

# LAB 2 (Network Programming with Java)

Fatema Mirza & Mohammad Newaj Jamil

D7001D Network Programming and Distributed Applications





# LAB 2 (Network Programming with Java)

by

Fatema Mirza & Mohammad Newaj  
Jamil

Submission Deadline: November 5, 2021

Teachers:

Dr. Evgeny Osipov,  
Ahmed Afif Monrat,

LULEÅ TEKNISKA UNIVERSITET  
LULEÅ TEKNISKA UNIVERSITET



# Contents

1	Part I: Java GUI (To be performed either on a local machine or on an AWS instance)	1
2	Part II –Java netprog patterns –sockets & threads	4
2.1	Compile and debug the Lookup, TCPEchoServer and UDPEchoServer classes. . . . .	4
2.2	Be able to describe the details of the implementation of each class (to be tested during individual assessment by Evgeny). . . . .	6
2.3	Modify the Lookup class so, that it outputs your name in addition to the input parameters that you supply at runtime. . . . .	6
2.4	Modify both the TCPEchoServer and UDPEchoServer classes so that in addition to echoed input symbols the server would send back your name. . . . .	6
2.5	Compile the class Race0 in the threads part of the project. . . . .	7
3	Part III: Client-server application	10
3.1	The GUI runs as a client-side application. Add an additional field to your GUI where the user can enter the URL of the server-side application . . . . .	10
3.2	The entered commands should be executed and run as the server-side application. You can implement this assignments in either of the following two ways: . . . . .	12
3.3	Add logging capability to the server-side application . . . . .	13
3.4	The application should take an argument defining which level of debug should be logged (warning, info, error, debug) . . . . .	14
3.5	When adding different log levels to functions you have to explain why you add this level of logging to the specific function. Write this explanation as a comment in your source code . . . .	15
4	Part IV –Java netprog patterns –Simple Messaging Architecture	16
4.1	Compile and install the classes in the SMA module of the project. . . . .	16
4.2	Demonstrate the functioning classes during the lab assessment. Use Wireshark to capture the traffic of the SMA session. You should be able to explain the details of the implementation during the individual assessment.. . . .	16
4.3	Extend the architecture adding a new kind of service of your choice!For example a simplified DNS service resolving an IP address for a given host number. (This is only a suggestion though) .	17
5	Part V: Java netprog patterns – security	19
5.1	Compile and install (in the cloud) the classes in the security module of the project (JCE and JSSE) . . . . .	19
5.1.1	JCE: . . . . .	19
5.1.2	JSSE: . . . . .	22

# 1

## Part I: Java GUI (To be performed either on a local machine or on an AWS instance)

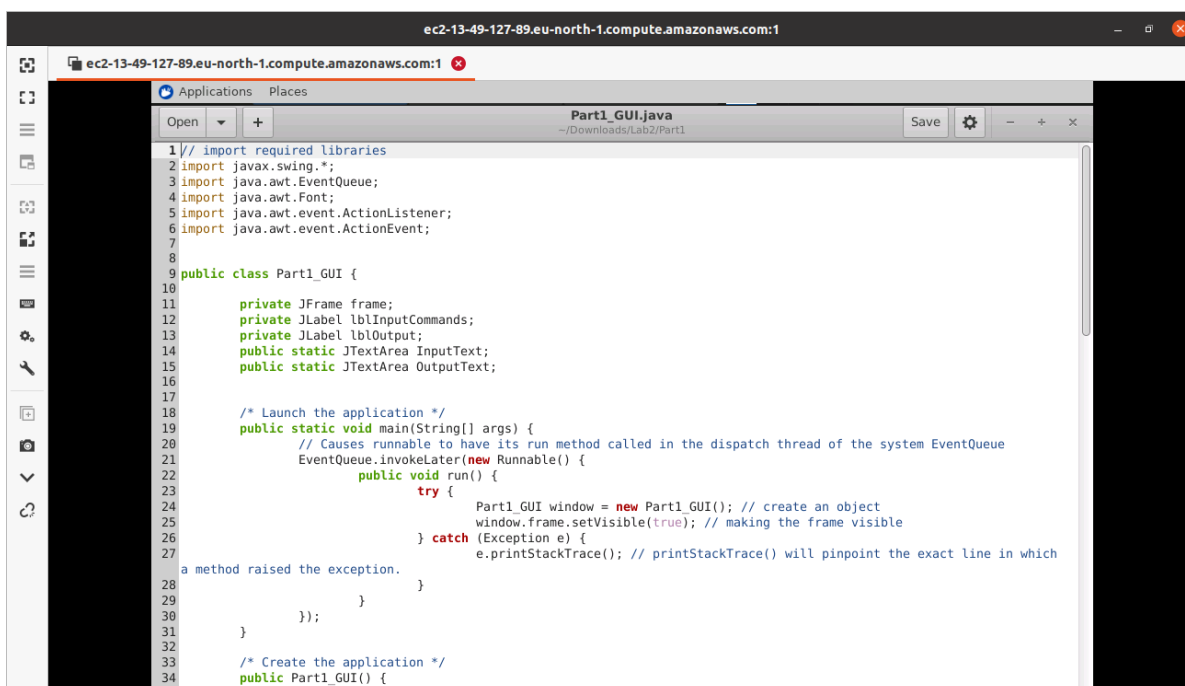
Write a simple java GUI with two fields and one button, where first field is used to input command and the second one is to display output. The application should be capable of executing the following Unix commands (or their counterparts in Windows):

- ls, pwd, uptime, cat file.txt, who, ps, grep, ifconfig
- It has to be possible to pipeline command. For example this command

– \$cd /home/ && pwd | ls -all

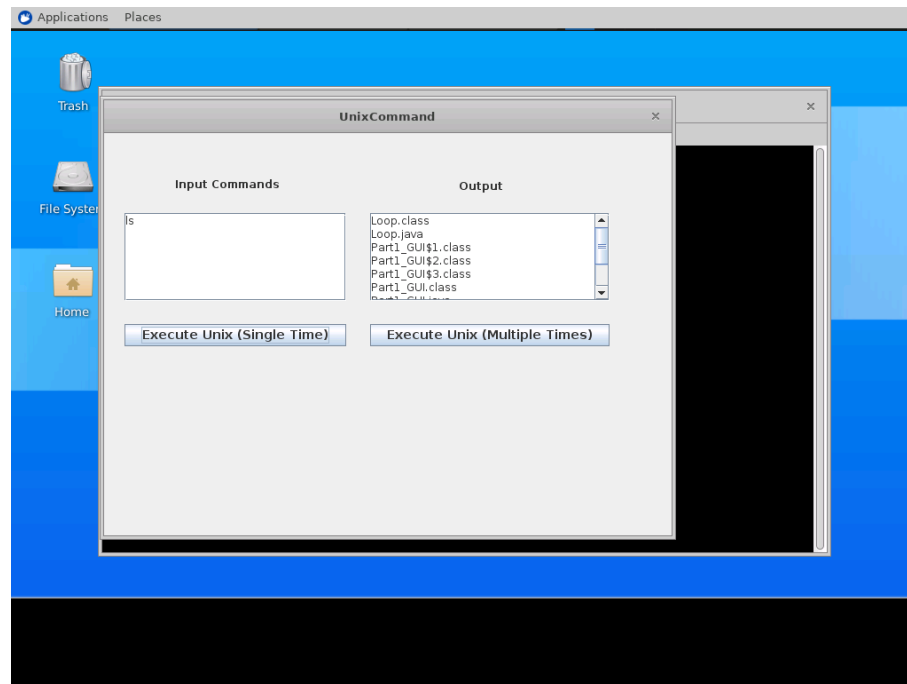
should return a detailed list of files and directories under /home/ folder.

The task has been performed on an AWS instance. A simple java GUI (Part1\_GUI.java) has been implemented with two fields and one button, where first field is used to input command and the second one is to display output, as shown below.

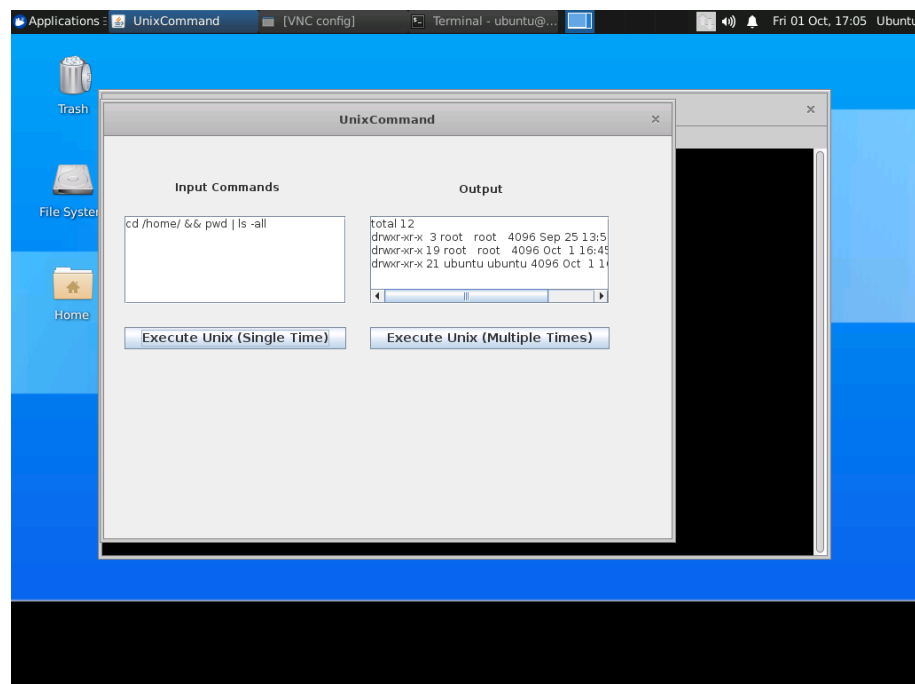


```
1 // import required libraries
2 import javax.swing.*;
3 import java.awt.EventQueue;
4 import java.awt.Font;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7
8
9 public class Part1_GUI {
10
11     private JFrame frame;
12     private JLabel lblInputCommands;
13     private JLabel lblOutput;
14     public static JTextArea InputText;
15     public static JTextArea OutputText;
16
17
18     /* Launch the application */
19     public static void main(String[] args) {
20         // Causes runnable to have its run method called in the dispatch thread of the system EventQueue
21         EventQueue.invokeLater(new Runnable() {
22             public void run() {
23                 try {
24                     Part1_GUI window = new Part1_GUI(); // create an object
25                     window.frame.setVisible(true); // making the frame visible
26                 } catch (Exception e) {
27                     e.printStackTrace(); // printStackTrace() will pinpoint the exact line in which
28                     a method raised the exception.
29                 }
30             }
31         });
32     }
33
34     /* Create the application */
35     public Part1_GUI() {
36         initComponents();
37     }
38
39     private void initComponents() {
40         // TODO add your initialization code here
41     }
42
43     private void btnExecuteActionPerformed(java.awt.event.ActionEvent evt) {
44         // TODO add your handling code here:
45         String command = InputText.getText();
46         OutputText.setText("");
47         try {
48             Process p = Runtime.getRuntime().exec(command);
49             BufferedReader reader = new BufferedReader(new InputStreamReader(p.getInputStream()));
50             String line;
51             while ((line = reader.readLine()) != null) {
52                 OutputText.append(line + "\n");
53             }
54         } catch (IOException e) {
55             e.printStackTrace();
56         }
57     }
58 }
```

The application is capable of executing the Unix commands mentioned in the question. For example, if a user wants to see the list of files in the current working directory, he can write the command "ls" in the input field and click on the "Execute Unix (Single Time)" button. Then the GUI will show the output as shown below.

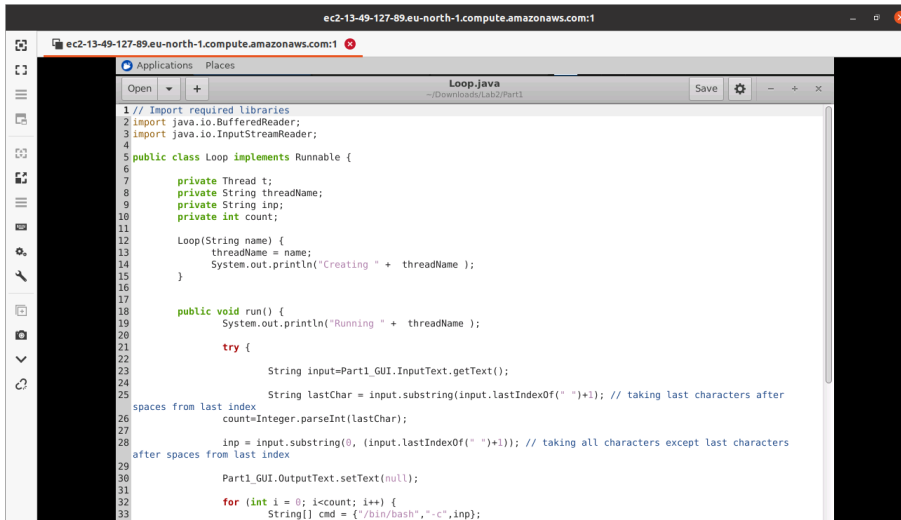


It is possible to pipeline command. For example, the command `$cd /home/ && pwd | ls -all` will return a detailed list of files and directories under /home/ folder as shown below.



**Now write another Java class, which will executes an empty loop. The program should take the number of loop iterations as a parameter, print this in the output field, launch the loop. Launch this program from the GUI!**

Another Java class (Loop.java) has been created; as shown below.

