Advance Java - SERVLETS

Interview Readiness Program
January, 2015

People matter, results count.

Objectives of Java-Servlet

- **Purpose**:
  - Basic understanding of Servlets as server-side technology to develop web applications.
  - To understand benefit of Servlet.
- **Product**:
  - Understand Server-side programming
  - Understand the static and dynamic pages.
  - To be able to write a servlet program
  - Understand the Servlet Architecture and Life cycle of the servlet.
  - Understand the deployment steps and execution steps.
  - Servlets and packages and some utilities that help build HTML.
- **Process**:
  - Theory Sessions followed by couple of assignments
  - A review at the end of the session and a Quiz.

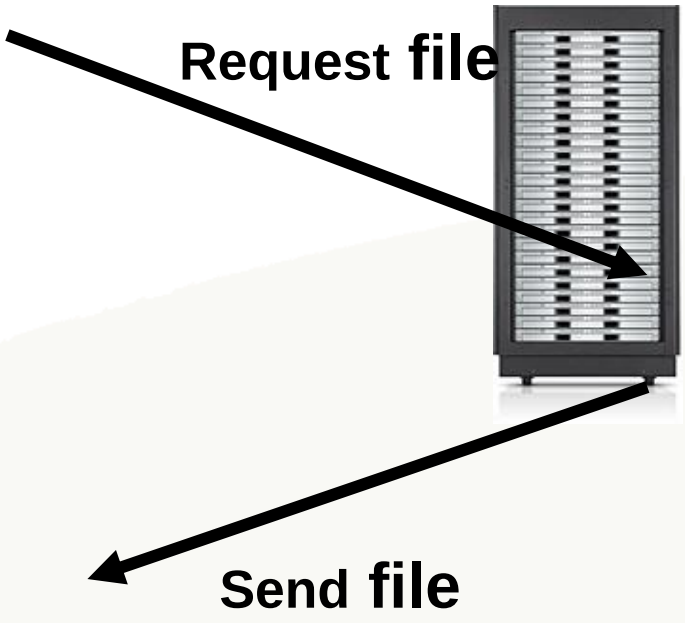
Introduction

- Java Servlets are server side components that provides a powerful mechanism for developing server side of web applications.
- With Java servlets web developers can create fast and efficient server side application and can run it on any Servlet enabled web server.
- Servlets runs entirely inside the Java Virtual Machine. Because the Servlet is running on the server side, it does not depend on browser compatibility. It just send the result in html formats.
- Servlets are secure, portable, and easy to use replacement for CGI
- Servlet is a dynamically loaded module that services requests from a Web server

Server-side programming

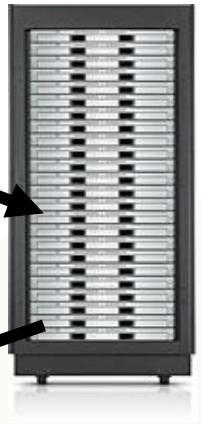
- In many cases, client-side applications will be insufficient
  - Heavy processing
  - Communication with other clients
  - Data available on server-side only
- It may be useful to send the request to the server, and to process it there.
- A number of technologies available: CGI, Servlets, JSP, ASP, PHP and others

