Create a Secure Chatroom Using Java and Secure Sockets Layer

Prince Patel

Contents

[Introduction 3](#_Toc69733773)

[Creating a Secure Chatroom Using Java and Secure Sockets Layer 4](#_Toc69733774)

[Create a Glassfish Server 4](#_Toc69733775)

[Generating a Key 5](#_Toc69733776)

[Trusting the Certificate 5](#_Toc69733777)

[Applying the SSL 6](#_Toc69733778)

[Generating a Java Chatroom 6](#_Toc69733779)

[Conclusion 11](#_Toc69733780)

[**References** 12](#_Toc69733781)

# Introduction

One of the most important things to the users of the Internet is confidentiality. This means that they should be able to browse the web without having to worry about their personal information being intercepted by other users who have malicious intent. It is important that anyone that is interacting/communicating over the Internet has a secure way of relaying information to the intended user in a way that cannot be intercepted or spoofed by an adversary; especially when sensitive information is being exchanged. This report will demonstrate the implementation of Secure Sockets Layer (SSL) in a chatroom created using Java, the limitation that the program may have, and an overview of the program itself.

# Creating a Secure Chatroom Using Java and Secure Sockets Layer

## Create a Glassfish Server

The first step in making a chatroom secure, we must first establish connections and certificates using SSL. First, we must create a local server on our machine in order to establish a connection between the source and destination chatrooms. Using the guide provided in the instructions, an installation of Java Development Kit Version 8 was necessary. The next step was to actually download the Glassfish application to create a server. During this phase, it was quite challenging to overcome the errors that were being created. With extensive methods using trial and error, the server was able to be ran successfully.

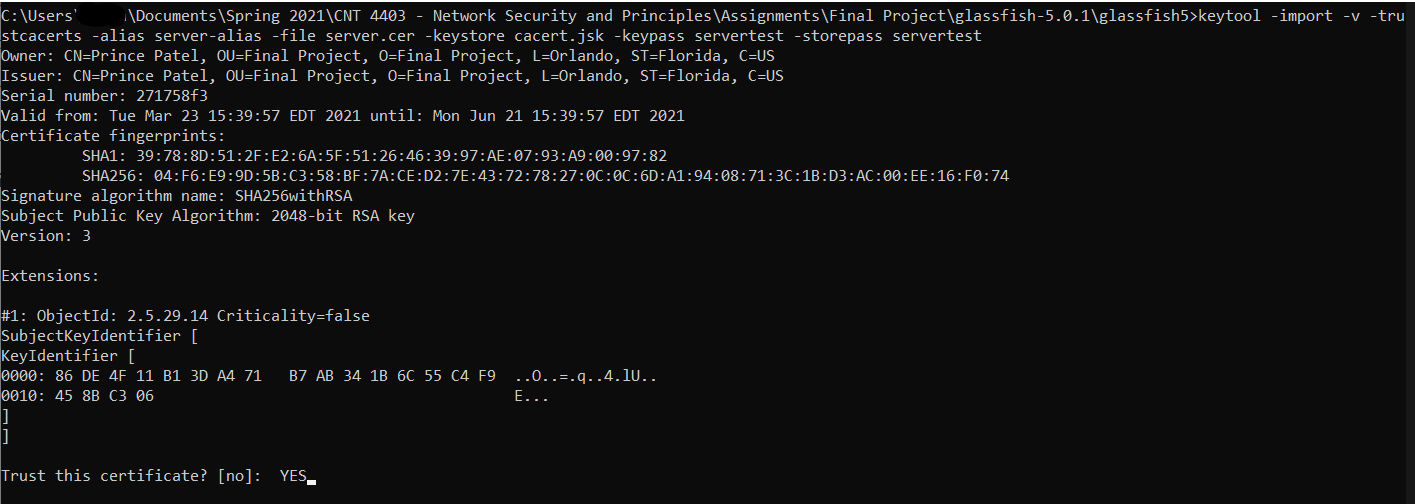
To being, the NetBeans IDE was actually necessary to actually used to create a server. NetBeans allows services to create a running server using their IDE, which is needed because the server utilizes the repositories of Glass Fish 5 which coincides with Java EE8. Thus, a .bat file was created to allow a simple transition in the command-shell from the current version of Java that was installed on the CPU to the previous Java EE8 version. To ensure that a Glass Fish server is running successfully, simply go to a web browser and type “**localhost:[portnumber]**” to navigate to the Glass Fish console. If the console displays Glass Fish server properties, then the server has successfully been created and is running. Once the Java version is switched to the appropriate one that works with Glass Fish 5 and a Glass Fish server is running using NetBeans, a command must be executed in the command-shell to begin the process of created an SSL certificate.

## Generating a Key

In order to establish a secure chatroom, we must first ensure a SSL connection with the corresponding certificates. This can be done by launching the location of the glass in a command-shell and then executing the “keytool” command provided in Java Code Geeks reference. Essentially what this command does is ask some basic information about the user and the organization (if applicable) and then generates certificates. Once the **keytool** is used to generate a certificate, the certificate must then be exported. This can be done by using the **keytool** command in the command-shell once again, however this time it must be the “**export**” command must be executed.

