Session 3:

Servlet Programming

Contents

- Overview of Servlet technology
- Servlet request & response model
- Servlet life cycle
- Servlet scope objects
- Servlet request
- Servlet response: Status, Header, Body

What is a Servlet?

- Java's answer to the Common Gateway Interface (CGI).
- Applet: a java program that runs within the web browser.
- Servlet: a java program that runs within the web server.
- Rapidly becoming the standard for building web applications.

The Problems with Applets and CGI

- Java Applets have three major drawbacks:
  - Take time to load onto client
  - May not work as planned (depends on JVM)
  - Security risk for client
- Server-side code is preferred for business logic

- CGI allows an application to run on server but creates server performance problems
- Eeach time a separate process must be spawned

Servlets

- Servlets overcome this problem
- Servlets relie on a Servlet Engine (Application Server) to manage multiple requests for the same application
- Java servlets have the advantages of Java applets but run on the server side
- The Jserv engine from SUN was the first Java **servlet engine**

Servlet Engine

Three Types of Servlet Engine

- Standalone Servlet Engine

  e.g., The Jakarta Tomcat Server, IBM's Websphere Application Server. etc.

- Add-on Servlet Engine

  e.g., The Jakarta Tomcat Server. Java-Apache Project's JServ Module , Allaire's JRun Web Server , etc.

- Embeddable Servlet Engine

  e.g., The Jakarta Tomcat Server, Acme Acme.Serve

– Please see the link below

http://www.servlets.com/engines/

Applet vs. Servlet

Applet

- Client side
- Takes time to load
- JVM varies with browser
- Require compatible browser
- Security issue if client side program needs to access sensitive data via browser

Servlet

- Server side
- Runs on request
- Constant JVM
- No GUI required, can generate HTML, Javascript, Applet code
- Server side programming and data access preferred for business applications.

CGIs vs. Servlets

CGI programs

- Separate process for each CGI program
- Mod_perl and FastCGI improves performance of CGI but not to level of servlets
- Have difficult time maintaining state across requests

Servlets

- Run under single JVM (better performance)
- Servlets loaded into memory with first call, and stay in memory
- Have built in state preservation methods
- Java's inherent security
- Proprietary source code can be retained by only giving up *.class files to the server

What can you build with Servlets?

- Search Engines
- Personalization Systems
- e-Commerce Applications
- Shopping Carts
- Product Catalogs
- Intranet Applications
- Groupware Applications: bulletin boards, file sharing, etc.

Advantages of Servlets

- Servlets have six main advantages:
  - Efficient
  - Convenient
  - Powerful
  - Portable
  - Secure
  - Inexpensive

Advantage 1: Efficient

- For each browser request, the servlet spawns a light weight thread.
- This is faster and more efficient that spawning a new operating system process.
- Hence, servlets have better performance and better scalability than traditional CGI.

Advantage 2: Convenient

- Servlets include built-in functionality for:
  - Reading HTML form data
  - Handling cookies
  - Tracking user sessions
  - Setting HTTP headers

Advantage 3: Powerful

- Servlets can talk directly to the web servers.
- Multiple servlets can share data:
  - Particularly important for maintaining database connections.
- Includes powerful techniques for tracking user sessions.

Advantage 4: Portable

- One of the advantages of Java is its portability across different operating systems.
- Servlets have the same advantages.
- You can therefore write your servlets on Windows, then deploy them on UNIX.
- You can also run any of your servlets on any Java-enabled web server, with no code changes.

Advantage 5: Secure

- Traditional CGI programs have a number of known security vulnerabilities.
- Hence, you usually need to include a separate Perl/CGI module to supply the necessary security protection.
- Java has a number of built-in security layers.
- Hence, servlets are considered more secure than traditional CGI programs.

Advantage 6: Inexpensive

- You can download free servlet kits for development use.
- You can therefore get started for free!
- Nonetheless, production strength servlet web servers can get quite expensive.

