```
package com.msoe.ce4960.lab6;

import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

/**
 * Acts as a basic webserver
 *
 * Experiences:
 * This lab was extremely straightforward.  Thanks to the discussions and topics
 * reviewed in class, parsing the headers and data was a fairly easy task.  Once
 * there all it required was writing output.
 *
 * Design:
 * The server starts listening on port 8888.  When a client connects, it spins
 * off another thread to handle the request.  This thread parses the request.
 * If the request is a GET request, the file is fetched, if it is available.  If
 * it is not available, a 404 error is returned.  If it is a POST request, the
 * thread processes the post data and echos fields that it expected back to the
 * user.
 *
 * Deficiencies:
 * The main deficiency in this program is its lack of flexibility.  Right now, it
 * can only serve raw files and parse 1 type of POST request.  Ideally, the server
 * would be able to look at where the file was POSTed to and react differently.
 * That would be a relatively simple task, but is out of the scope for this lab.
 *
 * Ideas for Improvement:
 * Make this a more defined lab.  Maybe have it act more like a webserver where
 * multiple forms have to be processed.  I would also add a requirement to serve
 * images.
 *
 * Usage:
 * Start the server and go to {@link http://localhost:8888}.  Fill out the form
 * (which was retrieved via a GET request) and submit it.  The POST data will
 * be processed and your data will be echoed back to you.
 *
 * Note that this has only been tested with Firefox 12
 *
 * @author Erik Sommer
 */

/**
 * Main class.  Starts the webserver
 * @author Erik Sommer
 *
 */
public class Webserver {

    /**
     * The port the server should listen on
     */
```

```
    private static final int SERVER_PORT = 8888;

    /**
     * Main method.  Starts the webserver
     * @param args  ignored
     */
    public static void main(String[] args){

        // Socket to listen for connections on
        ServerSocket socket;
        try {

            // Create the socket to listen for connections on
            socket = new ServerSocket(SERVER_PORT);

            // Listen for connections
            while(true){

                // Accept a client connection
                Socket clientSocket = socket.accept();

                // Spin off a new thread;
                new Thread(new ClientHandler(clientSocket)).start();
            }

        } catch (IOException e) {
            System.err.println("An I/O exception occured");
            e.printStackTrace();
        }
    }
}
```

```java
package com.msoe.ce4960.lab6;

import java.io.BufferedReader;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.net.Socket;
import java.util.HashMap;
import java.util.Map;

import com.msoe.ce4960.lab6.HTTPRequest.HTTPRequestType;

/**
 * Handles the client requests that are sent to the server
 * @author Erik Sommer
 *
 */
public class ClientHandler implements Runnable {

    /**
     * Size of the buffer to use when writing out the file
     */
    private static final int BUFFER_SIZE = 1024;

    /**
     * Socket that the client is connected from
     */
    Socket mSocket;

    /**
     * Constructor
     * @param socket    the socket that the client is connected form
     * @throws IllegalArgumentException if {@code socket} is {@code null}
     */
    public ClientHandler(Socket socket){

        // Parameter validation
        if(socket == null){
            throw new IllegalArgumentException("socket is null");
        }

        mSocket = socket;
    }

    /**
     * Displays the POST output in a human-readable form
     * @param map   the map that contains the post data
     * @param os    the output stream to write the data to
     * @throws IOException  if there was an error getting the output stream
     */
```

```
    private void displayOutput(Map<String, String> map, DataOutputStream os)
            throws IOException {

        // Build the response header
        StringBuilder responseBuilder = new StringBuilder();
        responseBuilder.append("HTTP/1.1 200 OK\n");
        responseBuilder.append("Content-Type: text/html; charset=utf-8\n");

        // Build the response body
        StringBuilder bodyBuilder = new StringBuilder();
        bodyBuilder.append("<p><strong>Text Input</strong>: "
                + map.get("textarea"));
        bodyBuilder.append("<p><strong>Radio Button Choice</strong>: "
                + map.get("radio-choice-1"));
        bodyBuilder.append("<p><strong>Dropdown Choice</strong>: "
                + map.get("select-choic"));
        bodyBuilder.append("<p><strong>Text Area</strong>: "
                + map.get("textarea"));
        bodyBuilder.append("<p><strong>Value is checked</strong>: "
                + (map.containsKey("checkbox") ? (map.get("checkbox").equals("on") ? "Yes" :
                "No") : "No"));

        // Set the response header content-length
        responseBuilder.append("Content-Length: " + bodyBuilder.toString().getBytes().length +
        "\n\n");

        // Write the header followed by the content
        os.write(responseBuilder.toString().getBytes());
        os.write(bodyBuilder.toString().getBytes());
    }

    /**
     * Processes a GET request for a file
     * @param request   the request
     * @param os        output stream to write data to
     * @throws IOException  if there was an error reading or witing to the streams
