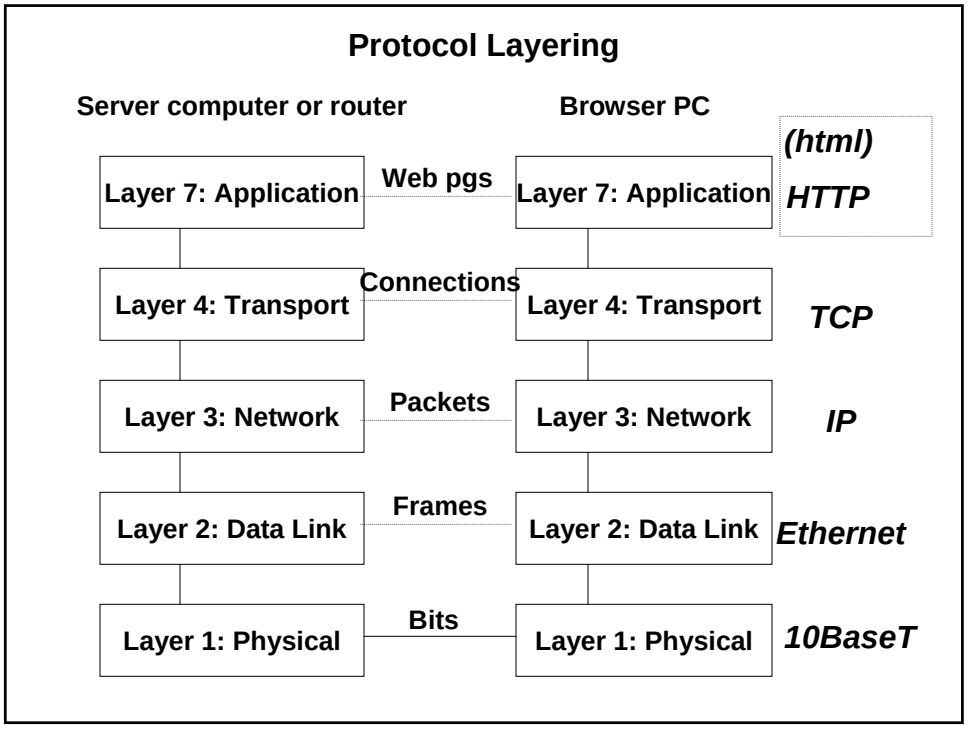
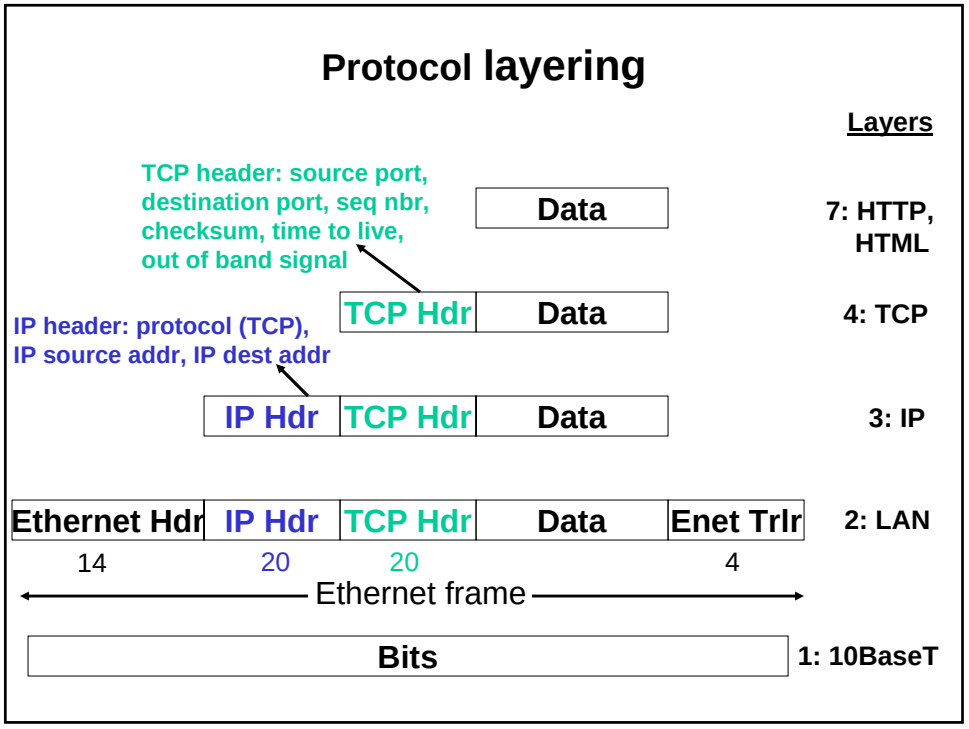
1.00 Lecture 37