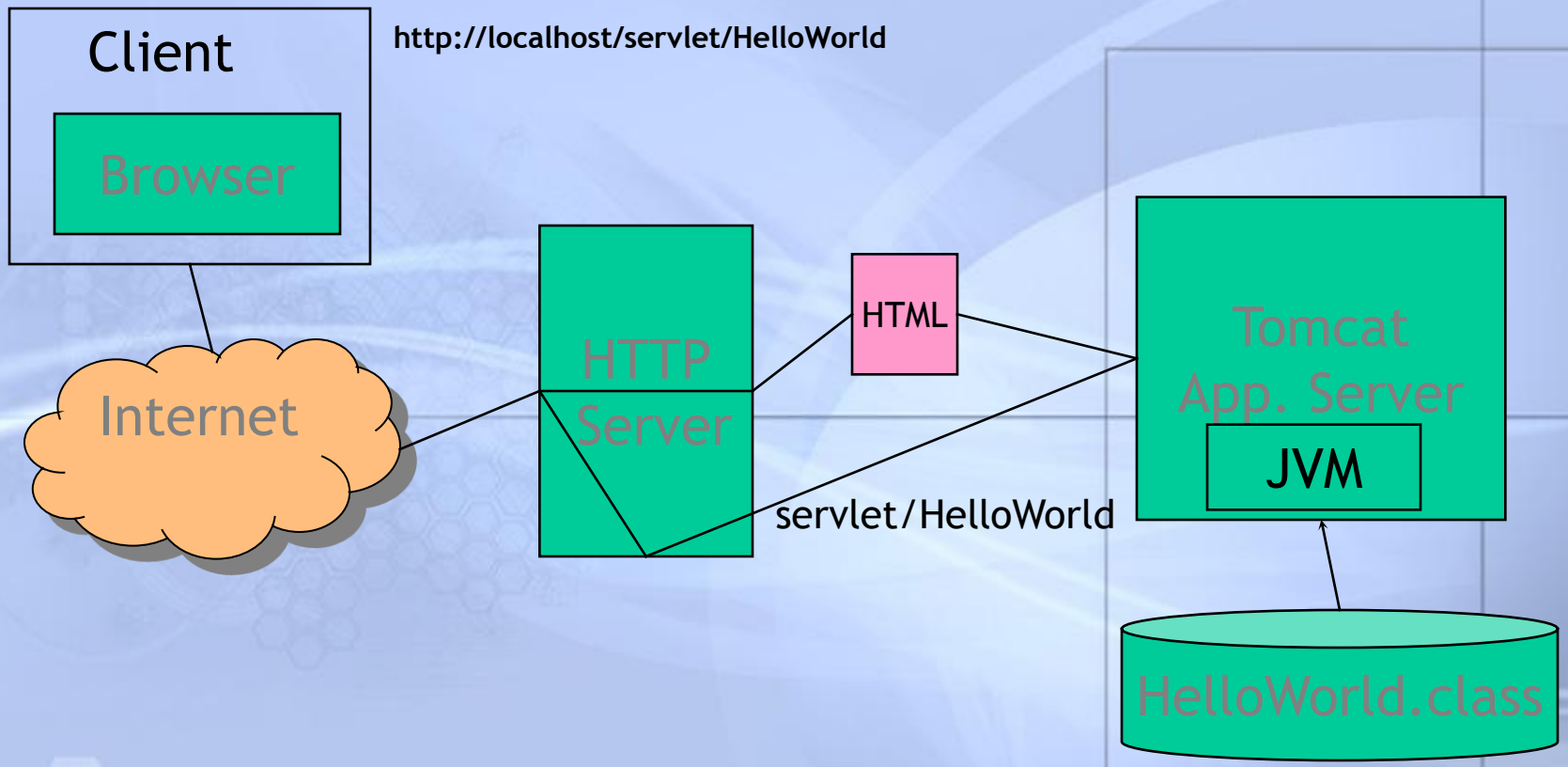
First Servlet Code

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

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest request, HttpServletResponse response)
      throws ServletException, IOException {
    // Use "request" to read incoming HTTP headers  (e.g. cookies) and HTML form data
      (e.g. data the user entered and submitted).


    // Use "response" to specify the HTTP response status  code and headers (e.g. the
      content type, cookies).


    PrintWriter out = response.getWriter();
    // Use "out" to send content to browser
  }
}
```

Example **HelloWorld** Servlet