## Trusting the Certificate

Once the certificate is exported, the user must trust the certificate. A trust must be placed on certificates created because this will assist in the process of authentication for future reference. In order to do this, we can use the keytool feature, however in this instance we can use the “**trustcacerts**” command. This command essentially defines is function: trust Certificate Authority certificate. Once this command is running, it will output various information about the certificate that we created above. It will also input the fingerprints of the certificate in both SHA1 format and SHA256. At the very bottom, the user will be asked if they would like to trust this certificate. In this case the user would type out “YES”. Below is the example that was used for the purpose of this project.



## Applying the SSL

At this point, we have the majority of the steps completed to actually generate an SSL certificate. However, we must now ensure that our server is able to reach this certificate and actually apply this. In order to do so, the user must access the server console from the web-browser. This can be done by typing “**localhost:4848**”. The configuration for the port number can be altered, however we will keep it the “4848” for this project. Once at this website, we must add the SSL certificate name to the “http-listeners-2” configurations. Simply input the certificate alias that was created before.

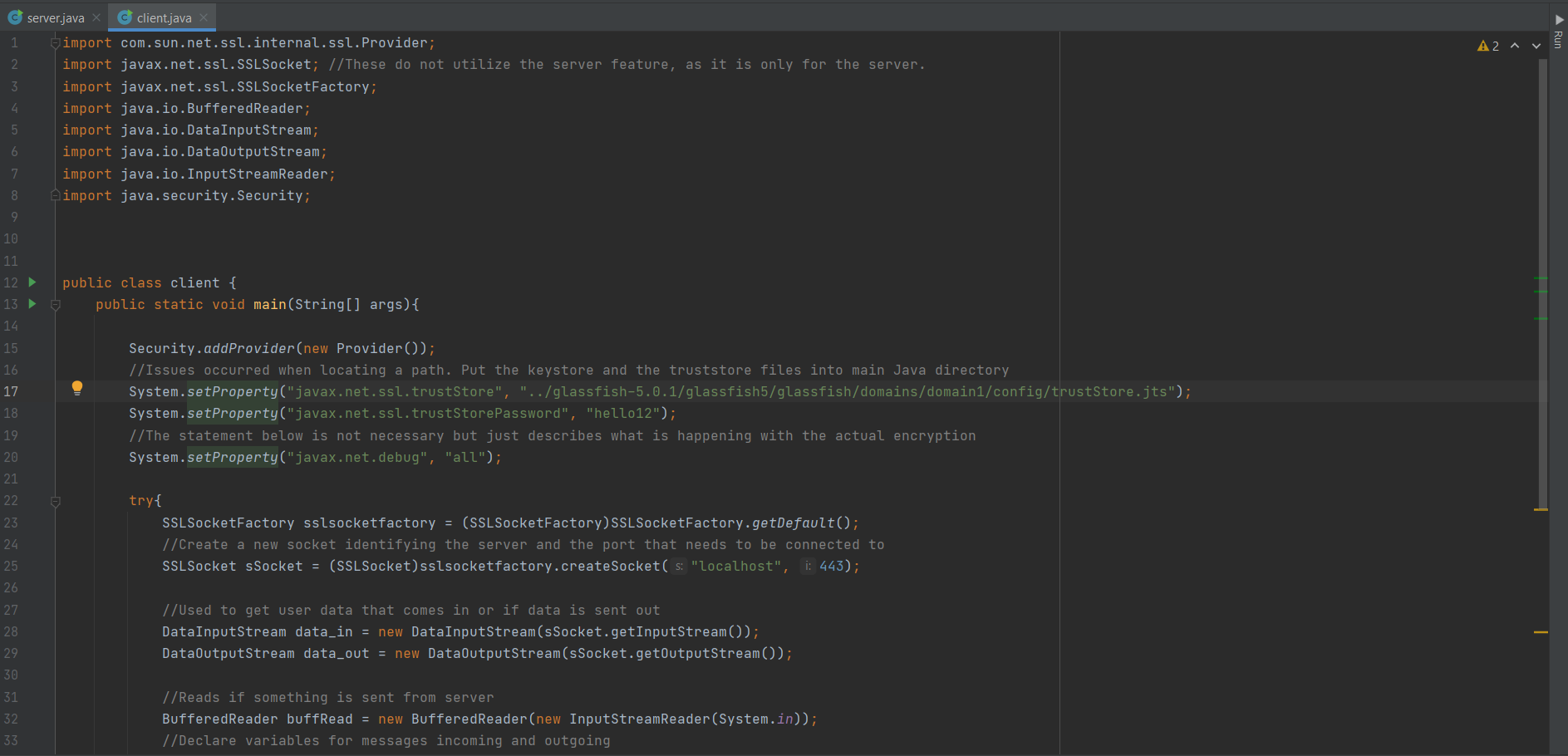
## Generating a Java Chatroom

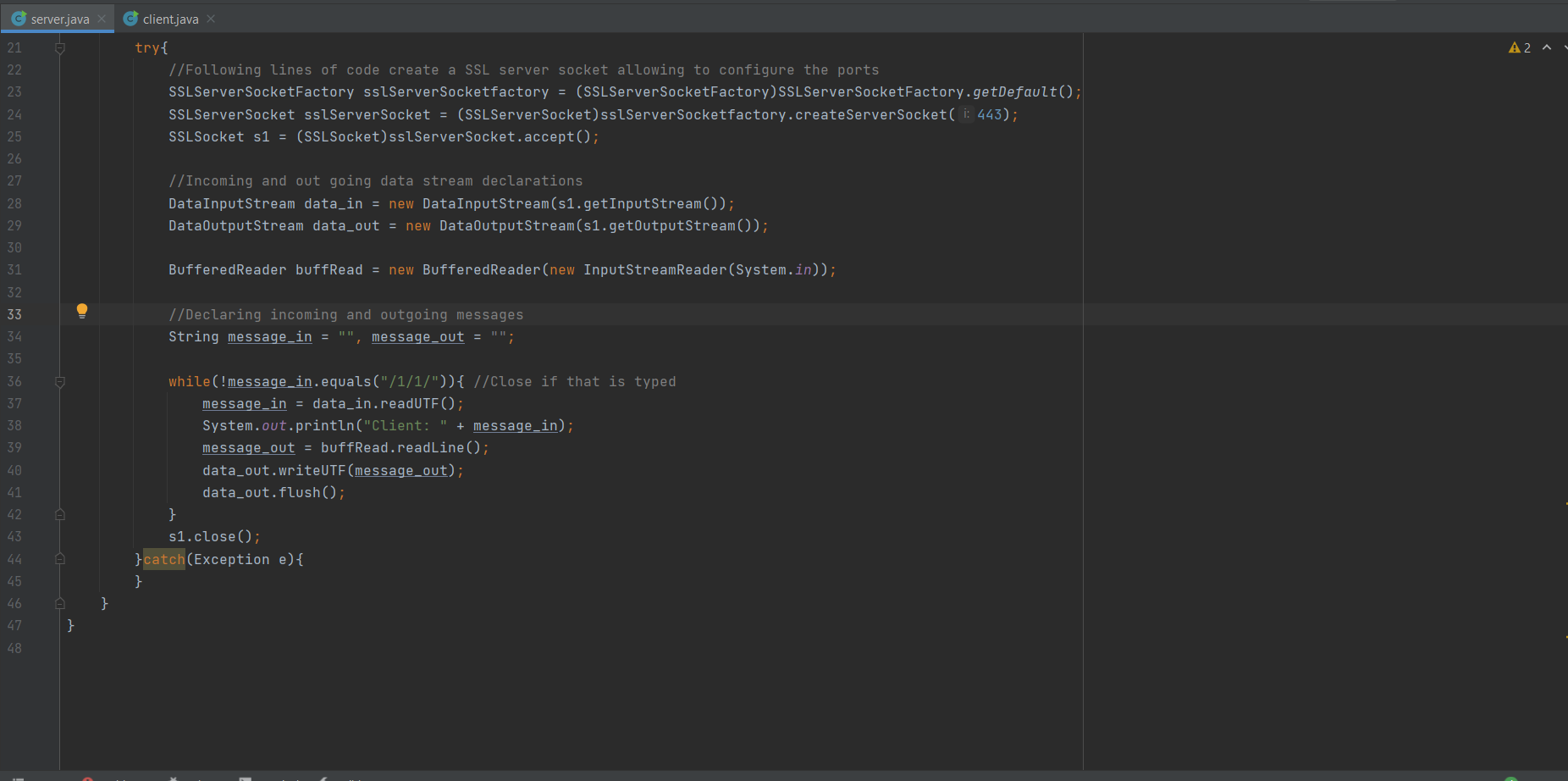
In order to generate a Java chatroom, I utilized the references mentioned below and those that were provided in the rubric for the project. Using this information, I was able to generate a Java application that is able to connect to a server and utilize ports and the SSL certificate we created above to have a secure means of communication. The information in this section will demonstrate how the Java application works and how it encrypts the data.

Some complication that I ran into when generating the application had to do a lot with the Java version I was using. The reference instructed me to use a specific Java library that was not available in the newer version of Java. Thus, I spent hours investigating why I was not able to use this information. To be more precise, the Java library I was not able to use was “**import com.sun.net.ssl.internal.ssl.Provider;**” I then was able to resolve this issue by compiling the code with JDK8.

