



Obligatorisk oppgave 1

DFON2110 Operativsystemer, nettverk og programmeringsspråk
Sirajuddin Asjad (216988)

Innholdsfortegnelse

1	Introduksjon	2
1.1	Oppgavebeskrivelse	2
1.2	Valg av operativsystem og språk	2
2	Implementasjon	3
2.1	Beskrivelse av kode	3
2.2	Demonstrasjon	5
3	Oppsummering	6
3.1	Konklusjon	6
3.2	Referanser	6
3.3	Kildekode	7

1 Introduksjon

1.1 Oppgavebeskrivelse

Denne oppgaven går ut på å lage en TCP klient som skal kommunisere med en TCP server på en gitt IP adresse og port. Kommunikasjonen foregår ved bruk av en spesifikk protokoll med noen spesifikke krav som må oppfylles, slik at bruker av klienten får kommunisert riktig med serveren. Dersom man har koblet seg til serveren og oppfyller alle krav så får man oppgaveteksten til neste obligatoriske innlevering i retur.

1.2 Valg av operativsystem og språk

TCP klienten har blitt programmert og utviklet i Java og platformen som har blitt brukt er Linux. Grunnen til at utvikler har valgt å kode klienten i Java er rett og slett fordi ett av kravene i denne oppgaven er at man ikke får lov til å bruke samme utviklerspråk på neste obligatoriske oppgave. Siden utvikleren behersker C++ mye bedre enn Java, så har denne oppgaven blitt kodet i Java slik at neste oppgave kan kodes i C++. Ellers er Java et greit språk for å utvikle en TCP klient, fordi språket er veldig stort og det finnes mye dokumentasjon på nettet om hvordan man enkelt kan kode en klient.

2 Implementasjon

2.1 Beskrivelse av kode

Koden starter selvfølgelig med å importere de nødvendige pakkene som trengs for å bruke spesielle funksjoner til å blant annet opprette en socket og kommunisere med serveren. Denne klienten bruker disse 7 pakkene:

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintStream;
import java.net.Socket;
import java.net.InetAddress;
import java.util.Scanner;
```

Det finnes en alternativ måte å importere pakker på, hvor man skriver *import java.io.** for å importere alle klassene som ligger inne i pakken. Denne metoden har både noen fordeler og ulemper, hvor en fordel er at man for eksempel sparer noen linjer i koden og man slipper å huske de forskjellige pakkenavnene. Ulempen er at koden bruker lengre tid under kompilering og bruker generelt mer plass. Med andre ord er det larest å kun importere de pakkene man trenger.

Alle funksjonene i TCP klienten er lagt inn i en klasse kalt *TCPClient*, hvor alle private variabler ligger øverst i klassen. Den første funksjonen som kjøres er da selvfølgelig *void main(...)*:

```
public class TCPClient{
    private Socket sock;
    private String ip;
    private int port;
    private char bufferIn[] = new char[512];
    private char bufferOut[] = new char[10];

    public static void main(String args[]) throws Exception{
        try{
            TCPClient client = new TCPClient();
            client.askUser();
            client.createSocket();
            client.sendData();
            client.readData();
        }
    }
}
```

Dersom noen av disse funksjonene feiler og ikke klarer å kjøre så vil funksjonen kaste en exception, altså en feilmelding som tas imot nederst i koden. Den ser slik ut:

```
catch(Exception e){
    System.out.println("ERROR: Something went wrong!");
}
```

Det er totalt fem funksjoner i TCP klienten, hvor disse har hver sin oppgave. Før noen funksjoner kjøres så opprettes det en instans av *TCPClient* klassen, og da allokeres det plass i minnet som er reservert til det objektet som nettopp ble opprettet. Vi kan deretter kjøre de forskjellige funksjonene som ligger i objektet:

```
TCPClient client = new TCPClient();
client.askUser();
client.createSocket();
client.sendData();
client.readData();
```

Den første funksjonen som kjøres er *askUser()*. Denne funksjonen spør brukeren om å skrive inn serverens IP adresse og port nummer. Dersom IP og port er gyldig så får de private variablene verdien av disse. Hvis noen av operasjonene innenfor denne funksjonen feiler, så vil funksjonen kaste en exception, altså en feilmelding som vises før programmet avsluttes.