Java and the Web

Internet and TCP/IP

- **Internet is "just" a set of loosely interconnected networks**
  - **A set of local area networks connected via wide area networks**
  - **Network segments interconnect via routers:**
    - **Dedicated computers that manage packets of data**
  - **TCP/IP is the universal data protocol on the network**
  - **Actual format, content is left to higher-level protocols, like the Web**
- **TCP/IP connections**
  - **Client is typically a data consumer that sends short requests**
    - **On Web, client is a browser**
  - **Server is typically a data provider that sends long responses**
    - **Listen for requests and transmit desired data, static or dynamic**
    - **On Web, servers talk a protocol called HTTP on port 80**
  - **TCP/IP connection is active only long enough to exchange data**
    - **Avoid overhead of many communication channels, but lose state**


Protocol Layering
Server computer or router
Browser PC
(html)
HTTP
Layer 7: Application
Web pgs
Layer 7: Application
Layer 4: Transport
Connections
Layer 4: Transport
TCP
Layer 3: Network
Packets
Layer 3: Network
IP
Layer 2: Data Link
Frames
Layer 2: Data Link
Ethernet
Layer 1: Physical
Bits
Layer 1: Physical
10BaseT


Protocol layering
Layers
TCP header: source port, destination port, seq nbr, checksum, time to live, out of band signal
Data
7: HTTP, HTML
IP header: protocol (TCP), IP source addr, IP dest addr
TCP Hdr
Data
4: TCP
IP Hdr
TCP Hdr
Data
3: IP
Ethernet Hdr
IP Hdr
TCP Hdr
Data
Enet Trlr
2: LAN
14
20
20
4
Ethernet frame
Bits
1: 10BaseT

Exercise: TCP/IP Connection

```
// Download and run TcpipTest. What is its output?
// At what layer (1, 2, 3, 4, or 7) is this program operating?
import java.net.*;
import java.io.*;

public class TcpipTest {
   public static void main(String[] args) {
     try {     // Socket is tcp/ip connection: ip address, port
        Socket s= new Socket("time-a.nist.gov", 13);
        InputStreamReader is= new
                        InputStreamReader(s.getInputStream());
        BufferedReader b= new BufferedReader(is); // Same as file!
        String currentLine = "";
        while ((currentLine = b.readLine()) != null)
          System.out.println(currentLine);
        b.close();
     } catch (IOException e) {
        System.out.println(e);
     }
   }
}  // Socket programs usually use threads, due to delays, losses…
```

