DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Subject Name: Web Technology

Day: 26

Topics Covered: Java Servlet

Prerequisites, Objectives and Outcomes

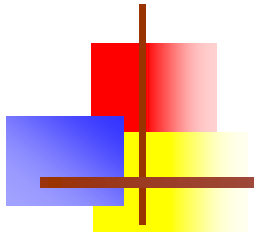
Prerequisite of topic: Basic concepts related to web programming

Objective: To make students aware about the server side programming using Servlet.

Outcome : 1. Students will be able to use Servlet as a server side technology.

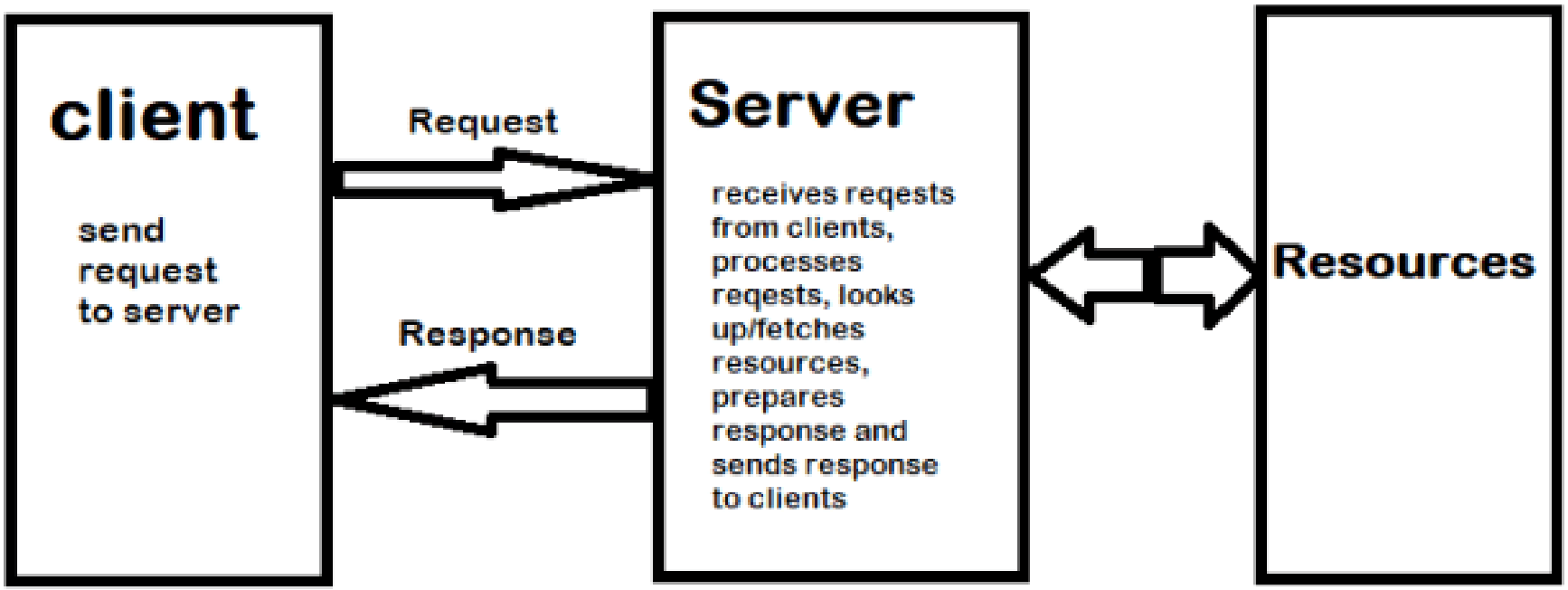
2. Students will be able to use web server along with deployment of application