     */
    private void processGET(HTTPRequest request, OutputStream os) throws IOException {

        // Log that the server is processing a GET request
        System.out.println("Sending get");

        StringBuilder responseBuilder = new StringBuilder();

        File requestFile = request.getFile();

        if(!requestFile.exists()){
            // Send a 404 header if the file is not found
            responseBuilder.append("HTTP/1.1 404 FILE NOT FOUND\n");
            responseBuilder.append("Content-Type: text/html; charset=utf-8\n");
            responseBuilder.append("Content-Length: 0\n\n");
        }else{
            // Send the 200 header if the file is found
```

```
            responseBuilder.append("HTTP/1.1 200 OK\n");
            responseBuilder.append("Content-Type: text/html; charset=utf-8\n");
            responseBuilder.append("Content-Length: " + requestFile.length() + "\n\n");
        }

        // Write the header
        byte respBuff[] = responseBuilder.toString().getBytes();
        os.write(respBuff);

        if(!requestFile.exists()){
            // Don't return any more data if the file doesn't exist
            return;
        }

        // Open the file for reading and write out the data
        FileInputStream reader = new FileInputStream(requestFile);

        byte buff[] = new byte[BUFFER_SIZE];
        int numRead = reader.read(buff);;

        while(numRead > 0){

            os.write(buff, 0, numRead);
            numRead = reader.read(buff);
        }
    }

    /**
     * Processes a POST request
     * @param request   the request that was received
     * @param reader    reader to read data from
     * @param os        output to write results to
     * @throws IOException  if there was an error reading or writing to the streams
     */
    private void processPOST(HTTPRequest request, BufferedReader reader,
            DataOutputStream os) throws IOException {

        // Log that the server is processing a POST request
        System.out.println("Processing post");

        // Read in the amount of data specified in the request
        char buff[] = new char[request.getLength()];
        reader.read(buff);

        // Split the POST data into key-value pairs
        String params[] = (new String(buff)).split("&");
        Map<String, String> map = new HashMap<String, String>();

        for(String param : params){
            String keyvalue[] = param.split("=");

            if(keyvalue.length > 1){
                map.put(keyvalue[0], keyvalue[1]);
```

```
            }else{
                map.put(keyvalue[0], null);
            }
        }

        // Display the values
        displayOutput(map, os);
    }

    /**
     * Executed by the thread to handle the request
     */
    @Override
    public void run() {

        try {
            // Read and parse the request
            BufferedReader reader = new BufferedReader(new InputStreamReader(
                    mSocket.getInputStream()));
            HTTPRequest request = HTTPRequest.fromStream(reader);

            // Create an output stream and process the request
            DataOutputStream os = new DataOutputStream(mSocket.getOutputStream());
            HTTPRequestType requestType = request.getRequestType();

            if(requestType == HTTPRequestType.GET){
                processGET(request, os);
            }else if(requestType == HTTPRequestType.POST){
                processPOST(request, reader, os);
            }else{
                System.err.println("Unsupported request type received");
            }

            // Flush and close the output stream.
            os.flush();
            os.close();

            // Close the connection once transmission is complete
            mSocket.close();
        } catch (IOException e) {
            System.err.println("I/O exception occured reading data");
            e.printStackTrace();
        }

    }

}
```

```java
package com.msoe.ce4960.lab6;

import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.net.InetAddress;

/**
 * Parses an HTTP request
 * @author Erik Sommer
 *
 */
public class HTTPRequest {

    /**
     * Enum used to specify the type of request
     * @author Erik Sommer
     *
     */
    public enum HTTPRequestType{
        /**
         * A DELETE request
         */
        DELETE,

        /**
         * A GET request
         */
        GET,

        /**
         * A POST request
         */
        POST,

        /**
         * A PUT request
         */
        PUT;
    }

    /**
     * Root directory for files
     */
    private static final String FILE_ROOT = "files/";

    /**
     * Key for the Accept field in the request
     */
    private static final String KEY_ACCEPT = "Accept";

    /**
     * Key for the Accept-Encoding field in the request
```

```
     */
    private static final String KEY_ACCEPT_ENCODING = "Accept-Encoding";

    /**
     * Key for the Accept-Language field in the request
     */
    private static final String KEY_ACCEPT_LANGUAGE = "Accept-Language";

    /**
     * Key for the Cache-Control field in the request
     */
    private static final String KEY_CACHE_CONTROL = "Cache-Control";

    /**
     * Key for the Connection field in the request
     */
    private static final String KEY_CONNECTION = "Connection";

    /**
     * Key for the Content-Length field in the request
     */