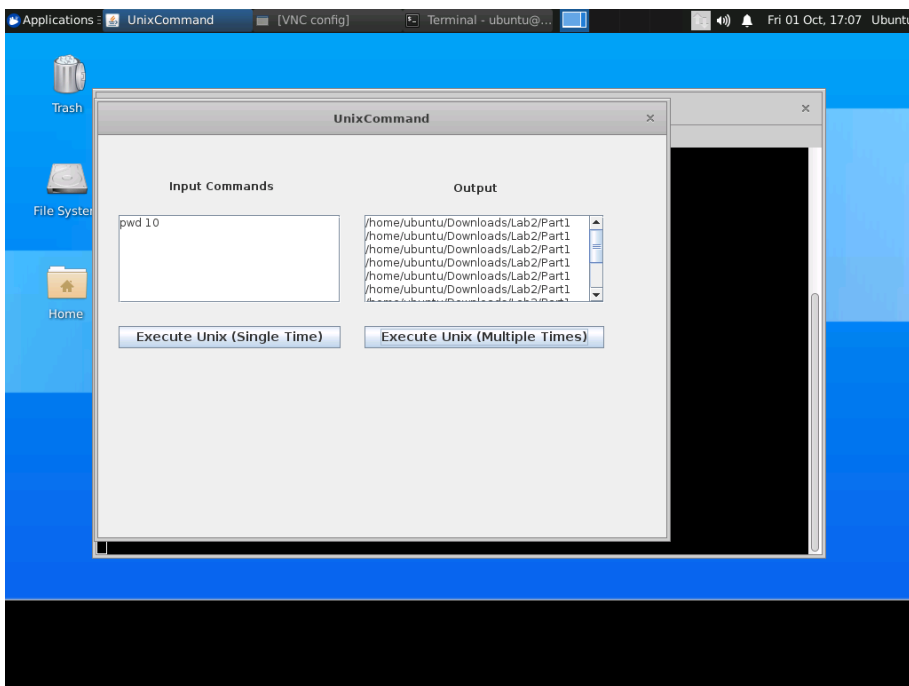


```

1 // Import required Libraries
2 import java.io.BufferedReader;
3 import java.io.InputStreamReader;
4
5 public class Loop implements Runnable {
6
7     private Thread t;
8     private String threadName;
9     private String inp;
10    private int count;
11
12    Loop(String name) {
13        threadName = name;
14        System.out.println("Creating " + threadName );
15    }
16
17    public void run() {
18        System.out.println("Running " + threadName );
19
20        try {
21
22            String input=Part1_GUI.InputText.getText();
23
24            String lastChar = input.substring(input.lastIndexOf(" ") +1); // taking last characters after
25            spaces from last index
26            count=Integer.parseInt(lastChar);
27
28            inp = input.substring(0, (input.lastIndexOf(" ") +1)); // taking all characters except last characters
29            after spaces from last index
30
31            Part1_GUI.OutputText.setText(null);
32
33            for (int i = 0; i<count; i++) {
34                String[] cmd = {"/bin/bash","-c",inp};

```

In the GUI, a user can specify how many times he wants to see the output. For example, if a user wants to see the name of the current working directory 10 times, he can write the command “pwd 10” in the input field and click on the “Execute Unix (Multiple Times)” button. Then the GUI will show the output 10 times, as shown below.



The GUI and the command executions are implemented in separate classes. Threads have been utilized to implement the command execution.

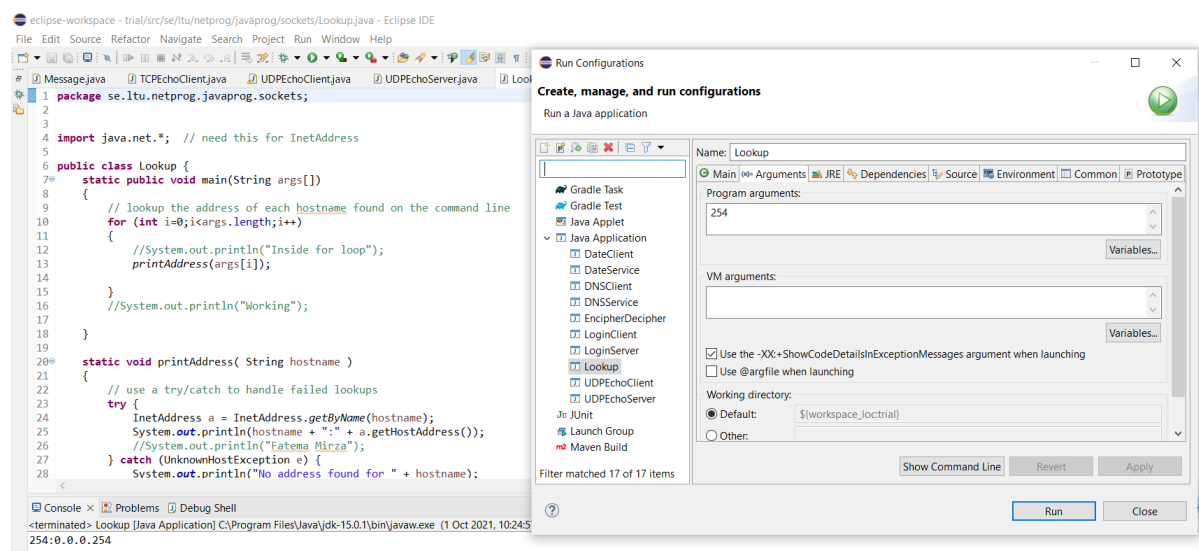
# 2

## Part II –Java netprog patterns –sockets & threads

Copy CodeExamples\_JavaProgrammingModule.zip file from Fronter to your home directory either on your local computer or on your AWS instance.Import the source files into your favorite IDE, for example Eclipse.

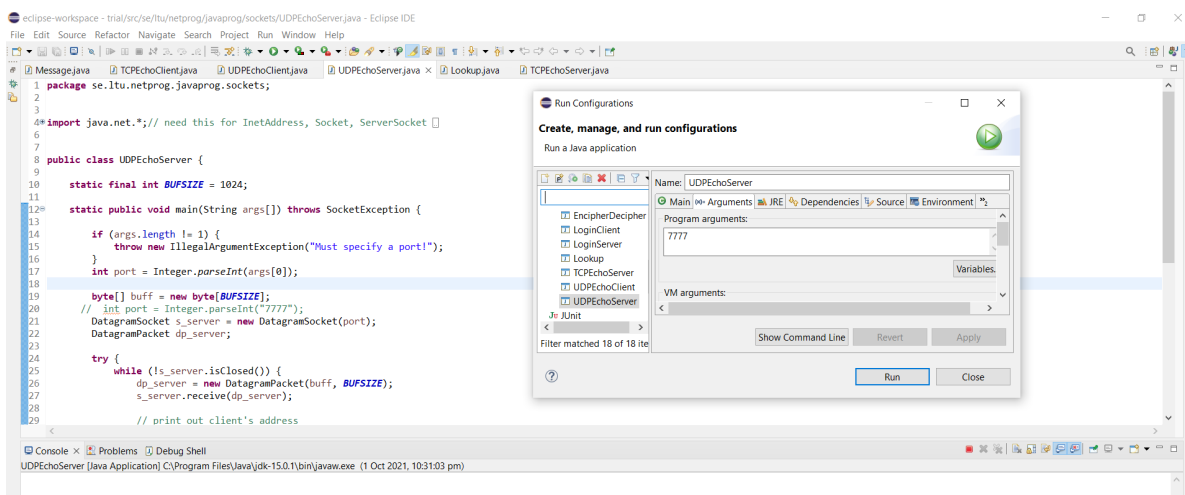
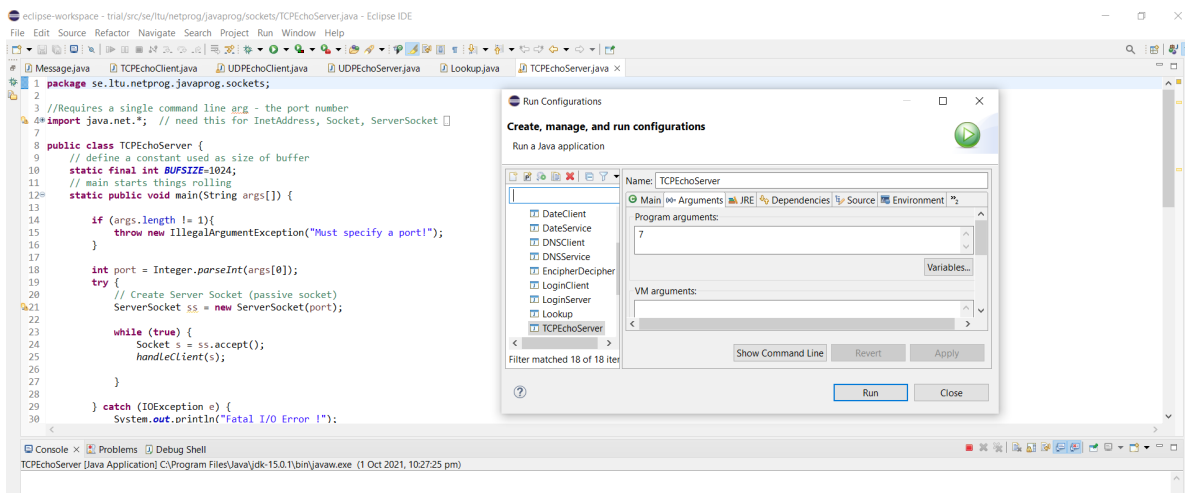
### 2.1. Compile and debug the Lookup, TCPEchoServer and UDPEchoServer classes.

It can be observed that the Lookup classes displays the IP address of a host given at execution time.

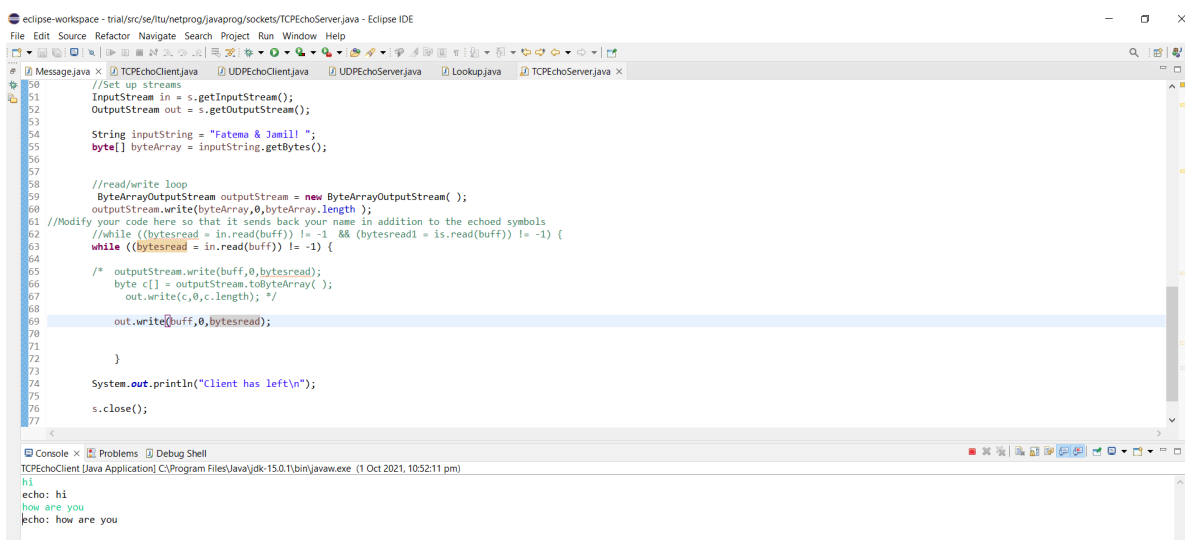


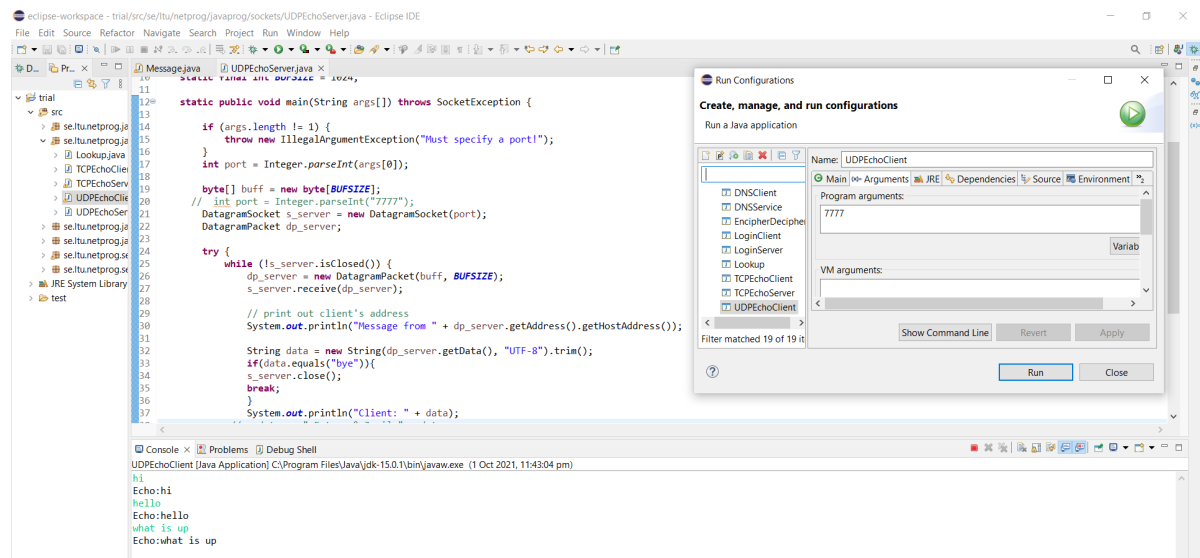
It can be observed that while both the TCPEchoServer and UDPEchoServer classes are working perfectly, that is not showing any error, it is not being able to echo anything as of yet.





This is because `TCPEchoServer` and `UDPEchoServer` does not have its corresponding `TCPEchoClient` and `UDPEchoClient` classes. After the client classes are written, the following can be observed:



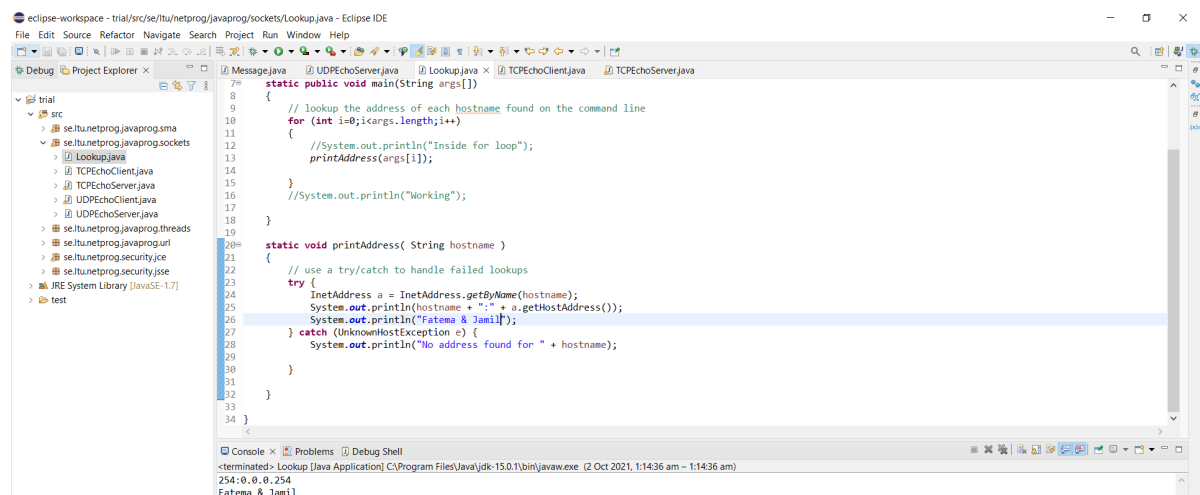


## 2.2. Be able to describe the details of the implementation of each class (to be tested during individual assessment by Evgeny).

