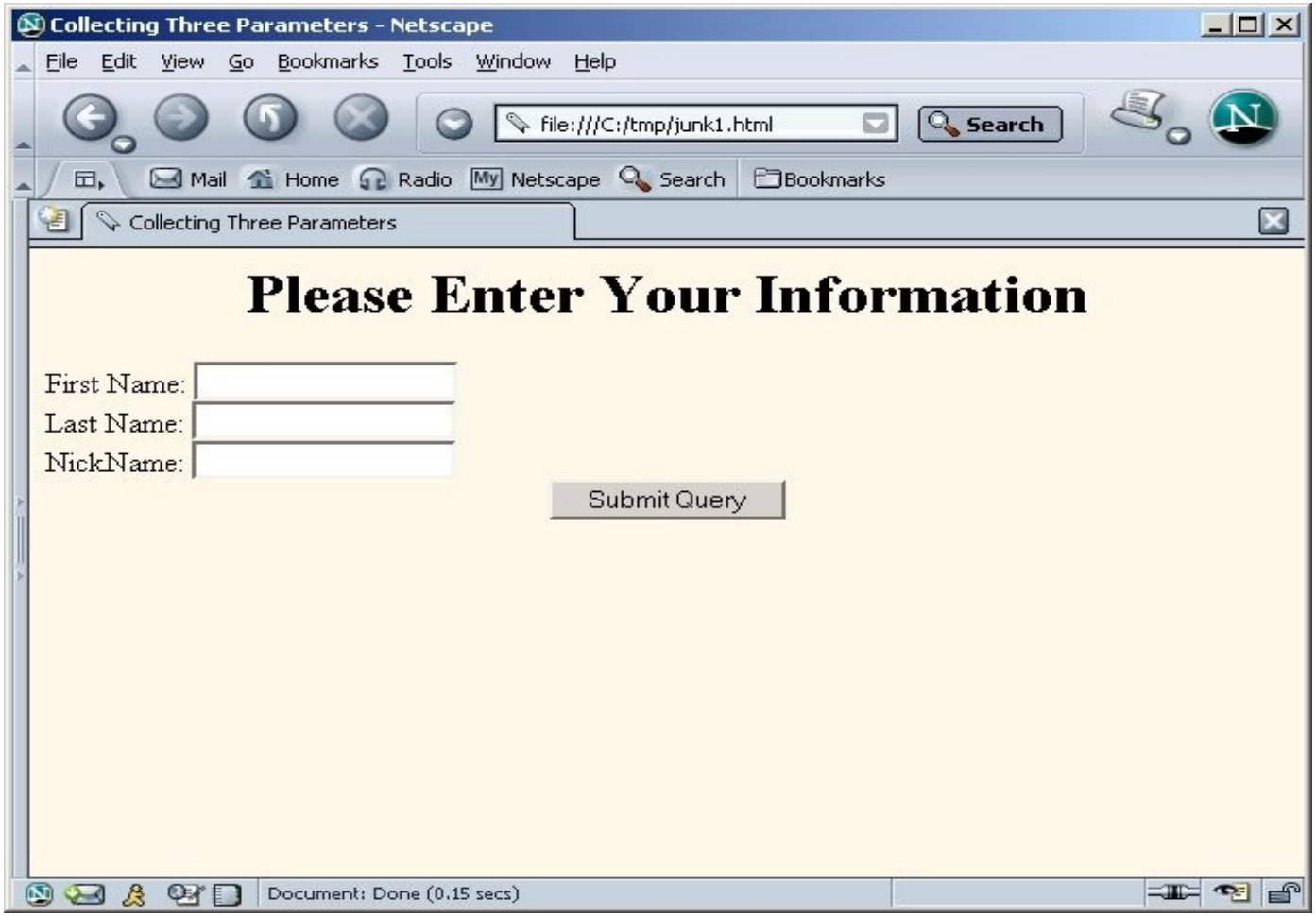
Getting Client Sent Parameters

- 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

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>
```

A Sample FORM using GET



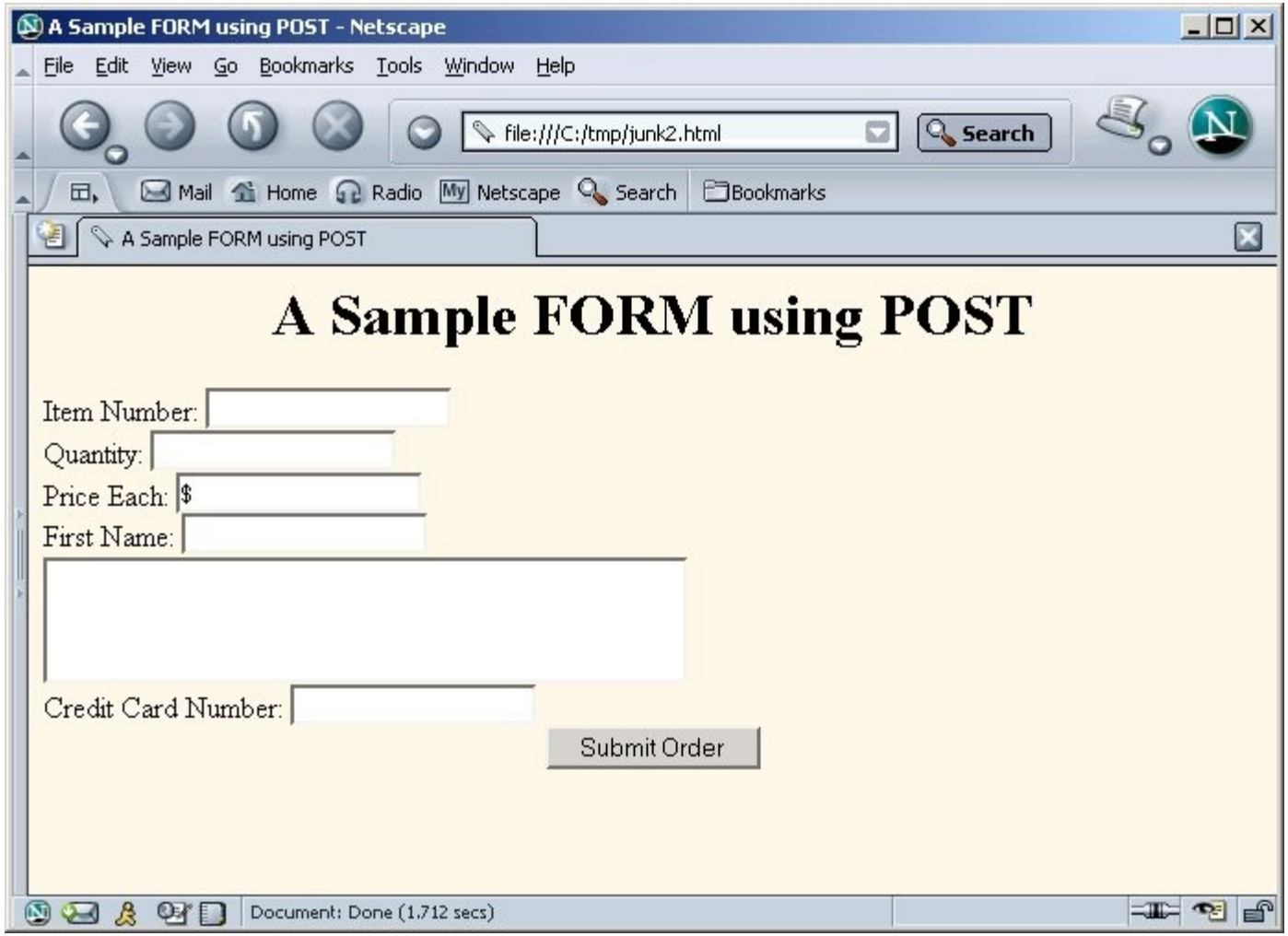
A FORM Based Servlet: Get

```
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>");
  }
}
```

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>
```

A Sample FORM using POST



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);

  }

}
```

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

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);
      }
```

Getting Client Information

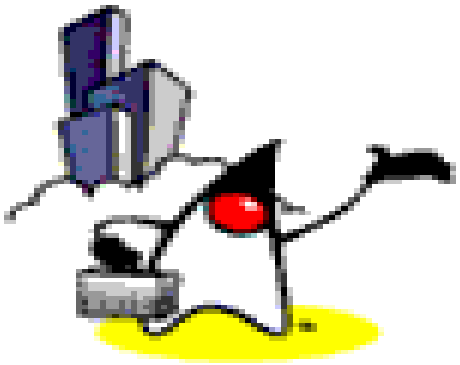
- Servlet can get client information from the request
  - String request.getRemoteAddr()
    - Get client's IP address
  - String request.getRemoteHost()
    - Get client's host name

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"

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()



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

HTTP Request URL

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

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

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
    - Add To Cart
    - 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")