Static Pages



Dynamic Pages

Request service



Do Computation

Generate HTML page with results of computation

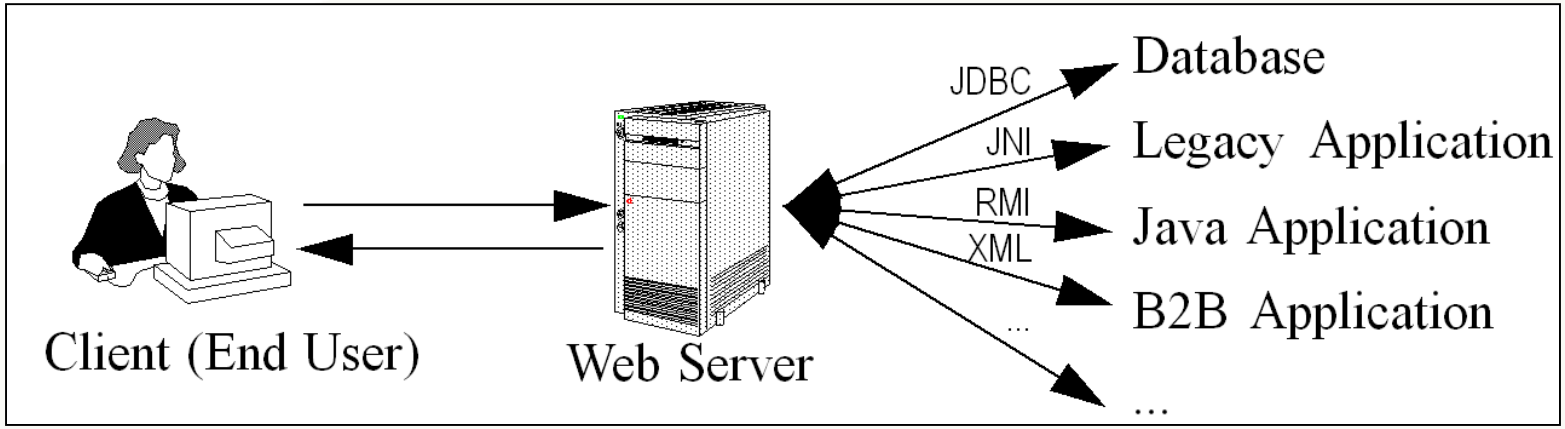
Return dynamically generated HTML file

Why Build Web Pages Dynamically?

- The Web page is based on data submitted by the user
  - E.g., results page from search engines and order-confirmation pages at on-line stores
- The Web page is derived from data that changes frequently
  - E.g., a weather report or news headlines page
- The Web page uses information from databases or other server-side sources
  - E.g., an e-commerce site could use a servlet to build a Web page that lists the current price and availability of each item that is for sale.

A Servlet's Job

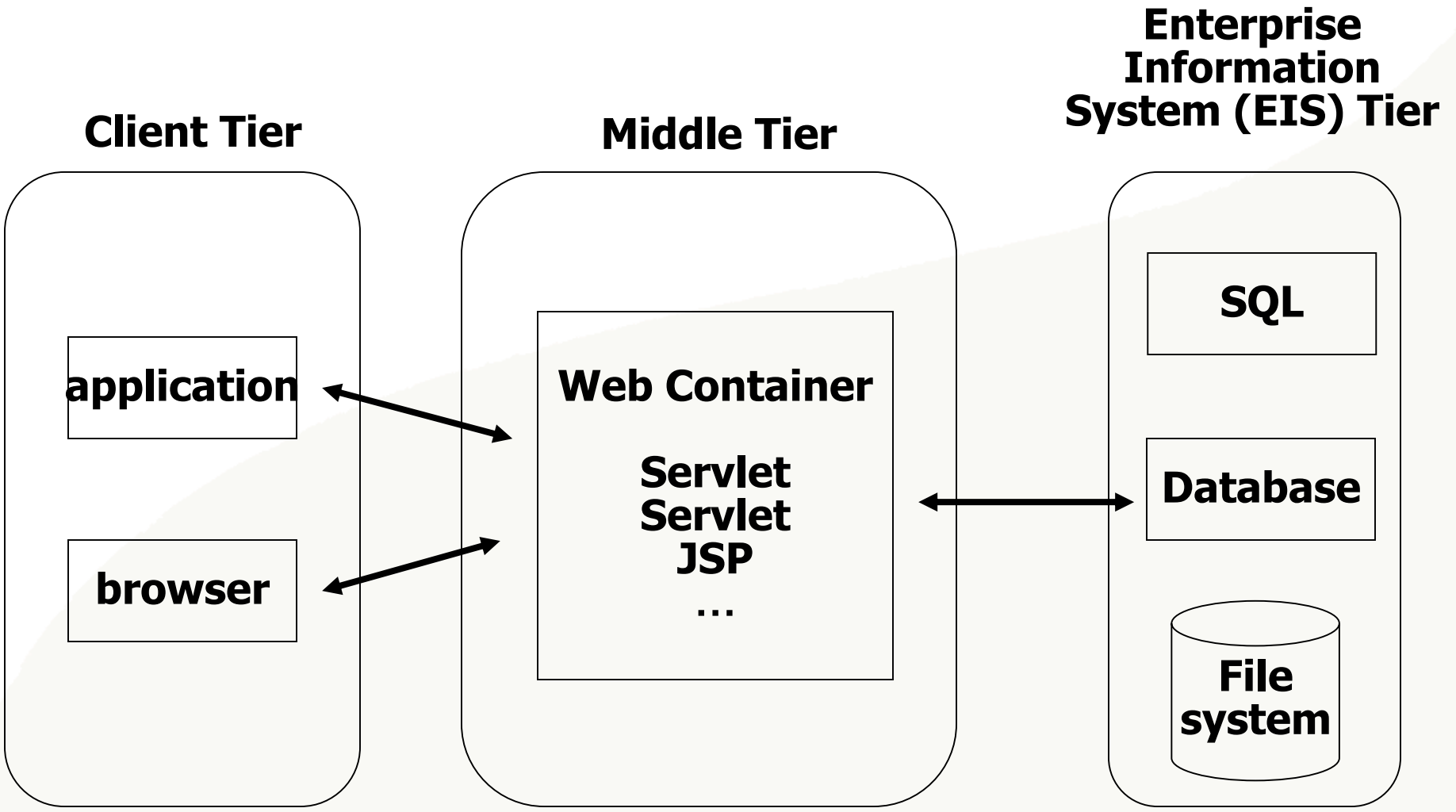
- Read explicit data sent by client (form data)
- Read implicit data sent by client (request headers)
- Generate the results
- Send the explicit data back to client (HTML)
- Send the implicit data to client (status codes and response headers)