Source code has been provided.

## 2.3. Modify the Lookup class so, that it outputs your name in addition to the input parameters that you supply at runtime

The edited part of the code as well as the output has been displayed below.



## 2.4. Modify both the TCPEchoServer and UDPEchoServer classes so that in addition to echoed input symbols the server would send back your name.

The edited part of the code as well as the output has been displayed below.

```

eclipse-workspace - trial/se/ttu/netprog/javaprog/sockets/TCPEchoServer.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
TCPEchoServer.java
46 int bytesRead = 0;
47 //print out client's address
48 System.out.println("Connection from " + s.getInetAddress().getHostAddress());
49
50 //Set up streams, printwriter and bufferedreaders
51 InputStream in = s.getInputStream();
52 OutputStream out = s.getOutputStream();
53 PrintWriter writer = new PrintWriter(out, true);
54
55 BufferedReader reader = new BufferedReader(new InputStreamReader(in));
56
57 //read/write loop
58
59 //Modify your code here so that it sends back your name in addition to the echoed symbols
60 //get the data from the client side
61 String data;
62
63 do {
64     data = reader.readLine();
65     data = " Fatema & Jamil " + data;
66
67     //send data back to client after adding our name
68     writer.println("Server: " + data);
69 } while(!data.equals("quit"));
70
71 System.out.println("Client has left\n");
72
73 s.close();
74
Console x Problems Debug Shell
TCPEchoClient [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (2 Oct 2021, 1:23:26 am)
hi
echo: Server: Fatema & Jamil hi
hello
echo: Server: Fatema & Jamil hello
how are you
echo: Server: Fatema & Jamil how are you

```

```

eclipse-workspace - trial/se/ttu/netprog/javaprog/sockets/UDPEchoServer.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
UDPEchoServer.java
26 //recieve packet from the client
27 dp_server = new DatagramPacket(buff, BUFSIZE);
28 s_server.receive(dp_server);
29
30 // print out client's address
31 System.out.println("Message from " + dp_server.getAddress().getHostAddress());
32
33 //convert packet data to string
34 String data = new String(dp_server.getData(), "UTF-8").trim();
35
36 if(data.equals("bye")){
37     s_server.close();
38     break;
39 }
40
41
42 // Add name to data packets
43 data = " Fatema & Jamil " + data;
44
45 //make datagram packet to send back to the client
46 dp_server = new DatagramPacket(data.getBytes(), data.getBytes().length, dp_server.getAddress(), dp_server.getPort());
47
48 //send to th client
49 s_server.send(dp_server);
50
51 buff = new byte[BUFSIZE];
52
53 }
54
Console x Problems Debug Shell
UDPEchoClient [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (2 Oct 2021, 1:25:56 am)
hi
Echo: Fatema & Jamil hi
hello
Echo: Fatema & Jamil hello
how are you
Echo: Fatema & Jamil how are you

```

## 2.5. Compile the class Race0 in the threads part of the project.

1. What kind of behavior do you observe?
2. Make now both dif() and bump() methods synchronized. Compile the classes and run.
3. How the behavior is changed now?

1. The behaviour that is displayed is called **Race Condition** - a condition that arises when two or more threads attempt to access a shared resources simultaneously. There the shared resources, in this instance the integer values of x and y fall under race conditions. There is a change for x or y to be read by the thread calling dif() while executing the bump() call leading at times to a value of 1 ( prints an X) and rest of the times 0 ( prints an . ).

```

package se.ltu.netprog.javaprog.threads;

public class Race0 extends Thread {
    //static Shared0 s;
    static Shared0 s;

    static volatile boolean done=false;
    public static void main(String[] args){
        Thread t=new Race0();
        s=new Shared0();
        try{
            t.start();
            while(!done){
                s.bump();
                sleep(30);
            }
        }catch (InterruptedException e)
        {
            return;
        }
    }

    public void run(){
        int i;
        try{
            for(i=0;i<1000;i++){
                if(i%60==0)

```

Console output (2 Oct 2021, 1:42:35 am):

```

X...X.X.X...X...X.X.X...X.X.X.X...X.X...
X...X.X.X...X.X.X...X.X.X...X.X.X...X
X...X.X.X...X.X.X...X.X.X...X.X.X...X
X.X...X.X...X.X.X...X.X.X...X.X.X...X
X.X.X.X...

```

2. The following output is observed:

```

package se.ltu.netprog.javaprog.threads;

public class Race0 extends Thread {
    //static Shared0 s;
    static Shared0 s;

    static volatile boolean done=false;
    public static void main(String[] args){
        Thread t=new Race0();
        s=new Shared0();
        try{
            t.start();
            while(!done){
                s.bump();
                sleep(30);
            }
        }catch (InterruptedException e)
        {
            return;
        }
    }

    public void run(){
        int i;
        try{
            for(i=0;i<1000;i++){
                if(i%60==0)

```

Console output (2 Oct 2021, 1:42:29 am):

```

X...X.X.X...X...X.X.X...X.X.X.X...X.X...
X...X.X.X...X.X.X...X.X.X...X.X.X...X
X...X.X.X...X.X.X...X.X.X...X.X.X...X
X.X...X.X...X.X.X...X.X.X...X.X.X...X
X.X.X.X...

```

3. The Race Condition has been eliminated with use of the keyword synchronized. The shared resources are now locked when the methods bump() or dif() is executed which means that they will not be able to use the shared resources at the same time or until one method has stopped executing. This leads to the value of x and y being same at all times, and the method dif() always returning a 0 and hence always printing a . .

The screenshot shows the Eclipse IDE with a project named 'trial'. The 'src' folder contains several packages, including 'se.ltu.netprog.javaprog.threads'. The 'Race0.java' file is open, showing the following code:

```
13 while(!done){
14     s.bump();
15     s.sleep(30);
16 }
17 io.join();
18 }catch (InterruptedException e)
19 {
20     return;
21 }
22 }
23 }
24 public void run(){
25     int i;
26     try{
27         for(i=0;i<1000;i++){
28             if(i%60==0)
29                 System.out.println();
30             System.out.print("X".charAt(s.dif()));
31             sleep(20);
32         }
33         System.out.println();
34         done=true;
35     }catch (InterruptedException e) {return;}
36 }
37 }
38 }
39 }
40 }
```

The console output shows the application terminated:

```
<terminated> Race0 [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (2 Oct 2021, 2:14:10 am - 2:14:24 am)
```

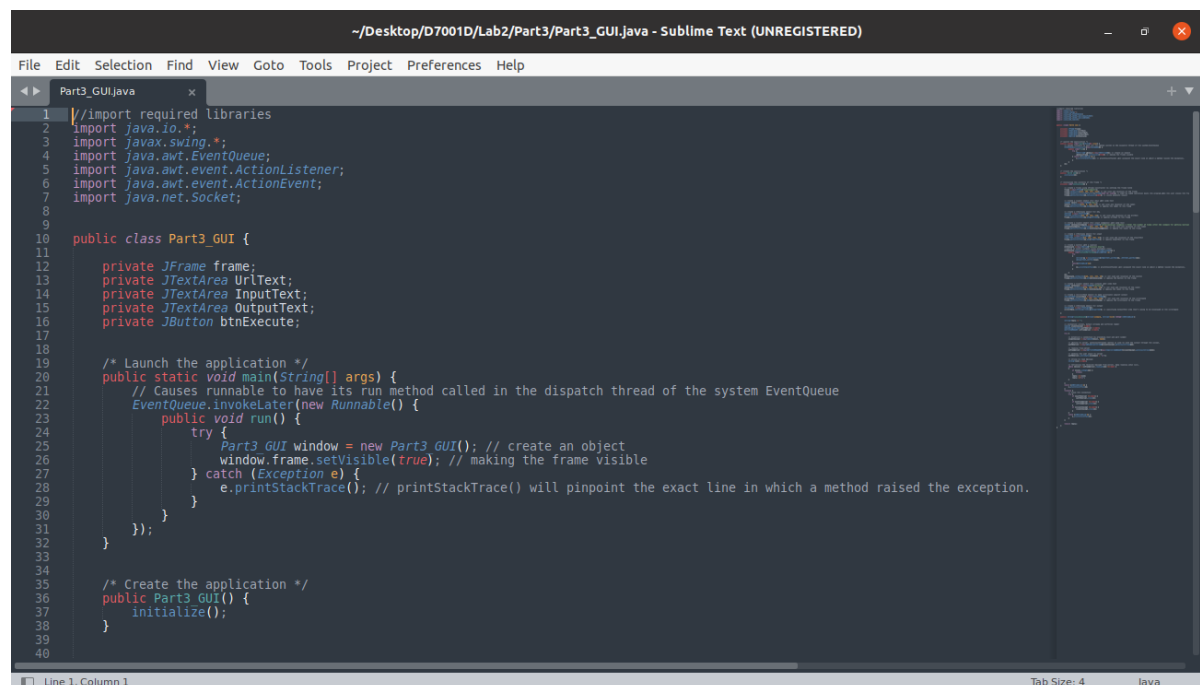
# 3

## Part III: Client-server application

Modify Part I of this Lab 2 so that:

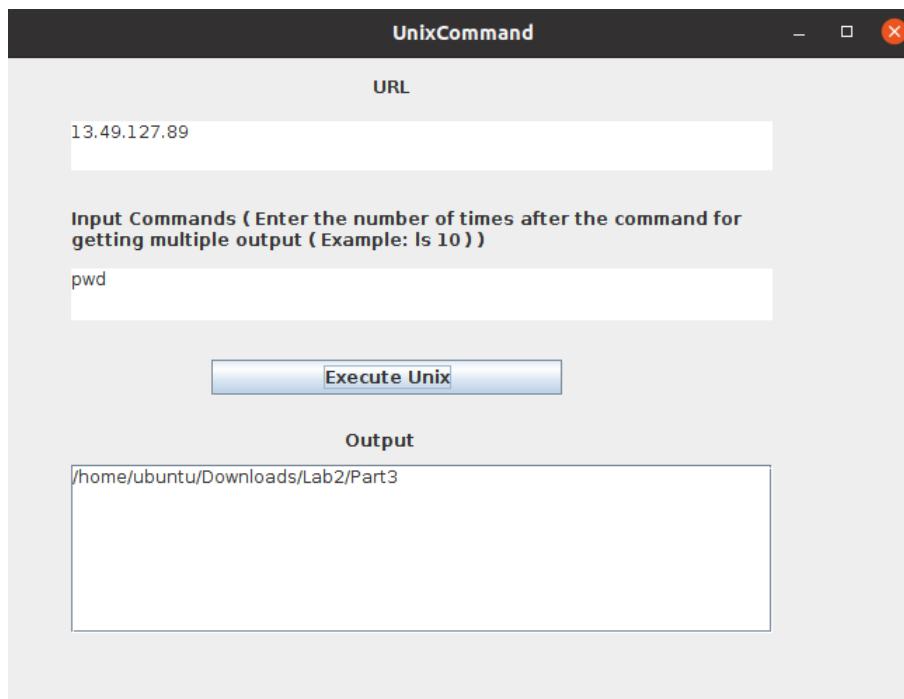
### 3.1. The GUI runs as a client-side application. Add an additional field to your GUI where the user can enter the URL of the server-side application

A GUI (Part3\_GUI.java) has been implemented on a local machine that runs as a client-side application, which is shown below.



```
1 //import required libraries
2 import java.io.*;
3 import javax.swing.*;
4 import java.awt.EventQueue;
5 import java.awt.event.ActionListener;
6 import java.awt.event.ActionEvent;
7 import java.net.Socket;
8
9
10 public class Part3_GUI {
11
12     private JFrame frame;
13     private JTextArea UrlText;
14     private JTextArea InputText;
15     private JTextArea OutputText;
16     private JButton btnExecute;
17
18
19     /* Launch the application */
20     public static void main(String[] args) {
21         // Causes runnable to have its run method called in the dispatch thread of the system EventQueue
22         EventQueue.invokeLater(new Runnable() {
23             public void run() {
24                 try {
25                     Part3_GUI window = new Part3_GUI(); // create an object
26                     window.frame.setVisible(true); // making the frame visible
27                 } catch (Exception e) {
28                     e.printStackTrace(); // printStackTrace() will pinpoint the exact line in which a method raised the exception.
29                 }
30             }
31         });
32     }
33
34     /* Create the application */
35     public Part3_GUI() {
36         initialize();
37     }
38
39
40 }
```

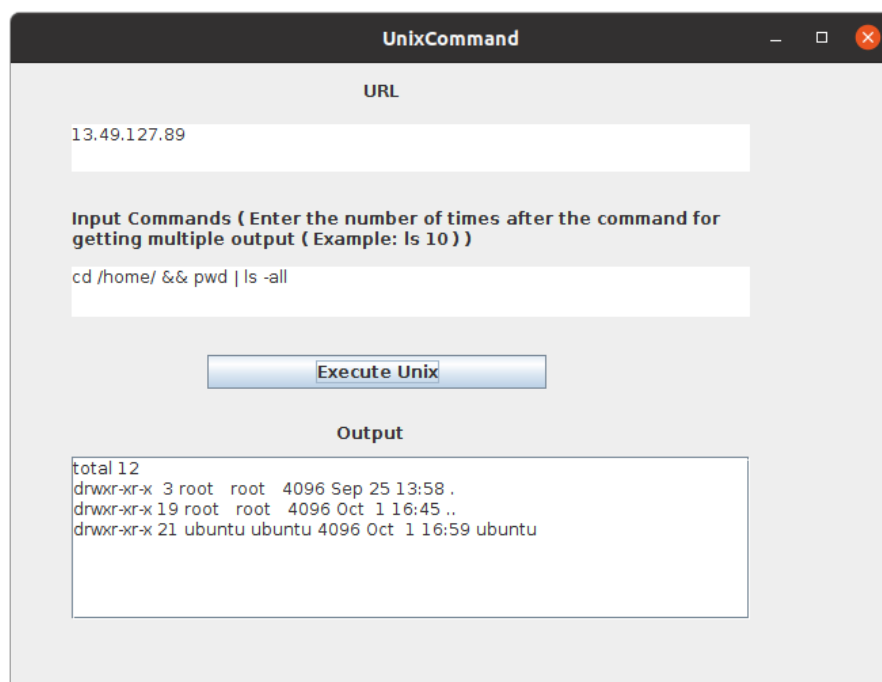
In the GUI, an additional field has been added where the user can enter the URL of the server-side applications. If a user wants to see the name of the current working directory, he can write the command “pwd” in the input field and click on the “Execute Unix” button. Then the GUI will show the output as shown below.



