BINUS UNIVERSITY
BINUS INTERNATIONAL

Assignment Cover Letter
(Individual Work)

Student Information:	***Surname***	***Given Names***	***Student ID Number***
	Roselynn	Callista	2301944462

Course Code	**: COMP6510**	**Course Name**	**: Programming Languages**
Class	**: L2BC**	**Name of Lecturer**	**:** Mr. Jude Martinez
Major	**: Computer Science**		
Title of Assignment	: Chat Application S2 Project		
Type of Assignment	**: Final project**		
Submission Pattern			
Due Date	**: 20 June, 2020**	**Submission Date**	**: 20 June, 2020**

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer's instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

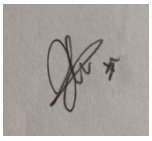
Plagiarism/Cheating

BINUS International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

Declaration of Originality

By signing this assignment, I understand, accept and consent to BINUS International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my/our* own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student: *(Name of Student)*



Callista Roselynn

CHAT APPLICATION

Programming Language

By: Callista Roselynn

Student ID: 2301944462

Submitted to: Sir Jude Martinez

Course: Programming Language

TABLE OF CONTENTS

PROJECT SPECIFICATION

PURPOSE

This document is intended as a program manual to obtain further information regarding this project. This project utilizes the java programming language to recreate a chat application to attempt a functional program.

SCOPE

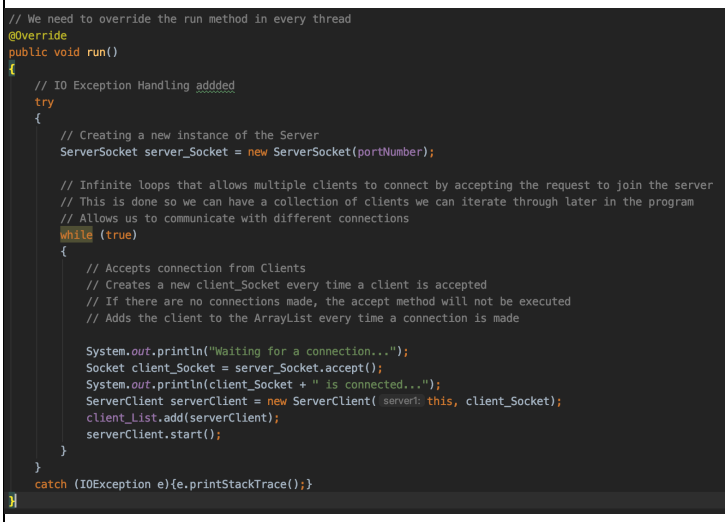
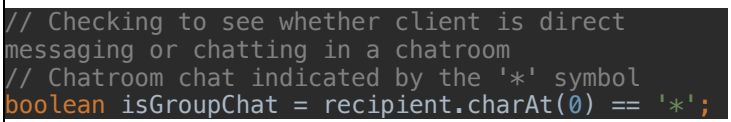
As mentioned above, this semester I attempted to recreate a chat Application. This system should utilize Java libraries and modules, telnet, as well as some other imported libraries to create the connections and relay information to and from each other. This chat application should function just like a Chat Application normally should. It must be capable of connecting to the internet, sending and receiving text messages as well as detecting online and offline presence of users. By implementing such features with the aid of Java libraries, I aim to expand my knowledge regarding web network programming as well as further my skills in Java. I also hope to learn more on what Java has to offer.

REQUIREMENTS

REQUIREMENTS	EXAMPLE
Primitive Data	`int portNumber = 2402;`
Imported Classes	`import org.apache.commons.lang3.StringUtils;`
Built in Classes & Methods	`public class Server extends Thread { // VARIABLES private final int portNumber; private ArrayList<ServerClient> client_List = new ArrayList<>(); // C // CONSTRUCTOR public Server(int portNumber) { this.portNumber = portNumber; } // METHODS // Method created to enable clients to have access to other clients fr public List<ServerClient> getClientList() { return client_List; }`

Java Collection	`// MODULES USED:`
	`import java.io.IOException;`
	`import java.net.ServerSocket;`
	`import java.net.Socket;`
	`import java.util.ArrayList;`
	`import java.util.List;`