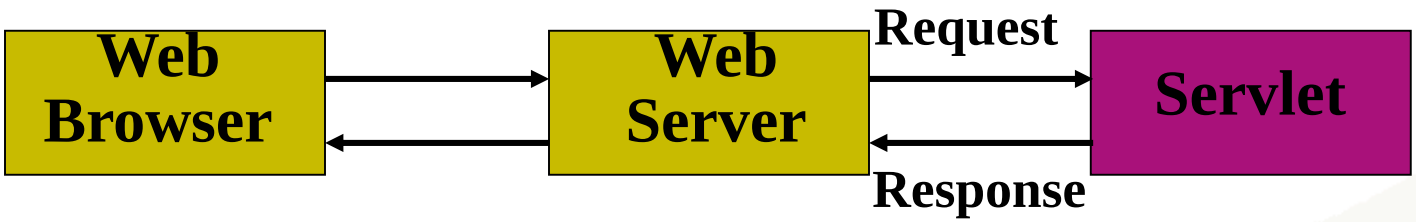
Servlet Architecture: 3-Tier system

- Tier 1: Client
  - HTML browser
  - Java client
- Tier 2: Servlets
  - embody business logic
  - secure, robust
- Tier 3: Data Sources
  - Java can talk to SQL, JDBC, OODB, files, etc…

Web Application model



Execution of Java Servlet



Java Servlet Alternatives

- CGI – Common Gateway Interface
  - New process for every CGI request
    - Slow response time
    - If CGI program terminates before responding to web server, the browser just waits for a response until it times out
- Proprietary APIs
  - NSAPI – Netscape Server API
  - ISAPI – IIS Server API
    - Dynamic link libraries
- Server-Side JavaScript
  - Embedding javascript into precompiled HTML pages – only few servers support it

Advantages of Servlets

- Efficiency
  - More efficient – uses lightweight java threads as opposed to individual processes
- Persistency
  - Servlets remain in memory
  - Servlets can maintain state between requests
- Portability
  - Since servlets are written in Java, they are platform independent
- Robustness
  - Error handling, Garbage collector to prevent problems with memory leaks
  - Large class library – network, file, database, distributed object components, security, etc.