Exercise 2: IP Addresses

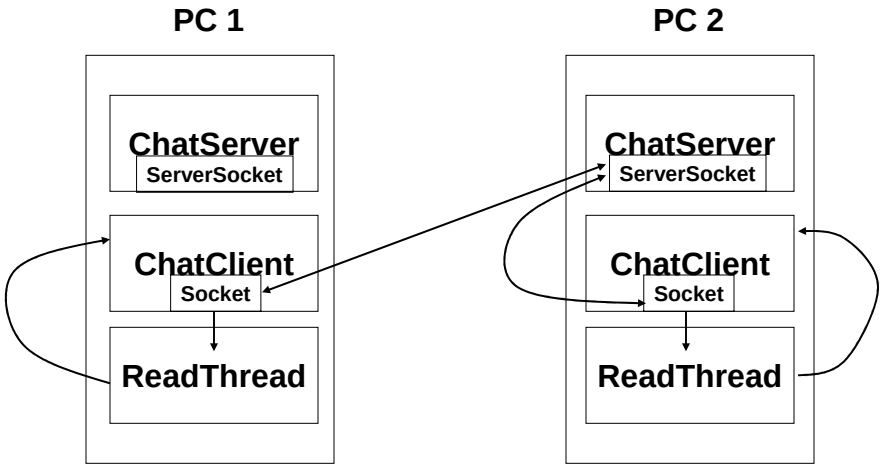
```
// Download and run AddressTest to see IP addresses
import java.net.*;
import javax.swing.*;

public class AddressTest {
   public static void main(String[] args) {
     try {
        InetAddress local= InetAddress.getLocalHost();
        System.out.println("Local host: "+ local);

        String input= JOptionPane.showInputDialog
                ("Enter host (e.g., web.mit.edu): ");
        InetAddress other= InetAddress.getByName(input);
        System.out.println("Other host: "+ other);

     } catch (Exception e) {
        System.out.println(e);
     }
   }
}
```

Exercise 3: Chat Program



Exercise 3: Chat Program

- **Download ChatClient, ChatServer and ReadThread**
  - **We'll read them together on the next several slides**
- **While we read we will:**
  - **Complete ChatServer (two short methods)**
  - **Complete ChatClient (one short method)**
  - **No changes in ReadThread**

ChatServer, p.1

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.net.*;
import java.io.*;

public class ChatServer extends JFrame {
   private JPanel serverPanel;     // Panel for UI elements
   private JLabel serverLabel;     // Label for server name
   private JTextField serverName;  // Input server to contact
   private JButton startButton;    // Button to request a session
   private JButton quitButton;     // Button to end application

   private static int CHATPORT = 5002;  // Port # for chat service
   private static int NCHATS = 16;      // max. # of requests

// Port 5002 is assigned to Radio Free Ethernet; we should really
// use an unassigned port!
```

ChatServer, p.2

```
 public ChatServer() {
     Container c = getContentPane();
     setBounds(200,200,500,200);

     serverPanel = new JPanel();
     serverLabel = new JLabel("Server = ");
     serverName = new JTextField("Enter server here", 40);
     startButton = new JButton("Request Connection");
     quitButton = new JButton("Quit");

     serverPanel.add(serverLabel);
     serverPanel.add(serverName);
     serverPanel.add(startButton);
     c.setLayout(new BorderLayout() );
     c.add("Center", serverPanel);
     c.add("North", quitButton);
```

ChatServer, p.3

```
startButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        String svr = serverName.getText();
        try {
        // Your code here:
        // Get the Internet address of svr (see AddressTest)
        // Create new Socket w/ address of server and chat port
        // (see TcpipTest)
        // Create new ChatClient, with Socket object as its arg
              }
        // Catch IOException: print "Unable to connect to "+svr
      }
    });
 quitButton.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        System.exit(0);
      }
   });
 setVisible(true);
 setDefaultCloseOperation(EXIT_ON_CLOSE);
 startListening();
 }    // End ChatServer constructor
```

Solution: ChatServer startButton

```
startButton.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
     String svr = serverName.getText();
     try {
        new ChatClient(new Socket(InetAddress.getByName(svr),
                       ChatServer.CHATPORT) );
     }
     catch(IOException e1) {
          System.out.println("Unable to connect to " + svr);
     }
   }
});
```

ChatServer, p.4

