```java
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.ArrayList;
import java.util.List;
import java.util.StringTokenizer;

import org.dom4j.Document;
import org.dom4j.DocumentException;
import org.dom4j.Element;
import org.dom4j.Node;
import org.dom4j.io.SAXReader;

public class User {
    private Socket s;
    private DataOutputStream dos;
    private DataInputStream is;
    private BufferedReader dis;
    private ArrayList<AvailableFile> availableFiles;
    private String userName;
    private String localHost;
    private String serverPort;
    private String connectionSpeed;
    private boolean loggedOn;

    /****
     *
     * Forms a connection to the Centralized_Server and starts the local server
     * thread.
     *
     ****/
    public void makeConnection(String userName, String serverHostName, String
    serverPort, String connectionSpeed,
            String localHost, String localPort) throws IOException {

        InetAddress ip = InetAddress.getByName("localhost");
        // Connection to server.
        s = new Socket(ip, Integer.parseInt(serverPort));

        // IP of the server to connect to.
        this.localHost = localHost;
        this.userName = userName;
        this.connectionSpeed = connectionSpeed;

        // Set up input and output stream to send and receive messages.
        is = new DataInputStream(s.getInputStream());

        dos = new DataOutputStream(s.getOutputStream());

        // Sends the initial connectionString that holds information about the client.
        dos.writeUTF(userName + " " + localHost + " " + connectionSpeed + " " +
        localPort);

        // Checks to see in the client has a XML file called fileList that holds
        // information of files available for download.
        File fileList = new File("./filelist.xml");
        if (fileList.exists()) {

            // Code From
```

```
68              // https://www.tutorialspoint.com/java_xml/java_dom4j_parse_document.htm
69              // This code parses the XML file with the filelist on it and sends the file
70              // information to the Central server.
71              try {
72
73                  SAXReader reader = new SAXReader();
74                  Document document = reader.read(fileList);
75                  System.out.println("Root element :" +
                    document.getRootElement().getName());
76
77                  Element classElement = document.getRootElement();
78
79                  List<Node> nodes = document.selectNodes("/filelist/file");
80
81                  // Let the server know data is coming and how much
82                  dos.writeUTF("200" + " " + nodes.size());
83
84                  for (Node node : nodes) {
85
86                      String fileName = node.selectSingleNode("name").getText();
87                      String fileDescription =
                        node.selectSingleNode("description").getText();
88                      dos.writeUTF(fileName + "$" + fileDescription);
89
90                  }
91              } catch (DocumentException e) {
92                  e.printStackTrace();
93              }
94
95          } else {
96
97              // If the XML file isnt found let the server and client know!
98              dos.writeUTF("505");
99              System.out.println("You need a XML file with your fileList!");
100         }
101
102         loggedOn = true;
103
104         // Handles other clients when they need to get a file from this localServer.
105         Thread localServer = new Thread(new Runnable() {
106             public void run() {
107                 while (loggedOn) {
108                     try {
109                         ServerSocket localServer = new
                            ServerSocket(Integer.parseInt(localPort)); // localServerPort
110                         Socket client = localServer.accept();
111
112                         DataOutputStream out = new
                            DataOutputStream(client.getOutputStream());
113                         DataInputStream in = new
                            DataInputStream(client.getInputStream());
114
115                         // Read in the request from the server.
116                         String command = in.readUTF();
117                         StringTokenizer tokens = new StringTokenizer(command);
118
119                         String targetFile = tokens.nextToken();
120                         targetFile = tokens.nextToken();
121
122                         // Checks to see if the targetFile exists on this server.
123                         File file = new File("./" + targetFile);
124                         if (file.exists()) {
125
126                             // Tell the client we have the file.
127                             out.writeUTF("200");
128                             BufferedReader contentRead = new BufferedReader(new
                                FileReader(targetFile));
129
130                             PrintWriter pwrite = new PrintWriter(out, true);
```

```
131
132                             String str;
133                             while ((str = contentRead.readLine()) != null) {
134                                 pwrite.println(str);
135                             }
136                             contentRead.close();
137                         } else {
138
139                             // Tell the client the file could not be found.
140                             out.writeUTF("505");
141                         }
142                         client.close();
143                         localServer.close();
144
145                     } catch (IOException e) {
146                         // TODO Auto-generated catch block
147                         e.printStackTrace();
148                     }
149                 }
150             }
151         });
152
153         // Start up the local Server.