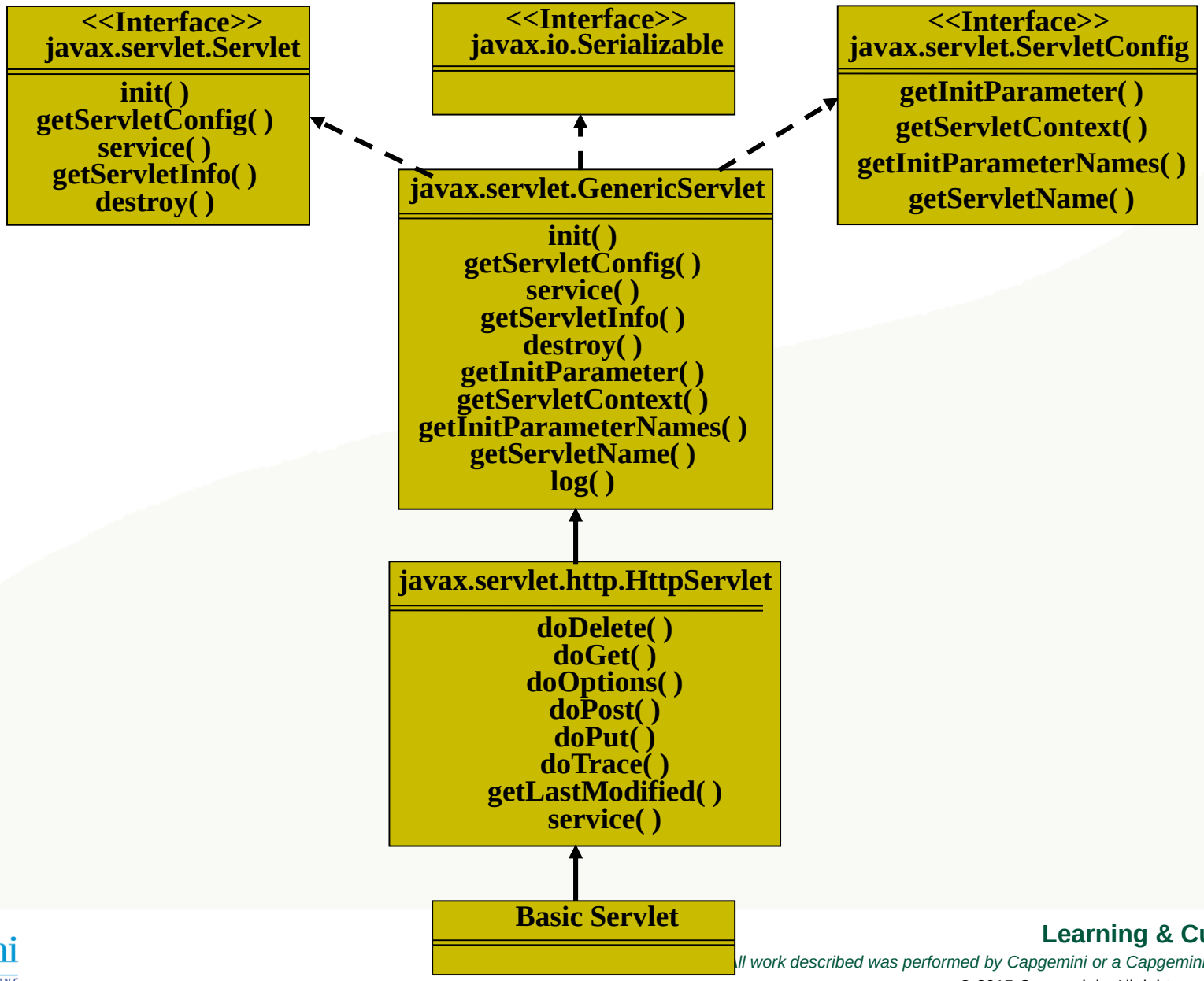
Advantages of Servlets

- Extensibility
  - Creating new subclasses that suite your needs
    - Inheritance, polymorphism, etc.
- Security
  - Security provided by the server as well as the Java Security Manager
  - Eliminates problems associated with executing cgi scripts using operating system “shells”
- Powerful
  - Servlets can directly talk to web server
  - Facilitates database connection pooling, session tracking etc.
- Convenient
  - Parsing and decoding HTML form data, reading and setting HTTP headers, handling cookies, etc.

Java Servlet Framework

- Two packages make up the servlet architecture
  - *javax.servlet*
    - Contains generic interfaces and classes that are implemented and extended by all servlets
  - *javax.servlet.http*
    - Contains classes that are extended when creating HTTP-specific servlets
- The heart of servlet architecture is the interface class *javax.servlet.Servlet*
- It provides the framework for all servlets
- Defines five basic methods – init, service, destroy, getServletConfig and getServletInfo

Object model of Servlet Framework



GenericServlet & HttpServlet

- *HttpServlet* class is extended from *GenericServlet* class
- *GenericServlet.service()* method has been defined as an abstract method
- The two objects that the service() method receives are *ServletRequest* and *ServletResponse*
- ServletRequest Object
  - Holds information that is being sent to the servlet
- ServletResponse Object
  - Holds data that is being sent back to the client

GenericServlet & HttpServlet

- Unlike the *GenericServlet*, when extending *HttpServlet*, don't have to implement the *service()* method. It is already implemented for you.

- When *HttpServlet.service( )* is invoked, it calls *doGet( )* or *doPost( ),* depending upon how data is sent from the client

- *HttpServletRequest* and *HttpServletResponse* classes are just extensions of *ServletRequest* and *ServletResponse* with HTTP-specific information stored in them

Life Cycle of a Servlet

- Applet life cycle methods: *init( ), start( ), paint( ), stop( ), and destroy( )* – appropriate methods called based on user action

- Similarly, servlets operate in the context of a request and response model managed by a servlet engine

- The engine does the following
  - Loads the servlet when it is first requested
  - Calls the servlet's ***init( )*** method
  - Handles any number of requests by calling the servlet's ***service( )*** method
  - When shutting down, calls each servlet's ***destroy( )*** method

Life Cycle – *init( )* method

- Request for a servlet received by the servlet engine
- Checks to see if the servlet is already loaded
- If not, uses a class loader to get the required servlet class and instantiates it by calling the constructor method
- After the servlet is loaded, but before it services any requests, the *init ( )* method is called
- Inside *init( ),* the resources used by the servlet are initialized. E.g.: establishing database connection
- This method is called only once just before the servlet is placed into service
- The init( ) method takes a *ServletConfig* object as a parameter
- Most common way of doing this is to have it call the *super.init( )* passing it the *ServletConfig* object

Life Cycle – *service( )* method

- The *service( )* method handles all requests sent by a client
- It cannot start servicing requests until the *init( )* method has been executed
- Only a single instance of the servlet is created and the servlet engine dispatches each request in a single thread
- The *service( )* method is used only when extending *GenericServlet* class
- Since servlets are designed to operate in the HTTP environment, the *HttpServlet* class is extended
- The *service(HttpServletRequest, HttpServletResponse)* method examines the request and calls the appropriate *doGet()* or *doPost()* method.
- A typical Http servlet includes overrides to one or more of these subsidiary methods rather than an override to *service()*

Life Cycle – *destroy( )* method

- This method signifies the end of a servlet's life
- The resources allocated during init( ) are released
- Save persistent information that will be used the next time the servlet is loaded
- The servlet engine unloads the servlet
- Calling destroy( ) yourself will not acutally unload the servlet. Only the servlet engine can do this

Servlet Life Cycle Summary

- init
  - Executed once when the servlet is first loaded.
    Not called for each request.
- service
  - Called in a new thread by server for each request.
    Dispatches to doGet, doPost, etc.
    Do not override this method!
- doGet, doPost, doXxx
  - Handles GET, POST, etc. requests.
  - Override these to provide desired behavior.
- destroy
  - Called when server deletes servlet instance.
    Not called after each request.