    private static final String KEY_CONTENT_LENGTH = "Content-Length";

    /**
     * Key for the Host field in the request
     */
    private static final String KEY_HOST = "Host";

    /**
     * Key for the User-Agent field in the request
     */
    private static final String KEY_USER_AGENT = "User-Agent";

    /**
     * Creates an {@link HTTPRequest} from the input stream
     * @param reader    read to read the data from
     * @return          an {@code HTTPRequest} if it is successfully parsed,
     *                  {@code null} otherwise
     * @throws IOException  if there was an I/O error reading the stream
     */
    public static HTTPRequest fromStream(BufferedReader reader) throws IOException{

        String request;
        HTTPRequest returnRequest = new HTTPRequest();

        // Read the line of the request
        request = reader.readLine();

        // While the line isn't empty (not just "\n")
        while(!request.isEmpty()){

            // Split the request into key-value pairs
            String splitRequest[] = request.split(":");
```

```
            if(splitRequest.length == 1){
                // If there isn't a ":" then it's the first line of the request
                // process it
                processRequest(splitRequest[0], returnRequest);
            }else{

                // Separate the key and value
                String key = splitRequest[0];
                String value = splitRequest[1].trim();

                // Look at the key and process it appropriately
                if(key.equalsIgnoreCase(KEY_HOST)){
                    returnRequest.mHostAddress = InetAddress.getByName(value);
                    returnRequest.mPort = Integer.parseInt(splitRequest[2]);
                }else if(key.equalsIgnoreCase(KEY_USER_AGENT)){
                    StringBuilder builder = new StringBuilder();

                    boolean first = true;
                    for(int i = 1; i < splitRequest.length; i++){

                        if(!first){
                            builder.append(":");
                        }else{
                            first = false;
                        }

                        builder.append(splitRequest[i]);
                    }
                    returnRequest.mUserAgent = builder.toString();
                }else if(key.equalsIgnoreCase(KEY_ACCEPT)){
                    returnRequest.mAccept = value.split(",");
                }else if(key.equalsIgnoreCase(KEY_ACCEPT_LANGUAGE)){
                    returnRequest.mAcceptLanguage = value.split(",");
                }else if(key.equalsIgnoreCase(KEY_ACCEPT_ENCODING)){
                    returnRequest.mAcceptEncoding = value.split(",");
                }else if(key.equalsIgnoreCase(KEY_CONNECTION)){
                    returnRequest.mConnection = value;
                }else if(key.equalsIgnoreCase(KEY_CACHE_CONTROL)){
                    returnRequest.mCacheControl = value;
                }else if(key.equalsIgnoreCase(KEY_CONTENT_LENGTH)){
                    returnRequest.mLength = Integer.parseInt(value);
                }
            }

            // Read the next line of data
            request = reader.readLine();
        }

        return returnRequest;
    }

    /**
```

```
     * Processes the request from an HTTP header
     * @param request       the request to process
     * @param returnRequest the {@link HTTPRequest} to fill in
     */
    private static void processRequest(String request, HTTPRequest returnRequest) {

        // Split the string on spaces and grab the parameters
        String splitString[] = request.split(" ");
        String requestType = splitString[0].trim();
        String fileName = splitString[1].trim();
        String protocol = splitString[2].trim();

        // Set the protocol type
        if(requestType.equals("GET")){
            returnRequest.mRequestType = HTTPRequestType.GET;
        }else if(requestType.equals("POST")){
            returnRequest.mRequestType = HTTPRequestType.POST;
        }

        // If the root file is requested, actually use "index.html"
        if(fileName.equals("/")){
            fileName = "index.html";
        }

        // Set the request file and the protocol
        returnRequest.mFile = new File(FILE_ROOT + fileName);
        returnRequest.mProtocol = protocol;
    }

    /**
     * Accepted content types of the request
     */
    private String[] mAccept;

    /**
     * Accepted encodings of the request
     */
    private String[] mAcceptEncoding;

    /**
     * Accepted languages of the request
     */
    private String[] mAcceptLanguage;



    /**
     * Cache-control flags of the request
     */
    private String mCacheControl;

    /**
     * Connection type of the request
```

```
     */
    private String mConnection;

    /**
     * Requested file
     */
    private File mFile;

    /**
     * Host address of the request
     */
    private InetAddress mHostAddress;

    /**
     * Length of the request
     */
    private int mLength;

    /**
     * Port for the request
     */
    private int mPort;

    /**
     * Protocol for the request
     */
    private String mProtocol;

    /**
     * Type of the request
     */
    private HTTPRequestType mRequestType;

    /**
     * User agent of the request
     */
    private String mUserAgent;

    /**
     * Gets the content-types that are accepted
     * @return the content-types that are accepted
     */
