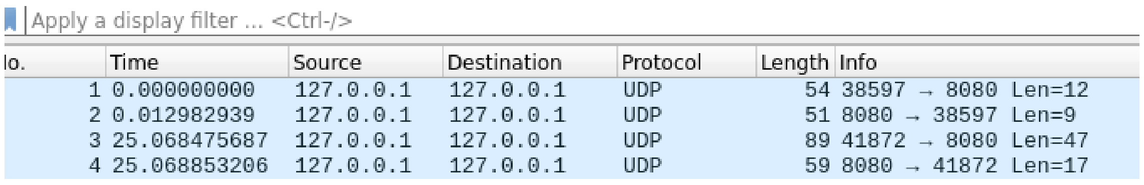
Practical Exercises 5

Name, Surname 1: Jakub Hejhal
Name, Surname 2: -
Degree (Ing. del Software/Ing. Informática/Ing. Computadores): Erasmus
Group (A/B/C): B
PC Label in the lab (e.g. 012): personal laptop

Trace 1 (**p5e1-5.pcapng**):

Exercise 1. How many messages are exchanged in the interaction between the client and the server? If the protocol had been TCP, how many messages would have been sent (you can suppose that the option and the text are in the same message)?

Four packets are exchanged in total, 2 for each client.



```
Apply a display filter ... <Ctrl-/>
lo.  Time          Source     Destination  Protocol  Length Info
   1 0.000000000   127.0.0.1  127.0.0.1    UDP           54 38597 → 8080 Len=12
   2 0.012982939   127.0.0.1  127.0.0.1    UDP           51 8080 → 38597 Len=9
   3 25.068475687  127.0.0.1  127.0.0.1    UDP           89 41872 → 8080 Len=47
   4 25.068853206  127.0.0.1  127.0.0.1    UDP           59 8080 → 41872 Len=17
```

Had those messages been sent through TCP, there would be 3 packets for the init handshake, 3 packets for the closing of the connection, and for each message there would be also an acknoledgement, that makes it 3 (SYN) + 3 (FIN) + 2 (client request) + 2 (server response) = 10 packets for 1 client (at least, supposing some packets aren't resent, fragmented etc.), so for 2 clients it would be at least 20 packets.

Exercise 2. What is the port used by the client? What is the one of the server? What kind of ports are they?
Client1.port = 38597
Client2.port = 41872
Server.port = 8080

Client ports are so called “Ephemeral ports”, which are short-lived ports chosen randomly by the os from range 49152 to 65535.

Server.port 8080 is a “Registered port” (assigned by IANA), that is used often as an alternative port to 80, but which can be used without root privilages.

Exercise 3. Wireshark has the option “Follow UDP stream” but in UDP the concept of flow of messages does not exist. How can Wireshark decide that a UDP datagram is from a specific flow of messages?

Each conversation is identified by tuple (Client IP, Client Port, Server IP, Server Port).

Exercise 4. Analyze a message sent by the server and check the length field of the UDP header. Is this value the same as the size of the payload (application data) transported by the datagram? Why?

The length field of the UDP header refers to the size of the UDP header **and** the payload.

Exercise 5. Analyze the long message (more than 20 characters). What is the size of the payload in the datagram sent by the client? And in the datagram sent by the server? What happens with the data that cannot be included in the datagram because of the size of the reception buffer?

Client sends 47B of payload data to the server, from which 3B include the command and the rest is text content to be transformed. Server then reveices the datagram, saves 20B from the payload and ignores the rest. It then transforms 20 – 3 = 17B of text data and sends it back to the client. In our case, no data transformation has been done, so all 17B are sent back.

Trace 2 (**p5e6-7.pcapng**):

Exercise 6. Why is it possible to send a message when there is not a server running? Is an answer received? If the answer is yes, what is the meaning of this message.

UDP is connectionless protocol, client doesn’t have to establish any connection and can start sending datagrams rightaway.

In our case, the client receives ICMP port unreachble message, notifying it, that the destination port is closed and there isn’t running anything on it.

Exercise 7. If the execution of the client is interrupted, will a closing message be sent? Why?

Execution of the client is interrupted, because ICMP port unreachable exception is thrown. My client doesn’t send any more messages after that and exits. That’s how I programmed it and it doesn’t make sense to send any more messages when we know the server isn’t running. The alternative would be to try send datagrams repeatedly until server responds or it timeouts.