Servlet interface

- Central abstraction in the Servlet API
- All servlets implement this interface
  - Either directly, or
  - By extending another class that implements it
- Defines abstract methods for managing the servlet and its communications with clients
- Servlet writers provide these methods
  - While developing servlets
  - Implementing the interface

Servlet classes

- GenericServlet class
  - implements Servlet
  - also implements Serializable, ServletConfig
  - implements all Servlet methods

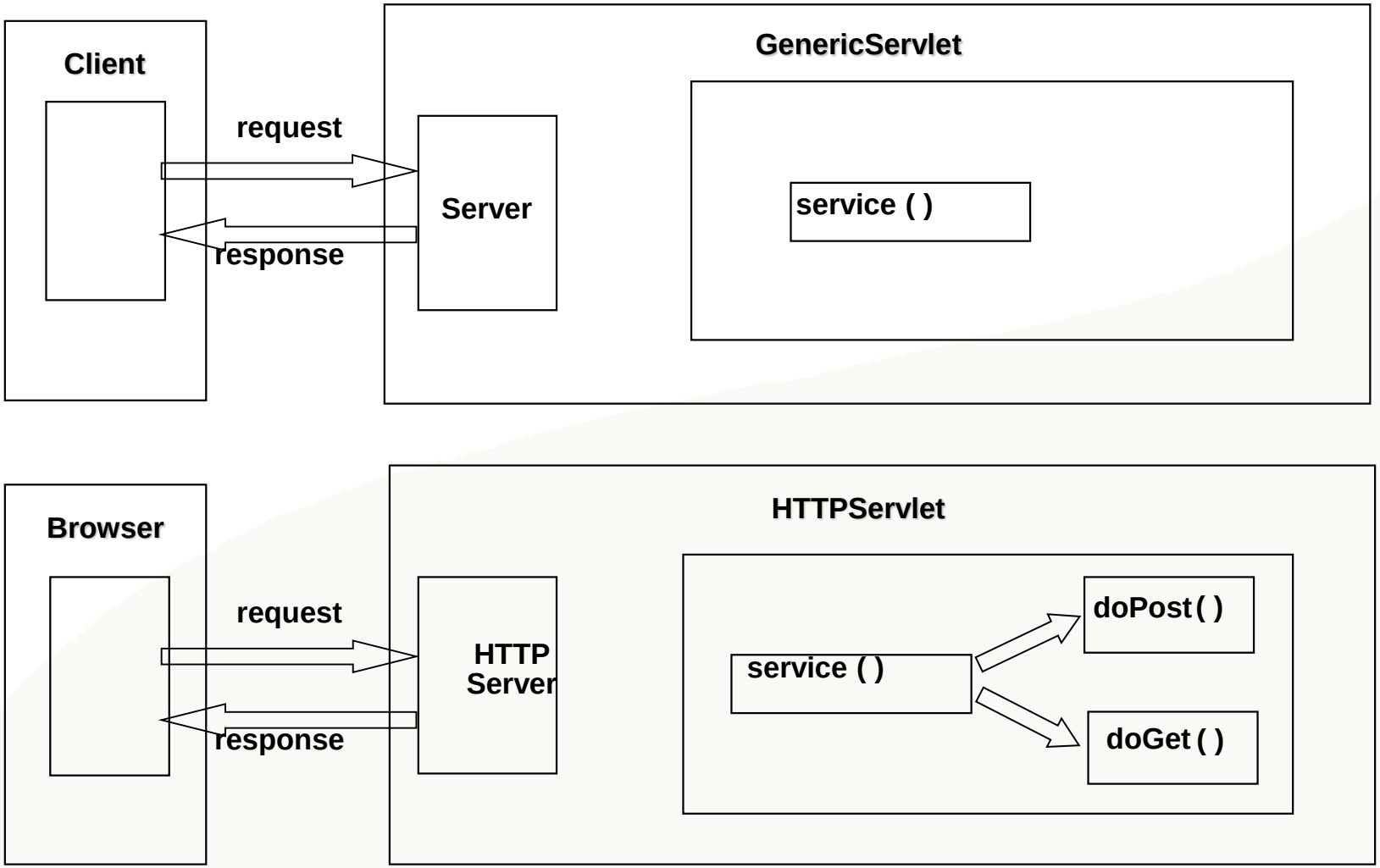
- HttpServlet class
  - extends the GenericServlet class
  - provides a framework for handling the HTTP protocol
  - has its own subclasses of ServletRequest and ServletResponse that do HTTP things

