

World Wide Web

- The World Wide Web (WWW) is an architectural framework for accessing linked documents over the Internet.
- It started in 1989 at CERN, the European center for nuclear research, by Tim Bernes-Lee.
- Every WWW page is identified by its Uniform Resource Locator (URL). A URL has three parts: the protocol, the DNS name of the machine on which the page is located, and a local name uniquely identifying the filename of a web page. An optional component of a URL is the port number that the server uses to listen for incoming requests.
- Web pages are written using a standard language called Hypertext Markup Language (HTML) to allow all Web browsers to decipher and to properly display the web page contents.
- The following steps outline the process of viewing a web page on the client side:
 - The user on the client clicks on a hyperlink or types in a URL address.
 - The browser checks with the DNS for the IP address of the hostname.
 - DNS returns the IP address; the browser establishes a connection to port 80 on that IP address.
 - The browser sends a request asking for the file corresponding to the web page.
 - The server answers by sending the file requested by the client browser.
 - The browser displays the contents of the file that was received.

HyperText Transport Protocol (HTTP)

- HTTP is the transfer protocol used throughout the World Wide Web.
- It specifies the messages that are exchanged by clients and servers over the Internet and uses a TCP connection on default port 80 of the server.
- HTTP is defined in RFC2616.
- Neither the browser nor the server has to worry about the status of messages—this is handled by the TCP protocol.
- In the earlier version of the protocol, a typical session goes as follows: a connection is established, a single request is sent, a single response is received, and finally, the connection is released. With the new version,

persistent connection is established enabling multiple request and response activities.

- Each request is introduced by an operation called a method. A method can be one of the following:
 - GET request to read a web page
 - HEAD request to read a web page header
 - PUT request to store a web page
 - POST append to a named resource.
 - DELETE remove the web page
 - TRACE echo the incoming request
 - CONNECT reserved for future use
 - OPTIONS query certain options
- Every request gets a response consisting of a status line and some additional information. The status line contains a three-digit code indicating the status of the request. Here is a summary of the codes:
 - 1xx Information
 - 2xx Success
 - 3xx Redirection
 - 4xx Client Error (e.g. 404: file not found)
 - 5xx Server Error (e.g. 501: server not available)
- The request line may be followed by additional lines called the request headers. These act like parameters in a procedure or function call. Likewise, responses have also their own response headers. Some request headers are:
 - User-agent: the information about the browser and the platform;
 - Host: the server's DNS name;
 - Date: the date and time the request/response was made;
 - Content-encoding: the encoding type;
 - Content-length: the page's length in bytes;
 - Last-modified: the time and date the page was last modified; and
 - Set-cookie: the server wants the client to save a cookie.
- To test your Web server, you may use the following commands:

```
telnet someWebserver.com 80
GET /index.html HTTP/1.1
Host: someWebserver.com

close
```

```

import java.net.*;
import java.io.*;
import java.util.*;
/*****
 * This is an implementation of an HTTP server
 * using TCP/IP.
 *
 *****/
public class WebServer {
    private static String doc_Root; // the webserver root directory
    private static int portNum=8081;// port number for web service
    private static ServerSocket sSocket; //the web server socket

    public static void main(String [] args) throws Exception{
        InputStreamReader in = new InputStreamReader(System.in);
        BufferedReader bufR = new BufferedReader(in);
        System.out.println("Please provide the server parameters...");
        System.out.print("Enter the Web document root: ");
        doc_Root = bufR.readLine();
        System.out.print("Enter the port number for the HTTP service: ");
        portNum = Integer.parseInt(bufR.readLine());

        //create a TCP ServerSocket using the given port number
        sSocket = new ServerSocket(portNum);
        System.out.print("Web Server is up!...");
        System.out.println("Check it out using a browser!");
        while (true){
            Socket newSocket = sSocket.accept();
            new appHandler(newSocket, doc_Root).start();
        }//end of while loop

    }//end of main
}

} //end of WebServer class definition

```

```

/*****
* The application handler class
*
*****/
import java.io.*;
import java.net.*;
import java.util.*;

public class appHandler extends Thread {
    final static int BUFSIZE = 8192;
    Socket socket;
    PrintWriter pw;
    BufferedOutputStream bufOS;
    BufferedReader bufR;
    File doc_Root;

    //the constructor method
    public appHandler(Socket inSocket, String inDocRoot) {
        socket = inSocket;
        try {
            //determine the absolute path
            doc_Root = new File(inDocRoot).getCanonicalFile();
        }catch (IOException e){
            System.out.println("ERROR: exception " + e);
        }
    }
    //end of constructor

    public void run(){

        try {
            // create a buffered reader and writer
            bufR = new BufferedReader(new
                InputStreamReader(socket.getInputStream()));
            bufOS = new BufferedOutputStream(socket.getOutputStream());
            pw = new PrintWriter(new OutputStreamWriter(bufOS));

            //read the HTTP request
            String line = bufR.readLine();
            //accept no more input
            socket.shutdownInput();

            if (line.toUpperCase().startsWith("GET")){
                StringTokenizer strTok = new
                    StringTokenizer(line, " ?");
                strTok.nextToken();
                String selectItem = strTok.nextToken();
                String name;
                if (selectItem.startsWith("\\\\")
                    name = doc_Root + selectItem;
                else
                    name = doc_Root + "\\\" + selectItem;
                //get the absolute file path
                File file = new File(name).getCanonicalFile();
                if (!file.exists()){
                    pw.println("HTTP Error: 404 File not found!");
                    pw.println();
                }
                else if (!file.canRead()){

```

```

        pw.println("HTTP Error: 403 Can not read file!");
        pw.println();
    }
    else if (file.isDirectory())
        listDir(file, selectItem);
    else
        readFile(file.getAbsolutePath());
} //end of if startsWith(GET)
else {
    pw.println("HTTP Error 501: Not Available");
    pw.println();
}
pw.flush();
bufOS.flush();
socket.close();
} catch (Exception e) {
    System.out.println("ERROR: exception " + e);
}
} //end of the run method

public void readFile(String fileName) throws Exception {

    BufferedInputStream inBuf = new BufferedInputStream(
        new FileInputStream(fileName));
    byte[] data = new byte[BUFSIZE];
    int read = inBuf.read(data);

    while (read != -1) {
        bufOS.write(data, 0, read);
        read = inBuf.read(data);
    }
    bufOS.flush();
} //end of readFile method

public void listDir(File dir, String folderName) throws Exception {
    pw.print("<HTML><HEAD><TITLE>FOLDER: " + folderName);
    pw.println("</TITLE></HEAD><BODY>");
    pw.println("DIRECTORY LISTING:<BR><P>");
    File [] dirContents = dir.listFiles();
    for (int i=0; i < dirContents.length; i++){
        pw.print("<A HREF=\" " + folderName +
            dirContents[i].getName());
        if (dirContents[i].isDirectory())
            pw.print("/");
        pw.print(">");

        pw.print(dirContents[i].getName());
        if (dirContents[i].isDirectory())
            pw.print("/");
        pw.println("</A><BR>");
    } //end of for loop
    pw.println("</BODY></HTML>");
    pw.flush();
} //end of listDir method

} //end of appHandler class definition

```