Solutions for Assignment Unit 7

```
package webserver;

import java.io.*;
import java.net.*;
import java.util.*;

/**
 * The main() program in this class is designed to read requests from
 * a Web browser and display the requests on standard output.  The
 * program sets up a listener on port 50505.  It can be contacted
 * by a Web browser running on the same machine using a URL of the
 * form  http://localhost:505050/path/to/resource.html  This method
 * does not return any data to the web browser.  It simply reads the
 * request, writes it to standard output and then closes the connection.
 * The program continues to run, and the server continues to listen
 * for new connections, until the program is terminated (by clicking the
 * red "stop" square in Eclipse or by Control-C on the command line).
 */
public class ReadRequest {

 /**
  * The server listens on this port.  Note that the port number must
  * be greater than 1024 and lest than 65535.
  */
 private final static int LISTENING_PORT = 50505;
       protected static Socket client;
       protected static DataInputStream in;
       protected static PrintStream out;
       static String requestedFile;


 /**
  * Main program opens a server socket and listens for connection
  * requests.  It calls the handleConnection() method to respond
  * to connection requests.  The program runs in an infinite loop,
  * unless an error occurs.
  * @param args ignored
  */
 public static void main(String[] args) {
  ServerSocket serverSocket;
  try {
   serverSocket = new ServerSocket(LISTENING_PORT);
  }
  catch (Exception e) {
   System.out.println("Failed to create listening socket.");
   return;
  }
  System.out.println("Listening on port " + LISTENING_PORT);
  try {
   while (true) {
    Socket connection = serverSocket.accept();
    System.out.println("\nConnection from "
      + connection.getRemoteSocketAddress());
                        ConnectionThread thread = new ConnectionThread(connection);
                        thread.start();
```

```
  }
 }
 catch (Exception e) {
  System.out.println("Server socket shut down unexpectedly!");
  System.out.println("Error: " + e);
  System.out.println("Exiting.");
 }
}

/**
 * Handle commuincation with one client connection.  This method reads
 * lines of text from the client and prints them to standard output.
 * It continues to read until the client closes the connection or
 * until an error occurs or until a blank line is read.  In a connection
 * from a Web browser, the first blank line marks the end of the request.
 * This method can run indefinitely,  waiting for the client to send a
 * blank line.
 * NOTE:  This method does not throw any exceptions.  Exceptions are
 * caught and handled in the method, so that they will not shut down
 * the server.
 * @param connection the connected socket that will be used to
 *    communicate with the client.
 */
private static void handleConnection(Socket connection) {

            String httpRootDir = "/Documents and Settings/";
 client = connection;
            try {
  //create input and output streams for conversation with client
                    in = new DataInputStream(client.getInputStream());
  out = new PrintStream (client.getOutputStream());

                    String line = null;     //read buffer
                    String req = null;      //first line of request

                        //read HTTP request -- the request comes in
                        //on the first line, and is of the form:
                        //      GET <filename> HTTP/1.x

                        req = in.readLine();

                        //loop through and discard rest of request
                        line = req;
                        while (line.length() > 0)
                        {
                        line = in.readLine();
                        }

                        // Create a token of the input line
                         StringTokenizer st = new StringTokenizer(req);

                         // Check if HTTP request it "GET"
                        if (!st.nextToken().equals("GET"))
                            {
                           // If not send HTTP Error 501
                            sendErrorResponse(501);
                            return;
                            }

                        //parse request -- get filename
                        requestedFile = st.nextToken();

                        //create File object
                        File f = new File(httpRootDir + requestedFile);
                        //check to see if file exists
                        if (!f.canRead())
```

```
                        {
                       // If not send HTTP Error 404
                        sendErrorResponse(404);
                        return;
                        }

                    //Send  HTTP response
                    sendResponseHeader(getMimeType(requestedFile),(int) f.length());
                    //Send File to the client
                    sendFile(f,client.getOutputStream());

  }
  catch (Exception e) {
   System.out.println("Error while communicating with client: " + e);
  }
  finally {  // make SURE connection is closed before returning!
   try {
    connection.close();
   }
   catch (Exception e) {
   }
   System.out.println("Connection closed.");
  }
 }

      //send an HTTP 200 OK header to the client
      //The first line is a status message from the server to the client.
      //The second line holds the mime type of the document
      //The third line holds the Content-Length of the document.
      //The fourth line holds the connection type.

      private static void sendResponseHeader(String type,int length)
      {
            out.println("HTTP/1.1 200 OK");
            out.println("Content-type: " +type);
            out.println("Content-Length: " +length);
            out.println("Connection: close " + "\r\n");
      }


      //Send a HTTP ERROR header to the client
      //The first line is a status message from the server to the client.
      //The second line holds  the connection type.
      //The third line holds the mime type of the document
      //The forth line holds more information of the error.
      private static void sendErrorResponse(int errorCode)
      {
            switch(errorCode) {
               case 404:
                  out.print("HTTP/1.1 404 Not Found");
                  out.println("Connection: close " );
                  out.println("Content-type: text/plain" +"\r\n");
                  out.println("<html><head><title>Error</title></head><body> <h2>Error: 404 Not Found</h2> <p>The resource that you
requested does not exist on this server.</p> </body></html>");
                  break;
               case 501:
                  out.print("HTTP/1.1 501 Not Implemented");
                  out.println("Connection: close " );
                  out.println("Content-type: text/plain" +"\r\n");
                  break;
            }
      }


      // Extract the MimeType of the file.
      private static String getMimeType(String fileName) {
```

```
            int pos = fileName.lastIndexOf('.');
            if (pos < 0)  // no file extension in name
                return "x-application/x-unknown";
            String ext = fileName.substring(pos+1).toLowerCase();
            if (ext.equals("txt")) return "text/plain";
            else if (ext.equals("html")) return "text/html";
            else if (ext.equals("htm")) return "text/html";
            else if (ext.equals("css")) return "text/css";
            else if (ext.equals("js")) return "text/javascript";
            else if (ext.equals("java")) return "text/x-java";
            else if (ext.equals("jpeg")) return "image/jpeg";
            else if (ext.equals("jpg")) return "image/jpeg";
            else if (ext.equals("png")) return "image/png";
            else if (ext.equals("gif")) return "image/gif";
            else if (ext.equals("ico")) return "image/x-icon";
            else if (ext.equals("class")) return "application/java-vm";
            else if (ext.equals("jar")) return "application/java-archive";
            else if (ext.equals("zip")) return "application/zip";
            else if (ext.equals("xml")) return "application/xml";
            else if (ext.equals("xhtml")) return"application/xhtml+xml";
            else return "x-application/x-unknown";
              // Note:  x-application/x-unknown  is something made up;
              // it will probably make the browser offer to save the file.
          }

          // Sending the file to the output Socket.
          private static void sendFile(File file, OutputStream socketOut) throws IOException {
            InputStream infile = new BufferedInputStream(new FileInputStream(file));
            OutputStream outfile = new BufferedOutputStream(socketOut);
            while (true) {
              int x = infile.read(); // read one byte from file
              if (x < 0)
                break; // end of file reached
              outfile.write(x);  // write the byte to the socket
            }
            outfile.flush();
          }

          // Create a Thread.
          private static class ConnectionThread extends Thread {
            Socket connection;
            ConnectionThread(Socket connection) {
              this.connection = connection;
            }
            public void run() {
              handleConnection(connection);
            }
          }

}
```

Last modified: Wednesday, 13 May 2020, 4:19 PM