HttpServlet methods

- HTTPServlet class provides helper methods for handling HTTP requests
  - doGet (GET and HEAD)
  - doPost (POST)
  - doPut, doDelete (rare)
  - doTrace, doOptions (not overridden)

- The service() method dispatches the requests to the appropriate do* methods

Generic Servlet vs. HTTP Servlet



ServletRequest class

- Encapsulates the client-server communication
- Allows the Servlet access to
  - Names of the parameters passed in by the client
  - The protocol being used by the client
  - The names of the remote host that made the request and the server that received it
  - The input stream, ServletInputStream, through which the servlet gets data from clients
- Subclasses of ServletRequest allow the servlet to retrieve more protocol-specific data
  - HttpServletRequest for accessing HTTP-specific header information

ServletRequest - Client Info

- getRemoteAddr()
  - Returns the IP address of the client that sent the request
- getRemoteHost()
  - Returns the fully qualified host name of the client that sent the request
- getProtocol()
  - Returns the protocol and version of the request as a string <protocol>/<major version>.<minor version>.

ServletRequest - URL Info

- getScheme()
  - Returns the scheme of the URL used in this request, for example "http", "https", or "ftp".
- getServerName()
  - Returns the host name of the server receiving the request
- getServerPort()
  - Returns the port number on which this request was received
- getServletPath()
  - Returns the URL path that got to this script, e.g. “/servlet/com.foo.MyServlet”
  - Useful for putting in a <FORM> tag

ServletRequest - Contents

- getContentLength()
  - Returns the size of the request data
- getContentType()
  - Returns the MIME type of the request data
- getInputStream()
  - Returns an input stream for reading binary data in the request body.
- getReader()
  - Returns a buffered reader for reading the request body.

ServletRequest - Parameters

- String getParameter(String)
  - Returns a string containing one value of the specified parameter, or null if the parameter does not exist.
- String[] getParameterValues(String)
  - Returns the values of the specified parameter as an array of strings, or null if the named parameter does not exist.
  - Useful for parameters with multiple values, like lists
- Enumeration getParameterNames()
  - Returns the parameter names as an enumeration of strings, or an empty enumeration if there are no parameters or the input stream is empty.

ServletResponse class

- Encapsulates the server→client communication
  - Gives the servlet methods for replying to the client
  - Allows the servlet to set the content length and MIME type of the reply
  - Provides an output stream, ServletOutputStream through which the servlet can send the reply data

- Subclasses of ServletResponse give the servlet more protocol-specific capabilities.
  - HttpServletResponse for manipulating HTTP-specific header information

ServletResponse

- Embodies the response
- Basic use:

```
response.setContentType("text/html");
PrintWriter out = response.getWriter();
out.println(
  "<HTML><BODY>Hello</BODY></HTML>");
```

- setContentType() is usually called before calling getWriter() or getOutputStream()

ServletResponse - Output

- getWriter()
  - for writing text data
- getOutputStream()
  - for writing binary data
  - or for writing multipart MIME
- And many other methods, similarly to the methods of ServletRequest
- Refer the documentation

Servlet Example

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

public class ServWelcome 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("<HEAD><TITLE>First Servlet Program</TITLE></HEAD>");
    out.println("<BODY>");
out.println("<H1>Welcome to Servlets</H1>");
    out.println("</BODY>");
    out.println("</HTML>");
        out.close();
    }
}
```

Servlets are not part of the standard SDK, they are part of the J2EE

Servlets normally extend HttpServlet

The response to be sent to the client

Details of the HTTP request from the client

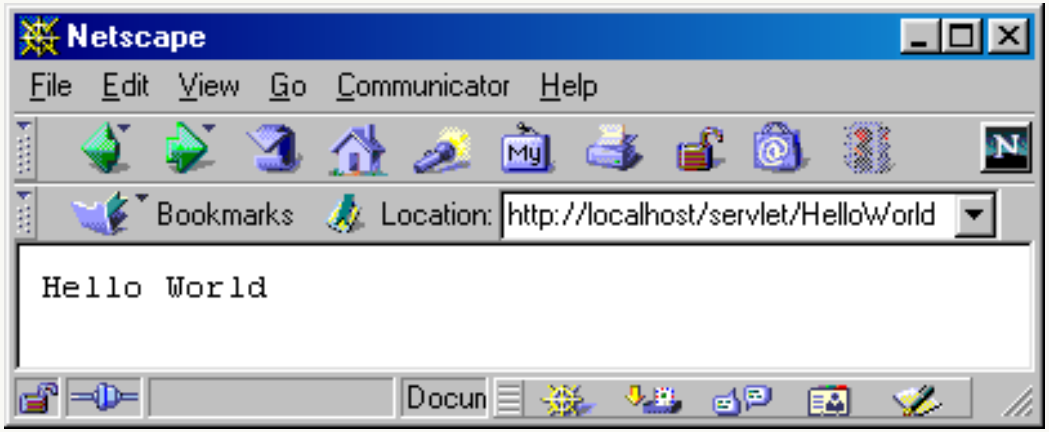
Set the response type to text/htm is normal)

Do not forget to close the connection with the client

This HTML text is sent to the client

A Simple Servlet That Generates Plain Text