Context, Path, Query, Parameter Methods

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

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)
```

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

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.

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!

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

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));
    }
    ...
  }
}
```

Request Headers Sample

Netscape

File Edit View Go Communicator Help

Back Forward Reload Home Search Netscape Print Security Shop Stop

RealPlayer

Bookmarks Location: http://localhost:8080/sample/servlet/ShowRequestHeaders What's Related

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] (Win95; 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

Document: Done

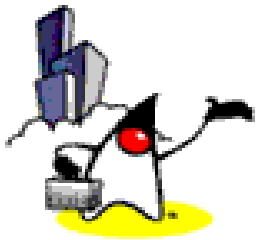
Authentication & User Security Information Methods

- 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

Cookie Method (in HTTPServletRequest)

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

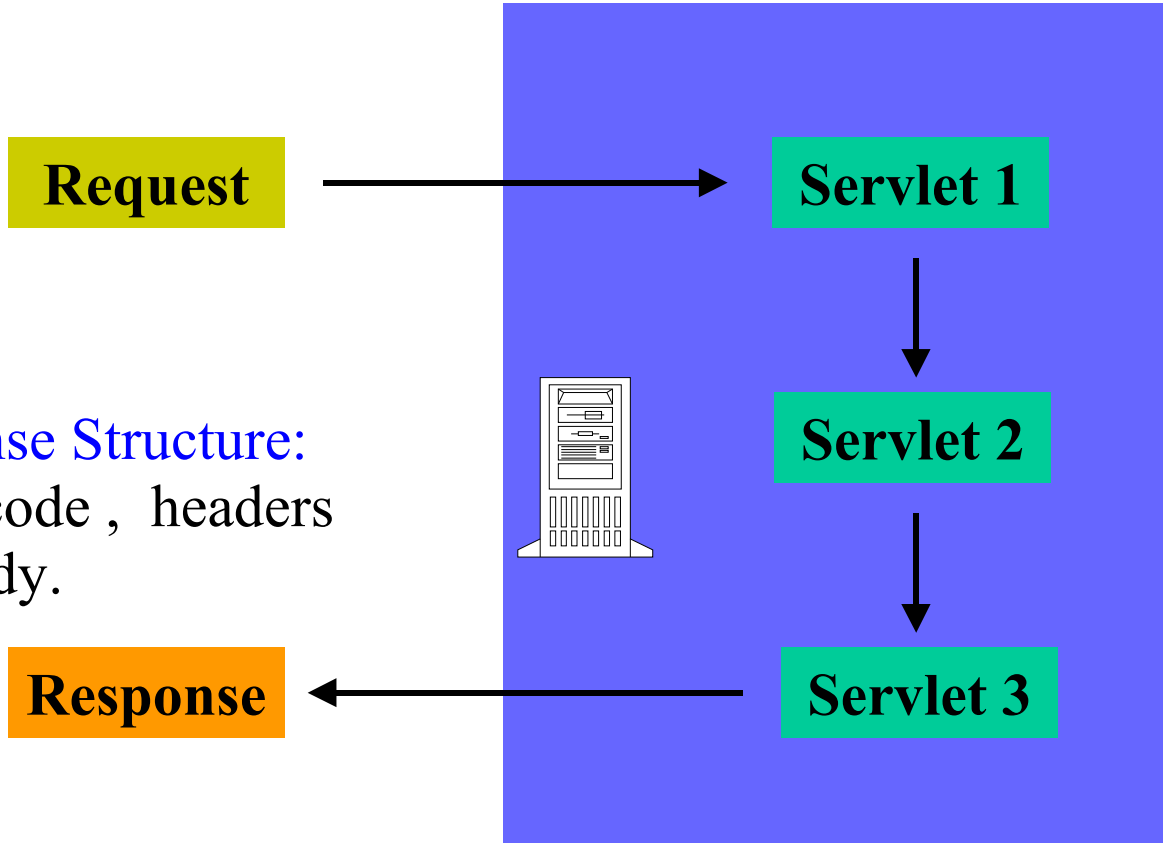
Servlet Response (HttpServletResponse)



What is Servlet Response?