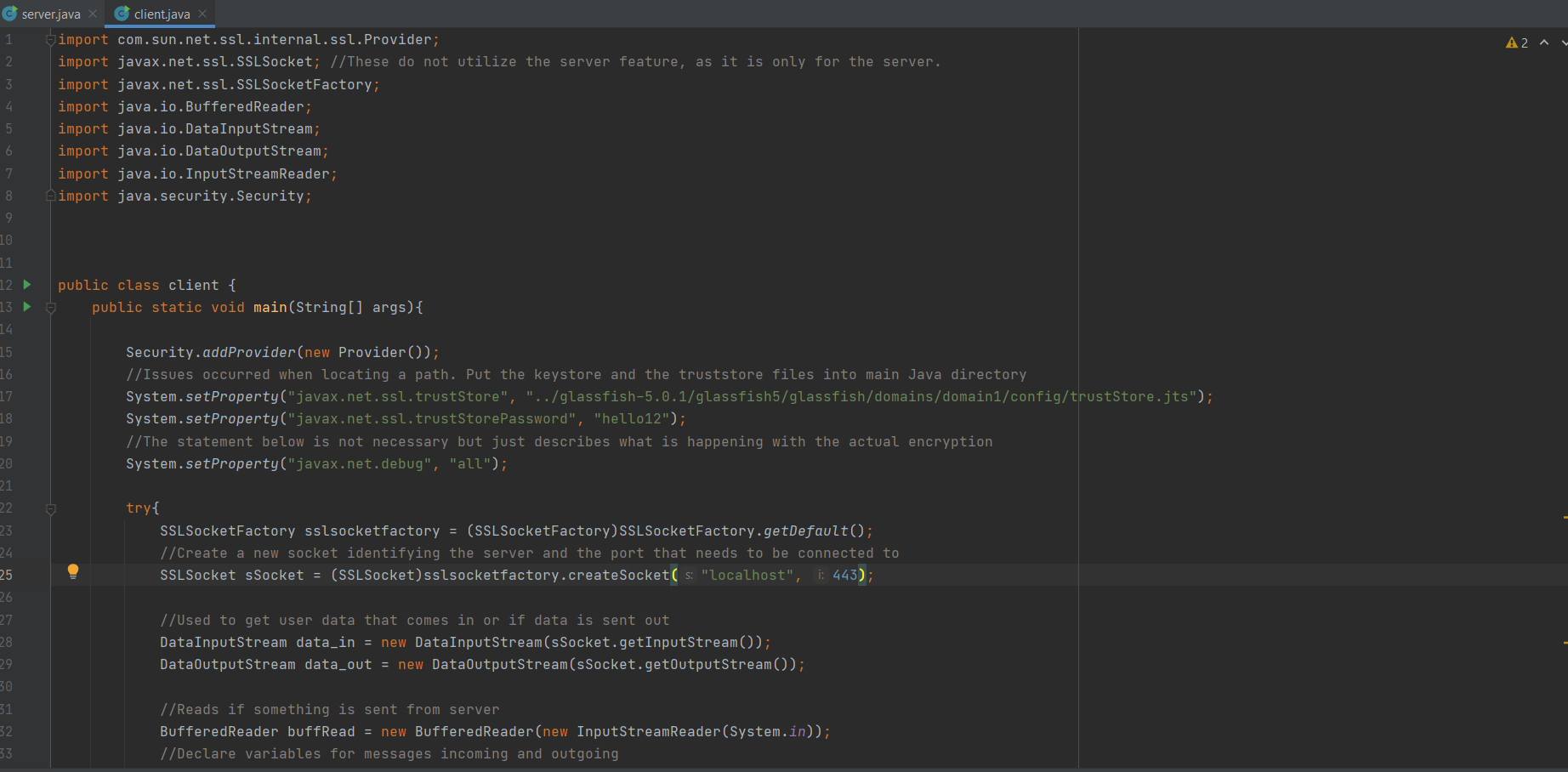
Below are images of the Java code that I developed with assistance to the references provided in the rubric and those I found on my own. There are comments that I have wrote demonstrating what important lines of code do.