```
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 {
                        PrintWriter out = response.getWriter();
                        out.println("Hello World");
        }
}
```



Generating HTML

- Set the Content-Type header
  - Use response.setContentType
- Output HTML
  - Be sure to include the DOCTYPE
- Use an HTML validation service
  - http://validator.w3.org/
  - http://www.htmlhelp.com/tools/validator/
  - If your servlets are behind a firewall, you can run them, save the HTML output, and use a file upload form to validate.

A Servlet That Generates HTML

```
public class HelloWWW extends HttpServlet {
        public void doGet(HttpServletRequest request,
                          HttpServletResponse response)
                          throws ServletException, IOException {
            response.setContentType("text/html");
            PrintWriter out = response.getWriter();
            String docType =
          "<!DOCTYPE HTML PUBLIC \"- //W3C//DTD HTML 4.0 "
            +  "Transitional//EN\">\n“;  out.println(docType
            +   "<HTML>\n"
            +   "<HEAD><TITLE>Hello WWW</TITLE></HEAD>\n"
            +   "<BODY>\n" +"<H1>Hello WWW</H1>\n"
            +   "</BODY></HTML>");
        }
}
```

Initializing Servlets

- Common in real-life servlets
  - E.g., initializing database connection pools.
- Use ServletConfig.getInitParameter to read initialization parameters
- Set init parameters in web.xml (ver 2.2/2.3)
  - …/WEB-INF/web.xml
  - Many servers have custom interfaces to create web.xml
- It is common to use init even when you don't read init parameters

A Servlet That Uses Initialization Parameters

```
public class ShowMessage extends HttpServlet {
    private String message;
    private String defaultMessage = "No message.";
    private int repeats = 1;

    public void init() throws ServletException {
        ServletConfig config = getServletConfig();
        message = config.getInitParameter("message");
        if (message == null) {
            message = defaultMessage;
        }
        try {
            String repeatString = config.getInitParameter("repeats");
            repeats = Integer.parseInt(repeatString);
        } catch(NumberFormatException nfe) {}
    }
```

ShowMessage Servlet

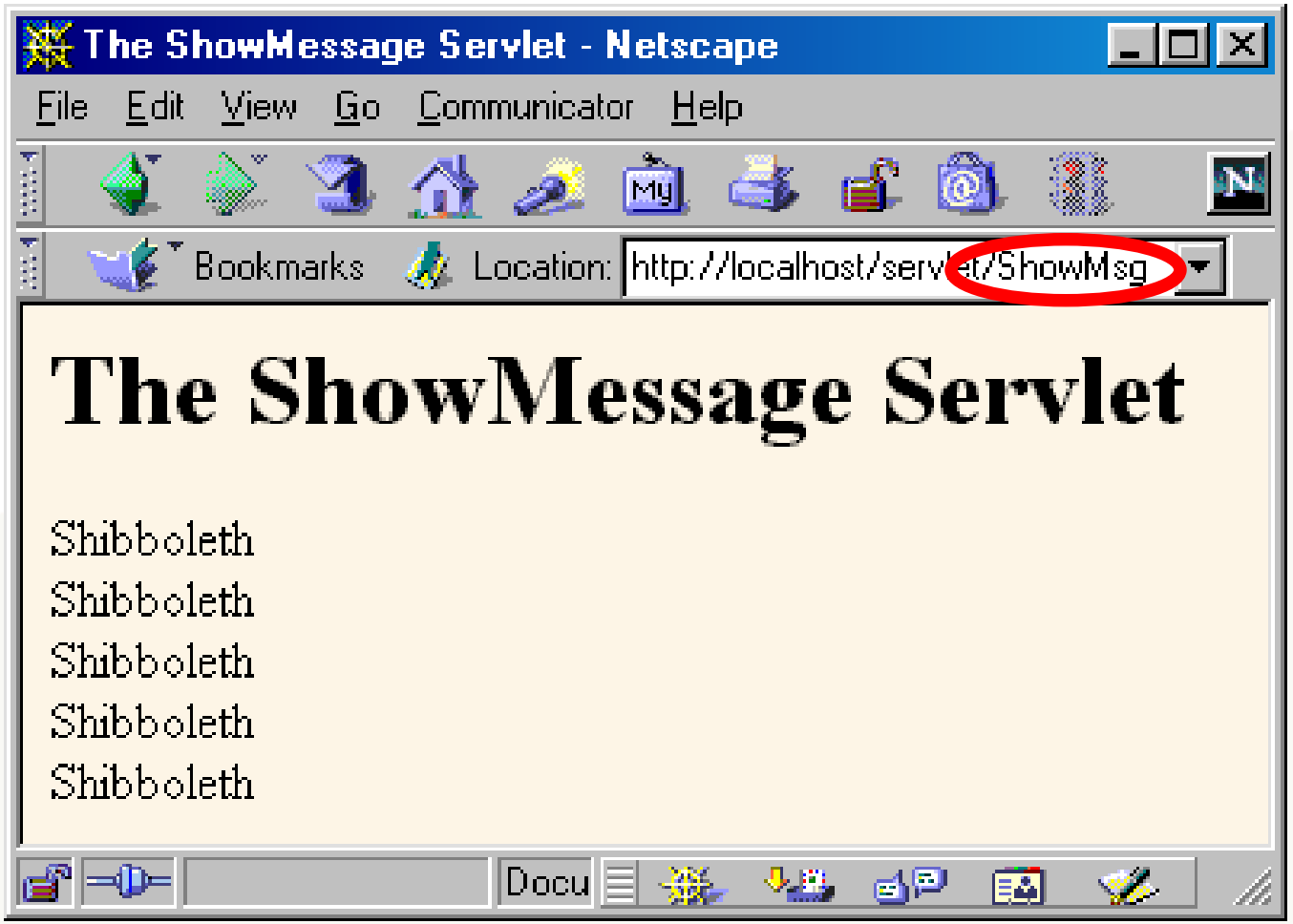
```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response)
                  throws ServletException, IOException {

