



Tecnológico de Monterrey

Final project

Programming languages

Benjamín Valdés Aguirre

Multi-user chat

Diana Estefanía Ortiz Ledesma

A01209403

[Abstract](#)

[Introduction](#)

[Objective](#)

[Scope of the project](#)

[Context of the problem](#)

[Features](#)

[Architecture](#)

[Implementation](#)

[Paradigm](#)

[Solution](#)

[Results](#)

[Conclusions](#)

[Setup instructions](#)

Abstract

In this digital age, the internet has become the main communication tool. It provides many options to keep us in contact with our family and friends. But usually, the most used applications are for instant messaging. This project shows the impact of these applications on our lives and how they work. The project is an implementation of a chat application based on Java. The application allows clients to transfer messages between only two clients and in group chats. Java, multithreading, and client-server concepts were used to develop this project.

Introduction

Chat applications and messaging apps have become popular since the appearance of smartphones. According to the latest messaging app usage statistics, the most used messaging app worldwide is WhatsApp, with 2.0 billion users. But there are numerous chatting applications available in the app stores. They offer different features such as group chat, the exchange of videos, images, documents, and even audio messages, as well as stickers, emoticons, and gifs. People find an application that works better for themselves and start using it. But not everyone knows and understands how the application works.

Objective

- Implement a chat system.
- Allow for private messaging.
- Allow group chat.
- Enable an easy method of communication.
- Enable an easy understanding of the chat applications.

Scope of the project

This project will have an impact for the people looking to understand the basic components of a chat application infrastructure. And it also provided a basic chat application that can be easily taken as a base project to make improvements for developing a better version of the application with additional features.

Context of the problem

Sometimes learning it's hard, because in the process we make mistakes or we don't know where to start, and then our confidence erodes. Understanding the system design and architecture of chat applications can be intimidating. But it also can be very useful, especially if you want to apply the concepts in your own projects. So let's get started.

Features

Communication between client and server

❖ Login

Typically, identifying oneself to a system is accomplished by entering one's username and password.

❖ Logout

Exit an account.

❖ Status

Once you login to your account, it will give you a welcome message that means you're logged in.

Communication between server and client

❖ Online

Send online messages when other users are online when you connect to the server and also send messages when a new client is connected.

❖ Offline

Send offline messages when other users disconnect from the chat.

Communication between client and client

❖ Private messaging

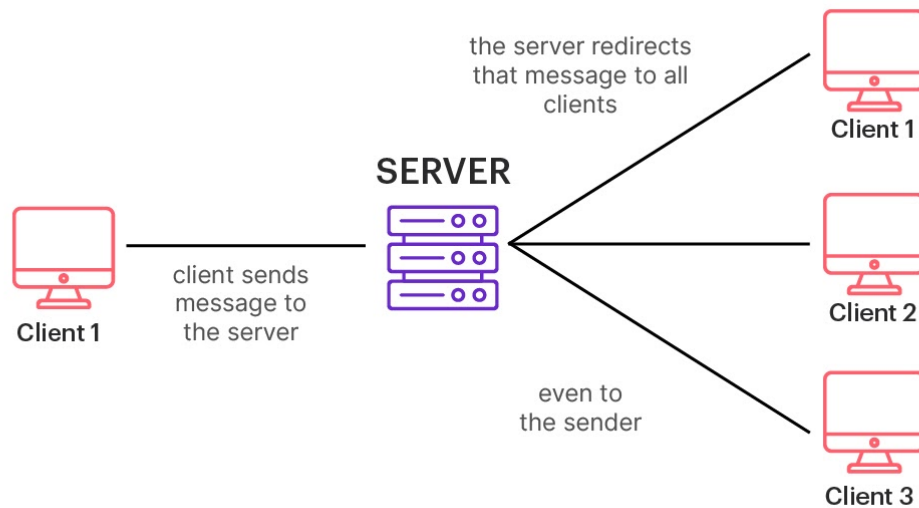
This is a private communication channel between two users where they can both send and receive messages simultaneously. In this project, this is achieved with the use of the Java multithreading concept.