    public String[] getAccept() {
        return mAccept;
    }

    /**
     * Gets the encodings that are accepted
     * @return the encodings that are accepted
     */
    public String[] getAcceptEncoding() {
        return mAcceptEncoding;
    }
```

```
    /**
     * Gets the languages that are accepted
     * @return the languages that are accepted
     */
    public String[] getAcceptLanguage() {
        return mAcceptLanguage;
    }

    /**
     * Gets the cache-control flags
     * @return the cache-control flags
     */
    public String getCacheControl() {
        return mCacheControl;
    }

    /**
     * Gets the connection flags
     * @return the connection flags
     */
    public String getConnection() {
        return mConnection;
    }

    /**
     * Gets the file requested
     * @return  the file requested
     */
    public File getFile() {
        return mFile;
    }

    /**
     * Gets the host address
     * @return the host address
     */
    public InetAddress getHostAddress() {
        return mHostAddress;
    }

    /**
     * Gets the length of the request
     * @return  the length of the request
     */
    public int getLength() {
        return mLength;
    }

    /**
     * Gets the port of the request
     * @return  the port of the request
     */
```

```
    public int getPort() {
        return mPort;
    }

    /**
     * Gets the protocol of the request
     * @return the protocol of the request
     */
    public String getProtocol() {
        return mProtocol;
    }

    /**
     * Gets the type of request
     * @return the type of request
     */
    public HTTPRequestType getRequestType() {
        return mRequestType;
    }

    /**
     * Gets the user agent
     * @return the user agent
     */
    public String getUserAgent() {
        return mUserAgent;
    }

    /**
     * Sets the content-types that are accepted
     * @param accept the content-types that are accepted
     */
    public void setAccept(String[] accept) {
        this.mAccept = accept;
    }

    /**
     * Sets the encodings that are accepted
     * @param acceptEncoding the encodings that are accepted
     */
    public void setAcceptEncoding(String[] acceptEncoding) {
        this.mAcceptEncoding = acceptEncoding;
    }

    /**
     * Sets the languages that are accepted
     * @param acceptLanguage    the languages that are accepted
     */
    public void setAcceptLanguage(String[] acceptLanguage) {
        this.mAcceptLanguage = acceptLanguage;
    }

    /**
```

```
     * Sets the cache-control flags
     * @param cacheControl  the cache control flags
     */
    public void setCacheControl(String cacheControl) {
        this.mCacheControl = cacheControl;
    }

    /**
     * Sets the connection flags
     * @param connection    the connection flags
     */
    public void setConnection(String connection) {
        this.mConnection = connection;
    }

    /**
     * Sets the file requested
     * @param file  the file requested
     * @throws IllegalArgumentException if {@code file} is {@code null}
     */
    public void setFile(File file) {

        // Parameter validation
        if(file == null){
            throw new IllegalArgumentException("file is null");
        }

        this.mFile = file;
    }

    /**
     * Sets the host address
     * @param hostAddress   the host address
     */
    public void setHostAddress(InetAddress hostAddress) {
        this.mHostAddress = hostAddress;
    }

    /**
     * Sets the length of the request
     * @param length    the length of the request
     * @throws IllegalArgumentException if {@code length} is less than 0
     */
    public void setLength(int length) {

        // Parameter validation
        if(length < 0){
            throw new IllegalArgumentException("length is less than 0");
        }

        this.mLength = length;
    }
```

```
    /**
     * Sets the port of the request
     * @param port  the port of the request
     * @throws IllegalArgumentException if {@code port} is not between 0 and
     *                                  65535
     */
    public void setPort(int port) {

        // Parameter validation
        if((port < 0) || (port > 65535)){
            throw new IllegalArgumentException("port is not between 0 and 65535");
        }
        this.mPort = port;
    }

    /**
     * Sets the protocol of the request
     * @param protocol  the protocol of the request
     * @throws IllegalArgumentException if {@code protocol} is {@code null} or
     *                                  empty
     */
    public void setProtocol(String protocol) {

        // Parameter validation
        if((protocol == null) || protocol.isEmpty()){
            throw new IllegalArgumentException("protocol is null or empty");
        }

        this.mProtocol = protocol;
    }

    /**
     * Sets the type of request
     * @param requestType   the type of the request
     * @throws IllegalArgumentException if {@code requestType} is {@code null}
     */
    public void setRequestType(HTTPRequestType requestType) {

        // Parameter validation
        if(requestType == null){
            throw new IllegalArgumentException("requestType is null");
        }

        this.mRequestType = requestType;
    }


    /**
     * Sets the user agent
     * @param userAgent the user agent
     */
    public void setUserAgent(String userAgent) {
        this.mUserAgent = userAgent;
```

```
        }
}
```