```
//  Create ServerSocket to listen for session requests
//  This is invoked as last line in ChatServer constructor
protected void startListening() {
   ServerSocket soc = null;
   try {
      soc =new ServerSocket(ChatServer.CHATPORT,ChatServer.NCHATS);
   } catch(IOException e) {
      System.exit(1);
   }
   // This loop waits for a connection request
   while(true) {
      try {
         Socket chatSocket = soc.accept();
         // Your code here:
         // Create new ChatClient with this socket as its argument
         // This client will be running on the server PC
      } catch(IOException e) {
         System.exit(1);
      }
   }
}
public static void main(String args[]) {
  ChatServer startFrame = new ChatServer(); }  }
```

Solution: ChatServer startListening

```
protected void startListening() {
      ServerSocket soc = null;
      try {
         soc = new ServerSocket(ChatServer1.CHATPORT,
                ChatServer1.NCHATS);
      } catch(IOException e) {
         System.exit(1);
      }
      // This loop waits for a connection request
      while(true) {
         try {
            Socket chatSocket = soc.accept();
            new ChatClient1(chatSocket);            // New
         } catch(IOException e) {
            System.exit(1);
         }
      }
   }
```

ChatClient, p.1

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import java.io.*;
import java.net.*;

public class ChatClient extends JFrame {
    private JFrame chFrame;
    private JTextArea sendArea, receiveArea;  // Send and receive text
    private JButton endButton;

    private Socket chatSocket;        // Session socket
    private final InetAddress addr;   // Chat server internet address
    private DataOutputStream output;  // Output stream for session
    private DataInputStream input;    // Input stream for session
    private ReadThread rThread;       // Thread to receive characters

    public final static char END_SESSION_CHAR = '\u0004';  // ctrl-D
    private static char NEWLINE = (char) 13;          // newline
    private static char CR = (char) 10;               // carriage return
    private static char BACKSPACE = (char) 8;         // backspace
    private static char TAB = '\t';                   // tab
```

ChatClient, p.2

```
public ChatClient(Socket cSoc) throws IOException {
       chatSocket = cSoc;
       addr= cSoc.getInetAddress();

       setTitle("Chat session  "+addr);
       Container c = getContentPane();
       setBounds(100,200,500,525);
       c.setLayout(null);

       sendArea = new JTextArea(30, 80);
       sendArea.setBounds(25,225,400,175);
       receiveArea = new JTextArea(30,80);
       receiveArea.setBounds(25,25,400,175);
       receiveArea.setEditable(false);
       endButton = new JButton("End Session");
       endButton.setBounds(150,425,150,50);

       c.add(sendArea);
       c.add(receiveArea);
       c.add(endButton);
       System.out.println("New chat session "+addr);
```

ChatClient, p.3

```
endButton.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent e) {
    try {
       output.writeChar(ChatClient.END_SESSION_CHAR);
    } catch(IOException ev) {
       System.out.println("IO error in chat session "+addr); }
    endSession();
   }
} );
sendArea.addKeyListener(new KeyAdapter() {
  public void keyTyped(KeyEvent e) {
    sendCharacter(e.getKeyChar() );
  }
} );

input = new DataInputStream(chatSocket.getInputStream() );
rThread = new ReadThread(this, input);  // Give input to thread to
rThread.start();                         // read and handle
output = new DataOutputStream(chatSocket.getOutputStream() );
c.validate();
setVisible(true);
setDefaultCloseOperation(EXIT_ON_CLOSE);  }    // End constructor
```

ChatClient, p.4

```
public void sendCharacter(char c) {
     try {
        if (isAllowedCharacter(c))
             output.writeChar(c);
             if(c == ChatClient.END_SESSION_CHAR)
                 endSession();
     } catch (IOException e) {
             System.out.println("IOException in chat "+addr);
     }
}
public void waitForMessage(char c) {
     if (c == ChatClient.END_SESSION_CHAR)
         endSession();
     else {
         final char ch= c;
         // Your code here, same as last lecture
         // Use SwingUtilities.invokeLater() method to insert an
         // event on the Swing event list. Its argument is
         // a new anonymous object that implements Runnable
         // The object's public void run() method will call the
         // updateReceivedMessage() method with char ch (next slide)
} }
```