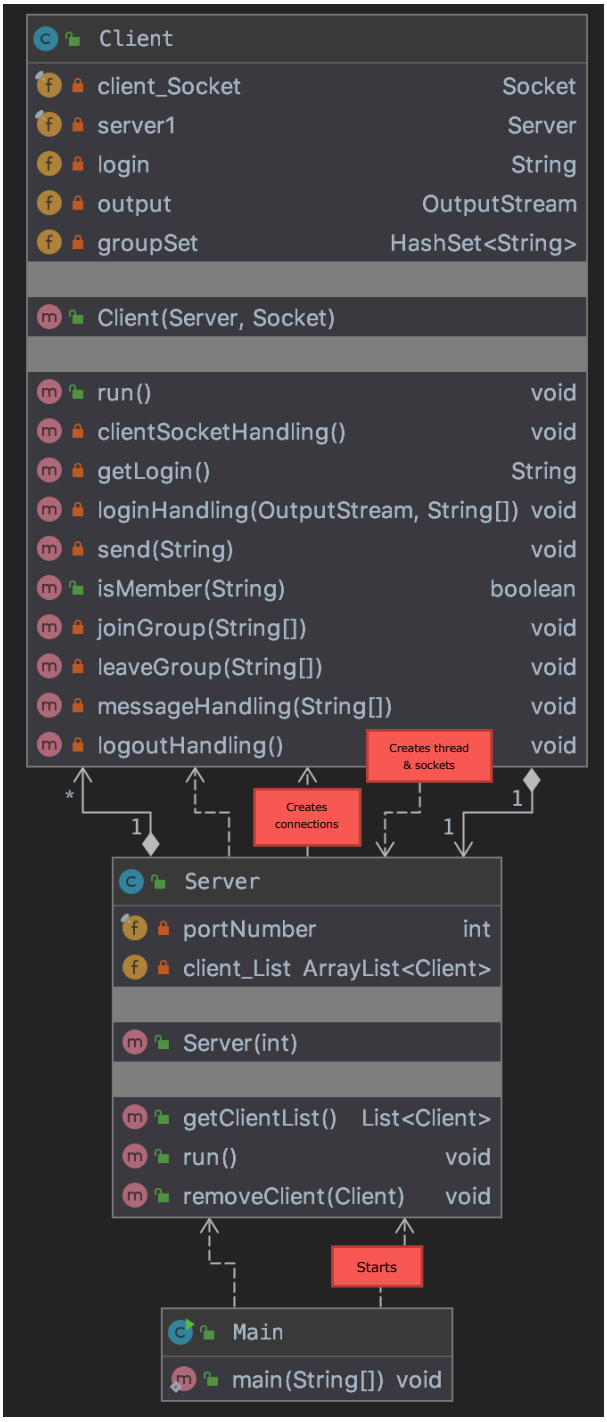
Exception Handling	`private void send(String message) throws IOException`
Interface, Inheritance, Polymorphism	Inheritance: `public class Server extends Thread { // VARIABLES private final int portNumber; private ArrayList<Client> client_List = new ArrayList<>();` *Extends Thread Class*

REQUIREMENTS	EXAMPLE
	Polymorphism:  *Overrides the run method*
Documentation Commenting	
Method Commenting	
Code Commenting	