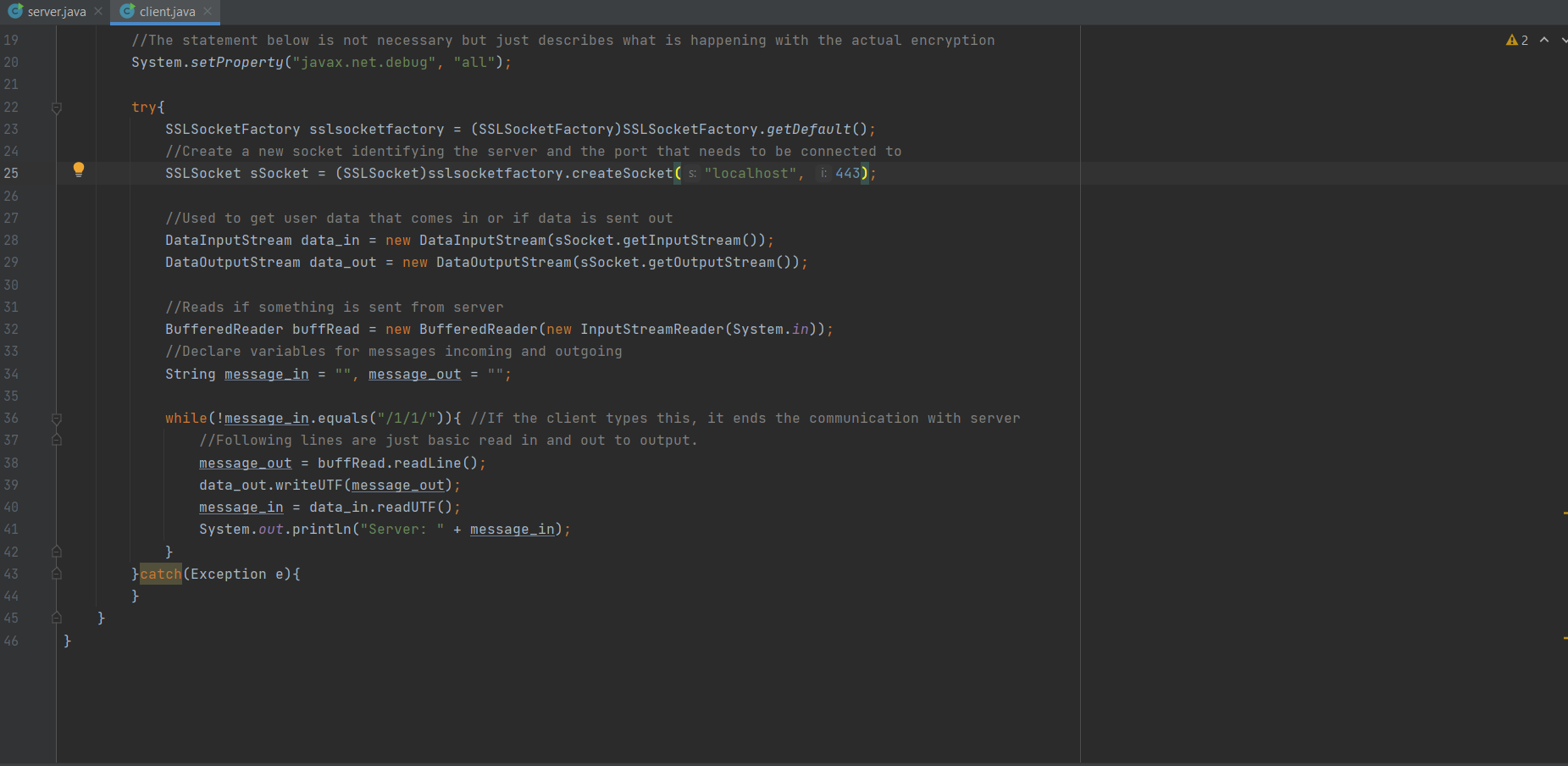
**SERVER**



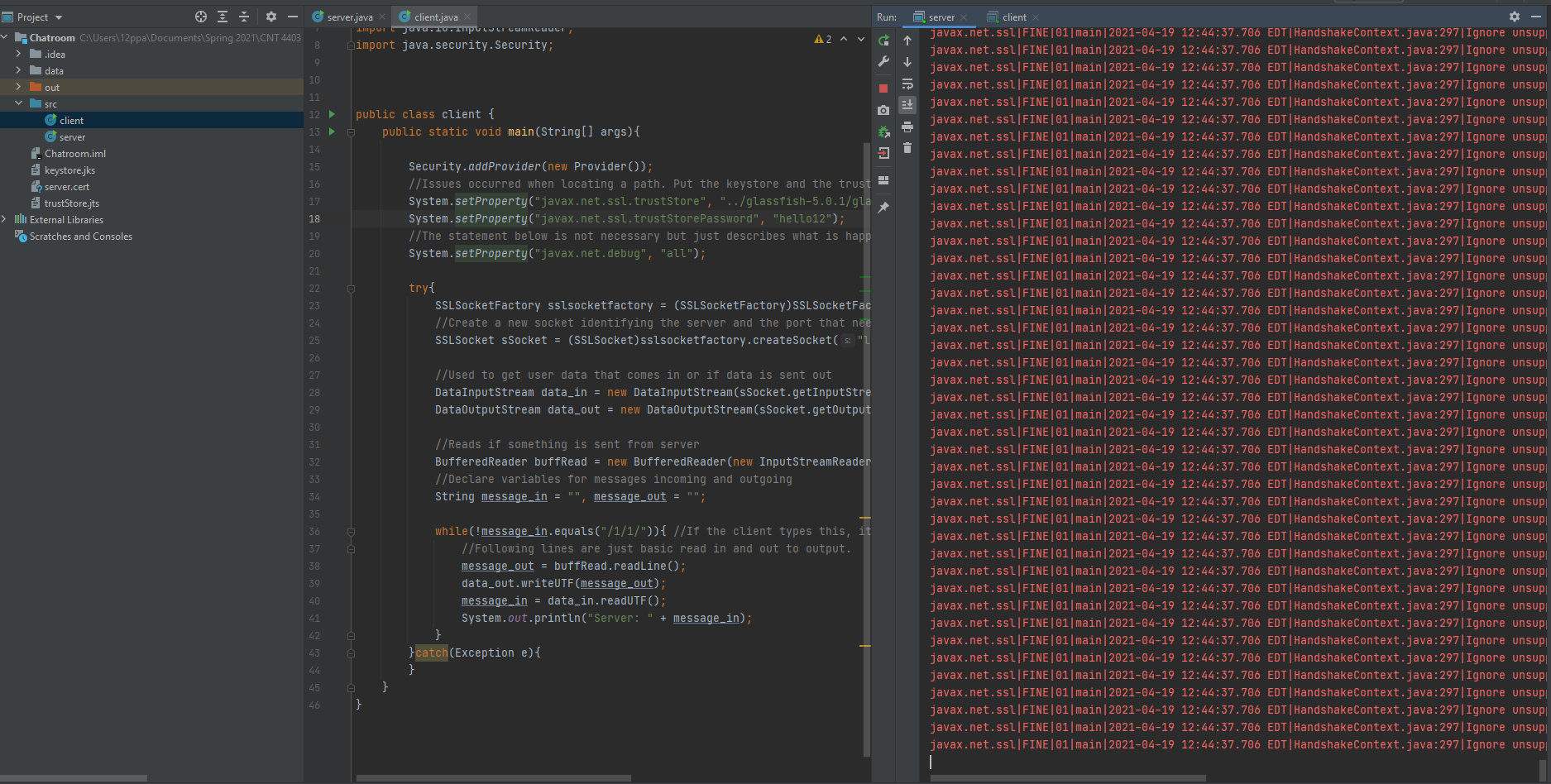


**CLIENT**

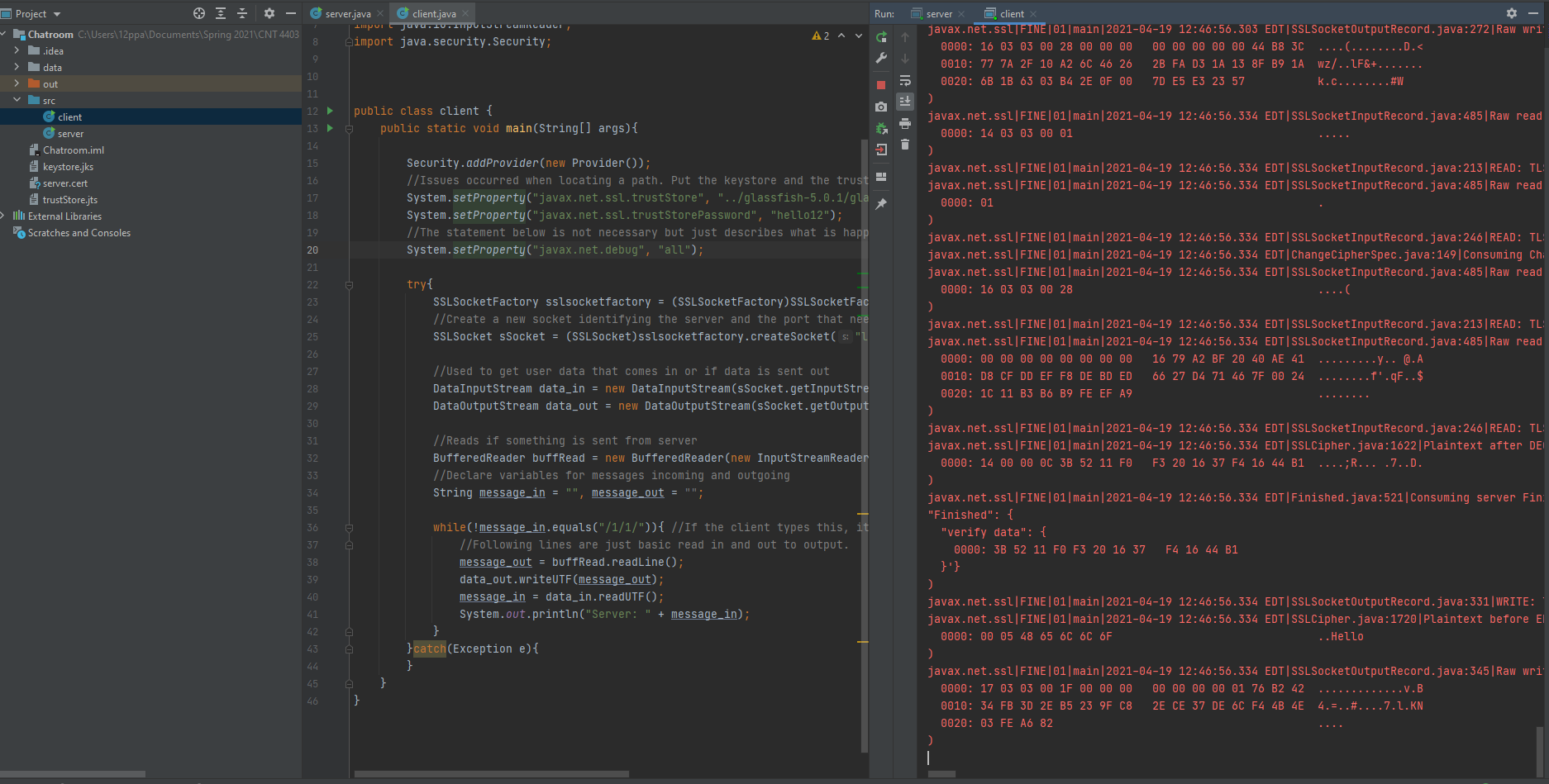
****



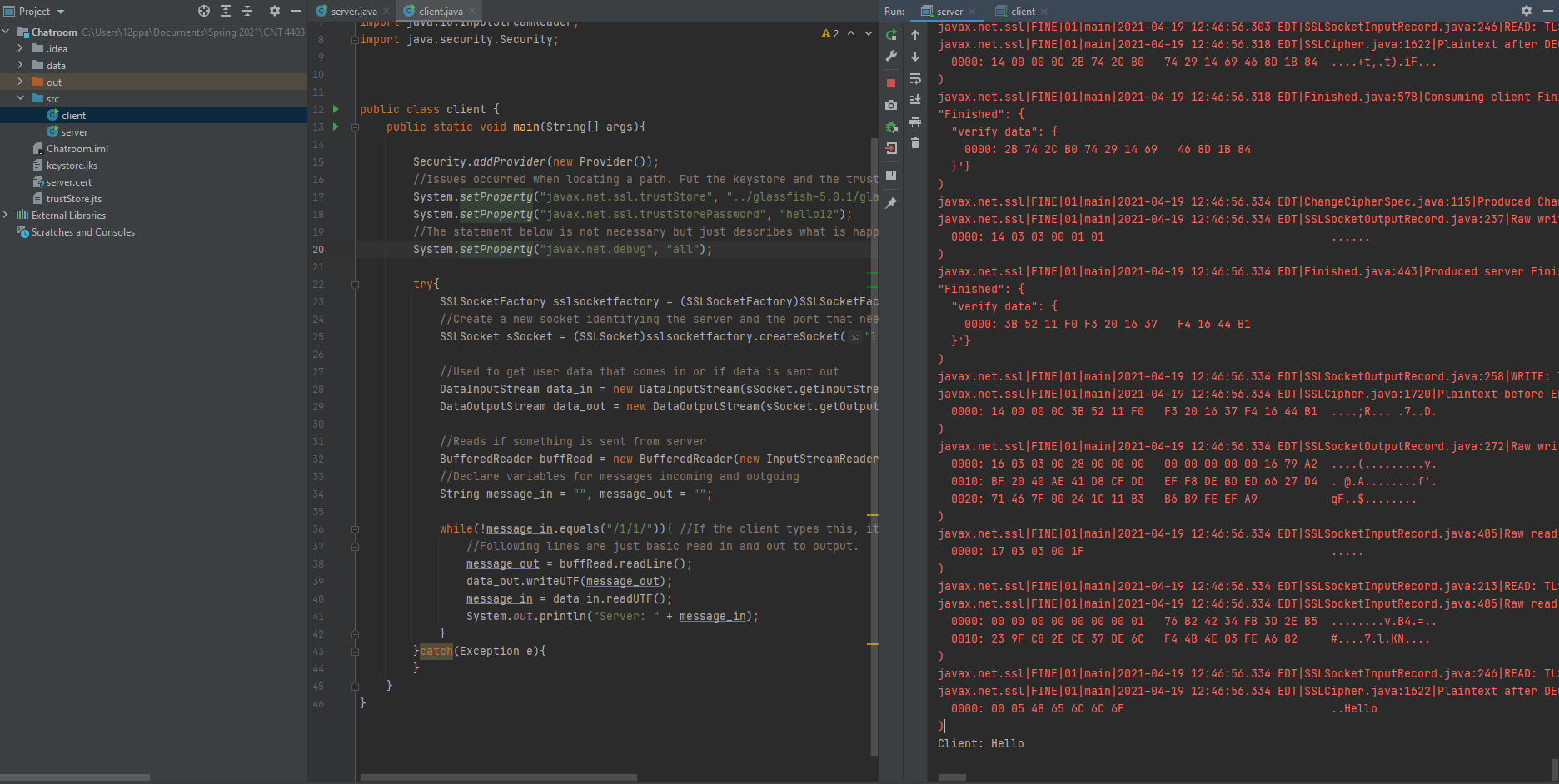
The way this program works is by first running the **server.java** file. Then the **client.java** file must be ran enabling it to connect to the server and the designated ports. Once that the server and client programs are running, the output window shows each step of the code execution and it clearly shows the encryption part of the communication. The images below will demonstrate this.