```
private void askUser() throws Exception{
    System.out.print("Enter the server IP address: ");
    Scanner input = new Scanner(System.in);
    String tmpIP = input.nextLine();
    if(tmpIP == "lekeplass") ip = "158.36.70.56"; else ip = tmpIP;

    System.out.print("Enter the server port: ");
    int tmpPort = input.nextInt();
    if(tmpPort <= 0 || tmpPort > 99999) System.out.println("ERROR: Invalid port!");
    else port = tmpPort;
}
```

Etter at IP adressen og port er satt så kjøres neste funksjon, altså *createSocket()*. Denne funksjonen oppretter en socket mellom klient og server ved bruk av gitt IP adresse og port. Dersom klienten får kontakt med serveren, så vil den forsøke å koble seg til. Dette gjøres ved hjelp av klassen *Socket* som ligger inne i pakken *java.net.Socket*, som vi inkluderte i begynnelsen av programmet.

```
private void createSocket() throws Exception{
    InetAddress address = InetAddress.getByName(ip);
    sock = new Socket(address, port);
}
```

Nå er det opprettet en socket mellom klient og server, og klienten kan nå kommunisere med serveren ved å sende et studentnummer. Serveren er satt opp med en spesifikk protokoll som kun godtar en liste med 10 bytes. Slik beskrevet i oppgaven så trenger man en melding ID, størrelse på meldingen, størrelsen på studentnummer og i tillegg sende studentnummer til serveren, og deretter vil serveren sjekke om klienten ber om riktig kommando, eventuelt om det er oppgitt feil studentnummer.

I *sendData()* funksjonen blir det spurt om å oppgi studentnummer, og deretter blir melding ID, melding størrelse og studentnummer størrelse satt inn i et array bestående av characters. Melding ID er en *char*, altså 1 byte, det samme gjelder størrelsen av studentnummer, mens størrelsen av meldingen er en *short*, altså 2 bytes. Sammen med studentnummer som består av 6 characters, så har vi en liste med tilsammen 10 bytes. Denne lista sendes videre til serveren ved hjelp av klassen *PrintStream*, som tar dataen som ligger inne i arrayet og overfører dette videre til serveren.

```
private void sendData() throws Exception{
    char msg_id = 0x01;
    short msg_size = 7;
    char studnr_size = 6;

    bufferOut[0] = msg_id; // Message ID
    bufferOut[1] = 0; // Size of stream (Big endian)
    bufferOut[2] = (char)msg_size; // Size of stream (Big endian)
    bufferOut[3] = studnr_size; // Size of studentnr

    System.out.print("Enter your student number: ");
    Scanner input = new Scanner(System.in);
    String studnr = input.next();
    for(int i = 0; i < studnr.length(); i++) bufferOut[i+4] = studnr.charAt(i);

    PrintStream ps = new PrintStream(sock.getOutputStream());
```

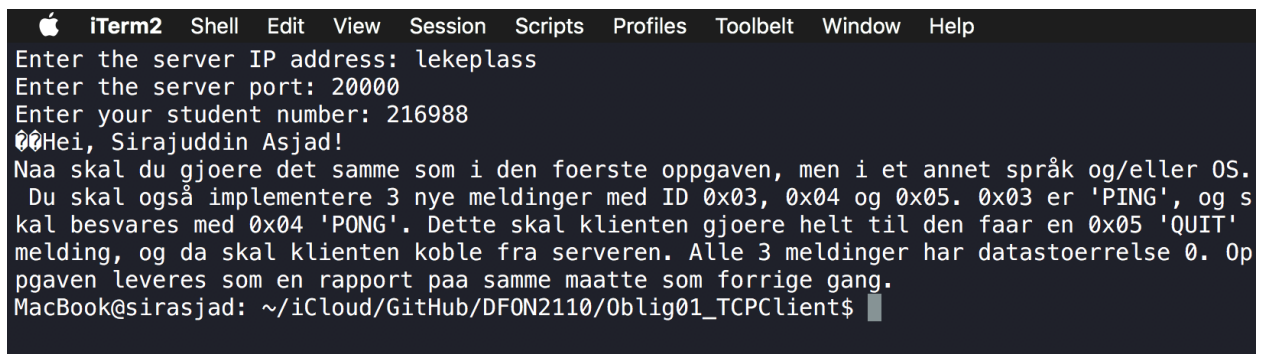
```
    ps.println(bufferOut);
}
```

Siste funksjon er å motta data fra serveren, altså motta en tilbakemelding på om vi har brukt riktig kommando, eller om vi har oppgitt riktig studentnummer. Dersom alle kravene er oppfylt og vi har sendt inn riktig data, så vil serveren returnere oppgaveteksten til obligatorisk innlevering 2. Denne teksten leses i form av characters, som oppbevares i et buffer som har nok plass å oppbevare 512 characters. Når oppgaveteksten er mottatt, så printes den ut og deretter avsluttes programmet.

