Lab 2 report

Part 1

GUI enables entering command or number of Fibonacci values and two separate buttons.

Every command starts a new thread, so, while numbers are being computed, it is possible to execute OS commands with “Run commands” button.

Command line is started automatically depending on OS type (Windows/Linux).

Part 2

Lookup – returns IP addresses of consecutively passed hostnames in the arguments

```
ubuntu@ip-172-31-3-115:~/jar-files$ java -jar Lookup_lab1part2.jar google.com
google.com:172.217.0.14. Name: Anastasiia
ubuntu@ip-172-31-3-115:~/jar-files$ java -jar Lookup_lab1part2.jar google.com facebook.com
google.com:172.217.0.14. Name: Anastasiia
facebook.com:31.13.71.36. Name: Anastasiia
```

TCPEchoServer – establishes TCP connection, gets the client input and sends it back. For appending a name, it should be converted to bytes and concatenated with the byte array of client input. Tested on AWS instance

```
C:\Users\aagri>telnet 18.224.69.160 9000
Trying 18.224.69.160...
Connected to 18.224.69.160.
Escape character is '^]'.
hey
 Anastasiia hey
How are you
 Anastasiia How are you
```

UDPEchoServer – establishes UDP connection, gets the client input and sends it back. For appending a name, it should be converted to bytes and concatenated with the byte array of client input. Datagrams should maintain accurate length and client address, if a new datagram is created. Tested on AWS and Ubuntu VM

```
anastasiiagrishina@anastasiiagrishina-VirtualBox:~$ nc -u 18.224.69.160 3004
hey
hey
Anastasiia

hello
hello
Anastasiia
```

Race0

a) Unpredictable strings of X and dots, since the shared resource is asynchronously accessed by two threads

c) Dots are displayed, since both sleep and increments are implemented and accessed synchronously.

Part 3

Version 1.

Two buttons initialize parallel client threads, where a URL, a command type (loop or OS command), command text, and text area pointer are passed. Clients connect to the multithreaded server by manipulating the GUI. On the server a listener is implemented. With every new connection a new thread is started on server, server Runnable implements run method, which executes commands and sends response – standard output or standard errors, or Fibonacci numbers. Client reads the response and updates text in the text area

Version 2.

Two buttons send http GET requests passing a URL, a command type (loop or OS command), command text to a method of HttpRequest class. In HttpRequest class, the request is formed from the passed parameters and sent to the server. A multithreaded server starts new thread for every TCP connection and runs a CmdWrkrkRunnable class run method. The method decodes url, extracts the query, executes corresponding commands and forms an encoded response with the output of the commands execution. Server response is read in the GUI as a returned parameter of send http request function, is decoded and passed to the text area in correspondence with what command was executed.

Both servers tested on the AWS instance

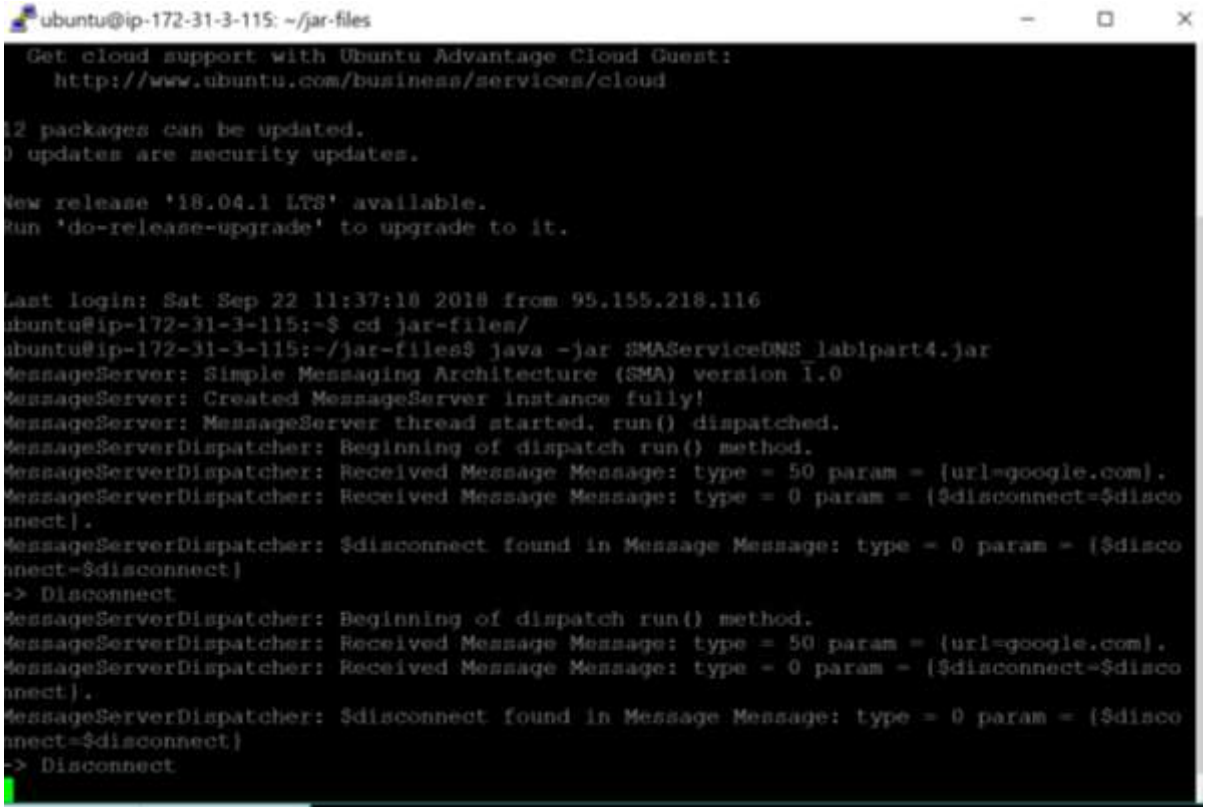
Logger levels:

