

# Socket client and server exercises

## Exercise 1.

Create a simplified FTP (file transport) client and server where the client can send or download text files from the server:

### General use-cases

1. ) Client connects to the server and sends a 'file listing' message
2. ) Server sends back the list of the downloadable files
3. ) Client lists the files and asks the user what action they want to take? Upload or download? ('u' or 'd')
4. ) In both cases the user must give the full file name with extension
5. ) The client sends the selected file to the server (upload) or downloads the selected file from the server to a specific directory.

### Server viewpoint

1. ) After connecting, it reads the files from the /store subdirectory and sends the file names to the client after receiving the listing message.
2. ) We are waiting for the client's 'u' or 'd' operation
3. ) We get a filename from the client and if the action is 'd' (download), we read the file content and return its contents
4. ) If the operation is 'u' (upload), we open a new file with the specified name and wait for the data to be written to the file.

### Client viewpoint

1. ) The client connects and waits for the list of files coming back and writes it to the console
2. ) We ask for the "u" or "d" key
3. ) Then we'll ask for the file-name as well.
4. ) The client reads the files from the /files folder, or creates the downloaded file here
5. ) If you press "d", it creates /files/ and writes data from the server
6. ) If you press "u", /files/ is sent to the server

## TCP style

### Starting point of implementation

#### 1.) Traditional blocked TCP based socket server class in Java

## Socket server source code

```
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

public class Server {
    ServerSocket providerSocket;
    Socket connection = null;
    ObjectOutputStream out;
    ObjectInputStream in;
    String message;

    Server() {
    }

    void run() {
        try {
            // 1. create a socket server listening to port 8080
            providerSocket = new ServerSocket(8080, 10);
            // 2. waiting for the connection (here we are waiting until
next connection)
            connection = providerSocket.accept();
            // 3. create Input and Output streams
            out = new ObjectOutputStream(connection.getOutputStream());
            in = new ObjectInputStream(connection.getInputStream());
            // 4. socket communication
            do {
                try {
                    message = (String) in.readObject();
                    System.out.println("client>" + message);
                    if (message.equals("bye")) {
                        sendMessage("bye");
                    }
                } catch (ClassNotFoundException classnot) {
                    System.err.println("Data received in unknown
format");
                }
            } while (!message.equals("bye"));
        } catch (IOException ioException) {
            ioException.printStackTrace();
        } finally {
            // 4: close connection
            try {
                in.close();
                out.close();
                providerSocket.close();
            }
        }
    }
}
```

```

        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    }
}

void sendMessage(String msg) {
    try {
        out.writeObject(msg);
        out.flush();
        System.out.println("server>" + msg);
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
}

public static void main(String args[]) {
    Server server = new Server();
    while (true) {
        server.run();
    }
}
}

```

## Socket client source

```

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;
import java.net.UnknownHostException;

public class Client {
    Socket requestSocket;
    ObjectOutputStream out;
    ObjectInputStream in;
    String message;

    Client() {
    }

    void run() {
        try {
            // 1. try to connect to the socket: localhost:8080
            requestSocket = new Socket("localhost", 8080);
            // 2. Input and Output streams
            out = new
ObjectOutputStream(requestSocket.getOutputStream());
            in = new ObjectInputStream(requestSocket.getInputStream());
            // 3: communications

```

```
        do {
            try {
                sendMessage("Hello server");
                sendMessage("bye");
                message = (String) in.readObject();
            } catch (Exception e) {
                System.err.println("data received in unknown
format");
            }
        } while (!message.equals("bye"));
    } catch (UnknownHostException unknownHost) {
        System.err.println("You are trying to connect to an unknown
host!");
    } catch (IOException ioException) {
        ioException.printStackTrace();
    } finally {
        // 4: close connection
        try {
            in.close();
            out.close();
            requestSocket.close();
        } catch (IOException ioException) {
            ioException.printStackTrace();
        }
    }
}

void sendMessage(String msg) {
    try {
        out.writeObject(msg);
        out.flush();
        System.out.println("client>" + msg);
    } catch (IOException ioException) {
        ioException.printStackTrace();
    }
}

public static void main(String args[]) {
    Client client = new Client();
    client.run();
}
}
```

## Traditional UDP style

The following Agent sends a message and waits for a response on port 8080, also with UDP. In the Eclipse IDE, the text you type on the console can be sent by pressing ctrl+z

**Exercise 2.**

Modify the code so that you can transfer a burned-in name and existing text or image file larger than 2 kbytes and verify that it was successfully sent.

```
package org.ait;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPClient {
    public static void main(String args[]) throws Exception {
        BufferedReader inFromUser = new BufferedReader(new
InputStreamReader(System.in));
        DatagramSocket clientSocket = new DatagramSocket();
        InetAddress IPAddress = InetAddress.getByName("localhost");

        byte[] sendData = new byte[1024];
        byte[] receiveData = new byte[1024];

        String sentence = inFromUser.readLine();
        sendData = sentence.getBytes();

        DatagramPacket sendPacket = new DatagramPacket(sendData,
sendData.length, IPAddress, 8080);
        clientSocket.send(sendPacket);

        DatagramPacket receivePacket = new DatagramPacket(receiveData,
receiveData.length);
        clientSocket.receive(receivePacket);
        String modifiedSentence = new String(receivePacket.getData());

        System.out.println("converted:" + modifiedSentence);
        clientSocket.close();
    }
}
```

2.b) The UDP server waits for the agents messages on port 8080 and converts them to uppercase letters and sends them back to the client UDP socket.

```
package org.ait;

import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;

public class UDPServer {
```

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

    DatagramSocket serverSocket = new DatagramSocket(8080);

    byte[] bytesReceived = new byte[1024];
    byte[] bytesSent = new byte[1024];

    DatagramPacket receivePacket = new DatagramPacket(bytesReceived,
bytesReceived.length);
    // here we are waiting for the packets
    serverSocket.receive(receivePacket);

    String textMessage = new String(receivePacket.getData());

    System.out.println("I got: " + textMessage);

    InetAddress IPAddress = receivePacket.getAddress();
    int port = receivePacket.getPort();

    String upperCaseText = textMessage.toUpperCase();
    bytesSent = upperCaseText.getBytes();

    // send back
    DatagramPacket sendPacket = new DatagramPacket(bytesSent,
bytesSent.length, IPAddress, port);
    serverSocket.send(sendPacket);
    serverSocket.close();

}
}
```

From:  
<https://edu.iit.uni-miskolc.hu/> - Institute of Information Science - University of Miskolc

Permanent link:  
[https://edu.iit.uni-miskolc.hu/tanszek:oktatas:tcp\\_socket\\_connection?rev=1677426801](https://edu.iit.uni-miskolc.hu/tanszek:oktatas:tcp_socket_connection?rev=1677426801)

Last update: 2023/02/26 15:53