```
private void readData() throws Exception{
    InputStreamReader isr = new InputStreamReader(sock.getInputStream());
    BufferedReader br = new BufferedReader(isr);
    br.read(bufferIn, 0, bufferIn.length);
    System.out.print(bufferIn);
}
```

2.2 Demonstrasjon

I figuren under har vi koblet til serveren med riktig IP adresse og port, og deretter oppgitt riktig studentnummer. Serveren vil da altså returnere oppgaveteksten til oblig 2.



```

iTerm2  Shell  Edit  View  Session  Scripts  Profiles  Toolbelt  Window  Help
Enter the server IP address: lekeklass
Enter the server port: 20000
Enter your student number: 216988
Hei, Sirajuddin Asjad!
Naa skal du gjoere det samme som i den foerste oppgaven, men i et annet språk og/eller OS.
Du skal også implementere 3 nye meldinger med ID 0x03, 0x04 og 0x05. 0x03 er 'PING', og s
kal besvares med 0x04 'PONG'. Dette skal klienten gjoere helt til den faar en 0x05 'QUIT'
melding, og da skal klienten koble fra serveren. Alle 3 meldinger har datastoerrelse 0. Op
pgaven leveres som en rapport paa samme maatte som forrige gang.
MacBook@sirasjad: ~/iCloud/GitHub/DF0N2110/Oblig01_TCPClient$
```

3 Oppsummering

3.1 Konklusjon

Denne TCP klienten fungerer fint mot serveren til Christian og besvarer på oppgaven og dens protokoll. Denne oppgaven har vært ekstremt lærerikt, og det har vært en veldig gøy opplevelse å kode en slik klient, med tanke på at fagstoffet er nytt og ukjent. I tillegg så er det øving i rapportskriving og LaTeX, og det er jo veldig gøy. Det var kanskje litt vanskelig å forstå oppgaven riktig i begynnelsen, men det løste seg når Christian forklarte oppgaven grundig. Det var ikke vanskelig å kode klienten, selv om det har oppstått noen små problemer underveis. Heldigvis finnes det veldig mye bra dokumentasjon og forklaring på nett, så da løste det meste seg etter ett par søk på Google. Ellers er dette en perfekt oppgave og det har vært en god opplevelse å utvikle noe slikt.

3.2 Referanser

- [1] Christian Scott, Universitetet i Sørøst-Norge
Avdelingsingeniør, lærer og guru ved USN Kongsberg
- [2] TCP/IP Sockets in C: Practical Guide for Programmers
Elsevier Science, 2009. Skrevet av Michael Donahoo, Kenneth Calvert
- [3] Java Tutorials: All About Sockets
<https://docs.oracle.com/javase/tutorial/networking/sockets/index.html>

3.3 Kildekode

```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.PrintStream;
import java.net.Socket;
import java.net.InetAddress;
import java.util.Scanner;

public class TCPClient{
    private Socket sock;
    private String ip;
    private int port;
    private char bufferIn[] = new char[512];
    private char bufferOut[] = new char[10];

    public static void main(String args[]) throws Exception{
        try{
            TCPClient client = new TCPClient();
            client.askUser();
            client.createSocket();
            client.sendData();
            client.readData();
        }
        catch(Exception e){
            System.out.println("ERROR: Something went wrong!");
        }
    }

    private void askUser() throws Exception{
        System.out.print("Enter the server IP address: ");
        Scanner input = new Scanner(System.in);
        String tmpIP = input.nextLine();
        if(tmpIP == "lekeplass") ip = "158.36.70.56"; else ip = tmpIP;

        System.out.print("Enter the server port: ");
        int tmpPort = input.nextInt();
        if(tmpPort <= 0 || tmpPort > 99999) System.out.println("ERROR: Invalid port!");
        else port = tmpPort;
    }

    private void createSocket() throws Exception{
        InetAddress address = InetAddress.getByName(ip);
        sock = new Socket(address, port);
    }
}
```



```

private void sendData() throws Exception{
    char msg_id = 0x01;
    short msg_size = 7;
    char studnr_size = 6;

    bufferOut[0] = msg_id; // Message ID
    bufferOut[1] = 0; // Size of stream (Big endian)
    bufferOut[2] = (char)msg_size; // Size of stream (Big endian)
    bufferOut[3] = studnr_size; // Size of studentnr

    System.out.print("Enter your student number: ");
    Scanner input = new Scanner(System.in);
    String studnr = input.next();
    for(int i = 0; i < studnr.length(); i++) bufferOut[i+4] = studnr.charAt(i);

    PrintStream ps = new PrintStream(sock.getOutputStream());
    ps.println(bufferOut);
}

private void readData() throws Exception{
    InputStreamReader isr = new InputStreamReader(sock.getInputStream());
    BufferedReader br = new BufferedReader(isr);
    br.read(bufferIn, 0, bufferIn.length);
    System.out.print(bufferIn);
}
}

```