```java
package ballmerpeak.turtlenet.remoteserver;

import ballmerpeak.turtlenet.shared.Message;
import java.io.*;
import java.net.*;
import java.util.Date;
import java.util.StringTokenizer;
import javax.xml.bind.DatatypeConverter;

public class Server
{
    public  static String shutdownPassword = "SHUTDOWN 83eea84d472df09f5e64468996fdff0e";
    private static ServerSocket socket;
    private static boolean running = true;

    public static void start (int port) {
        Socket incoming;
        Thread t;

        try {
            socket = new ServerSocket(port);

            while (running) {
                incoming = socket.accept();
                t = new Thread(new Session(incoming));
                t.start();
            }
        } catch (Exception e) {
            if (running)
                System.out.println("ERROR: " + e.getMessage());
        } finally {
            shutdown();
        }
    }

    public static void shutdown() {
        running = false;

        try {
            socket.close();
        } catch (Exception e) {
            System.out.println("ERROR: " + e.getMessage());
            System.exit(1);
        }
    }

    public static void main (String[] argv) {
        System.out.println("Server running...");
        start(31415);
    }
}

class Session implements Runnable
{
    private Socket client;

    Session (Socket s) {
        client = s;
    }

    // execute()s the clients command and then closes the connection.
    public void run() {
        System.out.println("Connection from " + client.getInetAddress().getHostAddress());
        BufferedReader in = null;
        PrintWriter out = null;

        try {
            in = new BufferedReader
              (new InputStreamReader(client.getInputStream()));
            out = new PrintWriter
              (new OutputStreamWriter(client.getOutputStream()));

            execute(in.readLine(), in, out);
        } catch (IOException e) {
            System.out.println("ERROR: " + e.getMessage());
        }
        out.flush();
        //close everything related to this session
        try {
            in.close();
        } catch (Exception e) {}

        try {
            out.close();
        } catch (Exception e) {}

        try {
            client.close();
        } catch (Exception e) {}
    }

    //Protocol:
    //NB: The universe came into existance at midnight on january 1st 1970
```

```
94        //A typical session is the following:
95        //    Connect -> Send command to server -> disconnect
96        //Valid commands are the following:
97        //    t          - request the number of milliseconds since midnight 1970-01-01
98        //    s <string> - request that a string be stored on the server
99        //    get <long> - get every message posted since <long> number of milliseconds past midnight 1970-01-01
100       //    c <claim message> - claim a username UNENCRYPTED PUBLICALLY KNOWN
101       //Responses are the following:
102       //s         - success
103       //e         - error
104       //<long>    - number of milliseconds since midnight on 1970-01-01
105       //<string>* - (0 or more strings) messages requested using get
106       public void execute(String cmd, BufferedReader in, PrintWriter out) {
107           System.out.println("Recieved \"" + cmd + "\"");
108
109           if (cmd.equals(Server.shutdownPassword)) {
110               System.out.println("WARNING: shutdown password should be loaded from config file");
111               System.out.println("Shutting down");
112               Server.shutdown();
113           }
114
115           else if (cmd.equals("t")) {
116               out.println(String.valueOf(new Date().getTime()));
117               out.println("s");
118           }
119
120           else if (cmd.length() > 2 && cmd.substring(0,1).equals("s")) {
121               try {
122                   String message = cmd.substring(2);
123                   System.out.println("Storing: " + message);
124                   BufferedWriter writer = new BufferedWriter(
125                                           new FileWriter(
126                                           new File("./data/" + (new Date()).getTime() +
127                                                   "_" + Hasher.hash(message))));
128                   writer.write(message);
129                   writer.close();
130                   out.println("s");
131               } catch (Exception e) {
132                   System.out.println("ERROR: Unable to save: " + e);
133               }
134           }
135
136           else if (cmd.length() > 4 && cmd.substring(0,3).equals("get")) {
137               System.out.println(cmd);
138               try {
139                   String timestamp = cmd.substring(4);
140                   long lastRead = Long.parseLong(timestamp);
141
142                   File dataDir = new File("./data");
143                   File[] files = dataDir.listFiles();
144                   for (int i = 0; i < files.length; i++) {
145                       if (lastRead <= getTimestamp(files[i])) {
146                           BufferedReader reader = new BufferedReader(
147                                                   new FileReader(files[i]));
148                           String msg = reader.readLine();
149                           out.println(msg);
150                       }
151                   }
152                   out.println("s");
153               } catch (Exception e) {
154                   System.out.println("ERROR: Cannot execute \"" + cmd + "\"");
155                   out.println("e");
156               }
157           }
158
159           else if (cmd.length() > 2 && cmd.substring(0,2).equals("c ")) {
160               Message claim = Message.parse(
161                                   new String(
162                                       DatatypeConverter.parseBase64Binary(
163                                           cmd.substring(2))));
164
165               String content = claim.getContent();
166               File data = new File("./data/" + (new Date()).getTime() + "_" + content);
167               if(userExists(content)) {
168                   out.println("e");
169               } else {
170                   try {
171                       BufferedWriter writer = new BufferedWriter(new FileWriter(data));
172                       writer.write(cmd);
173                       writer.close();
174                       out.println("s");
175                   } catch (Exception e) {
176                       System.out.println("ERROR: Could not write claim to disk");
177                       out.println("e");
178                   }
179               }
180           }
181
182           else {
183               System.out.println("Recieved \"" + cmd + "\", ignoring it");
184               out.println("e");
185           }
186
```

```
187             out.flush();
188         }
189 
190         //44634633434_HASH -> 44634633434
191         private long getTimestamp (File f) {
192             try {
193                 String fn = f.getName();
194                 if (fn.indexOf("_") != -1) {
195                     String ts = fn.substring(0, fn.indexOf("_"));
196                     return Long.parseLong(ts);
197                 }
198             } catch (Exception e) {
199                 System.out.println("ERROR: Could not parse file timestamp: " + e);
200             }
201             return 1;
202         }
203 
204         private Boolean userExists (String name) {
205             File dir = new File("./data");
206             File[] files = dir.listFiles();
207                 for (int i = 0; i < files.length; i++) {
208                     if (files[i].getName().indexOf("_") != -1) {
209                         String fname = files[i].getName();
210                         String[] tokens = new String[2];
211                         StringTokenizer tokenizer = new StringTokenizer(fname, "_", false);
212                         tokens[0] = tokenizer.nextToken();
213                         tokens[1] = "";
214                         while (tokenizer.hasMoreTokens())
215                             tokens[1] += tokenizer.nextToken();
216                         if (tokens[1].equals(name))
217                             return true;
218                     }
219                 }
220             return false;
221         }
222     }
```