```
import java.io.*;  import javax.servlet.*; import javax.servlet.http.*;
public class HelloWorld extends HttpServlet {
   public void doGet(HttpServletRequest request, httpServletResponse response)
   throws IOException, ServletException
   {  response.setContentType("text/html");
      PrintWriter out = response.getWriter();
      out.println("<html>");
            out.println("<body>");
                  out.println("<head>");
                        out.println("<title>Hello World!</title>");
                  out.println("</head>");
            out.println("<body>");
                  out.println("<h1>Hello World!</h1>");
            out.println("</body>");
      out.println("</html>");
   }
}
```

Java Servlet Request Processing

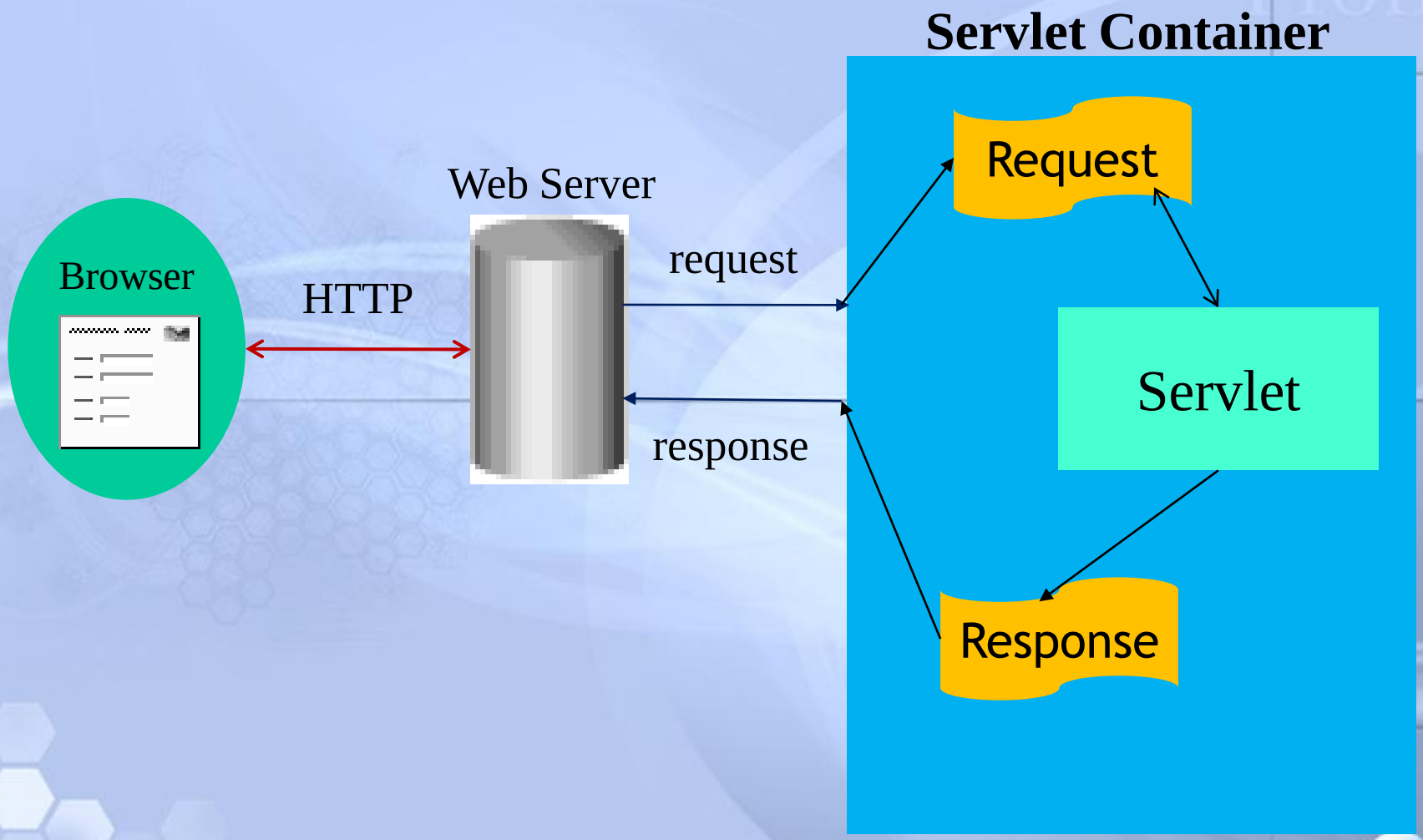


Servlet Request & Response Model

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

Servlet Request & Response Model



What does Servlet Do?