3. Students will be able to implement in practical applications.



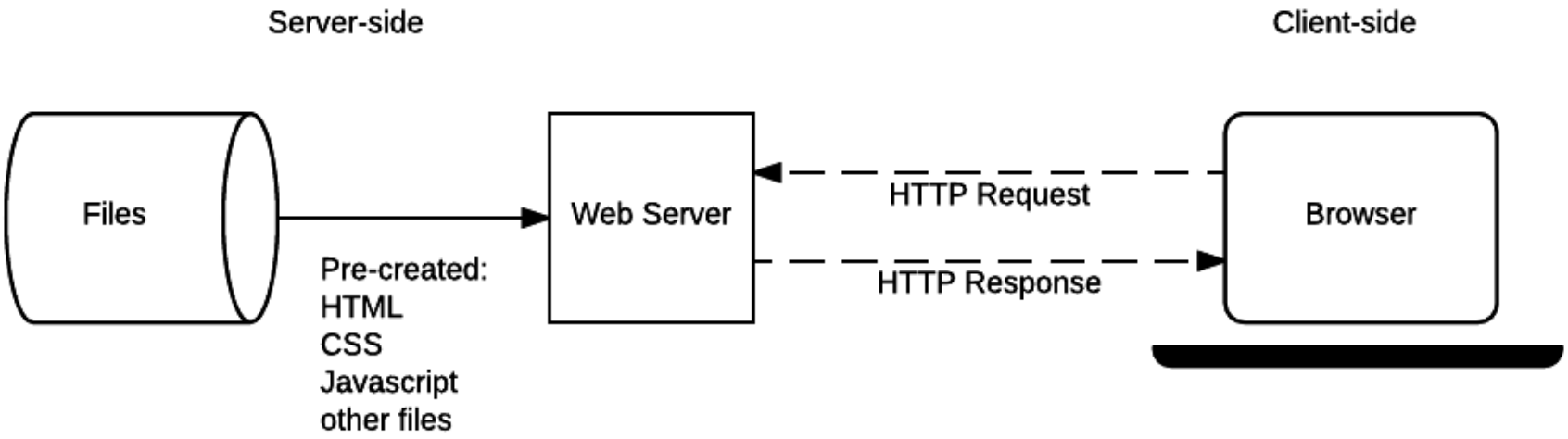
Servlets

Client Server Application

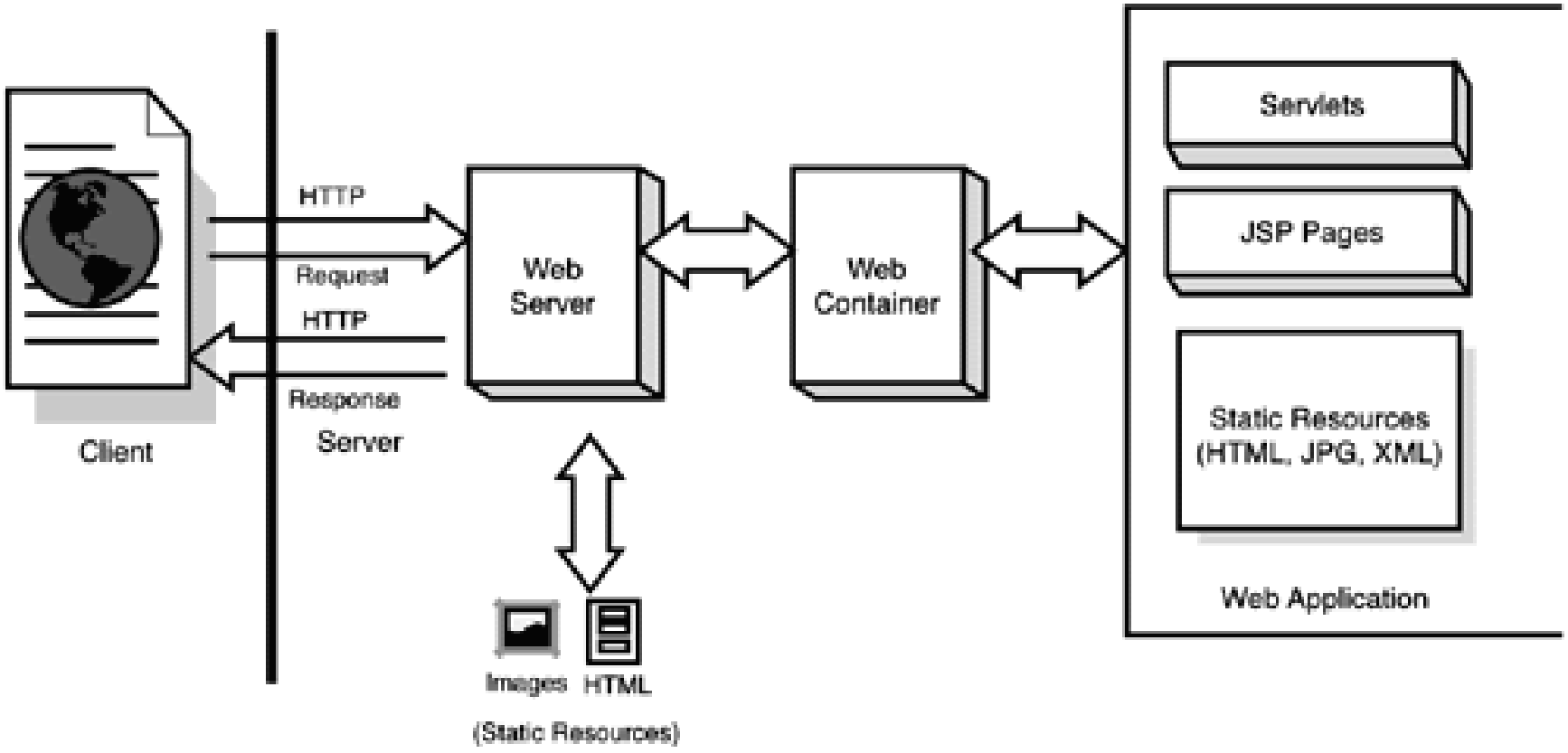


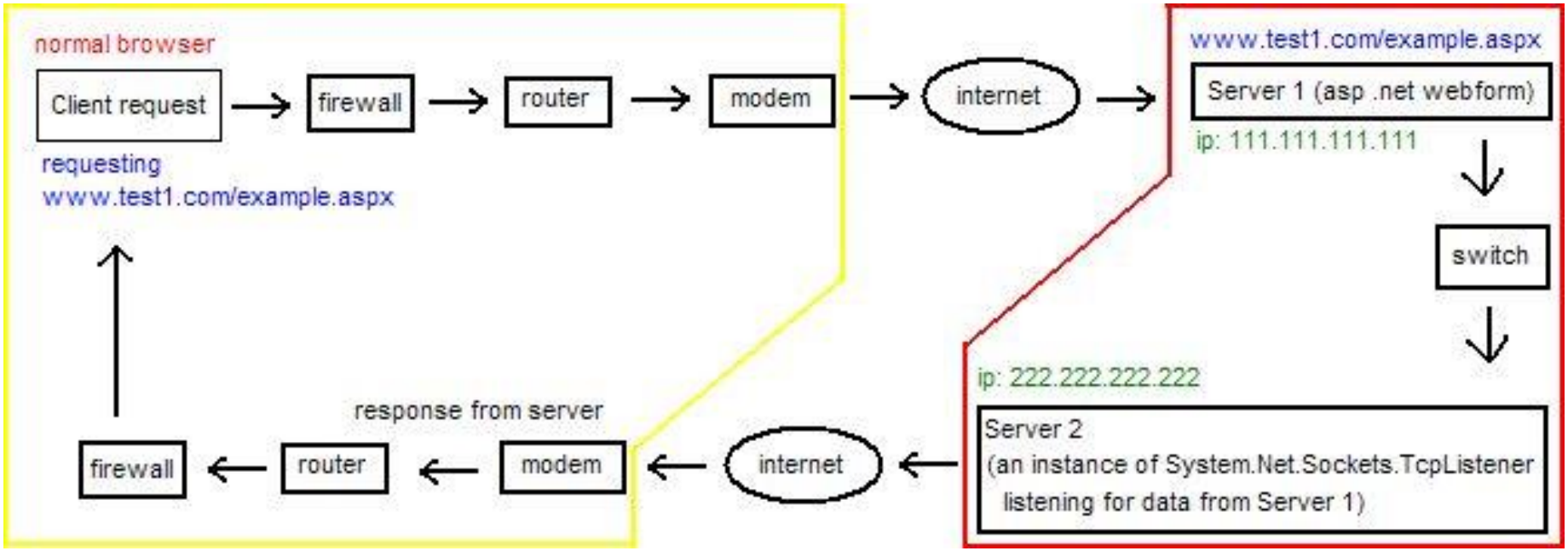
Request-Response Communication Model

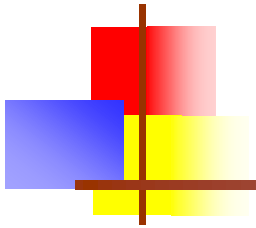
Client Server



Web Application

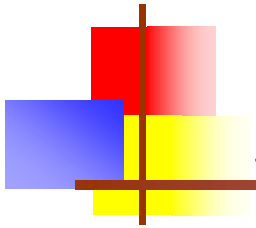



normal browser
Client request
firewall
router
modem
internet
www.test1.com/example.aspx
Server 1 (asp .net webform)
ip: 111.111.111.111
requesting
www.test1.com/example.aspx
switch
ip: 222.222.222.222
response from server
Server 2
(an instance of System.Net.Sockets.TcpListener
listening for data from Server 1)
firewall
router
modem
internet



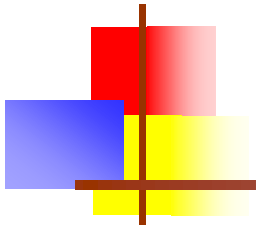
Servers

- A server is a computer that responds to requests from a client
  - Typical requests: provide a web page, upload or download a file, send email
- A server is also the software that responds to these requests; a client could be the browser or other software making these requests