MODULES & LIBRARY USED

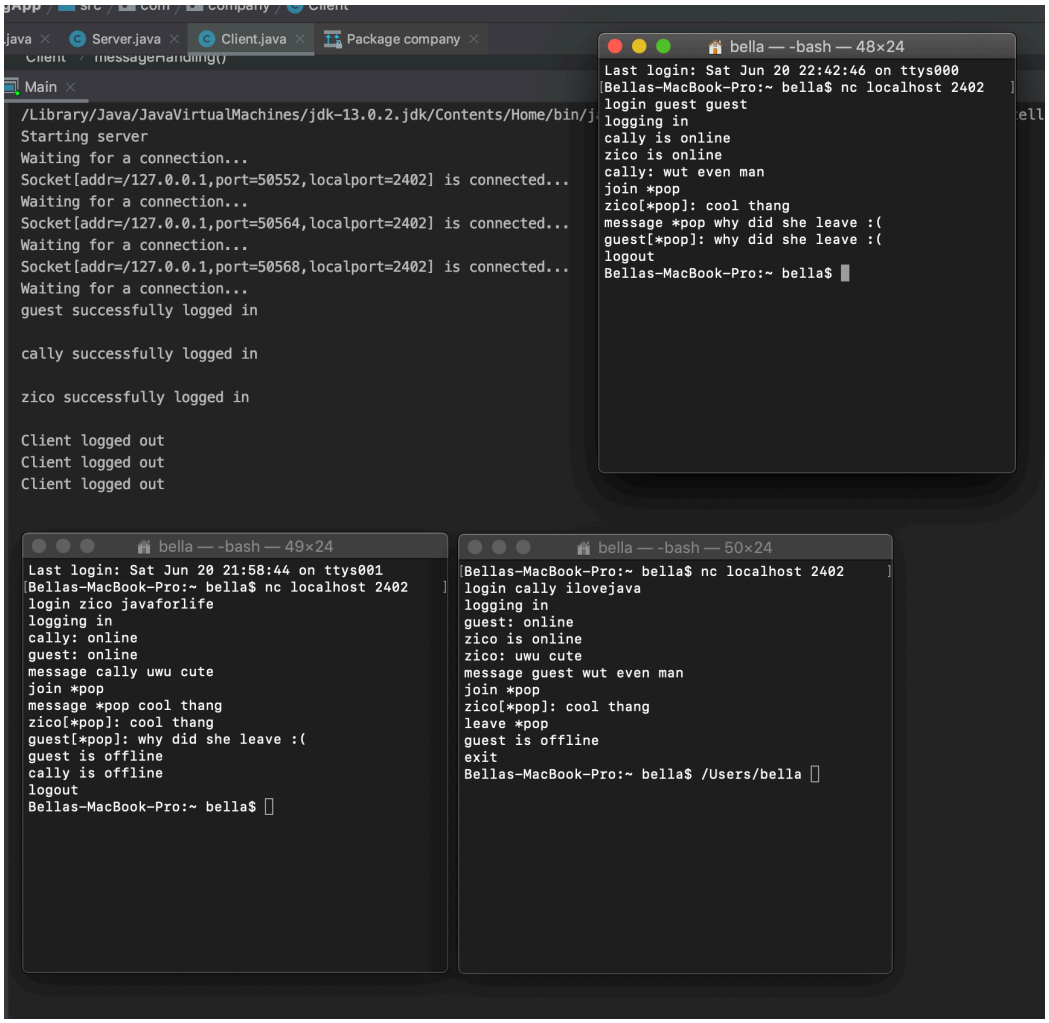
IMPORTED	USES
java.net.Socket & java.net.ServerSocket &	To create a Server and used to establish connections of clients via the internet
java.util.Array, java.util.List, java.util.Hashset	Used to store information: • keep track of online clients • existing group conversations / chatrooms.
java.io	Used to handle exceptions Used to take in data input (InputStream) & output (OuputStream) Used to read input
org.apache.commons.lang3 ***class:*** StringUtils	Used to split input into tokens. Tokens used for • initiating commands • getting the recipient • formatting message string

PROGRAM DESIGN



UML Class Diagram

EVIDENCE OF WORKING PROGRAM



This overall look of this project

Shows the main features: connecting to server, logging in, logging out, messaging one another, receving messages

HOW TO HANDLE CLIENT COMMANDS

```java
private void clientSocketHandling() throws IOException, InterruptedException
{
    if(Thread.interrupted()){throw new InterruptedException();}

    else
    {
        String line;

        // Reading and obtaining data or input from clients
        InputStream input = client_Socket.getInputStream();
        this.output = client_Socket.getOutputStream();

        // Bufferreader is used to read the data line by line
        BufferedReader reader = new BufferedReader(new InputStreamReader(input));

        // While loop to keep reading the input message per line until client types "exit"
        while((line = reader.readLine())!= null)
        {
            // if(Thread.interrupted()) { throw new InterruptedException(); }
            //else {}
            // Splitting our line into individual tokens (based on white spaces)
            String[] tokens = StringUtils.split(line);

            if (tokens != null && tokens.length >0)
            {
                // First token will be the one to initiate commands from clients
                String command = tokens[0];

                // COMMANDS:

                // If client types "exit", program will stop reading the line
                // Connection will be closed automatically by server
                if("logout".equalsIgnoreCase(command) || "exit".equalsIgnoreCase(command))
                {
                    logoutHandling();
                    break;
                }

                // If first index of what client types starts with login, they will be added to list of online clients
                else if("login".equalsIgnoreCase(command)) { loginHandling(output, tokens); }
```