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String title = "The ShowMessage Servlet";
    out.println(ServletUtilities.headWithTitle(title) +
      "<BODY BGCOLOR=\"#FDF5E6\">\n" +
      "<H1 ALIGN=CENTER>" + title + "</H1>");
    for(int i=0; i<repeats; i++) {
            out.println(message + "<BR>");
    }
    out.println("</BODY></HTML>");
  }
}
```

Setting Init Parameters (Servlets 2.2/2.3)

- ...\WEB-INF\web.xml
  - *tomcat_install_dir*\webapps\examples\WEB-INF\web.xml

```
<web-app>
  <servlet>
    <servlet-name>ShowMsg</servlet-name>
    <servlet-class>coreservlets.ShowMessage</servlet-class>
    <init-param>
      <param-name>message</param-name>
      <param-value>Shibboleth</param-value>
    </init-param>
    <init-param>
      <param-name>repeats</param-name>
      <param-value>5</param-value>
    </init-param>
  </servlet>
</web-app>
```

ShowMessage Result



Summary

- Servlets are efficient, portable, powerful, and widely accepted in industry
- Regardless of deployment server, run a free server on your desktop for development
- Getting started:
  - Set your CLASSPATH
    - Servlet JAR file
    - Top of your package hierarchy
  - Put class files in proper location
    - .../WEB-INF/classes
  - Use proper URL, usually http://*host*/servlet/*ServletName*

Summary (Continued)

- Main servlet code goes in doGet or doPost:
  - The HttpServletRequest contains the incoming information
  - The HttpServletResponse lets you set outgoing information
    - Call setContentType to specify MIME type
    - Call getWriter to obtain a Writer pointing to client
- One-time setup code goes in init
  - Servlet gets initialized and loaded once
  - Servlet gets invoked multiple times
  - Initialization parameters set in web.xml

Filters

- Request Filter
- Response Filter
- Request and Response Filters

- Implementations
  - Filter Interface
    - init()
    - destroy()
    - doFilter()
- Filter Configuration

Listeners

- 8 different Listeners

- Event classes
  - ServletRequestEvent
  - ServletContextEvent
  - ServletRequestAttributeEvent
  - ServletContextAttributeEvent
  - HttpSessionEvent
  - HttpSessionBindingEvent

Listeners

- Event interfaces
  - ServletRequestListener
  - ServletRequestAttributeListener
  - ServletContextListener
  - ServletContextAttributeListener
  - HttpSessionListener
  - HttpSessionAttributeListener
  - HttpSessionBindingListener
  - HttpSessionActivationListener

Summary

- Filters Usage
  - recording all incoming requests
  - logs the IP addresses of the computers from which the requests originate
  - conversion
  - data compression
  - encryption and decryption
  - input validation etc.
- Listeners
  - Events are basically occurrence of something. Changing the state of an object is known as an event.

Recap

Servlet

How to compile

Deploy in Web Server

Servlet Architecture

Advantages

Servlet Framework

Life Cycle

Servlet Classes and Interfaces

Thank You For Your Time

People matter, results count.