- Typically, your little computer is the client, and someone else's big computer is the server
  - However, any computer can be a server
  - It is not unusual to have server software and client software running on the same computer



Apache

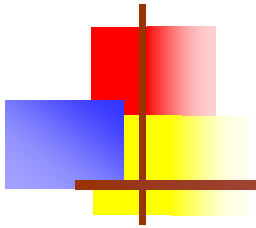
- Apache is a *very* popular server
  - 66% of the web sites on the Internet use Apache
- Apache is:
  - Full-featured and extensible
  - Efficient
  - Robust
  - Secure (at least, more secure than other servers)
  - Up to date with current standards
  - Open source
  - Free



Ports

- A port is a connection between a server and a client
  - Ports are identified by positive integers
  - A port is a software notion, not a hardware notion, so there may be very many of them
- A service is associated with a specific port
  - Typical port numbers:
    - 21—FTP, File Transfer Protocol
    - 22—SSH, Secure Shell
    - 25—SMTP, Simple Mail Transfer Protocol
    - 53—DNS, Domain Name Service
    - 80—**HTTP, Hypertext Transfer Protocol**
    - 8080—**HTTP (used for testing HTTP)**
    - 7648, 7649—CU-SeeMe
    - 27960—Quake III

} These are the ports of most interest to us



Ports II

- Our Web page is: http://www.ABC.org
- But it is *also:* http://www.ABC.org:80
- The http: at the beginning signifies a particular protocol (communication language), the Hypertext Transfer Protocol
- The :80 specifies a port
- By default, the Web server listens to port 80
  - The Web server could listen to any port it chose
  - This could lead to problems if the port was in use by some other server
  - **For testing servlets**, we typically have the server listen to port 8080
- In the second URL above, I explicitly sent my request to port 80
  - If I had sent it to some other port, say, 99, my request would either go unheard, or would (probably) not be understood

CGI Scripts

- CGI stands for “Common Gateway Interface”

Client sends a request to server

Server starts a CGI script

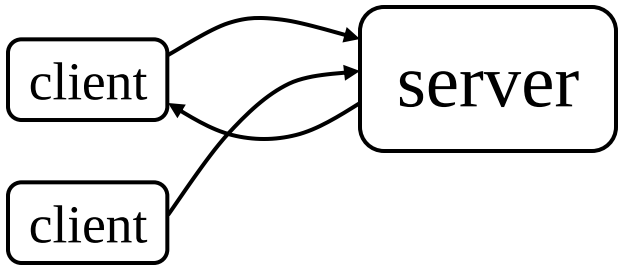
Script computes a result for server and quits

Server returns response to client

Another client sends a request

Server starts the CGI script again

Etc.