Solution: ChatClient

```
public void waitForMessage(char c) {
      if(c == ChatClient1.END_SESSION_CHAR)
        endSession();
      else {
         final char ch= c;
         SwingUtilities.invokeLater(new Runnable() {
           public void run() {
               ChatClient1.this.updateReceivedMessage(ch);
           }
         } ) ;
      }
   }
```

ChatClient, p.5

```
public void updateReceivedMessage(char c) {
    if (c == ChatClient.NEWLINE  || c == ChatClient.CR)
        receiveArea.append("\n");
    else if (c == ChatClient.BACKSPACE) {
        String s = receiveArea.getText();
        if (s.length() > 0)
           receiveArea.setText(s.substring(0,s.length()-1) );
    } else {
        char[] newChar = new char[1];
        newChar[0] = c;
        receiveArea.append(new String(newChar));
    }
}
public void endSession() {
    try {
       output.close();
       input.close();
       chatSocket.close();
       System.out.println("Connection closed "+addr);
    }
    catch(IOException e) {
        System.out.println("Unable to close connection "+addr); }
    dispose(); }          // isAllowedCharacter() code omitted
```

ReadThread

```
import java.io.*;

public class ReadThread extends Thread {
    private ChatClient chat;      // session to forward characters to
    private DataInputStream input;    // stream to read from
    public ReadThread(ChatClient c, DataInputStream s) {
        super();
        input = s;
        chat = c;
    }
    public void run() {
        char c='A';   // initialize c != END_SESSION_CHAR
        while(c != ChatClient.END_SESSION_CHAR) {
           try {
              c = input.readChar();  // Blocks until character is read
              chat.waitForMessage(c);
           }
           catch(IOException e) {
              break;
           }
        }
    }
}
```

Exercise 3 Solutions: Chat Program

- We completed ChatServer:
  - In startButton's action listener:
    - Created a new Socket with the server IP address and the chat port number
    - Created a new ChatClient on that socket. This ChatClient will be on the client PC.
  - In method startListening():
    - Created a new ChatClient on the socket that the server hears. This ChatClient will be on the server PC.
- We completed ChatClient:
  - In method waitForMessage(), which is called by the thread that reads characters from the other ChatClient:
    - It gets a character, ch
    - It calls SwingUtilities.invokeLater(), which take an anonymous object that implements Runnable as its argument
    - The anonymous object's public void run() method calls the updateReceivedMessage() method with character c
    - Refer to an outerClass instance by outerClass.this
- No changes in ReadThread
  - It reads characters from the other client, which blocks
  - By having the reader in a separate thread, the entire client doesn't block

Exercise Conclusion

- **Run ChatServer**
  - **If you enter your own computer name as the server, you will get two ChatClients on your PC**
    - **You can happily type at yourself**
  - **If you enter someone else's computer name who is running ChatServer, you will get a ChatClient on your PC and he or she will get a Chat Client on his or her PC**

HTTP protocol

- **Four phases:**
  - **Open tcp/ip connection: Based on URL**
  - **Http request: Browser opens connection to server and sends:**
    - **Request method, (and request data at bottom if POST or PUT request)**
    - **URL,**
    - **HTTP version number**
    - **Header information (informational, optional), terminated with blank line**
  - **Http response: Server processes request and sends:**
    - **HTTP protocol version and status code**
    - **Header information, terminated by blank line**
    - **Text (data)**
  - **Close tcp/ip connection**

HTTP request examples

Typical browser request: telnet web.mit.edu 80, then type:

GET /about-mit.html HTTP/1.1

Host: web.mit.edu ***(required)***

Accept: text/html, text/plain, image/jpeg, image/gif, */* ***(optional)***

(blank line)

Typical server response:

HTTP/1.1 200 OK

Server: Apache/1.3.3 Ben-SSL/1.28 (Unix)

Content-Type: text/html

Content-Length: 8300

(blank line)