❖ Group chat

This type of communication allows you to chat with multiple people at the same time. In this project, this is achieved when the users join a group chat and they start receiving messages from other users in the group.

Architecture

The design of a chat application can be broken down into two components: the client and the server.



cometchat

Figure 1. Client-server architecture

1. **Client**

When we talk about world Client, it means to talk of a person or an organization using a particular service. Similarly in the digital world a Client is a computer (Host) i.e. capable of receiving information or using a particular service from the service providers (Syed Modassir Ali, 2019).

2. **Server**

Similarly, when we talk about the word Servers, it means a person or medium that serves something. Similarly in this digital world a Server is a remote computer which provides information (data) or access to particular services (Syed Modassir Ali, 2019).

Implementation

According to this architecture, the client makes requests for data to the server and the server responds back as long as it's present in the database. But the server can't push data to the client if the client doesn't make a request. Unfortunately, for a chat application, this quickly causes inefficiency since the client would need to be asking the server for new data regularly. For the implementation of this project I used something called *sockets*.

Sockets

They are a mechanism that helps us establish a persistent link between the client and the server that runs independently of each other. Through the library *java.net*, Java provides us with two classes: *Socket* to implement the connection from the client side, this and *ServerSocket* that will allow us to manipulate the connection from the server side. Using sockets, the server can send data to the client without needing a request. That's how we solve the problem of inefficiency.

However, now we have another problem to solve. The application will be running on one thread on the server, processing the client connection, but it will block until this connection ends. So we will not be able to communicate between different clients, just the server and the client. And for the purpose of this project, that's not enough. So we need to use the multithreading concept.

Multithreading

When an application that can be used by multiple clients simultaneously is created, it's very common to use multithreading. Because we can assign a thread for each one of the instances that interact with the system. And following the implementation of this project, we could have many clients connected to the server. Although we need to be careful with the stability of the system and the consistency of the information. But what is behind this?

Paradigm

Concurrency is the ability to do more than one thing at the same time. In this digital world, that means a system can process more than one task of execution at the same time. For this, it is not necessary to have several CPUs. It can occur in a single-processor system since concurrency exists if the tasks coexist at the same instant of time or at least seemingly at the same time.

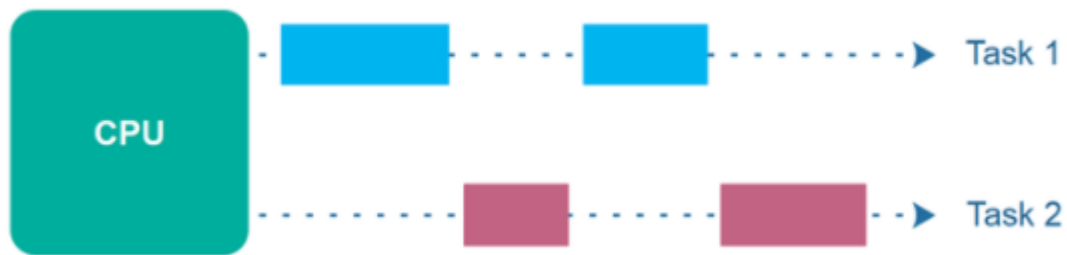


Figure 2. Concurrency diagram in one CPU

In Figure 2, we can see how concurrency works with two tasks in progress inside the application. The CPU switches between them during execution. This is known as **“time slicing”**. Furthermore, concurrent programming has two models: shared memory and message passing.

Shared memory

Means that concurrent computations communicate through reading and updating a shared location in memory. For example, two programs running on the same laptop share a common filesystem with files they can read and write. An advantage of this model is that it is faster, but it also may create problems such as synchronization and memory protection.