The screenshot shows a web application window titled "UnixCommand". It has a dark header bar with standard window controls. The main content area is light gray and contains the following elements:

- A label "URL" above a text input field containing "13.49.127.89".
- A label "Input Commands ( Enter the number of times after the command for getting multiple output ( Example: ls 10 ) )" above a text input field containing "pwd".
- A blue button labeled "Execute Unix" below the input fields.
- A label "Output" above a text area containing the output: "/home/ubuntu/Downloads/Lab2/Part3".

Likewise, the command `$cd /home/ && pwd | ls -all` will return a detailed list of files and directories under /home/ folder as shown below.

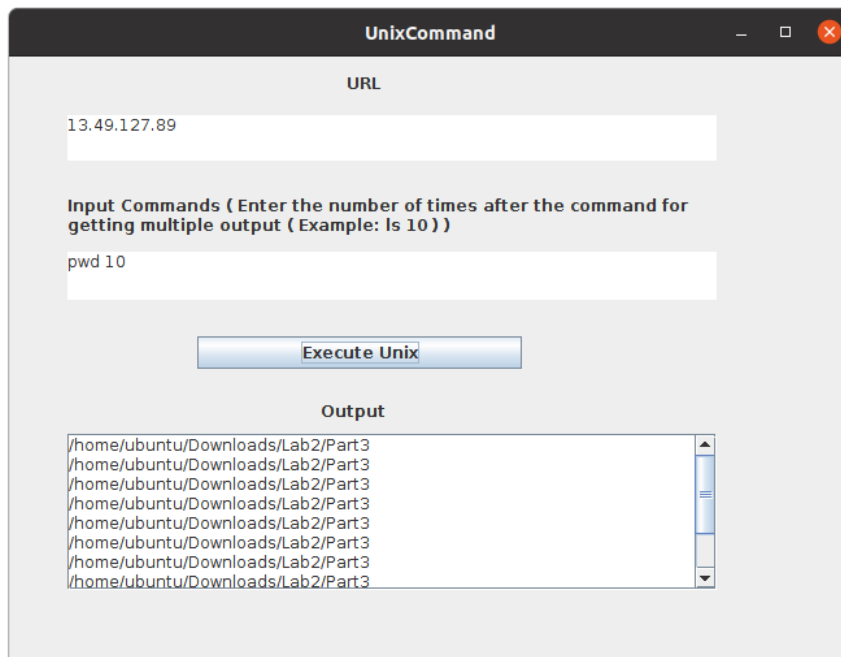


The screenshot shows the same "UnixCommand" web application window. The input fields and output are updated as follows:

- The "URL" field remains "13.49.127.89".
- The "Input Commands" field now contains the command: `cd /home/ && pwd | ls -all`.
- The "Execute Unix" button is still present.
- The "Output" text area now displays the output of the command: 

```
total 12
drwxr-xr-x 3 root  root  4096 Sep 25 13:58 .
drwxr-xr-x 19 root  root  4096 Oct 1 16:45 ..
drwxr-xr-x 21 ubuntu ubuntu 4096 Oct 1 16:59 ubuntu
```

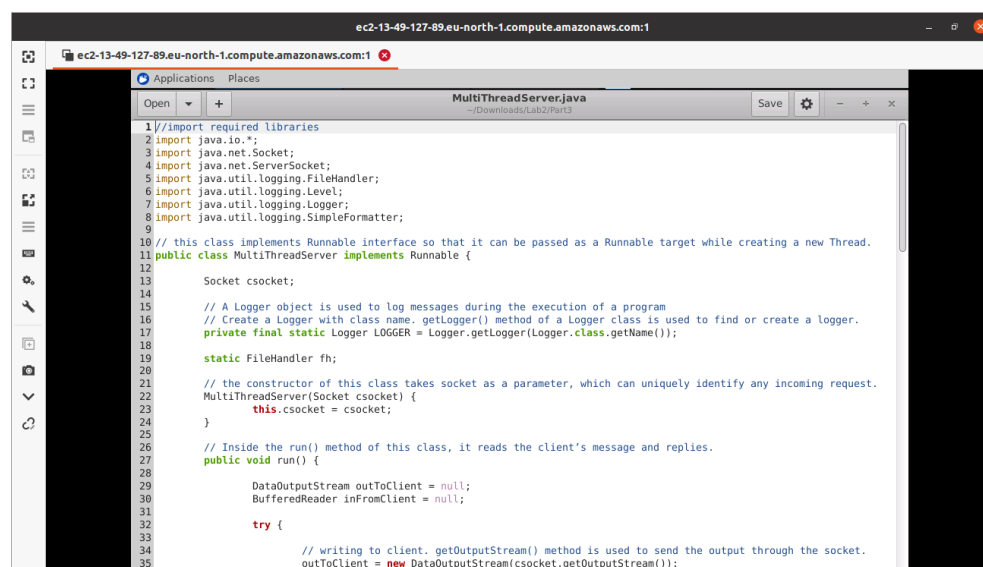
Finally, if a user wants to see the name of the current working directory 10 times, he can write the command "pwd 10" in the input field and click on the "Execute Unix (Multiple Times)" button. Then the GUI will show the output 10 times, as shown below.



### 3.2. The entered commands should be executed and run as the server-side application. You can implement this assignments in either of the following two ways:

1. A CGI program running at the remote webserver 2 (you would get minimum points sufficient to pass this task, however).
2. Implement your own multi-threaded TCP server (higher points will be granted).

A **multi-threaded TCP server** (MultiThreadServer.java) has been implemented and placed on an AWS instance, which runs as a server-side application and executes the commands provided in the GUI by a client, as shown below.





```

ec2-13-49-127-89.eu-north-1.compute.amazonaws.com:1
Applications: [VNC config] Terminal - ubuntu@ip-1...
Terminal - ubuntu@ip-172-31-5-18: ~/Downloads/Lab2/Part3
File Edit View Terminal Tabs Help
ubuntu@ip-172-31-5-18:~/Downloads/Lab2/Part3$ gedit MultiThreadServer.java
ubuntu@ip-172-31-5-18:~/Downloads/Lab2/Part3$ gedit MultiThreadServer.java
ubuntu@ip-172-31-5-18:~/Downloads/Lab2/Part3$ javac MultiThreadServer.java
ubuntu@ip-172-31-5-18:~/Downloads/Lab2/Part3$ java MultiThreadServer
Listening
Oct 01, 2021 5:21:36 PM MultiThreadServer main
INFO: Server is Listening to port: 6400

```

### 3.3. Add logging capability to the server-side application

Logging capability has been added to the server-side application as shown below.

```

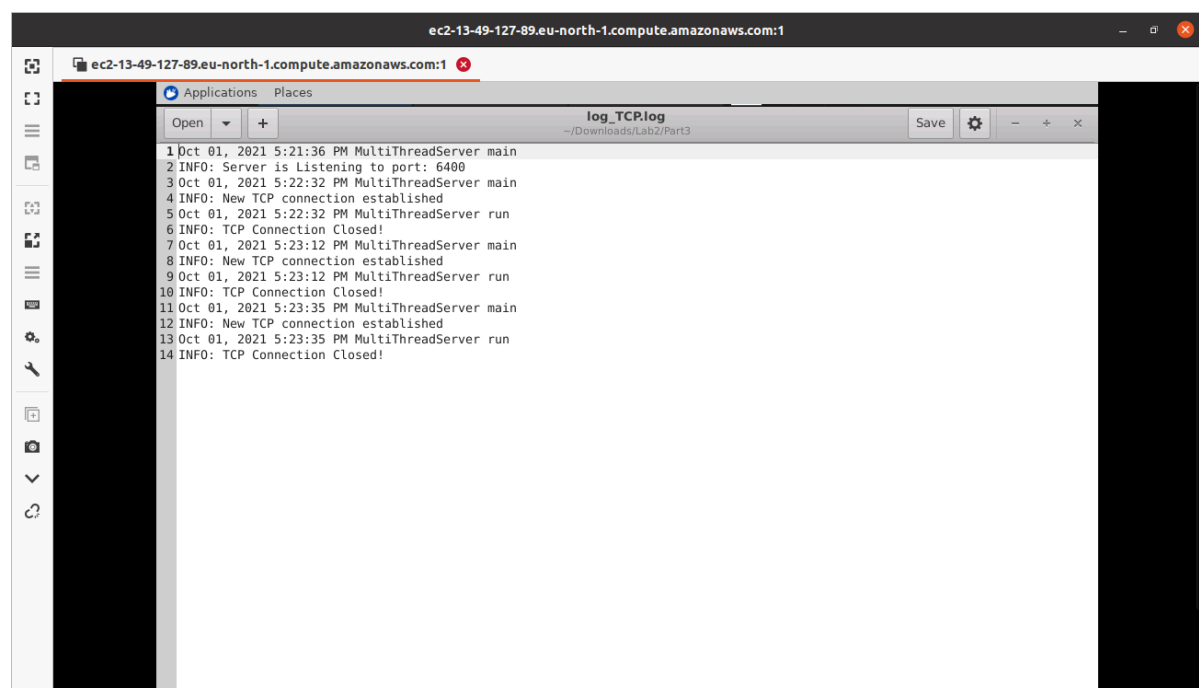
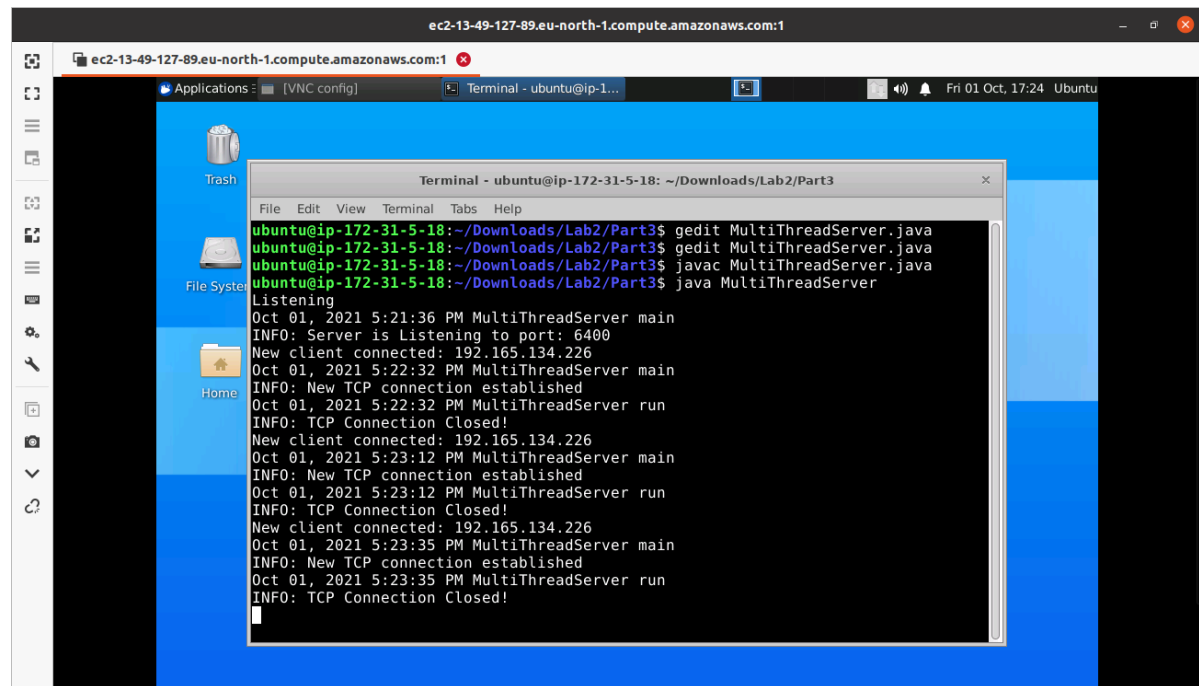
ec2-13-49-127-89.eu-north-1.compute.amazonaws.com:1
Applications: MultiThreadServer.j... [VNC config] Terminal - ubuntu@...
MultiThreadServer.java
~/Downloads/Lab2/Part3
Save
1 //import required libraries
2 import java.io.*;
3 import java.net.Socket;
4 import java.net.ServerSocket;
5 import java.util.logging.FileHandler;
6 import java.util.logging.Level;
7 import java.util.logging.Logger;
8 import java.util.logging.SimpleFormatter;
9
10 // this class implements Runnable interface so that it can be passed as a Runnable target while creating a new Thread.
11 public class MultiThreadServer implements Runnable {
12
13     Socket csocket;
14
15     // A Logger object is used to log messages during the execution of a program
16     // Create a Logger with class name, getLogger() method of a Logger class is used to find or create a logger.
17     private final static Logger LOGGER = Logger.getLogger(Logger.class.getName());
18
19     static FileHandler fh;
20
21     // the constructor of this class takes socket as a parameter, which can uniquely identify any incoming request.
22     MultiThreadServer(Socket csocket) {
23         this.csocket = csocket;
24     }
25

```

```

ec2-13-49-127-89.eu-north-1.compute.amazonaws.com:1
Applications: MultiThreadServer.j... [VNC config] Terminal - ubuntu@...
MultiThreadServer.java (~Downloads/Lab2/Part3) - gedit
~/Downloads/Lab2/Part3
Save
78
79
80
81     public static void main(String args[]) throws Exception {
82
83         // Initialize a FileHandler to write to the given filename, with optional append.
84         fh = new FileHandler("log_TCP.log", true);
85         LOGGER.addHandler(fh); // Add a log Handler to receive logging messages.
86
87         // create a TXT formatter
88         SimpleFormatter formatter = new SimpleFormatter();
89         fh.setFormatter(formatter);
90
91         // initialize server socket
92         ServerSocket server = null;

```



### 3.4. The application should take an argument defining which level of debug should be logged (warning, info, error, debug)

In the server-side application, which level of debug should be logged is defined with the corresponding argument, as shown below.