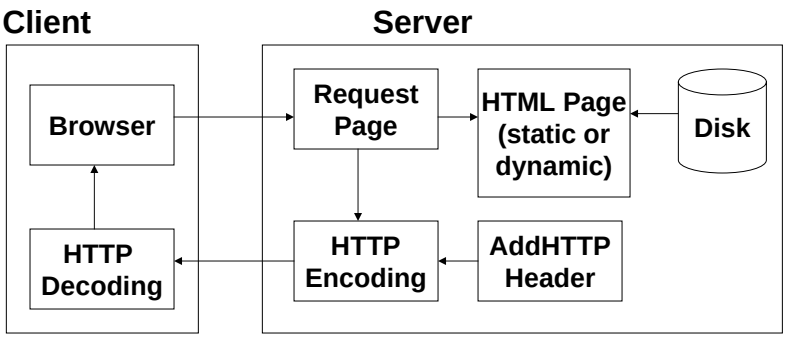
<HTML>

<HEAD><TITLE>About MIT</TITLE></HEAD>

<BODY>The mission of MIT…</BODY>

</HTML>

HTTP



- **These transactions are stateless. The connection is closed after each page and re-established: Server can't connect successive requests from the client**

Hypertext markup language (html) example

```
<HTML>
<HEAD>
<TITLE> Welcome to the Aircraft Parts System </TITLE>
</HEAD>
<BODY>
<H1> Welcome to the Aircraft Parts System </H1>
This system handles orders for aircraft and balloon parts.  We
    handle aircraft parts, and parts for all balloon manufacturers.
    We comply with the latest US regulations.
<P>
The use of this system is subject to <A HREF= MITRule.html>
    MIT rules and regulations. </A>
</BODY>
</HTML>
```

(Aircraft1.html)

HTML

- Tags (e.g. <TAG>) never display but direct the browser
- Often in pairs (e.g. <TAG> and </TAG>) to delimit section
- Some tags have attributes (e.g. <A HREF=abc.html> </A>)
- HTML document begins with <HTML>, ends with </HTML>
  - Two sections within document: HEAD and BODY
  - Head has identifying information not displayed
  - Body is displayed, with formatting:
    - Paragraph <P>
    - Header levels 1 through 6 <H1> through <H6>
    - Line break or carriage return
    - Anchor <A>, placed around text or graphics; used for hyperlinks

HTTP and HTML

- **HTTP**
  - **Is only direct form of interaction between browser and server**
  - **Was an extremely perceptive extension of email, ftp protocols by Tim Berners-Lee to enable Web browsers**
  - **Request-response paradigm**
  - **Connection made for each request/response pair**
  - **Most popular protocol on Internet**
- **HTML**
  - **Text description language, based on tags**
  - **High level description rather than specific formatting**
  - **HTML is static, which is not sufficient for Web applications**
  - **Many extensions and changes (scripting, Java) for dynamic content**

HTML forms

- **Used as front ends to server programs**
  - **Java Server Pages, servlets, .NET framework (Microsoft)**
- **Forms are user interface components to collect data from user and transmit it to the server application program**
  - **Forms are placed on Web pages that can also have other elements**
  - **Java applets can be used but rarely are**
  - **All of these run on the browser and are user interface components**

