```
 1: import java.io.*;
 2: import java.net.*;
 3: import java.util.Vector;
 4: import java.util.Enumeration;
 5:
 6: public class chatkaro12mServer
 7: {
 8:    private int port=5001;
 9:    private boolean li=true;
10:    private Vector clients=new Vector();
11:
12:    public static void main(String a[])
13:    {
14:       System.out.println(" Press ctrl-c to Quit.");
15:       new chatkaro12mServer().server();
16:    }
17:
18:    void server()
19:    {
20:       ServerSocket serverSock=null;
21:       try
22:       {
23:          InetAddress serverAddr=InetAddress.getByName(null);
24:          System.out.println("Waiting for" + serverAddr.getHostName() +
25:             "on port"+port);
26:          serverSock=new ServerSocket(port,50);
27:       }
28:       catch(IOException e)
29:       {
30:          System.out.println(e.getMessage()+":Failed");
31:          return;
32:       }
33:       while(li)
34:       {
35:          try
36:          {
37:             Socket socket=serverSock.accept();
38:             System.out.println("Accept "
39:             + socket.getInetAddress().getHostName());
40:             DataOutputStream remoteOut =
41:                new DataOutputStream(socket.getOutputStream());
42:             clients.addElement(remoteOut);
43:             new ServerHelper(socket,remoteOut,this).start();
44:          }
45:          catch(IOException e)
46:          {
47:             System.out.println(e.getMessage()+":Failed");
48:          }
49:       }
50:       if(serverSock !=null)
51:       {
52:          try
53:          {
54:             serverSock.close();
55:          }
56:          catch(IOException x)
57:          {
58:          }
59:       }
60:    }
61:
62:    synchronized Vector getClients()
63:    {
```

```
   64:       return clients;
   65:    }
   66:
   67:    synchronized void removeFromClients(DataOutputStream remoteOut)
   68:    {
   69:       clients.removeElement(remoteOut);
   70:    }
   71: }
   72:
   73: class ServerHelper extends Thread
   74: {
   75:    private Socket sock;
   76:    private DataOutputStream remoteOut;
   77:    private chatkaro12mServer server;
   78:    private boolean li=true;
   79:    private DataInputStream remoteIn;
   80:    ServerHelper(Socket sock, DataOutputStream remoteOut,
   81:       chatkaro12mServer server) throws IOException
   82:    {
   83:       this.sock=sock;
   84:       this.remoteOut=remoteOut;
   85:       this.server=server;
   86:       remoteIn=new DataInputStream(sock.getInputStream());
   87:    }
   88:
   89:    public synchronized void run()
   90:    {
   91:       String s;
   92:
   93:       try
   94:       {
   95:          while(li)
   96:          {
   97:             s=remoteIn.readUTF();
   98:             broadcast(s);
   99:          }
  100:       }
  101:       catch(IOException e)
  102:       {
  103:          System.out.println(e.getMessage()+"connection lost");
  104:       }
  105:       finally
  106:       {
  107:          try
  108:          {
  109:             cleanUp();
  110:          }
  111:          catch (IOException x)
  112:          {
  113:          }
  114:       }
  115:    }
  116:
  117:    private void broadcast(String s)
  118:    {
  119:       Vector clients=server.getClients();
  120:       DataOutputStream dataOut=null;
  121:
  122:       for(Enumeration e=clients.elements(); e.hasMoreElements(); )
  123:       {
  124:          dataOut=(DataOutputStream)(e.nextElement());
  125:          if(!dataOut.equals(remoteOut))
  126:          {
```

```
  127:             try
  128:             {
  129:                dataOut.writeUTF(s);
  130:             }
  131:             catch(IOException x)
  132:             {
  133:                System.out.println(x.getMessage()+"Failed");
  134:                server.removeFromClients(dataOut);
  135:             }
  136:          }
  137:       }
  138:    }
  139:
  140:    private void cleanUp() throws IOException
  141:    {
  142:       if(remoteOut!=null)
  143:       {
  144:          server.removeFromClients(remoteOut);
  145:          remoteOut.close();
  146:          remoteOut=null;
  147:       }
  148:
  149:       if(remoteIn!=null)
  150:       {
  151:          remoteIn.close();
  152:          remoteIn=null;
  153:       }
  154:
  155:       if(sock!=null)
  156:       {
  157:          sock.close();
  158:          sock=null;
  159:       }
  160:    }
  161:
  162:    protected void finalize() throws Throwable
  163:    {
  164:       try
  165:       {
  166:          cleanUp();
  167:       }
  168:       catch(IOException x)
  169:       {
  170:       }
  171:       super.finalize();
  172:    }
  173: }
```