### 3.5. When adding different log levels to functions you have to explain why you add this level of logging to the specific function. Write this explanation as a comment in your source code

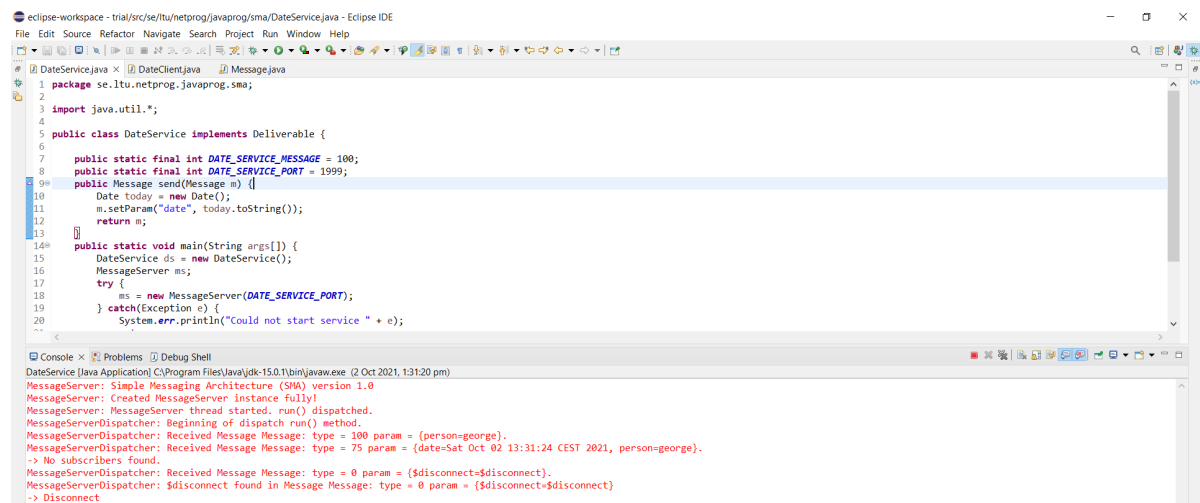
After adding different log levels, the explanation to add this level of logging is written as a comment in the source code, as illustrated in the above two Figures.

# 4

## Part IV –Java netprog patterns –Simple Messaging Architecture

### 4.1. Compile and install the classes in the SMA module of the project.

After compiling and installing the classes, the following output is achieved:



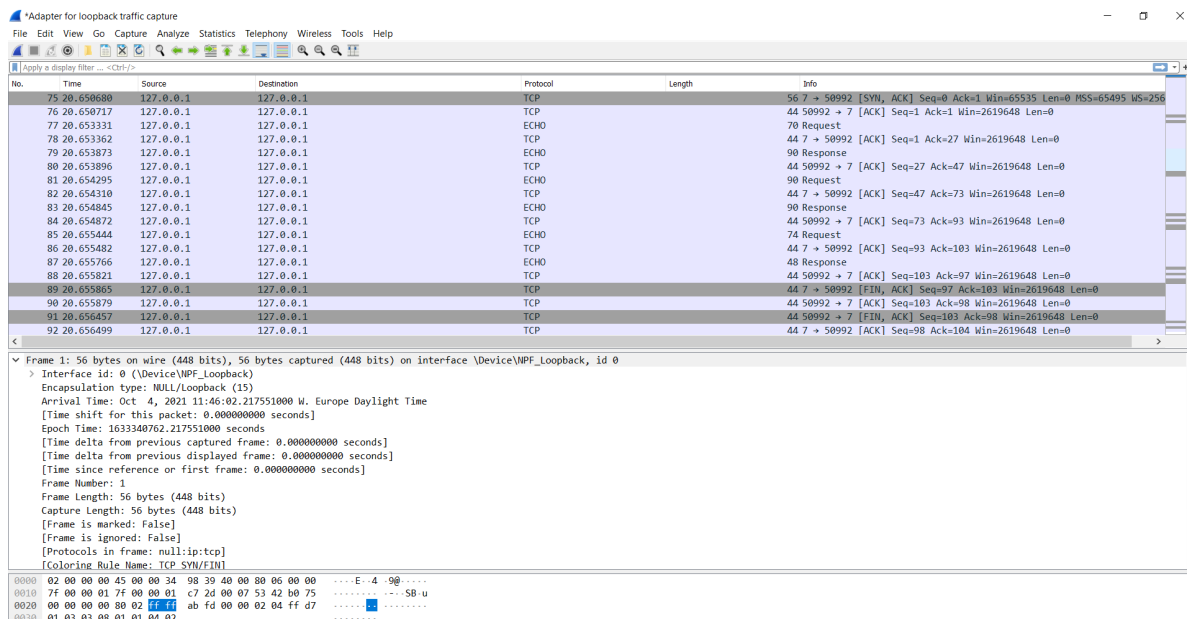
```
eclipse-workspace - trial/src/se/ltu/netprog/javaprog/sma/DateService.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
# DateService.java x DateClient.java Message.java
1 package se.ltu.netprog.javaprog.sma;
2
3 import java.util.*;
4
5 public class DateService implements Deliverable {
6
7     public static final int DATE_SERVICE_MESSAGE = 100;
8     public static final int DATE_SERVICE_PORT = 1999;
9     public Message send(Message m) {
10         Date today = new Date();
11         m.setParam("date", today.toString());
12         return m;
13     }
14     public static void main(String args[]) {
15         DateService ds = new DateService();
16         MessageServer ms;
17         try {
18             ms = new MessageServer(DATE_SERVICE_PORT);
19         } catch (Exception e) {
20             System.err.println("Could not start service " + e);
21         }
22     }
23 }
```

```
Console x Problems x Debug Shell
DateService [Java Application] C:\Program Files\Java\jdk-15.0.1\bin\javaw.exe (2 Oct 2021, 13:12:0 pm)
MessageServer: Simple Messaging Architecture (SMA) version 1.0
MessageServer: Created MessageServer instance fully!
MessageServer: MessageServer thread started. run() dispatched.
MessageServerDispatcher: Beginning of dispatch run() method.
MessageServerDispatcher: Received Message Message: type = 100 param = {person=george}.
MessageServerDispatcher: Received Message Message: type = 75 param = {date=Sat Oct 02 13:31:24 CEST 2021, person=george}.
-> No subscribers found.
MessageServerDispatcher: Received Message Message: type = 0 param = {disconnect=$disconnect}.
MessageServerDispatcher: $disconnect found in Message Message: type = 0 param = {disconnect=$disconnect}
-> Disconnect
```

### 4.2. Demonstrate the functioning classes during the lab assessment. Use Wireshark to capture the traffic of the SMA session. You should be able to explain the details of the implementation during the individual assessment.

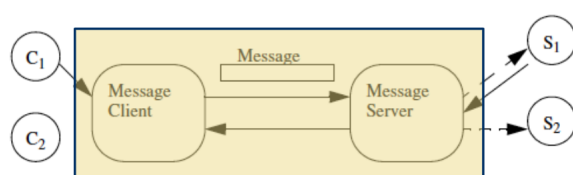
The source code is provided at:

A portion of the wireshark capture is provided below:

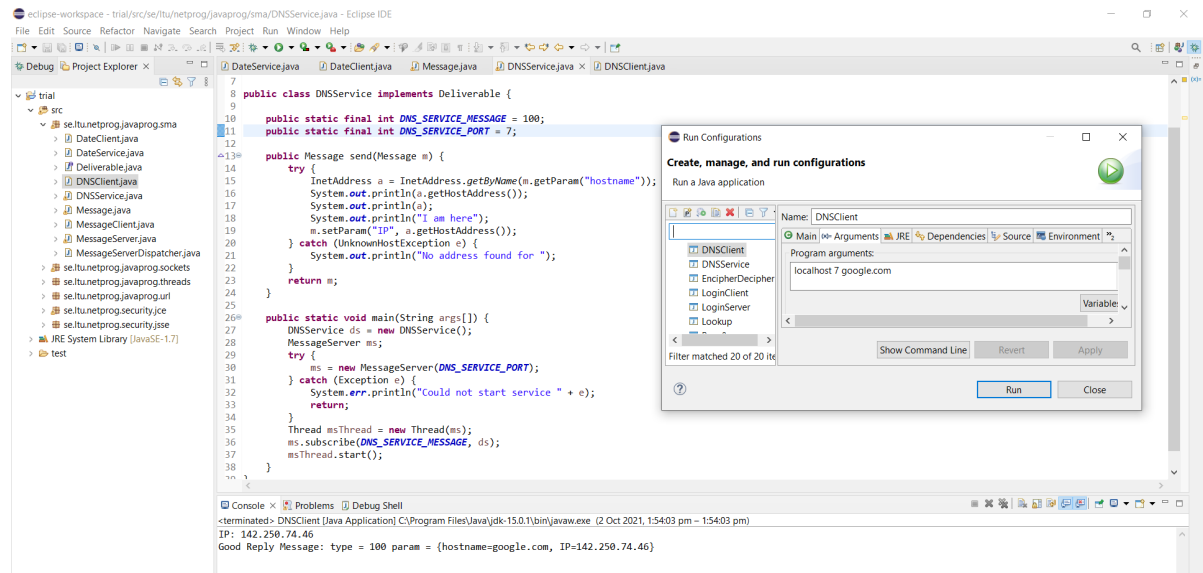


### 4.3. Extend the architecture adding a new kind of service of your choice! For example a simplified DNS service resolving an IP address for a given host number. (This is only a suggestion though)

DNS Service is implemented by adding a new service and client for that service.



Message represents the class used to represent a basic unit of communication (an abstraction that is built using a hash table, wherein the key and values are String instances). MessageServer denotes the the common functionality of a server. MessageClient denotes the common functionality of a client. DNSService is created to represent functionalities on top of the MessageServer whereas DNSClient is used to represent functionalities on top of the MessageClient.



# 5

## Part V: Java netprog patterns – security

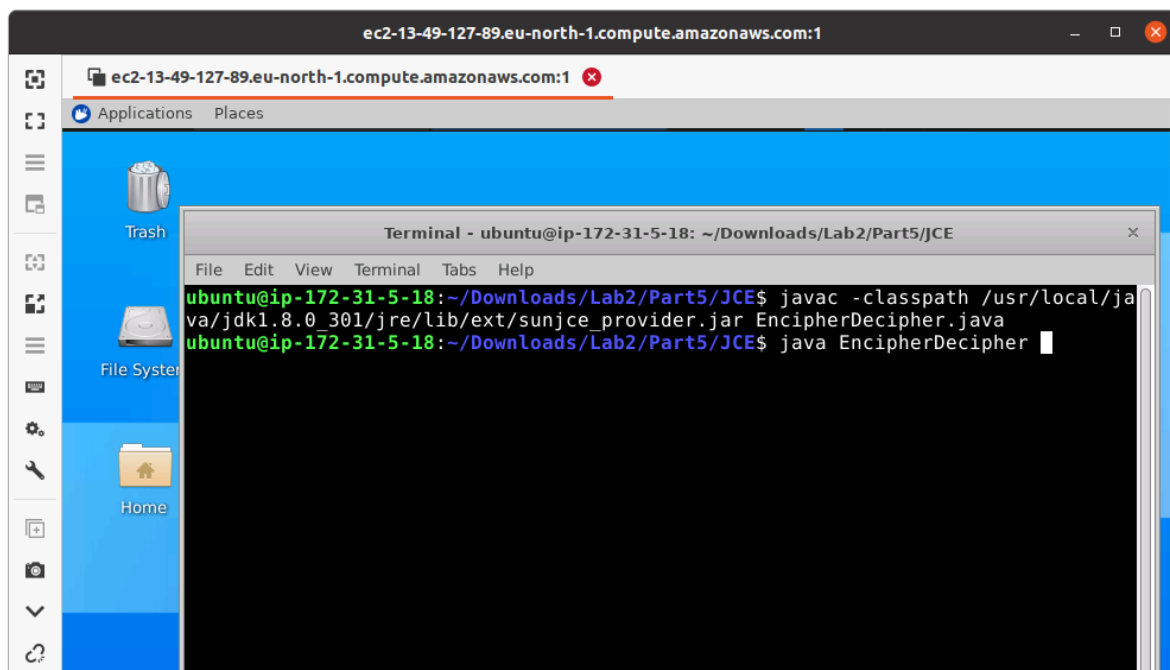
### 5.1. Compile and install (in the cloud) the classes in the security module of the project (JCE and JSSE)

#### 5.1.1. JCE:

The Java Cryptography Extension (JCE) provides Java applications with several security facilities (Deitel et al., 2002). JCE supports secret-key encryption, such as 3DES, and public-key algorithms, such as Diffie-Hellman and RSA (Deitel et al., 2002).