154         localServer.start();
155     }
156
157     /****
158      *
159      * Allows the client to search the Central Server for available files to
160      * download.
161      *
162      ****/
163     public boolean search(String keyword) {
164         StringTokenizer tokens;
165
166         try {
167
168             dos.writeUTF(keyword);
169             String str = "";
170
171             // If no files match the search 'str' will equal EOF
172             str = is.readUTF();
173
174             availableFiles = new ArrayList<AvailableFile>();
175
176             // While there are more files to read. Read them.
177             while (!str.equals("EOF")) {
178
179                 tokens = new StringTokenizer(str);
180                 String hostSpeed = tokens.nextToken();
181                 String hostName = tokens.nextToken();
182                 int hostPort = Integer.parseInt(tokens.nextToken());
183                 String hostFileName = tokens.nextToken();
184                 String hostUserName = tokens.nextToken();
185
186                 // Store the file information in an AvailableFile object.
187                 AvailableFile file = new AvailableFile(hostUserName, hostName,
                    hostPort, hostFileName, hostSpeed);
188                 availableFiles.add(file);
189                 str = is.readUTF();
190
191             }
192         } catch (IOException e) {
193             e.printStackTrace();
194         }
195
196         // Return whether or not matches were found
197         if (availableFiles.isEmpty())
198             return false;
```

```
199             else
200                 return true;
201         }
202 
203         /****
204          *
205          * Returns the List of Available Files.
206          *
207          ****/
208         public ArrayList<AvailableFile> getAvailableFiles() {
209             return availableFiles;
210         }
211 
212         /****
213          *
214          * Checks to see if the file requested is in the available files list on the
215          * server. If the file is available the client forms a socket with the server
216          * that holds the file. The client then downloads the file.
217          *
218          ****/
219         public boolean retrieve(String file) throws UnknownHostException, IOException {
220             boolean downloaded = false;
221             for (int i = 0; i < availableFiles.size(); i++) {
222 
223                 if (availableFiles.get(i).fileName.equals(file) &&
                    !availableFiles.get(i).hostUserName.equals(userName)) {
224 
225                     AvailableFile targetFile = availableFiles.get(i);
226                     InetAddress ip = InetAddress.getByName("localhost");
227 
228                     // New Socket for file Transfer.
229                     Socket ret = new Socket(ip, targetFile.port);
230 
231                     DataOutputStream out = new DataOutputStream(ret.getOutputStream());
232                     DataInputStream din = new DataInputStream(ret.getInputStream());
233                     BufferedReader in = new BufferedReader(new
                        InputStreamReader(ret.getInputStream()));
234 
235                     String command = "retr: " + file;
236                     out.writeUTF(command);
237 
238                     String response = din.readUTF();
239 
240                     // If the server has the file start the download else nothing.
241                     if (!response.equals("505")) {
242 
243                         String str = "";
244                         File newFile = new File("./" + file);
245                         FileWriter fw = new FileWriter("./" + file);
246                         PrintWriter writer = new PrintWriter(fw);
247 
248                         // Read in the file.
249                         while ((str = in.readLine()) != null) {
250                             writer.println(str);
251                         }
252                         writer.close();
253                         downloaded = true; // download flag.
254 
255                     } else {
256                         System.out.println("File Not Found!");
257                     }
258 
259                     in.close();
260                     ret.close();
261                 }
262             }
263             return downloaded;
264         }
265 
```

```
266         /****
267          *
268          * Sends a message to the server closing the connection.
269          *
270          ****/
271         public void quit() {
272 
273             try {
274                 dos.writeUTF("QUIT");
275                 s.close();
276             } catch (IOException e) {
277                 e.printStackTrace();
278             }
279 
280             // Ends the thread.
281             loggedOn = false;
282         }
283     }
284 
285     /*****************************************************************************************
    ***
286      *
287      * Holds the information for the available files that matched in the search.
288      *
289 
         *****************************************************************************************
         **/
290     class AvailableFile {
291 
292         public String hostUserName;
293         public String hostName;
294         public String fileName;
295         public String speed;
296         public int port;
297 
298         public AvailableFile(String hostUserName, String hn, int p, String fn, String
            speed) {
299             this.hostUserName = hostUserName;
300             this.hostName = hn;
301             this.port = p;
302             this.fileName = fn;
303             this.speed = speed;
304         }
305 
306     }
307 
```