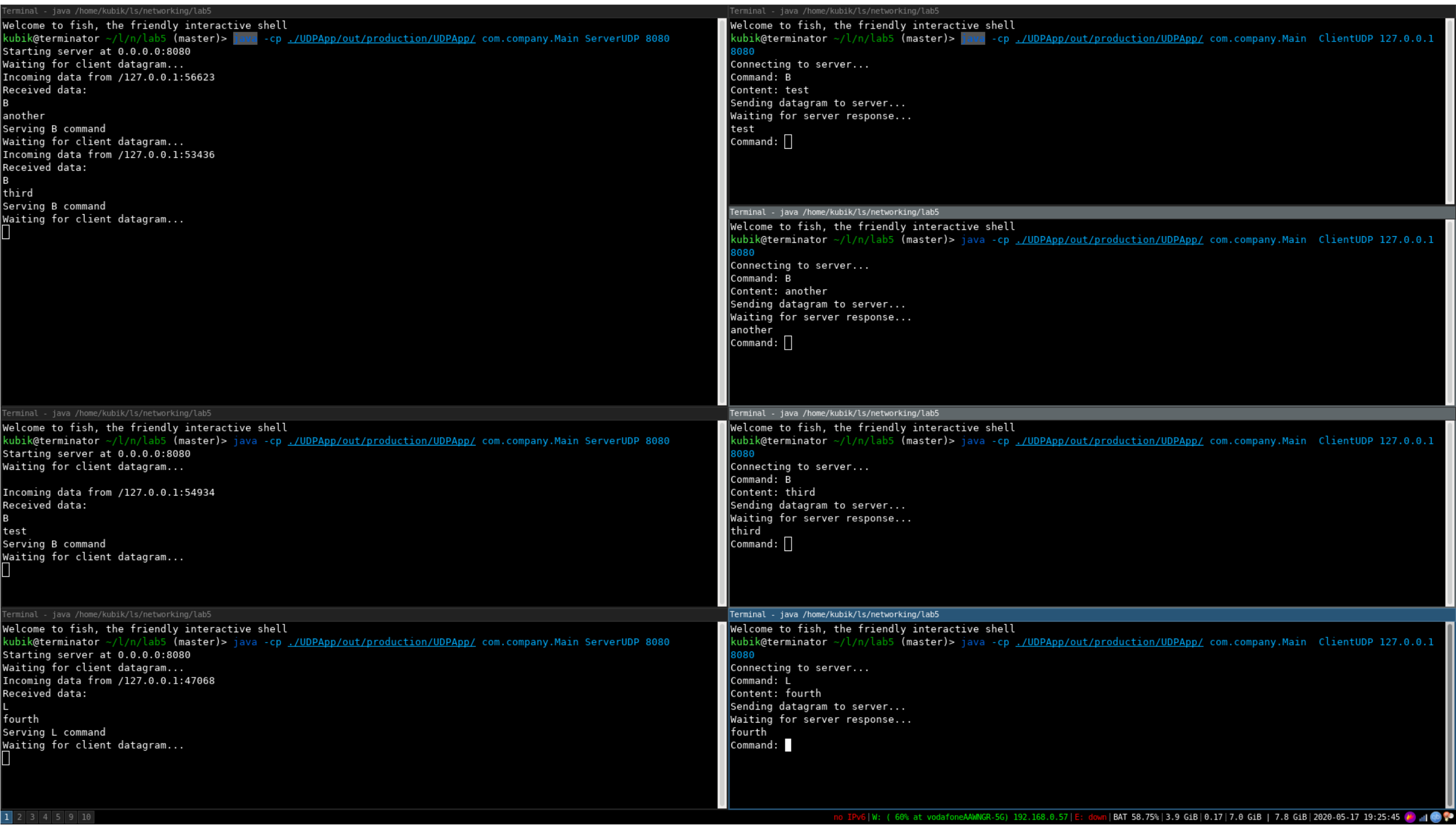
<u>No trace:</u>

Exercise 8. Try to open two times a server with the same input parameters and capture the Java exception thrown by the program. If the code was not prepared to capture that exception and show it (method *getMessage()*), modify the code for doing so. What kind of error is notified? How can you modify the server to avoid this error and to make it possible to have two servers with the same parameters running at the same time?

java.net.BindException: Address already in use (Bind failed) exception is thrown.

We can modify the code and set the kernel option SO_REUSEPORT (only since Linux kernel 3.9). Then the requests are randomly distributed between the running server instances.

(http://man7.org/linux/man-pages/man7/socket.7.html)



```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main ServerUDP 8080
Starting server at 0.0.0.0:8080
Waiting for client datagram...
Incoming data from /127.0.0.1:56623
Received data:
B
another
Serving B command
Waiting for client datagram...
Incoming data from /127.0.0.1:53436
Received data:
B
third
Serving B command
Waiting for client datagram...
```

```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main  ClientUDP 127.0.0.1
8080
Connecting to server...
Command: B
Content: test
Sending datagram to server...
Waiting for server response...
test
Command: 
```

```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main  ClientUDP 127.0.0.1
8080
Connecting to server...
Command: B
Content: another
Sending datagram to server...
Waiting for server response...
another
Command: 
```

```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main ServerUDP 8080
Starting server at 0.0.0.0:8080
Waiting for client datagram...

Incoming data from /127.0.0.1:54934
Received data:
B
test
Serving B command
Waiting for client datagram...
```