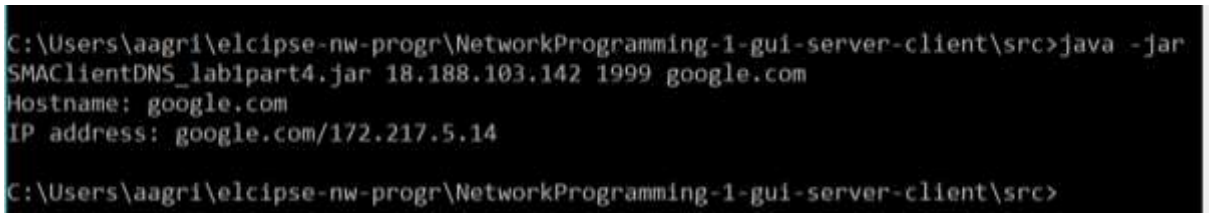
- severe is used when program crashes due to an error,
- warning is applied to cases when a program is still working, but desired functioning is deviated,
- info is for informative messages, like connection established/ended,
- fine, finer, finest – are informative and not at all urgent logs.

Part 4

Simple Messaging Architecture is created to allow developers focus on functionality other than on connection establishment and threads. By creating analogous classes to DateClient and DateService, it is possible send requests to a server, execute them and send back responses to the client. Send function on the server side should be used to execute and form the response. Message and port numbers should be updated and correlated in the newly created classes.

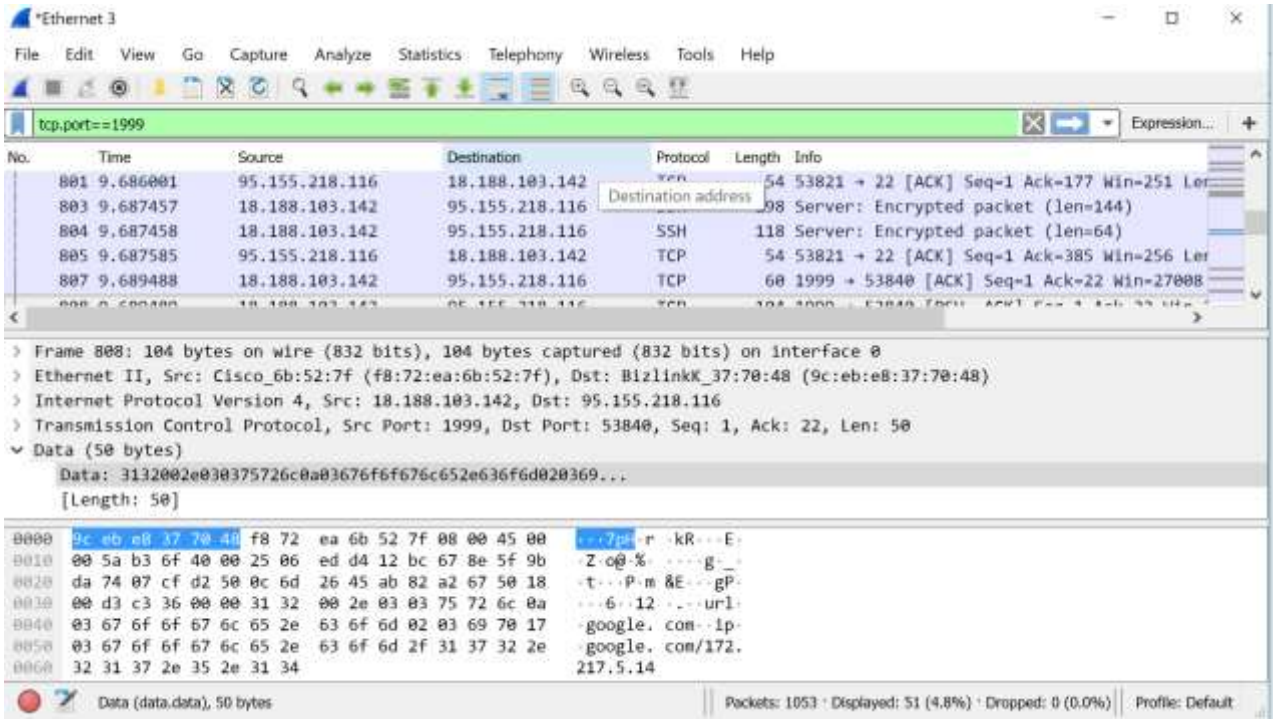
Server tested on AWS instance:





Wireshark capture

The reply message contained two parameter-value pairs: {“url”; <hostname>} and {“ip”; <hostname/ip>}.



Part 5

The concept: server has an SSL private key, which are integrated in the keystore. The client should create a public certificate and integrate it into the truststore locally. A key, a keystore and a truststore have their own different passwords. Truststore keeps trusted certificates, keystore has a private key to decrypt messages incoming with a public certificate.

InstallCert generates a certificate and puts it into the truststore called jssecacert – JSSE Certified Authority Certificate. While running InstallCert you can pass arguments in the following format:

`host[:port] [password]`

host – localhost or AWS IP, here we use AWS IP

port – we use 7070, but default 443 can be used. In the case of default port, https traffic should be allowed on port 443 in the security group.

password – specify it if you wish to use your own password. The password is *changeit* by default.

Side notes

To export jar file from Eclipse:

1. Right click on the class to export, Java, Runnable Jar file, next.

2. Choose configuration to launch. If the file does not appear on the drop-down list, close export window and try to run it with Ctrl+F11. Ignore possible not passed arguments error and repeat 1, 2.

3. Set a destination and a jar file name.

4. Tick export required libraries into generated JAR

5. Finish. Ok for warnings.

If there are problems with overwriting an existing jar, delete it and export again, as if it did not exist.

Enable ciphers on the server and client after creating a socket. Example for the server socket:

```
String[] supportedCiphers = serverSocket.getSupportedCipherSuites();
serverSocket.setEnabledCipherSuites(supportedCiphers);
```

AWS
Configure 7070 port for your future TCP traffic

1. enable ssl
   `sudo a2enmod ssl`

2. restart apache server
   `sudo  service apache2 restart`

3. generate a certificate and a keystore where keytool will automatically insert the certificate you are generating
   `keytool -genkey -keystore keystore_name -alias certificate_name`
   My certificate name: MyCert

4. enable pkcs12
   `keytool -importkeystore -srckeystore sslStore -destkeystore sslStore -deststoretype pkcs12`

5. set key password for alias or whatever it is
   "The destination entry will be protected using *destkeypass*. If *destkeypass* is not provided, the destination entry will be protected with the source entry password."

   ```
   keytool -importkeystore -srckeystore sslStore -destkeystore sslStore -
   deststoretype pkcs12 -destkeypass destination_password
   ```

6. Run the server to launch InstallCert on Client (after this step go to Client part):

   ```
   java -cp ServerLoginJSSE_lab1part5.jar -
   Djavax.net.ssl.keyStore=keystore_full_path -
   Djavax.net.ssl.keyStorePassword=keystore_password project_path.classname
   ```

7. Run the server to talk to the client (same as above)
   ```
   java -cp ServerLoginJSSE_lab1part5.jar
   -Djavax.net.ssl.keyStore= keystore_full_path
   -Djavax.net.ssl.keyStorePassword=keystore_password project_path.classname
   ```

Client – local machine

Import sslStore with scp/WinSCP or analogs from AWS to your local machine

1. Run Install Cert (while running the server) from command line
   ```
   java -jar jar_file_name.jar <AWS instance IP>:7070
   ```

2. Run the client
   ```
   java -cp jar-file-name.jar -Djavax.net.ssl.trustStore=client_truststore
   -Djavax.net.ssl.trustStorePassword=truststore_passwd
   project_path.classname server_host:server_port
   ```

   client_truststore – full path to client truststore, i.e. jssecacert if InstallCert was used to generate it
   truststore_passwd – truststore password, i.e. “changeit” if InstallCert was used