上海交大—巴黎高科卓越工程师学院
SJTU-ParisTech Elite Institute of Technology

上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

Java Socket Programming — Instant Messenger

Deng Ruofan 516261910008

Contents

1 System diagram

The diagram of the program design is as following:

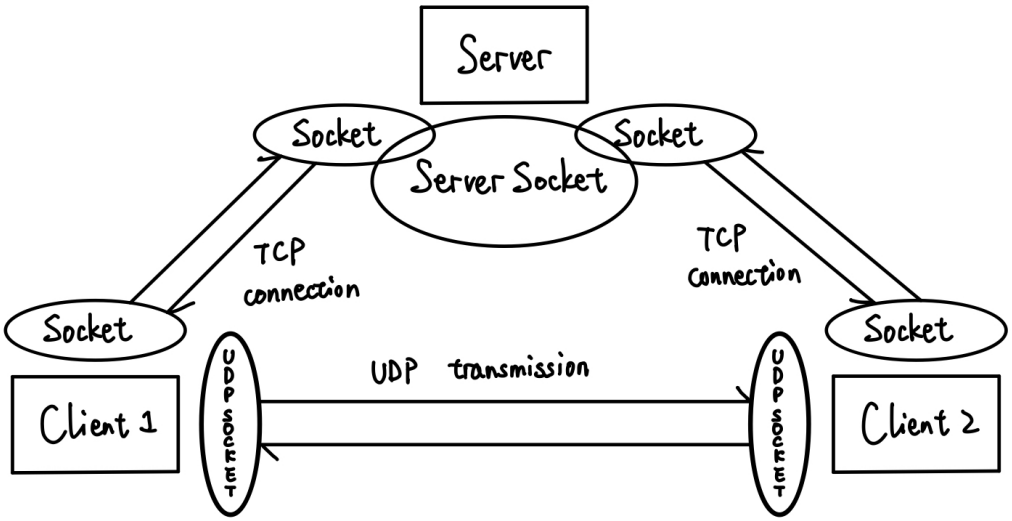

Figure 1: Diagram of the program design

Then taking all the classes and methods into consideration, we obtain:

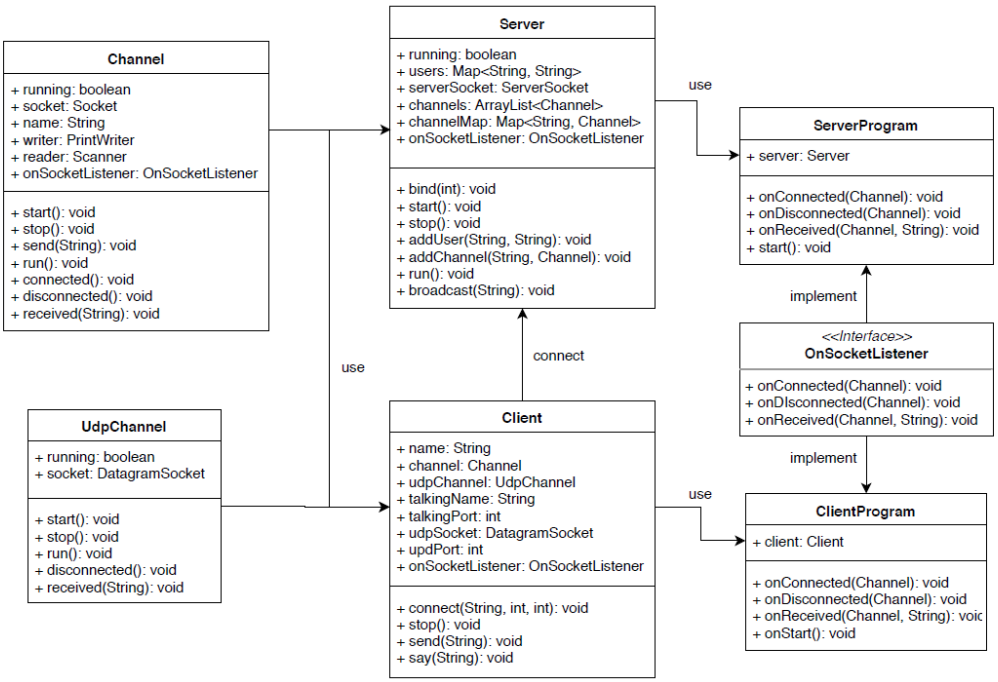

Figure 2: System diagram of the project with classes and methods defined

2 Functions description

2.1 *Channel.java*

The aim of the file *Channel.java* is to create a TCP channel between the server and a client.