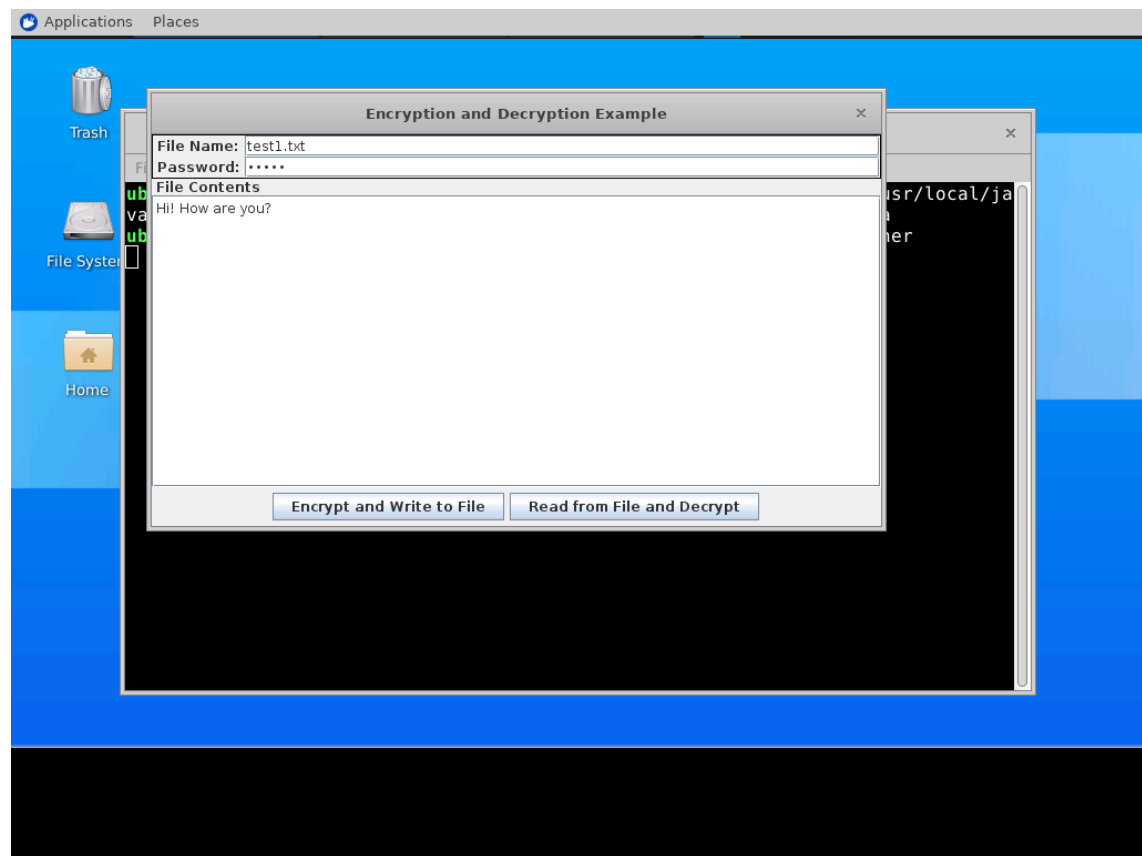
In the JCE module of the project, Class EncipherDecipher uses JCE to demonstrate Password-Based Encryption (PBE). Class EncipherDecipher provides users with a graphical user interface that allows them to specify the file name of a file that the application will use to write to and read from, the contents of the file to encrypt/decrypt, and the password used to encrypt/decrypt the file (Deitel et al., 2002).

The task has been performed on an AWS instance. The following command is used to compile and run the class EncipherDecipher.

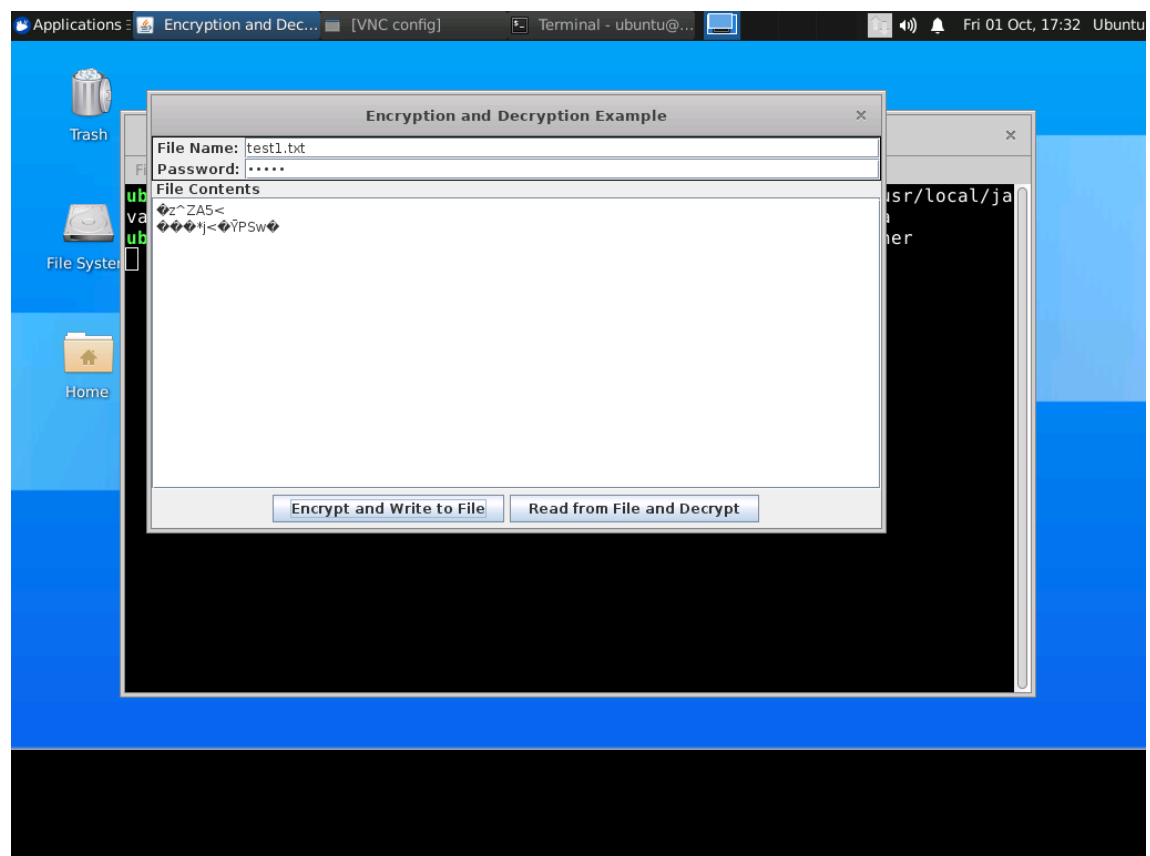


```
ec2-13-49-127-89.eu-north-1.compute.amazonaws.com:1
Applications Places
Trash
File System
Home
Terminal - ubuntu@ip-172-31-5-18: ~/Downloads/Lab2/Part5/JCE
File Edit View Terminal Tabs Help
ubuntu@ip-172-31-5-18:~/Downloads/Lab2/Part5/JCE$ javac -classpath /usr/local/java/jdk1.8.0_301/jre/lib/ext/sunjce provider.jar EncipherDecipher.java
ubuntu@ip-172-31-5-18:~/Downloads/Lab2/Part5/JCE$ java EncipherDecipher
```

The following figure displays the contents that application EncipherDecipher will encrypt and write to file 'test1.txt' using password 'jamil'.

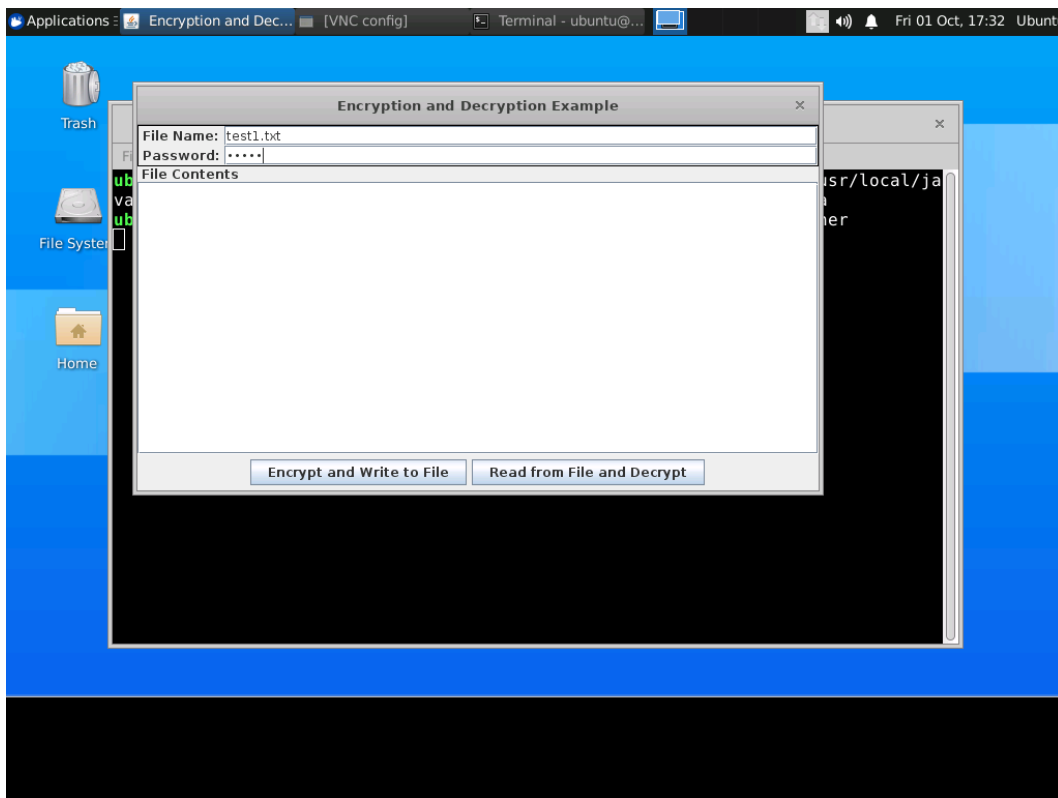


The following figure displays the contents of the file after pressing button 'Encrypt and Write to File'.

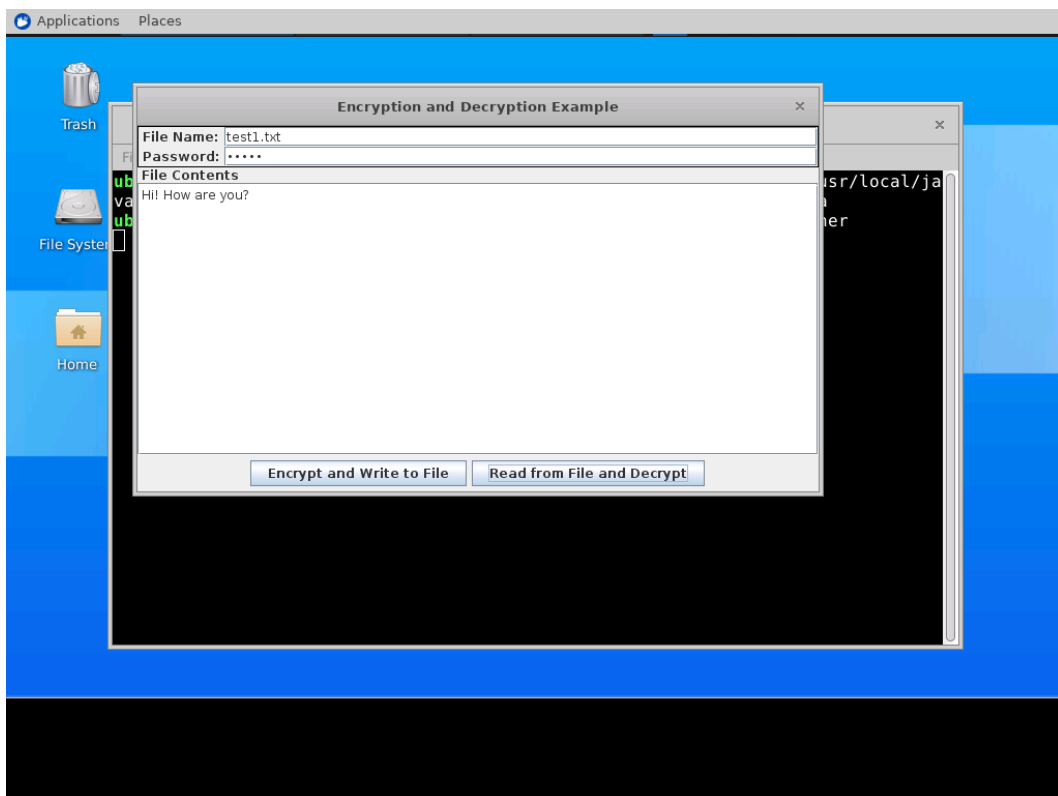




The following figure displays the contents that application EncipherDecipher will read from file and decrypt 'test1.txt' using password 'jamil'.



The following figure displays the contents of the file after pressing button 'Read from File and Decrypt'.



### 5.1.2. JSSE:

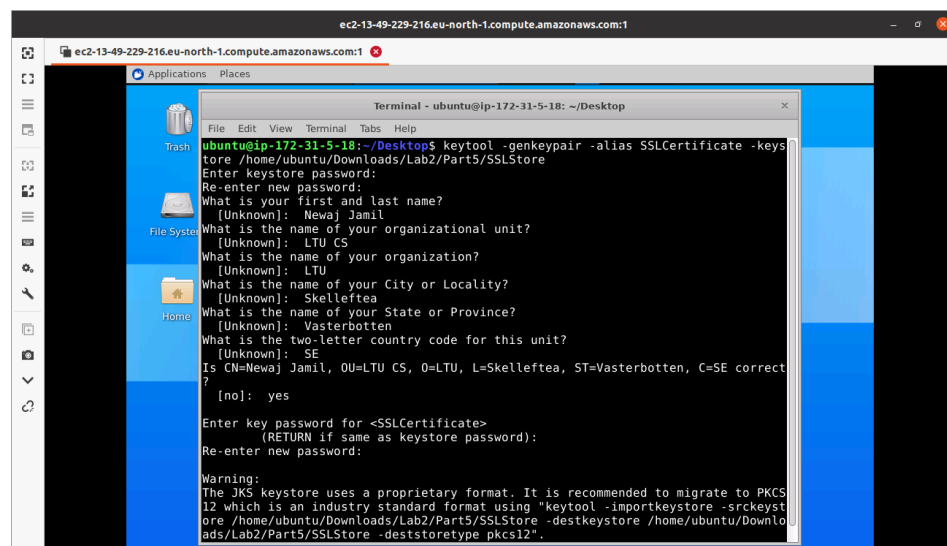
SSL encryption has been integrated into Java technology through the Java Secure Socket Extension (JSSE) (Llewellyn, 2006). JSSE uses keystores to secure storage of key pairs and certificates used in PKI (Public Key Infrastructure which integrates public-key cryptography with digital certificates and certificate authorities to authenticate parties in a transaction) (Llewellyn, 2006). A truststore is a keystore that contains keys and certificates used to validate the identities of servers and clients (Llewellyn, 2006).

In the JSSE module of the project, the classes involves a client application (LoginClient.java) that attempts to logon to a server (LoginServer.java) using SSL.