Regardless of the application, servlets usually carry out the following routine:

1) Read any data sent by the user
   - Capture data submitted by an HTML form.

2) Look up any HTTP information
   - Determine the browser version, host name of client, cookies, etc.

3) Extract some information from the request and Generate the Results
   - Connect to databases, connect to legacy applications, etc.
   - Do content generation or business logic process ,invoking EJBs, etc

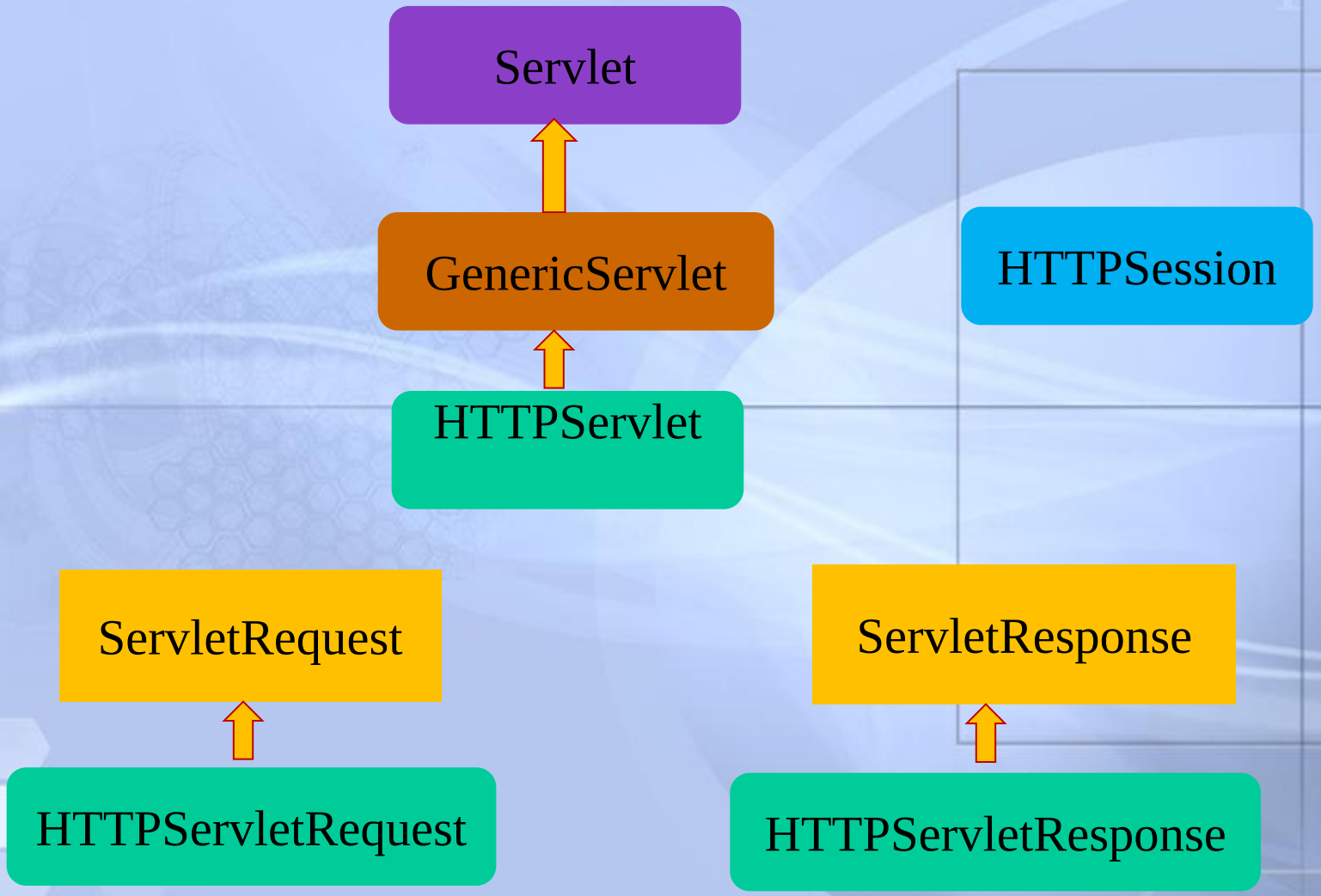
What does Servlet Do?

4) Format the Results
   - Generate HTML on the fly

5) Set the Appropriate HTTP headers
   - Tell the browser the type of document being returned or set any cookies.

6) Create and send response to client (mostly in the form of HTTP response) or forward the request to another servlet or JSP page

Servlet Interfaces & Classes

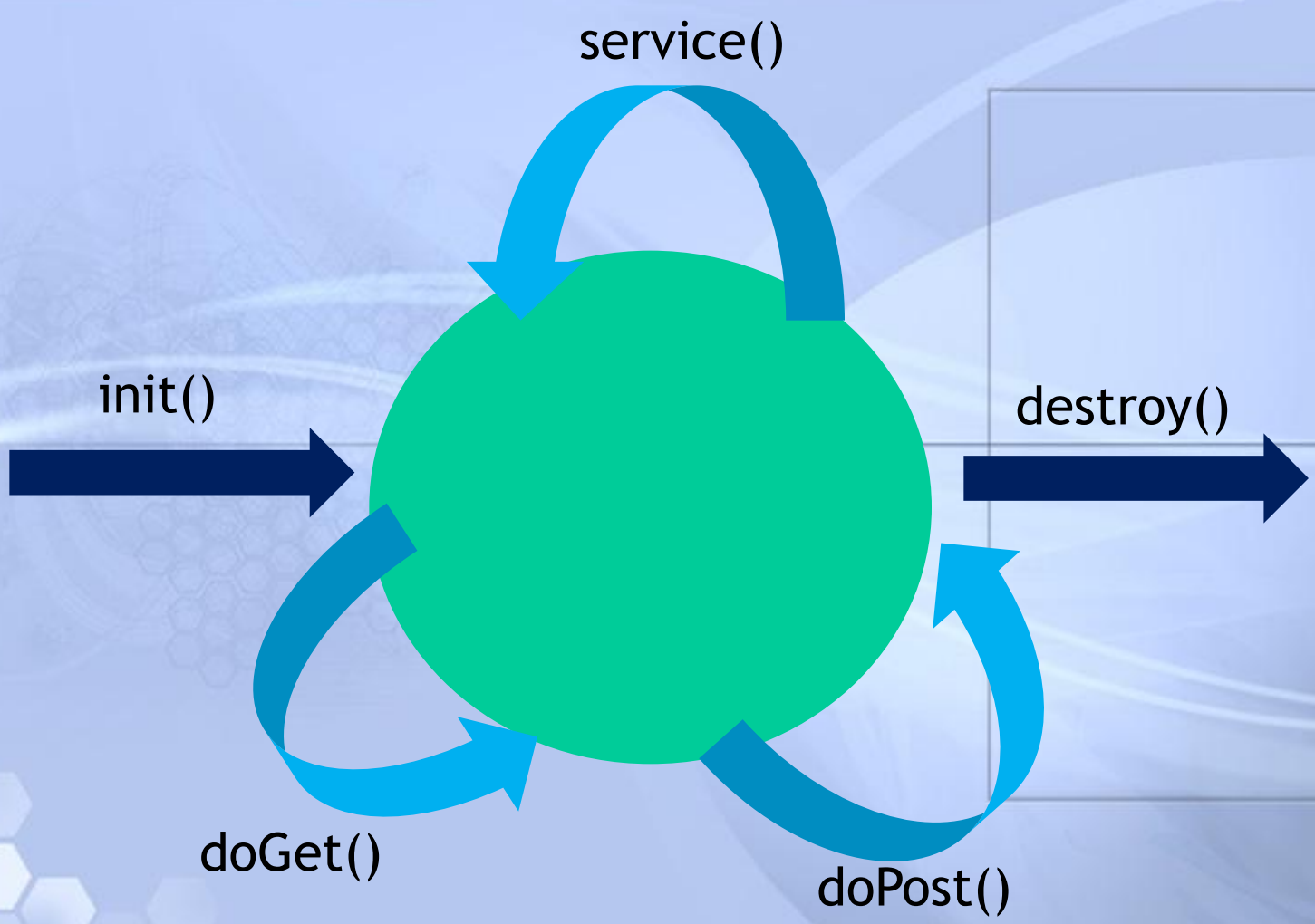


Servlet Life-Cycle

Fundamental parts of a Servlet

1. import javax.servlet.*; and import javax.servlet.http.*;
   - packages of servlet classes that implement the Java Servlet API
2. public class HelloWorld extends HttpServlet {
   - extends the HTTPServlet class
3. init()
   -intializes servlet after loading into memory
   - place to perform operations done only once at start-up
      - reading a current properties
      - clearing log files, notifying other services that the servlet is running
4. service(), doGet(), doPost()
   - this is where work is done
   - each time the servlet is called a new thread of execution begins
   - input is passed to the Java code via either HTTP GET or POST commands
5. destoy()
   - executed when server engine calls servlet to terminate
   - used to flush I/O, disconnect from database

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

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>ChartServlet</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>
```