OUTPUT:

```
bella — -bash — 50×18
Last login: Sat Jun 20 16:16:33 on ttys000
Bellas-MacBook-Pro:~ bella$ telnet localhost 2402
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
login cally ilovejava
logging in
zico is online
message zico hello uwu
zico: yo whatssup
logout
Connection closed by foreign host.
Bellas-MacBook-Pro:~ bella$
```

```
Bellas-MacBook-Pro:~ bella$ telnet localhost 2402
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
login guest guest
logging in
what even
Unrecognized command what even
```

To ensure the program will be able to do the things that the client wants it to do, it must understand certain command that the clients have inputted into their command line. To do this I have created a method called clientHandling() where we obtain client input, read it line by line and get the output to the server so we can display the correct responses to the client's commands.

To create the commands, we have simply created a String variable called line which will serve to store whatever it is the client typed into their terminal. Next we use a BufferReader and InputStreamReader to read the input and read it line per line. If the inputted string is not an empty string, the program will use the StringUtils class from apache to split this line into tokens. This is where we will be able to get our commands. From this point onwards, I've set our command to always be in the first position of our client's input, at tokens[0]. We then check to ensure that our token is not empty (null).

Using an if-else statement, we then further break down our program to detect different commands and execute different blocks of code as per desired by the client.

An example would be the logout or exit commands. We check to see if the first word (command) that the client has inputted is logout or exit. If it is, it will call the method to logout called logoutHandling(). Afterwards, it will break the program which will close the socket or connection with the particular client.

```
// Method that handles clients when they logout or quit the program
private void logoutHandling() throws IOException
{
    // Removes the client from the list
    server1.removeClient(this);

    // Iterating through the list of clients
    List<Client> client_List = server1.getClientList();

    // Notify other clients that someone logged out
    String status = login + " is offline\n";
    for (Client client : client_List)
    {  if(!login.equals(client.getLogin())) { client.send(status); } }

    client_Socket.close();
    System.out.println("Client logged out");
}
```

```
logout
Connection closed by foreign host.
Bellas-MacBook-Pro:~ bella$
```

Here, we can see that I've thrown some exception handling and called the .removeClient() which accepts parameter of the Client instance. We passed in this to indicate that we are talking about the current user in our program. We then called the .getClientList(), return a list of clients online at the moment and iterate through the list. When it finds the match to the current client, the program will change their status to offline by notifying the other users and printing a message on our console.

HOW TO HANDLE CLIENTS LOGGING IN

```java
// Method to handle the login process of clients
private void loginHandling(OutputStream output, String[] tokens) throws IOException {
    if(tokens.length == 3)
    {
        String username = tokens[1];
        String password = tokens[2];

        // Allows guests to join the server
        if(username.equals("guest") && password.equals("guest") || username.equals("cally") && password.equa
        {
            String message = "logging in\n";
            output.write(message.getBytes());

            // Setting the username of the client
            this.login = username;

            // Notifying that client has connected
            System.out.println(username + " successfully logged in \n");
```

The Login process is handled using if else statements, for loop and exception handling. For the parameters of this method, we pass the output variable to send appropriate response messages to the server. We also pass the String array of tokens to able to obtain information on the client's command, username, and password. If the username and password that the user passed is indeed part of the ones registered in our code, then we use the .write() method to send the server response and indicate to the user that the login was successful.