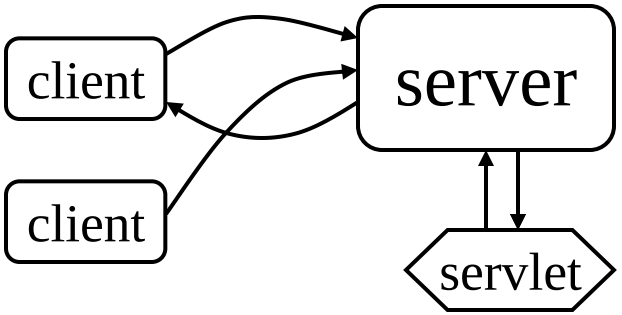


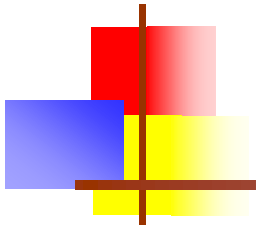
Servlets

- A servlet is like an applet, but on the server side

  - Client sends a request to server
  - Server starts a servlet
  - Servlet computes a result for server and *does not quit*
  - Server returns response to client
  - Another client sends a request
  - Server calls the servlet again

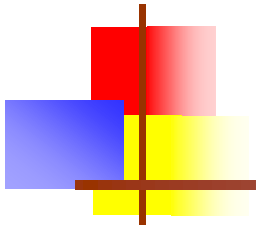
  Etc.





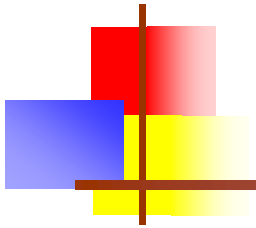
Servlets vs. CGI scripts

- Advantages:
  - Running a servlet doesn't require creating a separate process each time
  - A servlet stays in memory, so it doesn't have to be reloaded each time
  - There is only one instance handling multiple requests, not a separate instance for every request
  - Untrusted servlets can be run in a "sandbox"
- Disadvantage:
  - Less choice of languages (**CGI scripts can be in any language**)



Tomcat

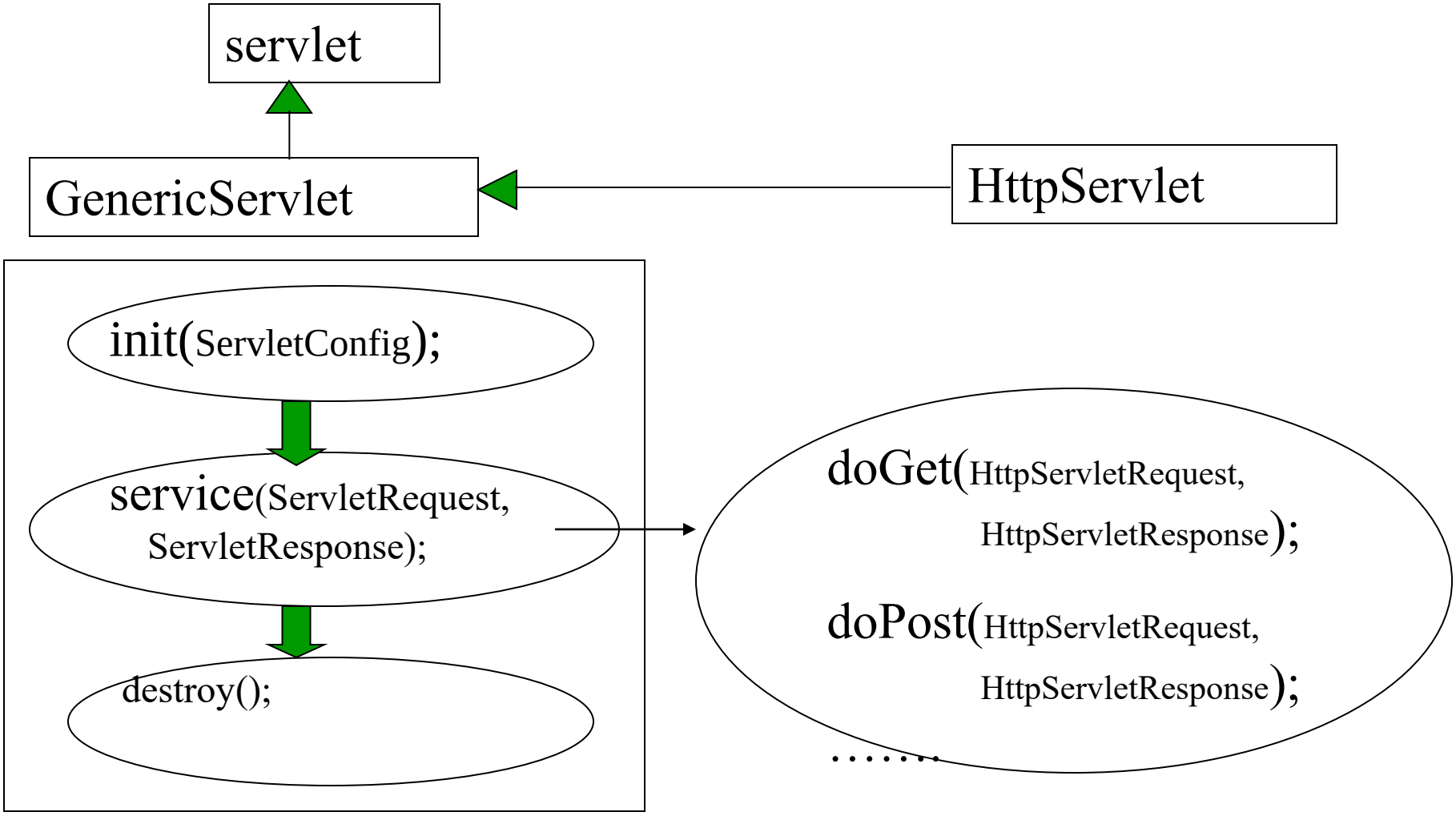
- Tomcat is the Servlet Engine than handles servlet requests for Apache
  - Tomcat is a “helper application” for Apache
  - It’s best to think of Tomcat as a “servlet container”
- Apache can handle many types of web services
  - Apache can be installed without Tomcat
  - Tomcat can be installed without Apache
- It’s easier to install Tomcat standalone than as part of Apache
  - **By itself, Tomcat can handle web pages, servlets, and JSP**
- Apache and Tomcat are open source (and therefore free)