Servlet Life Cycle Methods (Cont.)

- 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 **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)

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("<H1>Hello J2EE Programmers! </H1>");
    }
}
```

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 (ServletConext)**
  - javax.servlet.ServletContext
- **Session**
  - javax.servlet.http.HttpSession
- **Request**
  - subtype of javax.servlet.ServletRequest: javax.servlet.http.HttpServletRequest
- **Page**
  - javax.servlet.jsp.PageContext

What is ServletContext For?

- Used by servets to
  - Set and get context-wide (application-wide) object-valued attributes
  - Get RequestDispatcher
    - 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“
    - 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
  - There is one ServletContext object per "web application" per Java Virtual Machine

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

Using RequestDispatcher Object

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


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

        (or)

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

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

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

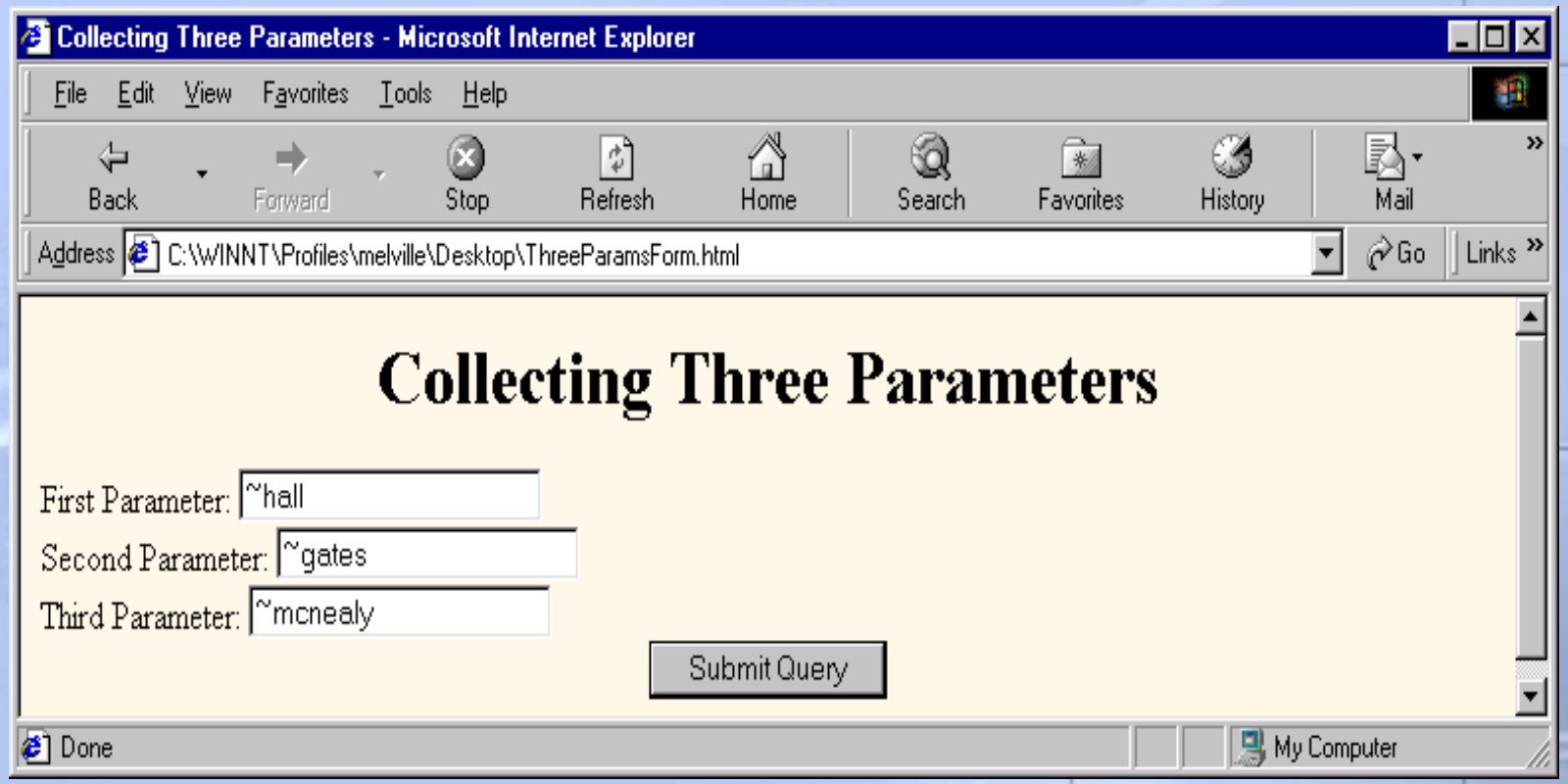
Example **HTML Post Form**

```
<FORM ACTION="/servlet/hall.ThreeParams"  METHOD="POST">

    First Parameter:  <INPUT TYPE="TEXT" NAME="param1"><BR>
    Second Parameter: <INPUT TYPE="TEXT" NAME="param2"><BR>
    Third Parameter:  <INPUT TYPE="TEXT" NAME="param3"><BR>