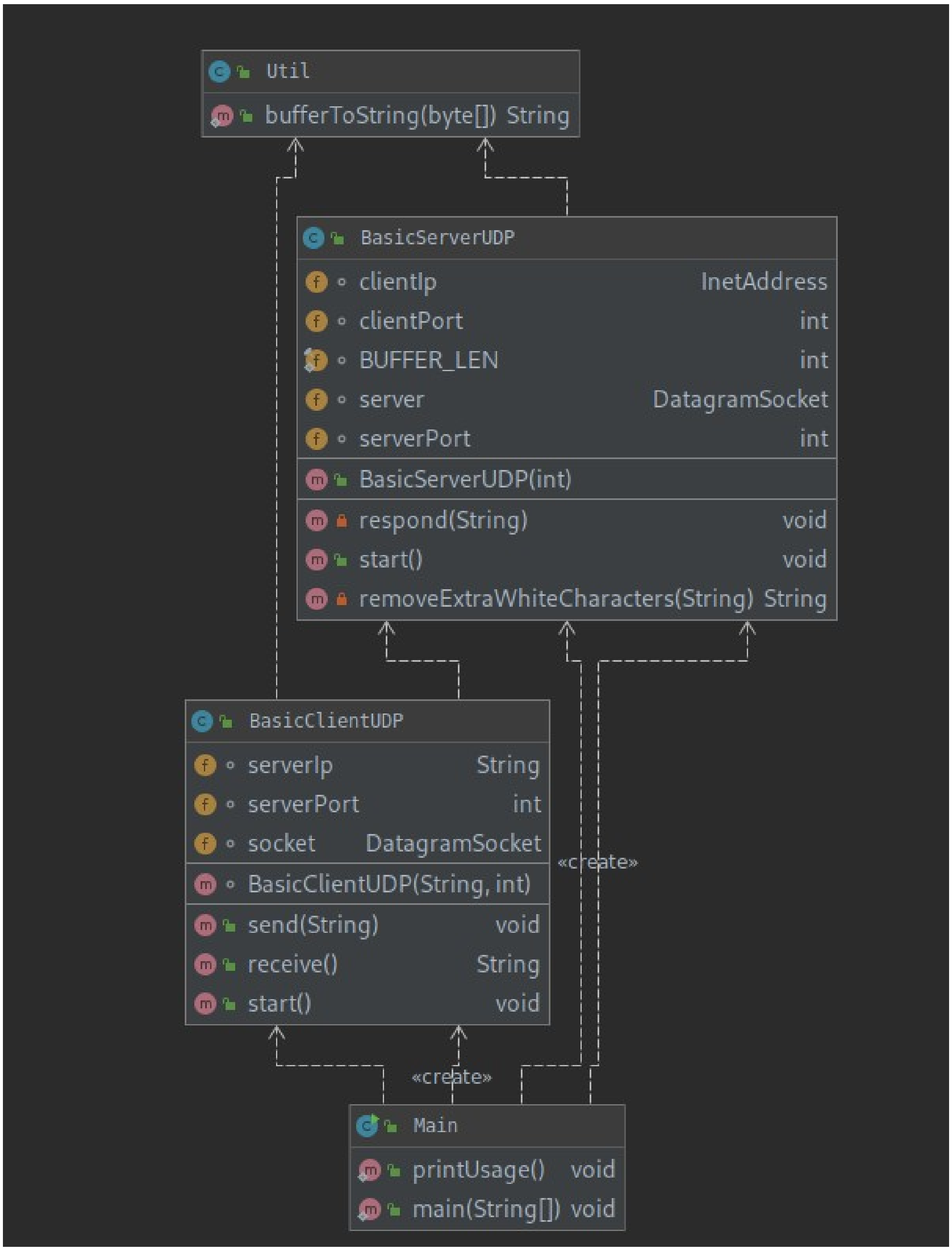
```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main  ClientUDP 127.0.0.1
8080
Connecting to server...
Command: B
Content: third
Sending datagram to server...
Waiting for server response...
third
Command: 
```

```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main ServerUDP 8080
Starting server at 0.0.0.0:8080
Waiting for client datagram...
Incoming data from /127.0.0.1:47068
Received data:
L
fourth
Serving L command
Waiting for client datagram...
```

```
Terminal - java /home/kubik/ls/networking/lab5
Welcome to fish, the friendly interactive shell
kubik@terminator ~/l/n/lab5 (master)> java -cp ./UDPApp/out/production/UDPApp/ com.company.Main  ClientUDP 127.0.0.1
8080
Connecting to server...
Command: L
Content: fourth
Sending datagram to server...
Waiting for server response...
fourth
Command: 
```

```
1 2 3 4 5 9 10
no IPv6|W: ( 60% at vodafoneAAWNGR-5G) 192.168.0.57|E: down|BAT 58.75%|3.9 GiB|0.17|7.0 GiB | 7.8 GiB|2020-05-17 19:25:45
```

Code UML


Util
bufferToString(byte[]) String
BasicServerUDP
clientIp InetAddress
clientPort int
BUFFER_LEN int
server DatagramSocket
serverPort int
BasicServerUDP(int)
respond(String) void
start() void
removeExtraWhiteCharacters(String) String
BasicClientUDP
serverIp String
serverPort int
socket DatagramSocket
BasicClientUDP(String, int)
send(String) void
receive() String
start() void
«create»
«create»
Main
printUsage() void
main(String[]) void