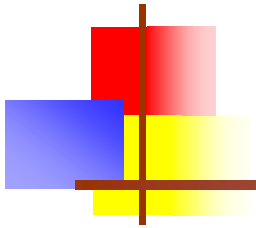


Servlets

- A servlet is any class that implements the javax.servlet.Servlet interface
  - In practice, most servlets extend the javax.servlet.http.HttpServlet class
  - Some servlets extend javax.servlet.GenericServlet instead
- Servlets, like applets, usually lack a main method, but must implement or override certain other methods

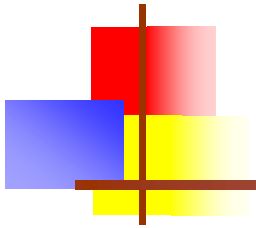
Life Cycle of Servlet





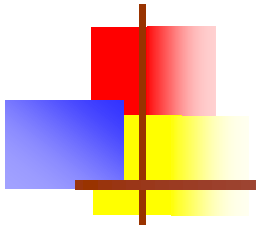
Important servlet methods, I

- When a servlet is first started up, its init(ServletConfig ***config***) method is called
  - init should perform any necessary initializations
  - init is called only once, and does not need to be thread-safe
- Every servlet request results in a call to service(ServletRequest ***request***, ServletResponse ***response***)
  - service calls another method depending on the type of service requested
  - Usually you would override the called methods of interest, not service itself
  - service handles multiple simultaneous requests, so it and the methods it calls *must be thread safe*
- When the servlet is shut down, destroy() is called
  - destroy is called only once, but must be thread safe (because other threads may still be running)



HTTP requests

- When a request is submitted from a Web page, it is almost always a GET or a POST request
- The HTTP <form> tag has an attribute action, whose value can be "get" or "post"
- The "get" action results in the form information being put after a ? in the URL
  - Example: http://www.google.com/search?hl=en&ie=UTF-8&oe=UTF-8&q=servlets
  - The & separates the various parameters
  - Only a limited amount of information can be sent this way
- "put" can send large amounts of information



Important servlet methods, II

- The service method dispatches the following kinds of requests: DELETE, GET, HEAD, OPTIONS, POST, PUT, and TRACE
  - A GET request is dispatched to the doGet(HttpServletRequest request, HttpServletResponse response) method
  - A POST request is dispatched to the doPost(HttpServletRequest request, HttpServletResponse response) method
  - These are the two methods you will usually override
  - doGet and doPost typically do the same thing, so usually you do the real work in one, and have the other just call it
  - public void doGet(HttpServletRequest request,
    HttpServletResponse response) {
    doPost(request, response);
    }