To establiash a TCP channel, we need sockets on both sides and the information needed. Thus, several variables are introduced to record these information. We have also defined different methodes needed in the class *Channel*. Here is the list of methodes:

- Functions *setName*, *getName* and *getSocket* are designed to update or to get value of variables needed.
- Function *send* is used for sending messages through the channel.
- Function *run* creates a thread to transfer messages between two users of the channel and to cut down the connection when the command *quit* is demanded by a client.
- Function *connected* verifies that the connection between the server and a client is successfully established.
- Function *disconnected* declares the state of disconnection when current connection is cut down (the server is offline).
- Function *received* is to send commands from a client to the server.

2.2 *UdpChannel.java*

The file *UdpChannel.java* defines a Udp connection between two clients.

The *Runnable* interface ensures the parallism of multi-threads. We have defined several methods to realize an Udp transmission. Here is the list of functions:

- Functions *start* and *stop* are designed to start a Udp connection and to stop a Udp connection.
- Function *disconnected* is used to stop a Udp connection while not running.
- Function *run* unpackets every received package for the client.

2.3 *Server.java*

The aim of the file *Server.java* is to create a server thread.

The *Runnable* interface is designed to realize a parallism of threads. When a TCP connection is established, we create a socket on both sides, that is to say, one at server side and one at client side. Besides, a channel is necessary to transfer messages. The list *channels* is used to store the channels which are already occupied. For server, it registers its own socket and two maps as variables. The first map is a map of clients and their correspondant port number. The second one is a map of clients and the channel occupied of their TCP connection with the server.

The server main thread have one child thread listening specific port as TCP server, a pop-up thread is generated upon reception of a request.

There is a list of methods defined in the Server. Here is the list of functions:

- Functions *bind*, *start*, *stop* are designed respectively to bind the serverSocket (for monitoring), to start a new server thread and to stop a thread.
- Functions *addUser*, *addChannel* register the information of the client and the channel needed at server side.
- Functions *getUser*, *getChannel* are used to derive the information of the client and the channel.
- Functions *removeUser*, *remove* can remove a client and its channel from the map.
- Function *run* creates a channel for the TCP connection and supervises its state. It receives requests from clients and stops the server once demanded.
- Function *broadcast* uses the TCP channels already established to broadcast a message to all the clients.

2.4 *ServerProgram.java*

The file *ServerProgram.java* defines the operations and the behaviors needed for a server.

The interface *OnSocketListener* is designed to listen to the sockets at server side all the time. Once there is a client request, it will catch it and make a response. There are several methods defined in the class. Here is the list of functions:

- Function *onConnected* connects a client to the server through a channel and get the information of sockets, ports, etc.
- Function *onDisconnected* disconnects a client from the server and eliminates its information stored in the server.
- Function *onReceived* distinguishs three different commands of the client, impliments correspondant operations, and receives feedbacks.
- Function *start* initiates a server and start to listenning to the socket. It is ready to receive requests from clients and to broadcast a message.

2.5 *Client.java*

The aim of the file *Client.java* is to create a client.

The three principle variables of a client are *name*, *UdpPort* number and *channel*. The first two variables are used to identify a client and to establish a connection between two clients. The third one is needed for the connection between the client and the server.

The other variables *talkingName*, *talkingPort* and *UdpSocket* store the information of the target client to which the current one wants to talk. Here is the list of functions in the Client class:

- Function *connect* is to create a long term connection between the client and server to transfer messages (commands) and to initiate the Udp ports which will be needed afterwards.

- Function *setTalking* is used to set talking with a specified client.

- Function *getTalkingName* and *getTalkingPort* are used to get the name and the port of the client to which the current client is talking.

- Function *send* sends a message to the server through the TCP channel.

- Function *say* sends a message to the other client through the UDP channel.

2.6 *ClientProgram.java*

The file *ClientProgram.java* defines the operations and the behaviors needed for a client. The interface *OnSocketListener* implimented is designed to monitor the channel between the client and the server. Several operational methods are defined. Here is the list of functions:

- Function *onConnected* connects the client to the server through the proper channel.

- Function *onDisconnected* declares that the client is disconnected from the server.

- Function *onReceived* receives the messages from the server and impliments correpondant operations according to the contents of received messages.

- Function *start* initiates a client thread, sets the values of parameters and tries to connect to the server. It also decides whether a message should be sent to the server or to the other client.