    <CENTER>
        <INPUT TYPE="SUBMIT">
    </CENTER>
</FORM>
```

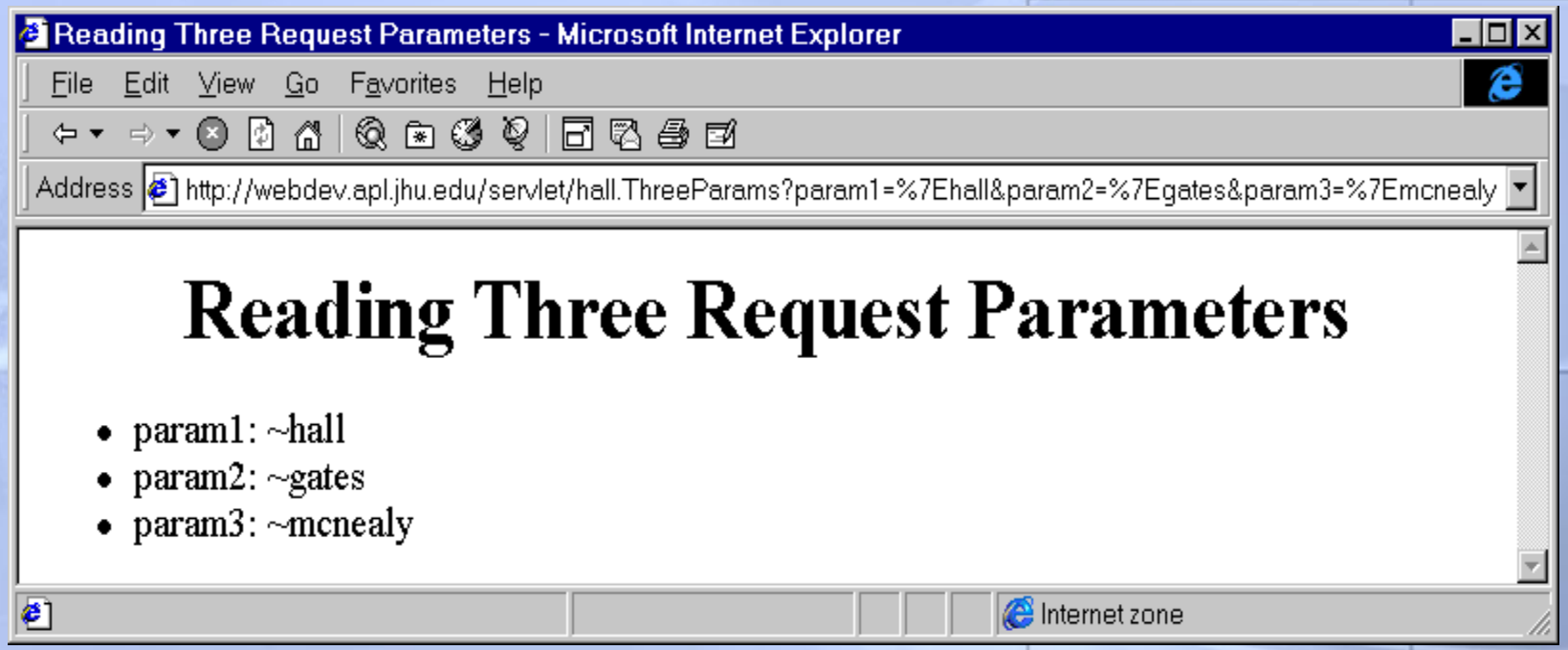
Reading Parameters