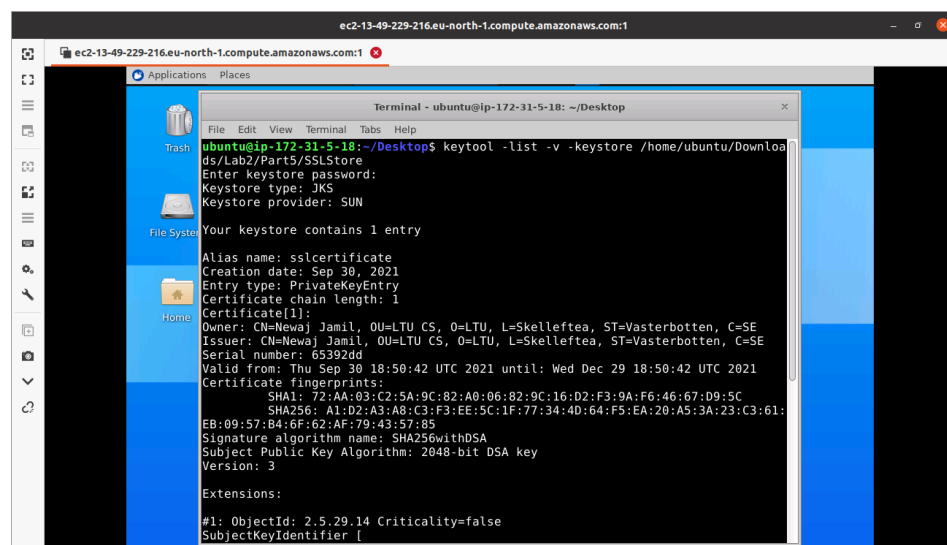
Before executing the LoginServer and LoginClient application using SSL, we have to create a keystore and certificate for the SSL to operate correctly. Utilizing the keytool (a key and certificate management tool) in Java, a keystore and a certificate for the server application can be generated.

The server (LoginServer.java) is placed on an AWS instance. First, we have created a self-signed certificate for the server (LoginServer.java) in the server's KeyStore. Then on a local machine (client machine), class InstallCert.java has to be run by providing the server's address along with port number and passphrase in order to add the server's certificate to the client's TrustStore with other trusted certificates. Finally, we can run the client application (LoginClient.java) in order to establish a secure connection with the server.

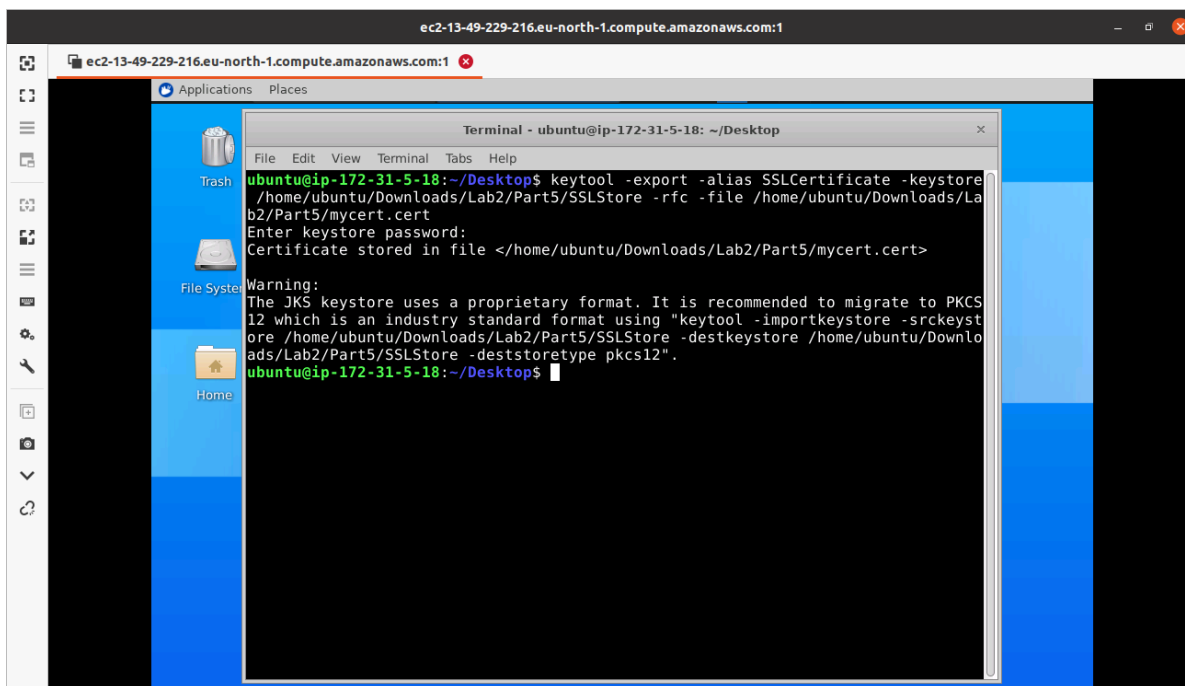
**Creating a Self-Signed Certificate:** Java keytool command is used to generate a self-signed certificate for the server and place it in the KeyStore.



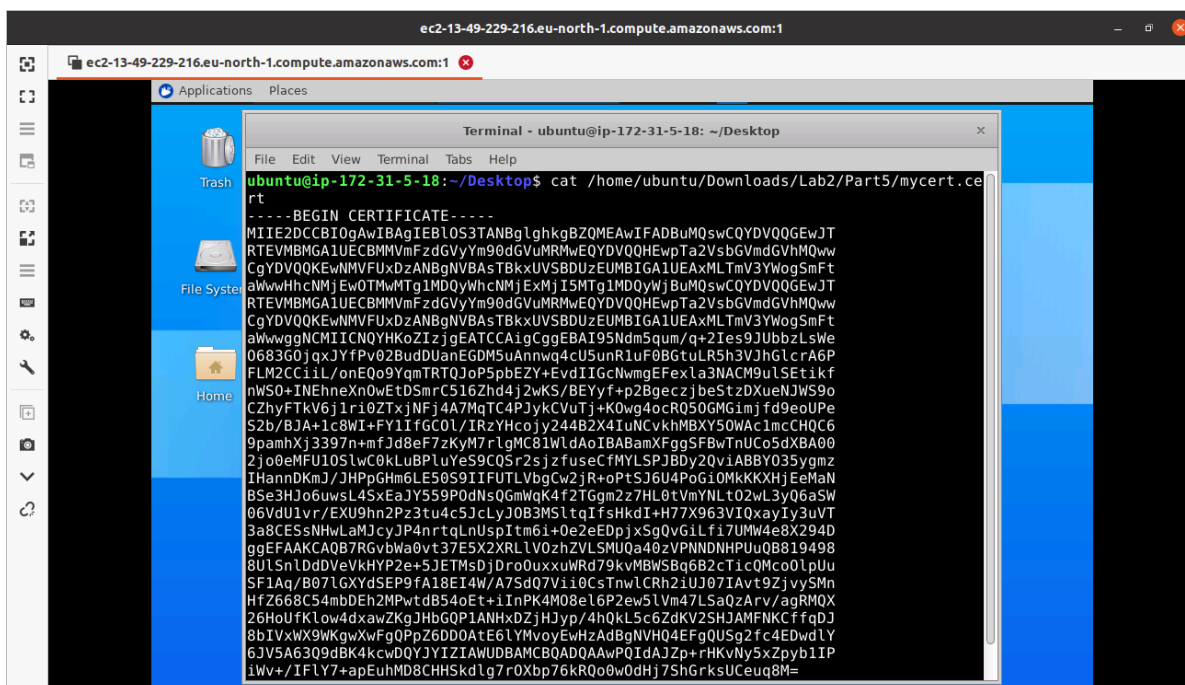
We can view the keystore contents after its creation using the following command:



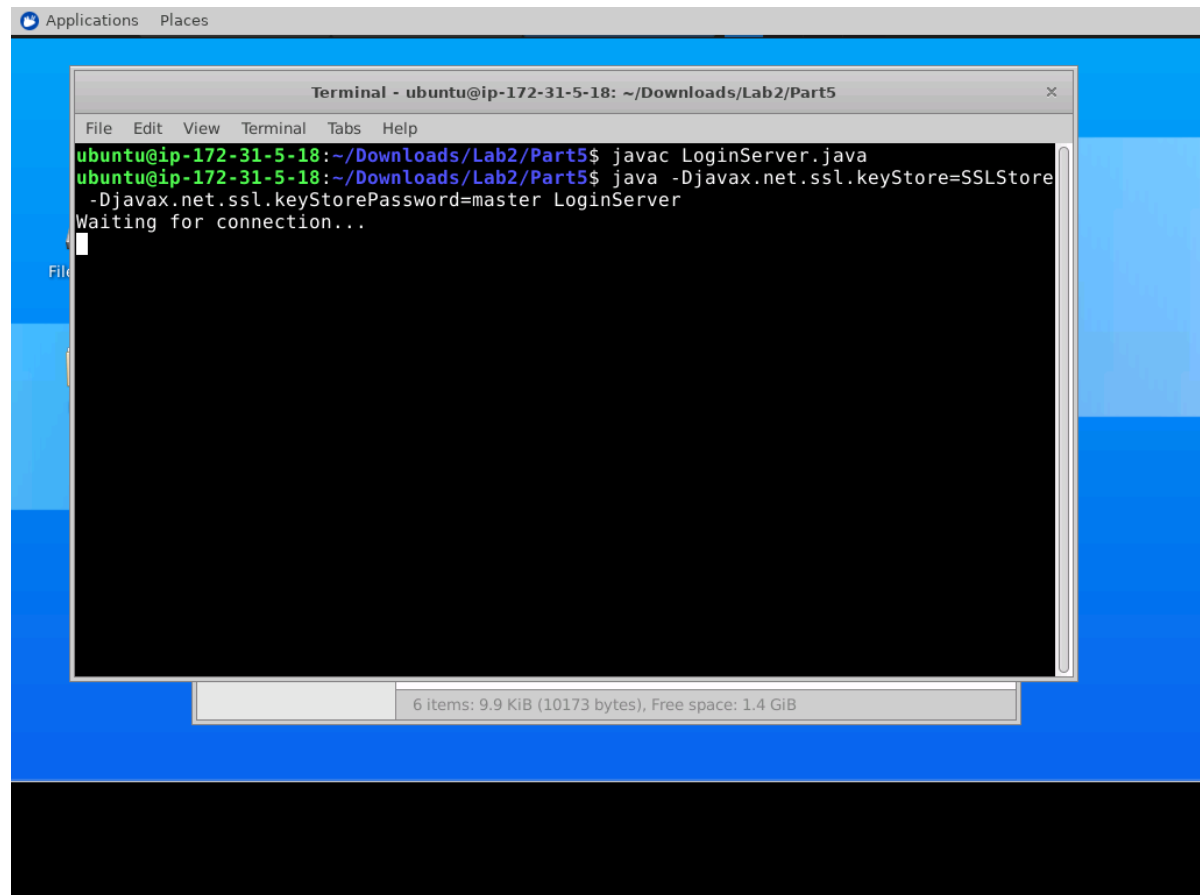
Then we have exported the certificate into a certificate file (mycert.cert).



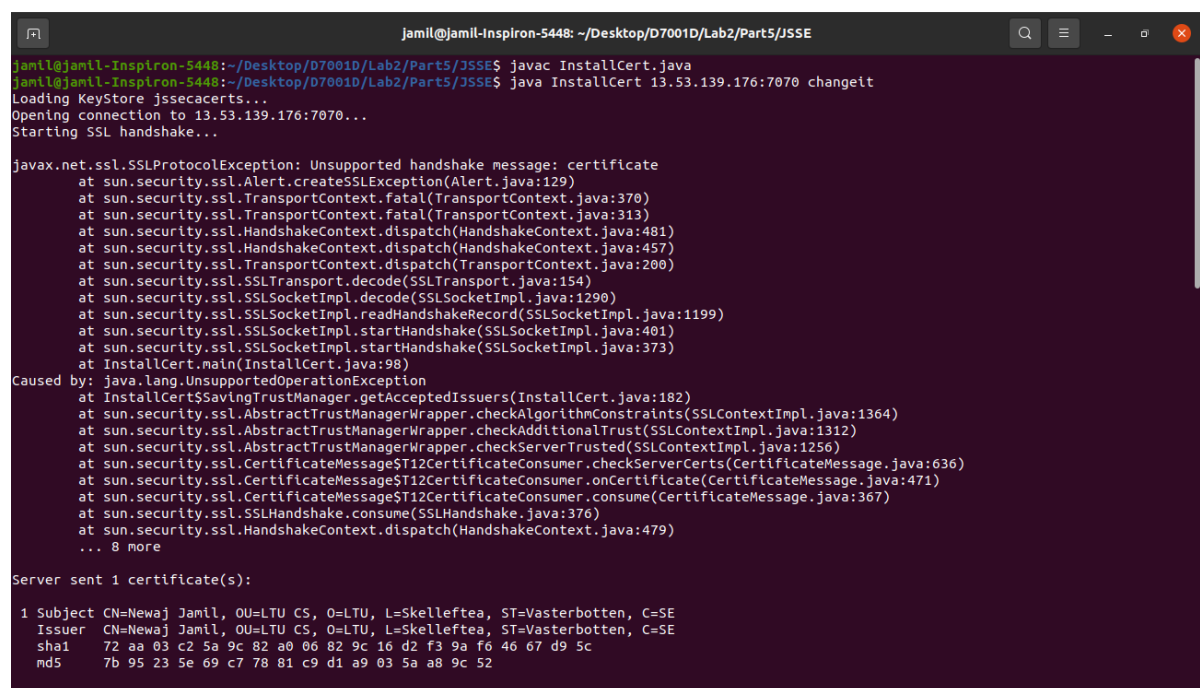
We can view the contents of the certificate using the following command:



**Launching LoginServer.java:** We can start the server from a terminal by specifying the keystore for the server. Once started, the server simply waits for a connection from a client. The following command is used to start the server.



