/\*

\* Copyright (c) 2004 David Flanagan. All rights reserved.

\* This code is from the book Java Examples in a Nutshell, 3nd Edition.

\* It is provided AS-IS, WITHOUT ANY WARRANTY either expressed or implied.

\* You may study, use, and modify it for any non-commercial purpose,

\* including teaching and use in open-source projects.

\* You may distribute it non-commercially as long as you retain this notice.

\* For a commercial use license, or to purchase the book,

\* please visit http://www.davidflanagan.com/javaexamples3.

\*/

package je3.net;

import java.io.\*;

import java.net.\*;

/\*\*

\* This program connects to a server at a specified host and port.

\* It reads text from the console and sends it to the server.

\* It reads text from the server and sends it to the console.

\*\*/

public class GenericClient {

public static void main(String[] args) throws IOException {

try {

// Check the number of arguments

if (args.length != 2)

throw new IllegalArgumentException("Wrong number of args");

// Parse the host and port specifications

String host = args[0];

int port = Integer.parseInt(args[1]);

// Connect to the specified host and port

Socket s = new Socket(host, port);

// Set up streams for reading from and writing to the server.

// The from\_server stream is final for use in the inner class below

final Reader from\_server=new InputStreamReader(s.getInputStream());

PrintWriter to\_server = new PrintWriter(s.getOutputStream());

// Set up streams for reading from and writing to the console

// The to\_user stream is final for use in the anonymous class below

BufferedReader from\_user =

new BufferedReader(new InputStreamReader(System.in));

// Pass true for auto-flush on println()

final PrintWriter to\_user = new PrintWriter(System.out, true);

// Tell the user that we've connected

to\_user.println("Connected to " + s.getInetAddress() +

":" + s.getPort());

// Create a thread that gets output from the server and displays

// it to the user. We use a separate thread for this so that we

// can receive asynchronous output

Thread t = new Thread() {

public void run() {

char[] buffer = new char[1024];

int chars\_read;

try {

// Read characters from the server until the

// stream closes, and write them to the console

while((chars\_read = from\_server.read(buffer)) != -1) {

to\_user.write(buffer, 0, chars\_read);

to\_user.flush();

}

}

catch (IOException e) { to\_user.println(e); }

// When the server closes the connection, the loop above

// will end. Tell the user what happened, and call

// System.exit(), causing the main thread to exit along

// with this one.

to\_user.println("Connection closed by server.");

System.exit(0);

}

};

// Now start the server-to-user thread

t.start();

// In parallel, read the user's input and pass it on to the server.

String line;

while((line = from\_user.readLine()) != null) {

to\_server.print(line + "\r\n");

to\_server.flush();

}

// If the user types a Ctrl-D (Unix) or Ctrl-Z (Windows) to end

// their input, we'll get an EOF, and the loop above will exit.

// When this happens, we stop the server-to-user thread and close

// the socket.

s.close();

to\_user.println("Connection closed by client.");

System.exit(0);

}

// If anything goes wrong, print an error message

catch (Exception e) {

System.err.println(e);

System.err.println("Usage: java GenericClient <hostname> <port>");

}

}

}

/\*

\* Copyright (c) 2004 David Flanagan. All rights reserved.

\* This code is from the book Java Examples in a Nutshell, 3nd Edition.

\* It is provided AS-IS, WITHOUT ANY WARRANTY either expressed or implied.

\* You may study, use, and modify it for any non-commercial purpose,

\* including teaching and use in open-source projects.

\* You may distribute it non-commercially as long as you retain this notice.

\* For a commercial use license, or to purchase the book,

\* please visit http://www.davidflanagan.com/javaexamples3.

\*/

package je3.net;

import java.io.\*;

import java.net.\*;

/\*\*

\* This program is a very simple Web server. When it receives a HTTP request

\* it sends the request back as the reply. This can be of interest when

\* you want to see just what a Web client is requesting, or what data is

\* being sent when a form is submitted, for example.

\*\*/

public class HttpMirror {

public static void main(String args[]) {

try {

// Get the port to listen on

int port = Integer.parseInt(args[0]);

// Create a ServerSocket to listen on that port.

ServerSocket ss = new ServerSocket(port);

// Now enter an infinite loop, waiting for & handling connections.

for(;;) {

// Wait for a client to connect. The method will block;

// when it returns the socket will be connected to the client

Socket client = ss.accept();

// Get input and output streams to talk to the client

BufferedReader in = new BufferedReader(

new InputStreamReader(client.getInputStream()));

PrintWriter out = new PrintWriter(client.getOutputStream());

// Start sending our reply, using the HTTP 1.1 protocol

out.print("HTTP/1.1 200 \r\n"); // Version & status code

out.print("Content-Type: text/plain\r\n"); // The type of data

out.print("Connection: close\r\n"); // Will close stream

out.print("\r\n"); // End of headers

// Now, read the HTTP request from the client, and send it

// right back to the client as part of the body of our

// response. The client doesn't disconnect, so we never get

// an EOF. It does sends an empty line at the end of the

// headers, though. So when we see the empty line, we stop

// reading. This means we don't mirror the contents of POST

// requests, for example. Note that the readLine() method

// works with Unix, Windows, and Mac line terminators.

String line;

while((line = in.readLine()) != null) {

if (line.length() == 0) break;

out.print(line + "\r\n");

}

// Close socket, breaking the connection to the client, and

// closing the input and output streams

out.close(); // Flush and close the output stream

in.close(); // Close the input stream

client.close(); // Close the socket itself

} // Now loop again, waiting for the next connection

}

// If anything goes wrong, print an error message

catch (Exception e) {

System.err.println(e);

System.err.println("Usage: java HttpMirror <port>");

}

}

}