Output:

```
Waiting for a connection...
Socket[addr=/127.0.0.1,port=61025,localport=2402] is connected...
Waiting for a connection...
cally successfully logged in
```

```
login cally ilovejava
logging in
```

HOW TO HANDLE CLIENT'S STATUS

```
// Method created to enable clients to have access to other clients from the list
public List<Client> getClientList() { return client_List; }
```

OUTPUT:

```
guest: online
zico is online
```

```
// Iterating through the list of clients
List<Client> client_List = server1.getClientList();

// Used to notify new client, which clients are already online
for (Client client : client_List)
{
    // Filtering out current client as to avoid notifying themselves when they go online
    if(client.getLogin() != null)
    {
        if(!username.equals(client.getLogin()))
        {
            String onlineStatus =  client.getLogin() + ": online\n";
            send(onlineStatus);
        }
    }
}

// Used to send a broadcast message to all clients notifying that someone went online
String status = username + " is online\n";
for (Client client : client_List)
{  if(!username.equals(client.getLogin())) { client.send(status); } }
```

The status handling process is used to display which users are online and to notify other user if someone decides to logout or exit the chat program. To do that, in the client class, login method, we created a list of the Client instance and store it in a variable called client_List. For every login or connection that was made, they will be stored in this list. We then call on the getClientList method which we created in the Server Class. This method will return the client_list enabling all the clients to have access to one another throughout the entire program. This is also useful when it comes to sending and receiving messages. The next step will be to iterate through all the clients stored in the client_list and if a client has logged into the program, using an if- else statement, our code will then look to see whether the login data of the current client matches to that of the ones stored in the list, if it doesn't, it will print out an onlineStatus which is used to show the current users, the list of clients who are logged in or online. The for-loop code below is used to send a broadcast message or notification to other users that the current client is now online. We send and display these notifications by using a send method which will be discussed later.

HOW TO HANDLE CLIENT MESSAGING

Group Messaging:

```
// A method to access the output stream and send messages to the clients
private void send(String message) throws IOException
{ if(login != null) { output.write(message.getBytes()); } }


// Method to verify whether the group is part of the groupSet or exists in the server connection
public boolean isMember(String groupName){ return groupSet.contains(groupName); }


// Method to handle clients when they join a chatroom
private void joinGroup(String[] tokens)
{
    if (tokens.length > 1)
    {
        String groupName = tokens[1];

        // Set is used for storing client's memberships in each chatroom
        // Adding the group or chatroom to the set
        groupSet.add(groupName);
    }
}


private void leaveGroup(String[] tokens)
{
    if (tokens.length > 1)
    {
        String groupName = tokens[1];

        // Removing the group or chatroom from the set
        groupSet.remove(groupName);
    }
}
```

In the above snippet, we find several methods. The send method is one we will use to access the input and output streams of the server and deliver or prints out our notifications, serer responses as well as messages to the shell window. It does so by first taking in the parameter of String message which will store our response, notifications and messages. Next, it will check whether or not the current client is logged in. If they are and the method is called, it will overwrite the output to be our message.getBytes(). The reason we use .getBytes() is simply because all the data inputted y the client will be converted into bytes for them to travel across the internet and to access them and display them so in a way we will understand them is by converting these data back.

We also have 3 other methods: isMember(), joinGroup(), and leaveGroup(). The isMember() method is used to check whether the chatroom exists in the HashSet that we stored into called groupSet as well as check if clients are part of the chatroom. This will be useful when deciding whether or not to send the message later on in the messageHandling method.

Next up, we have the joinGroup() and leaveGroup() which are self explanatory as its name suggests. To handle clients joining the group, we can again made sure to pass in the tokens to gather and set information of the name of the chatroom which comes right after the command and therefore stored in the second index tokens[1]. After obtaining user input on the name, we then store this chatroom in the HashSet that we made earlier by using the .add() method. For the leaveGroup() method, it is basically identical to that of the join except that now, instead of adding, we will be removing the chatroom from the HashSet by using the .remove() method.