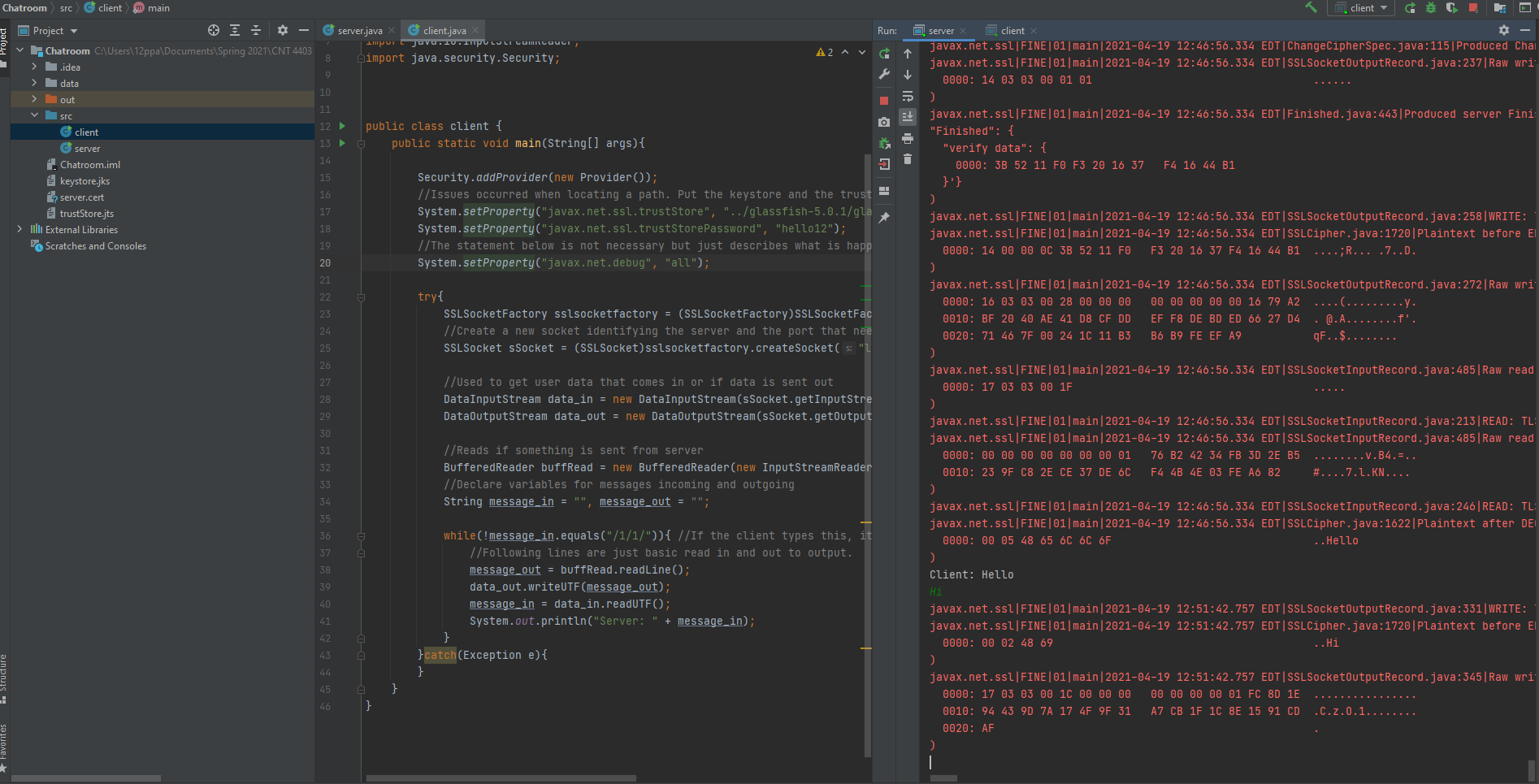
Once the client sends a message, it will use the SSL information to encrypt the message As shown below.



The image below will demonstrate what the server sees once the message is received.



The following image demonstrates what happens when the server replies to the client.



## Conclusion

To develop a secure chatroom that uses Java and the Secure Sockets Layer protocol, there are many steps that need to be taken. As mentioned in this paper, a server must be created first along with using the keytool Java feature that allows for key generation for the SSL certificate. Then a Java program must be created to implement the SSL certificate and keystore. The steps along with the comments are present in the source code of the file which show the steps to implement this. Although I encountered many issues in the implementation process, the learning experience to develop this program was phenomenal.

References

* <https://www.javacodegeeks.com/2013/06/java-security-tutorial-step-by-step-guide-to-create-ssl-connection-and-certificates.html>
* <https://www.youtube.com/watch?v=kqBmsLvWU14>
* <https://www.youtube.com/watch?v=uYRTpMGdf1g>
* <https://www.youtube.com/watch?v=uxWlxhrwRkA>