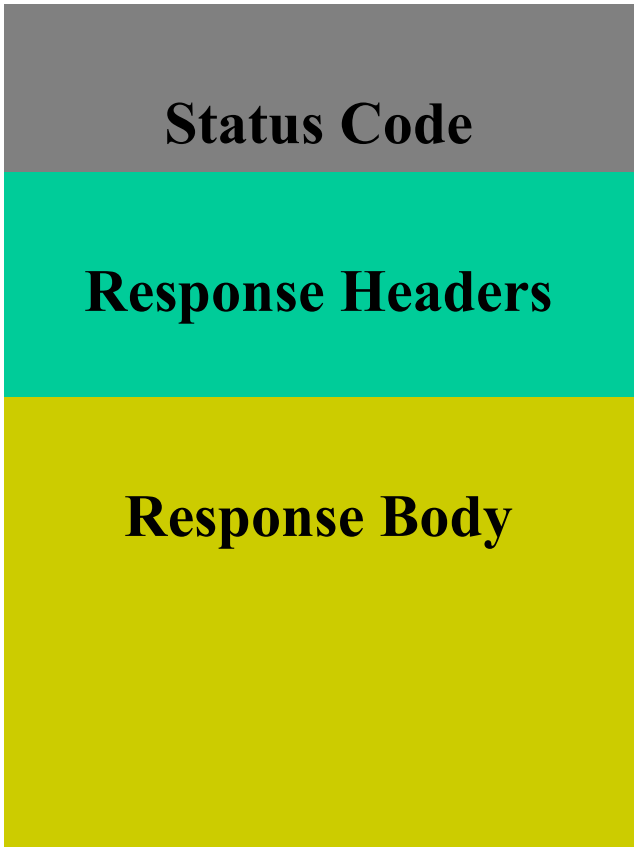
- Contains data passed from servlet to client
- All servlet responses implement ServletResponse interface
  - Retrieve an output stream
  - Indicate content type
  - Indicate whether to buffer output
  - Set localization information
- HttpServletResponse extends ServletResponse
  - HTTP response status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

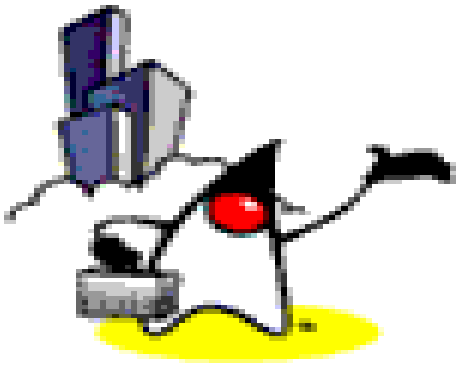
Responses



Response Structure:
status code , headers and body.

Response Structure





Status Code in Http Response

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

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)

Example of HTTP Response Status

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

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

Common Status Codes

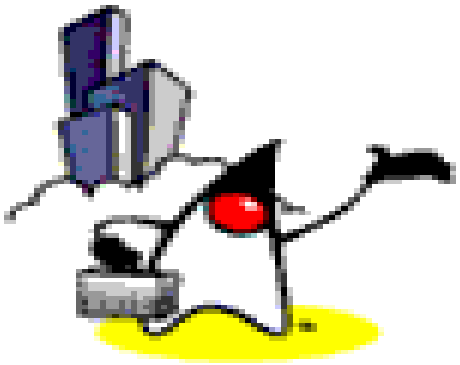
- 302 (SC_MOVED_TEMPORARILY)
  - Note the message is "Found"
  - Requested document temporarily moved elsewhere (indicated in Location header)
  - Browsers go to new location automatically
  - Servlets should use sendRedirect, not setStatus, when setting this header
- 401 (SC_UNAUTHORIZED)
  - Browser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

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

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);
}
```



Header in 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.

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.

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.

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.

Common HTTP 1.1 Response Headers (cont.)

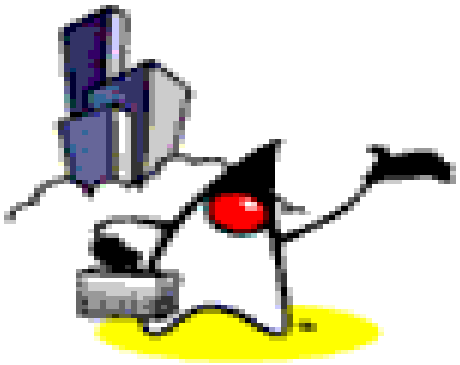
- Cache-Control (1.1) and Pragma (1.0)
  - A no-cache value prevents browsers from caching page. Send both headers or check HTTP version.
- Content- Encoding
  - The way document is encoded. Browser reverses this encoding before handling document.
- Content- Length
  - The number of bytes in the response.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

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.

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());
  }
}
```



Body in Http Response

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

Handling Errors

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

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>
```

Passion!