**Launching InstallCert.java:** On a local machine (client machine), class InstallCert.java has to be run by providing the server's address along with port number and passphrase in order to add the server's certificate to the client's TrustStore with other trusted certificates.



```
jamil@jamil-Inspiron-5448: ~/Desktop/D7001D/Lab2/Part5/JSSSE
Enter certificate to add to trusted keystore or 'q' to quit: [1]

[
Version: V3
Subject: CN=Newaj Jamil, OU=LTU CS, O=LTU, L=Skelleftea, ST=Vasterbotten, C=SE
Signature Algorithm: SHA256withDSA, OID = 2.16.840.1.101.3.4.3.2

Key: Sun DSA Public Key
Parameters: DSA
p: 8f7935d9 b9aa9bf abed887a cf4951b6 f32ec59e 3baf3718 e8eac496 1f3efd36
06e74351 a9c41833 39b809e7 c2ae1c53 9ba7475b 85d011ad b8b47987 75498469
5cac0e8f 14b33688 28a22ffa 27110a3d 62a99345 3409a0fe 696c4658 f84bdd20
819c3709 a01057b1 95adcd00 233dba54 04b6291f 9d648ef8 83448677 979ced04
b434a0ac 2675e998 5de23d00 292fc111 8c9ffa9d 8181e733 8db792b7 30d7b9e3
49592f60 09987215 3915aa3d 68bb4653 c633458f 803b32a4 c2e0f272 90256e4e
3f8a3b08 38a1c450 e4e18c1a 29a37ddf 5ea143de 4b66ff04 903ed5cf 1623e158
d487c008 e97f211c db1dca23 cb6e3807 65f822e3 42be484c 05763939 601cd667
q: baf696a6 8578f7df dee7fa67 c977c785 ef32b233 bae580c0 bcd5695d
g: 16a65c58 20485070 4e7502a3 9757040d 34da3a34 78c154d4 e4a5c02d 242ee04f
96e51e4b d09e4ab0 ac8f37ae b1e09f31 82d23c90 43cb42f7 8080a160 edf9ca09
b32076a7 9c32a027 f2473e91 879ba2c4 e744bd20 81544c55 5b802c30 8d1fa83e
d489e94e 0fa0688e 32428a5c 78c478c0 8d0527b7 1c9a3abb 0b0be12c 44689639
e7d3ce74 db101a65 aa2b87f6 4c0826db 3ec72f4b 5599834b b4edb02f 7c90e9a4
96d3a55d 535bebf6 45d4f619 f63f3ded bb873925 c2f224e0 7731296d a887ec1e
4748f87e fb5fdeb7 5484310b 2232dees 53ddaf02 112b0d1f 02da3097 3224fe27
aeda8b9d 4b2922d9 ba8be39e d9e103a6 3c52810b c688b7e2 ed4310e1 ef17dbde

y:
7b446bdb 59ad2fb7 7ec4e57d 9744b955 3b385954 b48c5106 b8d3354f 34d0cd1c
f52e401f 35f78f7c f149529e 50dd0d57 9590760f d9efb924 44ccb038 c3ae83ae
c71b9645 defd92f3 0159206a e81d9c4e 271031ca 0e96952e 485d40ab f074ee51
9761d484 3fd7c0d7 c108e16f c0ed2750 ed58a2d0 2b139f09 42461da2 509d3b20
0bedf598 efc92327 1df67aeb c0b9e266 c3121d8c 3f0b5d07 9e2812df a22273ca
e0c3bc7a 5e8fd9ec 399559b8 ecb49a43 302bbff6 a044c17f dba1e851 f2a5a30e
1dc5ac19 2a02476c 640fd403 47c43663 1c9ca9ff 885090be 5ce9974a 57648724
030534a0 9f7ea0c9 f1b215c5 65fd58a8 305f0160 40fa59e8 30ce02d1 3a95832f

Validity: [From: Thu Sep 30 20:50:42 CEST 2021,
To: Wed Dec 29 19:50:42 CET 2021]
Issuer: CN=Newaj Jamil, OU=LTU CS, O=LTU, L=Skelleftea, ST=Vasterbotten, C=SE
SerialNumber: [ 065392dd]

Certificate Extensions: 1
[1]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 4A 0D 9F 73 81 03 C1 D9 58 E8 95 79 03 AD D0 F5 J...S...X...Y....
0010: D0 4A E2 47 .J.G
]
]

Algorithm: [SHA256withDSA]
Signature:
0000: 30 3D 02 1D 00 96 69 FA B1 CA BC DC B9 C5 9A 72 0=...t.....r
0010: 6F 52 0F 89 68 FE FC B1 65 63 BF 9A A4 48 A1 30 0R...k...ec...K.0
0020: 3F 02 1C 74 A4 76 58 3B AC E5 DB A7 BE A4 45 0A ?...t.VX;.....E.
0030: 34 C0 E7 47 BF B4 A1 1A B9 2c 50 27 AE AB C3 4..G.....P'...

Added certificate to keystore 'jssecacerts' using alias '13.53.139.176-1'
jamil@jamil-Inspiron-5448: ~/Desktop/D7001D/Lab2/Part5/JSSSE
```

```
jamil@jamil-Inspiron-5448: ~/Desktop/D7001D/Lab2/Part5/JSSSE

aeda8b9d 4b2922d9 ba8be39e d9e103a6 3c52810b c688b7e2 ed4310e1 ef17dbde

y:
7b446bdb 59ad2fb7 7ec4e57d 9744b955 3b385954 b48c5106 b8d3354f 34d0cd1c
f52e401f 35f78f7c f149529e 50dd0d57 9590760f d9efb924 44ccb038 c3ae83ae
c71b9645 defd92f3 0159206a e81d9c4e 271031ca 0e96952e 485d40ab f074ee51
9761d484 3fd7c0d7 c108e16f c0ed2750 ed58a2d0 2b139f09 42461da2 509d3b20
0bedf598 efc92327 1df67aeb c0b9e266 c3121d8c 3f0b5d07 9e2812df a22273ca
e0c3bc7a 5e8fd9ec 399559b8 ecb49a43 302bbff6 a044c17f dba1e851 f2a5a30e
1dc5ac19 2a02476c 640fd403 47c43663 1c9ca9ff 885090be 5ce9974a 57648724
030534a0 9f7ea0c9 f1b215c5 65fd58a8 305f0160 40fa59e8 30ce02d1 3a95832f

Validity: [From: Thu Sep 30 20:50:42 CEST 2021,
To: Wed Dec 29 19:50:42 CET 2021]
Issuer: CN=Newaj Jamil, OU=LTU CS, O=LTU, L=Skelleftea, ST=Vasterbotten, C=SE
SerialNumber: [ 065392dd]

Certificate Extensions: 1
[1]: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: 4A 0D 9F 73 81 03 C1 D9 58 E8 95 79 03 AD D0 F5 J...S...X...Y....
0010: D0 4A E2 47 .J.G
]
]

Algorithm: [SHA256withDSA]
Signature:
0000: 30 3D 02 1D 00 96 69 FA B1 CA BC DC B9 C5 9A 72 0=...t.....r
0010: 6F 52 0F 89 68 FE FC B1 65 63 BF 9A A4 48 A1 30 0R...k...ec...K.0
0020: 3F 02 1C 74 A4 76 58 3B AC E5 DB A7 BE A4 45 0A ?...t.VX;.....E.
0030: 34 C0 E7 47 BF B4 A1 1A B9 2c 50 27 AE AB C3 4..G.....P'...

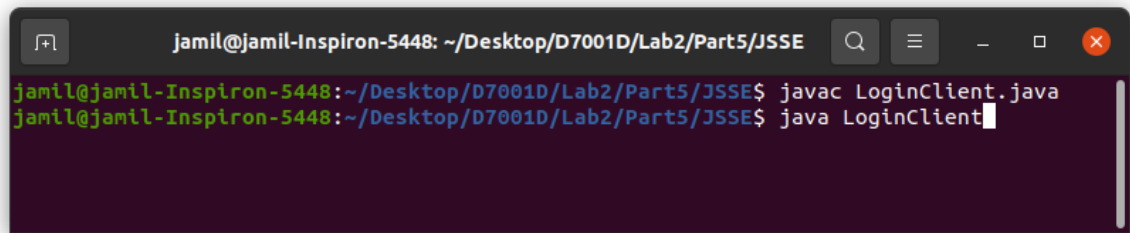
Added certificate to keystore 'jssecacerts' using alias '13.53.139.176-1'
jamil@jamil-Inspiron-5448: ~/Desktop/D7001D/Lab2/Part5/JSSSE
```

The certificate is added to keystore 'jssecacerts' in the current directory. We have to copy the file from the current directory to the security properties directory, 'java.home \lib \security \', where java.home is the runtime environment directory (the jre directory in the SDK). This will facilitate Java to use 'jssecacerts' to authenticate the servers.

```
jamil@jamil-Inspiron-5448: ~/Desktop/D7001D/Lab2/Part5/JSSSE

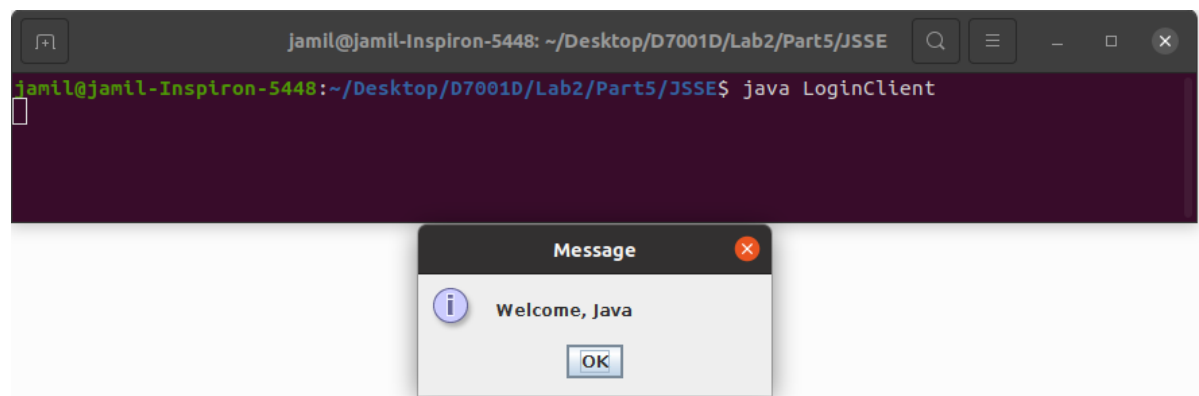
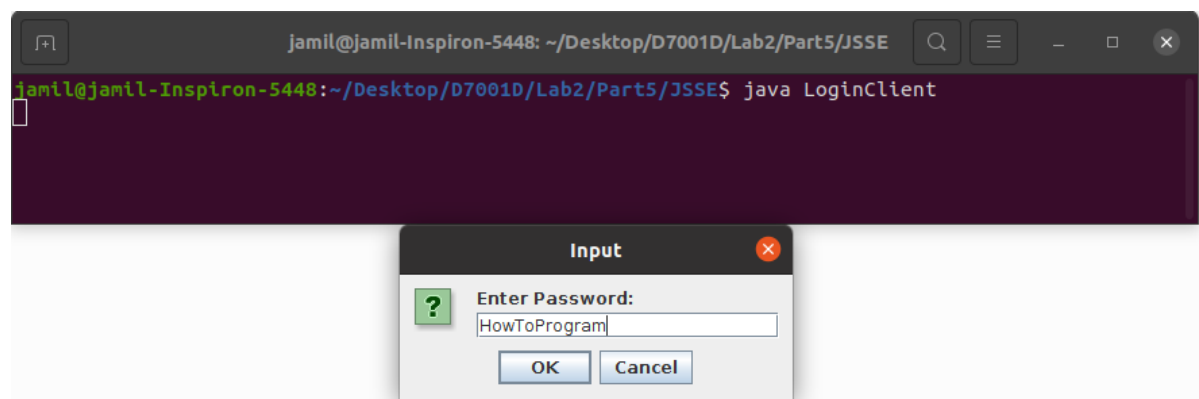
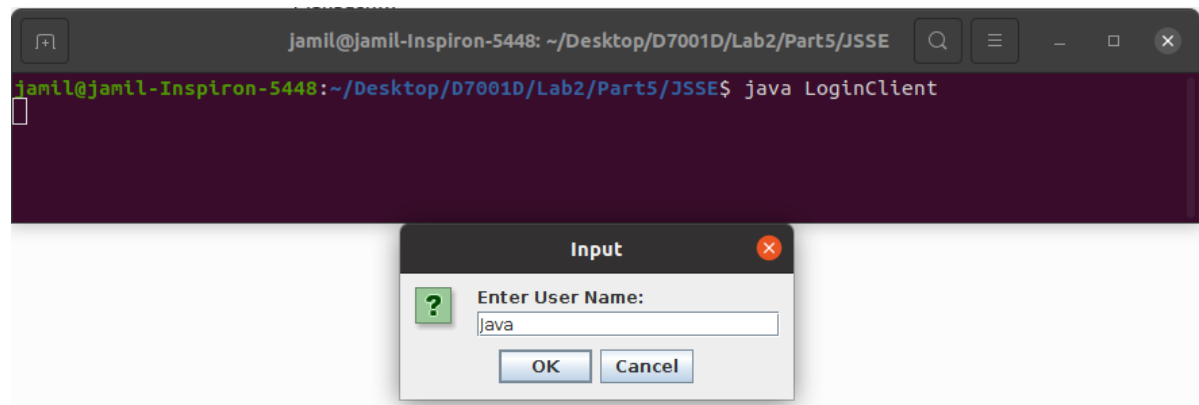
jamil@jamil-Inspiron-5448:~/Desktop/D7001D/Lab2/Part5/JSSSE$ sudo cp jssecacerts /usr/local/java/
jdk1.8.0_301/jre/lib/security/
[sudo] password for jamil:
jamil@jamil-Inspiron-5448:~/Desktop/D7001D/Lab2/Part5/JSSSE$
```

**Launching LoginClient.java:** Now we can start the client application to the client host from a terminal. Once the client establishes communication with the server, the authentication process begins. The following command is used to start the client application.



```
jamil@jamil-Inspiron-5448: ~/Desktop/D7001D/Lab2/Part5/JSSE
jamil@jamil-Inspiron-5448:~/Desktop/D7001D/Lab2/Part5/JSSE$ javac LoginClient.java
jamil@jamil-Inspiron-5448:~/Desktop/D7001D/Lab2/Part5/JSSE$ java LoginClient
```

After that, the client has to enter the username and password which are sent to the server. If the authentication is successful, the client is logged on.



# Bibliography

- Deitel, H., Deitel, P., & Santry, S. (2002). *Advanced java 2 platform: How to program*. Prentice Hall. <https://books.google.se/books?id=umtwQgAACAAJ>
- Llewellyn, M. (2006). *Cop 4610L: Java networking part 3*. Retrieved October 7, 2021, from <http://www.cs.ucf.edu/courses/cop4610L/fall2005/javanet3.pdf>