3 How to build/run

3.1 Create a server thread and two client threads

• Use "*javac ServerProgram.java*" and "*javac ClientProgram.java* to compile all the files needed

• Firstly create the server by "*java ServerProgram*"

• Secondly create the first client by "*java ClientProgram*"

• Define the name and port number of the first client

• Thirdly create the second client by "*java ClientProgram*"

• Define the name and port number of the second client

```
E:\Java_projects\Project1\src>java ServerProgram
Client connected from 127.0.0.1:50726
Client connected from 127.0.0.1:50727
Asterix >> /add Asterix 1234
Obelix >> /add Obelix 1235
Asterix >> /connect Obelix
Good job, every one!
Server >> Good job, every one!
Obelix >> /quit Asterix
```

Figure 3: Build and run the server program

3.2 In a client thread

• Use *"/add #Name #Port"* to register the name and port number of the client in the server

• Connect to the other client by *"/connect #Name"*

• If the UDP connection is successfully established, it will receive the UDP port number of the other client from the server as *"/TargetPort #Name #Port"*,

• Send messages to the other client freely through the UDP connection

• Terminate the current conversation by *"/quit"*

```
E:\Java_projects\Project1\src>java ClientProgram
Name : Asterix
Local UDP Port : 1234
Server Connected.
/add Asterix 1234
/connect Obelix
server >> /TargetPort Obelix 1235
The UDP port of Obelix is: 1235
Hi, Obelix! I'm Asterix.
Message is sent to the destination port
Obelix >> Hi, Asterix! Nice to meet you.




Server >> Good job, every one!
server >> /quit/
Quit talking with Obelix
```

Figure 4: Build and run the client program 1

3.3 In a server thread

• If the UDP connection between two clients are successfully established, the server will send target ports to the clients as *"/TargetPort #Name #Port"*

• Once the TCP connections between the server and clients are established, the server can broadcast messages freely to all the clients

• There will be a feedback information in the server's interface for all client requests

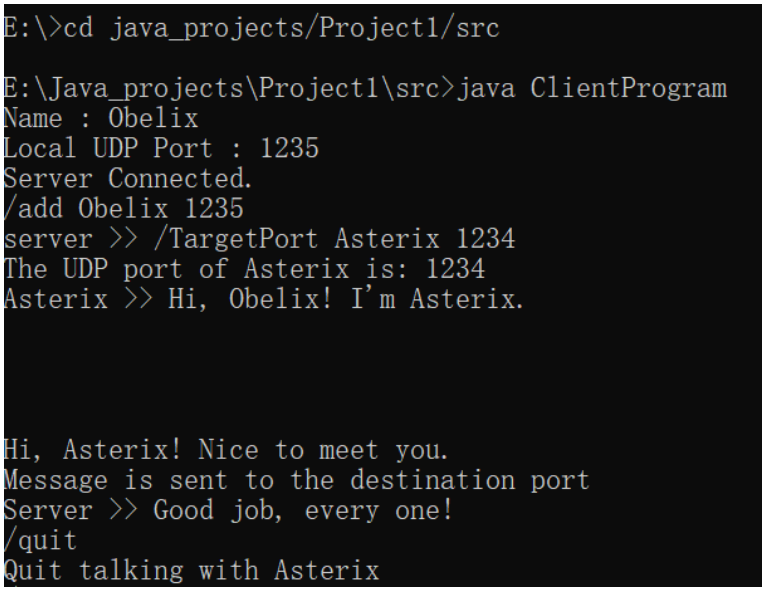

Figure 5: Build and run the client program 2

4 Problems and experiences

4.1 Troubleshooting experiences

- Lack of java programming experience is a big problem. By referring to books and CSDN blogs, I've learnt to read and understand the program and have tried to write some codes to finish the whole project.

- At the beginning of programming, I have no idea of where to start. After drawing a graph of the program design and referring to the functions already written, I started with the ClientProgram since it is simpler. Then, based on the functions needed, I've completed the codes of ServerProgram.

- A problem of the code met is that when the server broadcast a message to the clients, the message will be printed twice. I've eliminated a conditional branch *else* in *ClientProgram.java* to solve the problem.

4.2 Experiences and suggestions

- Some situations will possibly interrupt the program. For example, a spelling problem for a command, or a click on *Enter* accidentally.

- A limit is that clients can't send messages with a size larger than 1024 bytes.

- Small exercises of java programming could make the programming process more smoothly.

- Maybe more exercices during the lectures will be helpful to understand the course, since there is much information compressed in two hours.