```
public class ThreeParams extends HttpServlet {
 public void doGet(HttpServletRequest request, HttpServletResponse response)
    throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println(… +"<UL>\n" +
     "<LI>param1: " + request.getParameter("param1") + "\n" +
    "<LI>param2: " + request.getParameter("param2") + "\n" +
    "<LI>param3: " + request.getParameter("param3") + "\n" +
    "</UL>\n" + …);
    }
    public void doPost(HttpServletRequest request, HttpServletResponse
    response) throws ServletException, IOException {
          doGet(request, response);
  }
}
```

Form Example



Servlet Output



Reading All Params

- **Enumeration paramNames = request.getParameterNames();**
  - parameter names in unspecified order
- **String[] paramVals = request.getParameterValues(paramName);**
  - Array of param values associated with *paramName*

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 from the request
  - **String request.getServerName()**
    - E.g., www.sun.com
  - **String request.getServerPort()**
    - E.g., port number 8080

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

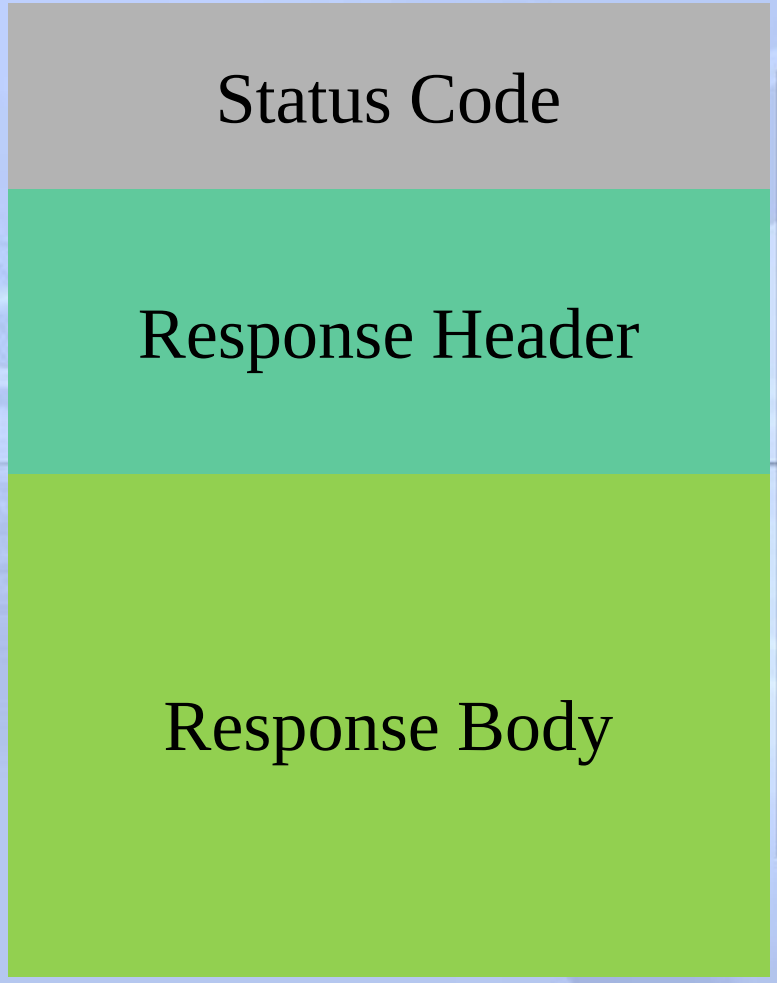
What is Servlet Response?

- 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

Response Structure



Status Code in Http Response

- 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)

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);
}
```

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.

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

Comparison between JSP & Servlet

Java Server Pages

- Related to Java Servlets
- Can be used alone or in conjunction with servlets
- Represent (yet) another method for creating server side applications

Servlets v. JSP

- **Servlets**
  - code looks like a regular Java program.
- **JSP**
  - embed Java commands directly within HTML

A Java Servlet : Looks like a regular Java program

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

public class HelloWorld extends HttpServlet {
  public void doGet(HttpServletRequest req, HttpServletResponse res)
   throws ServletException, IOException {
    res.setContentType("text/html");
    PrintWriter out = res.getWriter();
    out.println("<HTML>");
            out.println("<HEAD><TITLE>Hello World</TITLE></HEAD>");
            out.println("<BODY>");
                    out.println("<H2>Hello World</H2>");
                    out.println("The current time in milliseconds is " +
                                        System.currentTimeMillis() );
    out.println("</BODY></HTML>");
  }
}
```

A JSP Page : Looks like a regular HTML page

```
<html>
        <head>
                <title>Hello, World JSP Example</title>
        </head>
        <body>
              <h2> Hello, World!
                   The current time in milliseconds is
                            <%= System.currentTimeMillis() %>
             </h2>
        </body>
</html>
```

Embedded Java command to print current time.

Summary

- Servlet: a java program that runs within the web server.
- Servlets have lots of advantages over other server side scripting options.
- Servlets look like regular Java code with some HTML.
- Java Server Pages look like HTML with some Java.

Lets make exercise!

- **Anagrams** : Two words are anagrams if they have the same letters in the same frequency.
- e.g., **seven** and **evens** are anagrams. seven evens
- In this program, transfer everything to lower case. Your program will load words from a dictionary.
- Then, for each word, you will compute its representative by sorting the characters of the word to form a new string. seven and evens have the same representative eensv.
- Obviously, to find all the anagrams for a word, you compute its representative, and look for all other words with the same representative.
- So as you load the dictionary, you want to have a map in which the key is the representative, and the value is a list of words with that representative. Once the map is computed, answering anagram queries is trivial.

Cont.

- You need to create,
  - a simple HTML form accepting a word from the user.
  - the servlet will handle the logic, and transmit back a new HTML page with the results.

❖ You need to create a dictionary of some English words in advance.

❖ Some anagrams are :

❖ **tars star**

❖ **pots tops stop**

❖ **tub but**

❖ **tap pat**

❖ **owl low**

❖ **smart marts**

Thank You!