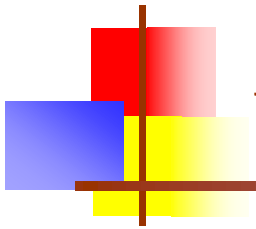
A “Hello World” servlet

(from the Tomcat installation documentation)

```
import javax.servlet.http.*;
import javax.servlet.*;
import java.io.*;
public class FirstServlet extends HttpServlet{
public void doGet(HttpServletRequest req,HttpServletResponse res)
throws ServletException,IOException
{
res.setContentType("text/html");//setting the content type
PrintWriter pw=res.getWriter();//get the stream to write the data

//writing html in the stream
pw.println("<html><body>");
pw.println("Welcome to servlet Hello World");
pw.println("</body></html>");

pw.close();//closing the stream
}}
}
```



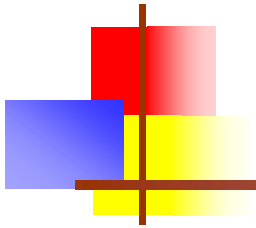
Web.xml: Deployment Descriptor

```
<web-app>

<servlet>
<servlet-name>FirstServlet l</servlet-name>
<servlet-class>FirstServlet </servlet-class>
</servlet>

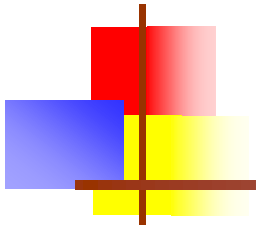
<servlet-mapping>
<servlet-name>FirstServlet </servlet-name>
<url-pattern>/welcome</url-pattern>
</servlet-mapping>

</web-app>
```



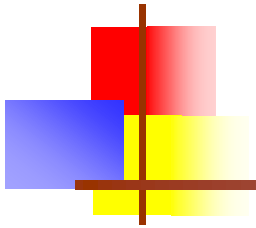
The superclass

- public class HelloServlet extends HttpServlet {
- Every class must extend GenericServlet or a subclass of GenericServlet
  - GenericServlet is "protocol independent," so you could write a servlet to process any protocol
  - In practice, you almost always want to respond to an HTTP request, so you extend HttpServlet
- A subclass of HttpServlet must override at least one method, usually one doGet, doPost, doPut, doDelete, init and destroy, or getServletInfo



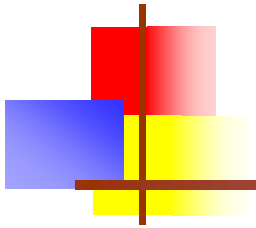
The doGet method

- public void doGet(HttpServletRequest ***request***,
  HttpServletResponse ***response***)
  throws ServletException, IOException {
- This method services a GET request
- The method uses request to get the information that was sent to it
- The method does not return a value; instead, it uses response to get an I/O stream, and *outputs* its response
- Since the method does I/O, it can throw an IOException
- Any other type of exception should be encapsulated as a ServletException
- The doPost method works *exactly* the same way



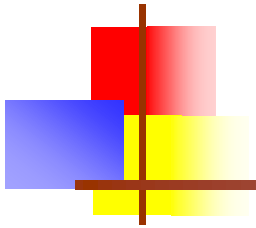
Parameters to doGet

- Input is from the HttpServletRequest parameter
  - Our first example doesn't get any input, so we'll discuss this a bit later
- Output is via the HttpServletResponse object, which we have named response
  - I/O in Java is very flexible but also quite complex, so this object acts as an "assistant"



Using the HttpServletResponse

- The second parameter to doGet (or doPost) is HttpServletResponse response
- Everything sent via the Web has a "MIME type"
- The first thing we *must* do with response is set the *MIME type* of our reply: response.setContentType("text/html");
  - This tells the client to interpret the page as HTML
- Because we will be outputting character data, we need a PrintWriter, handily provided for us by the getWriter method of response:
  PrintWriter out = response.getWriter();
- Now we're ready to create the actual page to be returned



Using the PrintWriter

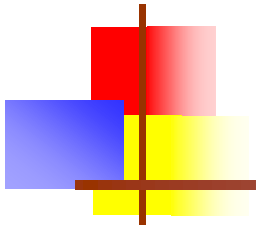
- From here on, it's just a matter of using our PrintWriter, named out, to produce the Web page
- First we create a header string:

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

  - This line is technically required by the HTML spec
  - Browsers mostly don't care, but HTML validators *do* care
- Then use the println method of out one or more times

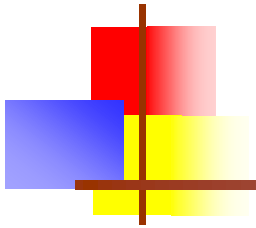
```
out.println(docType +
            "<HTML>\n" +
            "<HEAD> ... </BODY></HTML>");
```

- And we're done!