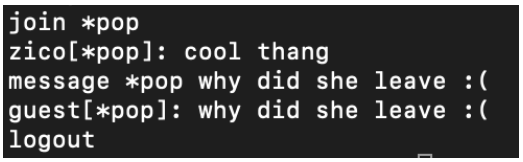
```
// Method to handle communications between clients
private void messageHandling(String[] tokens) throws IOException
{
    // The person who will receive the message
    String recipient = tokens[1];
    String text = tokens[2];

    // Checking to see whether client is direct messaging or chatting in a chatroom
    // Chatroom chat indicated by the '*' symbol
    boolean isGroupChat = recipient.charAt(0) == '*';

    // Iterating through the list of clients to find the recipient
    List<Client> client_List = server1.getClientList();
    for (Client client : client_List)
    {
        // Check whether conversation is really a group chat or not
        // Handles group messaging (sending messages to the group - broadcast messages to clients)
        if(isGroupChat)
        {
            // Checks to see if client is part of the chatroom
            if(client.isMember(recipient))
            {
                // Formats how the message will look like as well as specify the sender of the message
                String outputText = login + "[" + recipient + "]"+ ": " + text + "\n";
                client.send(outputText);
            }
        }
```

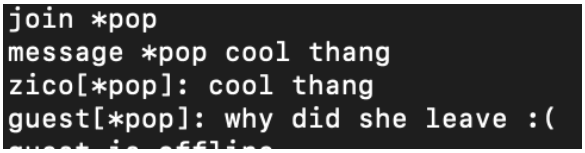
Here we can see the messagehandling method which takes care of both direct and group messaging. Here, we again pass in tokens as our parameters and get the data of our recipient as well as our message according to their index. To identify whether client is sending a message to a group conversation, I have created a boolean variable called isGroupChat which checks whether or not the first index of the token recipient is indeed "*". If it starts with "*" then it will return true and indicate that it is indeed a group convo. Next, we again call the getClientList to return the client_list

and use it to iterate through all the clients stored in the list just like we did in the login method.

Using an if-else statement, we can further confirm whether client is sending the message to the chatroom or directly. If the Boolean variable returns a true value, we can proceed to the next block of code which is another if-else statement to check whether client is a member of the group chat or whether they have joined or not. If they have, we can then send an output of the formatted string we have created for a chat conversation. We display the output by calling the send() method where we can access the output of server and write over it by using .write() and getBytes().

Output:



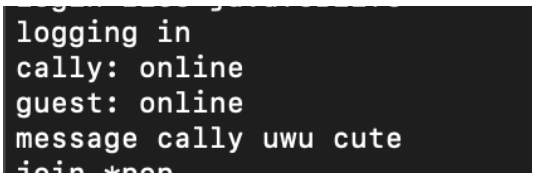


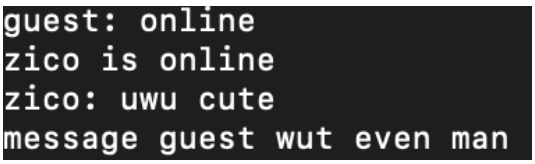
Direct Messaging:

```
// Handles direct messaging (sending messages from one client to another)
else
{
    // If the recipient is the same as that in the client list, message will be sent
    if (recipient.equalsIgnoreCase(client.getLogin()))
    {
        // Formats how the message will look like as well as specify the sender of the message
        String outputText = login + ": " + text + "\n";
        client.send(outputText);
    }
}
```

For direct messagin, we iterate through the list of clients and if the client in the list matches that of our recipient, we will then proceed to
send an output of the formatted string we have created for direct messaging, similar to the process of group messaging. We then display the output by calling the send() method as well where we can access the output of server and write over it by using .write() and getBytes().

Output:





REFERENCE & RESOURCES

- https://osxdaily.com/2018/07/18/get-telnet-macos/
- https://www.instructables.com/id/Creating-a-Chat-Server-Using-Java/
- http://commons.apache.org/proper/commons-net/download_net.cgi
- https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html
- http://commons.apache.org/proper/commons-lang/apidocs/org/apache/commons/lang3/StringUtils.html
- https://stackoverflow.com/questions/8660151/how-do-i-use-stringutils-in-java
- https://www.youtube.com/watch?v=-xKgxqG411c
- https://www.google.com/url?sa=i&url=http%3A%2F%2Fvietnamnews.vn%2Feconomy%2F260144%2Fsmartphones-surge-in-popularity-study.html&psig=AOvVaw1wo73M96XzT7bEV5JJwMII&ust=1592732722523000&source=images&cd=vfe&ved=0CAIQjRxqFwoTCJCC5fONkOoCFQAAAAAdAAAAABAY
- https://www.varonis.com/blog/netcat-commands/