How HTML forms transmit data

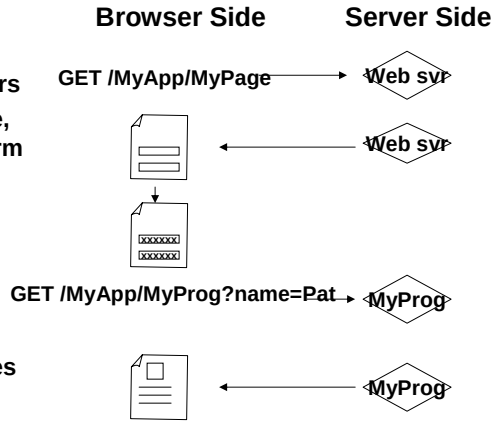
- Forms allow a series of components to be placed on the page
  - Each component has a name and a value
  - Entire form is associated with the URL of a server side program that will process the input data
  - Form data is sent when user presses 'Submit' button (component)
  - Data is sent to URL as string of form:
    - `Name1=Value1&Name2=Value2&…NameN=ValueN`
  - If data is sent with HTTP GET command, it is appended to end of GET string after a ?:
    - `GET /Index.html?Name1=Value1…`
  - If data is sent with HTTP PUT command, it is sent after the blank line as a string
  - Server programs (Java Server Pages, servlet) have methods to extract the data from the string and use it in the program

HTML 4.0 tags for forms

Tag	Type	Definition
<FORM>		Start a form
<INPUT TYPE=	text	Single line of text entry
	password	Single line password entry
	file	File to upload, with 'Browse' button
	checkbox	Checkbox
	radio	Radio button (option box)
	image	Image acting as button
	hidden	Track user, store predefined input, etc.
	submit	Submit button for form
	reset	Button to restore default values
<SELECT>		List box or combo box
<OPTION>		Item in scrolling list or popup menu
<TEXTAREA>		Start multiple-line text entry field

Browser-server interaction

- Step 1: Browser requests page without any parameters
- Step 2: Server fetches page, and returns empty input form
- Step 3: User fills out input form and presses 'submit' button
- Step 4: Browser packages form into query string and sends to server
- Step 5: Program synthesizes a response document and returns it



Java on servers

- Java servlets. Implement the following steps:
  - Read data sent by browser
    - Usually entered by user, but could be from applet or JavaScript
  - Look up other information from the HTTP request
    - Browser capabilities, cookies, host, etc. from HTTP headers
  - Generate results
    - Access database, execute program (possibly via CORBA/COM), etc.
  - Format the results as an HTML or other document
    - Servlets can generate GIF, gzip, many MIME types
  - Set HTTP response parameters in headers
  - Send document back to browser
- Java Virtual Machine (JVM) runs the servlets on the server
- These evolved from applets (limited use) into servlets (extremely useful, secure, standard)

Servlet example

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

public class Hello100 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 1.00</TITLE></HEAD>\n" +
                "<BODY>\n" +
                "<H1>Hello 1.00</H1>\n" +
                "</BODY></HTML>");}}
```

Java Server Pages (JSP)

- Allow us to mix static HTML with dynamically generated content from servlets
  - Many Web pages are primarily static
  - Changeable parts are in only a few locations on the page
- A servlet requires us to program the entire page
- JSP allows the Web designer to write static HTML and leave stubs for dynamic content
  - Java programmers can then write the stubs
- ASP.NET is very similar (Microsoft)

JSP example

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML> <HEAD>
<TITLE>JSP Example</TITLE>
<LINK REL=STYLESHEET
      HREF="JSP-Styles.css"
      TYPE="text/css">
</HEAD>
<BODY>
<H2>JSP Expressions</H2>
<UL>
  <LI>Current time: <%= new java.util.Date() %>
  <LI>Your hostname: <%= request.getRemoteHost() %>
  <LI>Your session ID: <%= session.getId() %>
  <LI>The <CODE>testParam</CODE> form parameter:
      <%= request.getParameter("testParam") %>
</UL>
</BODY>
</HTML>
```

JSP Example 2

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD> <TITLE>Color Test</TITLE> </HEAD>
<%
  String bgColor = request.getParameter("bgColor");
  boolean hasColor;
  if (bgColor != null) {
      hasColor = true; }
    else {
      hasColor = false;
      bgColor = "WHITE";}
%>
<BODY BGCOLOR="<%= bgColor %>">
<H2 ALIGN="CENTER">Color Testing</H2>
<%
if (hasColor) {
  out.println(bgColor + " used"); }
else {
  out.println("Default color (WHITE) used";
}
%>
</BODY>
</HTML>
```