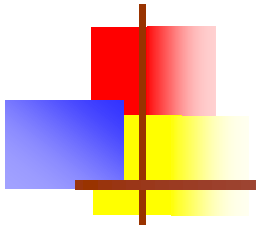
Input to a servlet

- A GET request supplies parameters in the form
  URL ? name=value & name=value & name=value
  - (Illegal spaces added to make it more legible)
  - Actual spaces in the parameter values are encoded by + signs
  - Other special characters are encoded in hex; for example, an ampersand is represented by %26
- Parameter names can occur more than once, with different values
- A POST request supplies parameters in the same syntax, only it is in the "body" section of the request and is therefore harder for the user to see



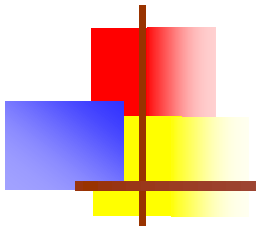
Getting the parameters

- Input parameters are retrieved via messages to the HttpServletRequest object request
  - Most of the interesting methods are inherited from the superinterface ServletRequest
- public Enumeration getParameterNames()
  - Returns an Enumeration of the parameter names
  - If no parameters, returns an empty Enumeration
- public String getParameter(String name)
  - Returns the value of the parameter name as a String
  - If the parameter doesn't exist, returns null
  - If name has multiple values, only the first is returned
- public String[] getParameterValues(name)
  - Returns an array of values of the parameter name
  - If the parameter doesn't exist, returns null



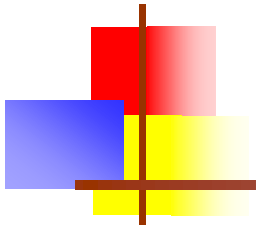
Enumeration review

- An Enumeration is almost the same as Iterator
  - It's an older class, and the names are longer
- Example use:
  - ```
    Enumeration e = myVector.elements();
    while (e.hasMoreElements()) {
        System.out.println(e.nextElement());
    }
    ```
    ```



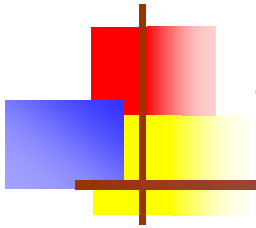
Example of input parameters

```
public void doGet(HttpServletRequest request,
                  HttpServletResponse response) {
 ... stuff omitted ...
    out.println("<H1>Hello");
      String names[] =
            request.getParameterValues("name");
      if (names != null)
        for (int i = 0; i < names.length; i++)
          out.println(" " + names[i]);
      out.println("!");
 }
```



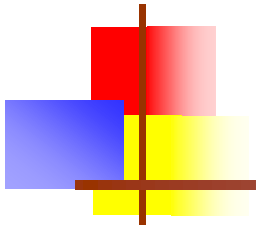
Java review: Data from Strings

- All parameter values are retrieved as Strings
- Frequently these Strings represent numbers, and you want the numeric value
  - int n = new Integer(***param***).intValue();
  - double d = new Double(***param***).doubleValue();
  - byte b = new Byte(***param***).byteValue();
    - Similarly for short, float, and long
    - These can all throw a NumberFormatException, which is a subclass of RuntimeException
  - boolean p = new Boolean(***param***).booleanValue();
- But:
  - char c = ***param***.charAt(0);



What's left?

- We've covered enough so far to write simple servlets, but not enough to write *useful* servlets
  - We still need to be able to:
    - Use configuration information
    - Authenticate users
    - Keep track of users during a session
    - Retain information across different sessions
    - Make sure our servlets are thread safe
    - Communicate between servlets
  - But remember: The most difficult program in any language is *Hello World!*



The End