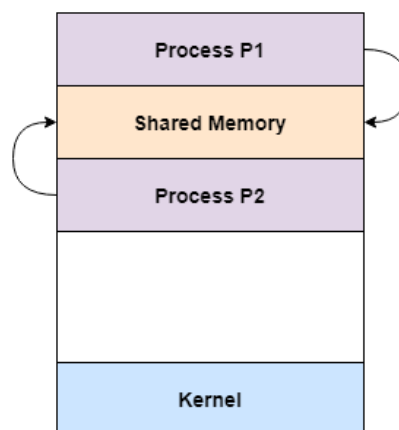


Figure 3. Shared memory diagram

Message passing

Means that concurrent modules interact through a communication channel, sending messages to each other. For example, two computers on a network communicate by network connections. An advantage of this model is that it makes it easier to build parallel hardware because of the tolerance of higher communication latencies. But it also has slower communication since the connection setup takes time.

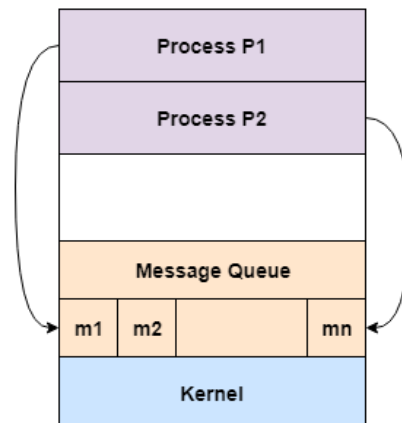


Figure 4. Message passing diagram

The concurrent modules themselves come in two different kinds: *processes and threads*.

Process

A process is an instance of a running program that is *isolated* from other processes on the same machine. In particular, it has its own private section of the machine's memory. The process abstraction is a *virtual computer* (Saman Amarasinghe, 2014).

Threads

A thread is a locus of control inside a running program. Think of it as a place in the program that is being run, plus the stack of method calls that led to that place to which it will be necessary to return through. The thread abstraction represents a virtual processor (Adam Chlipala, 2014).

Now that you know the basic concepts of this paradigm, we need to say that concurrent programming is hard. Despite its presence in so many applications, it can have subtle bugs. Like **race conditions**, where two or more threads can access the same data and try to **change** it at the same time. This is very hard to discover when testing because these bugs are **heisenbugs**, which means that they are nondeterministic and very difficult to reproduce. It's hard to make them happen the same way twice. For you to code a program that is safe to call by multiple threads simultaneously, it is important to know what resources are shared when executing.

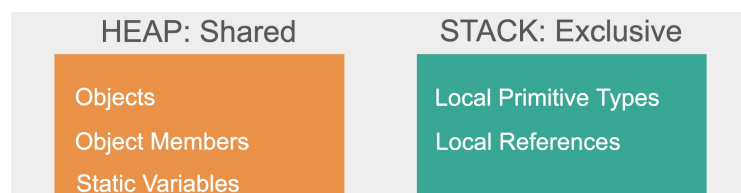


Figure 5. Shared and exclusive resources diagram

Local variables

They are stored in each thread's own stack. So they are never shared between threads.

Local object references

The reference itself is not shared, but all objects are stored in the shared heap. To be thread safe the object created locally never escapes the method it was created in.

Object member variables

They are stored on the heap along with the object.

If you want to determine if your code is thread safe you can use the thread control escape rule:

"If a resource is created, used and disposed within the control of the same thread, and never escapes the control of this thread, the use of that resource is thread safe" (Jakob Jenkov, 2015).

In conclusion, and in simple terms, concurrent programming is one way in which we can solve problems. Executing multiple tasks at the same time and not sequentially so one task can continue without the need for others to start or end. And while it is true that it could cause some problems, it is also true that if it's implemented correctly, we can improve the performance of our applications.

Solution

At this point, you are ready to understand the solution to this project. First of all, I decided to implement my program in Java because the platform is designed from the ground up to support concurrent programming. Also, it is really easy to implement threads and sockets. And these three concepts are needed to implement this solution.

Since the project is about a multi-user chat application, we need to start implementing the network server. This means we have to use the socket and socket server to create the link of communication. The server socket is going to be listening on a port (in this case, 8818) and it will need a new socket that receives another port number to establish communication and continue listening on the original port and serving the connections. We used stream sockets because of the Transmission Control Protocol that guarantees application programs and computing devices can exchange messages over a network.

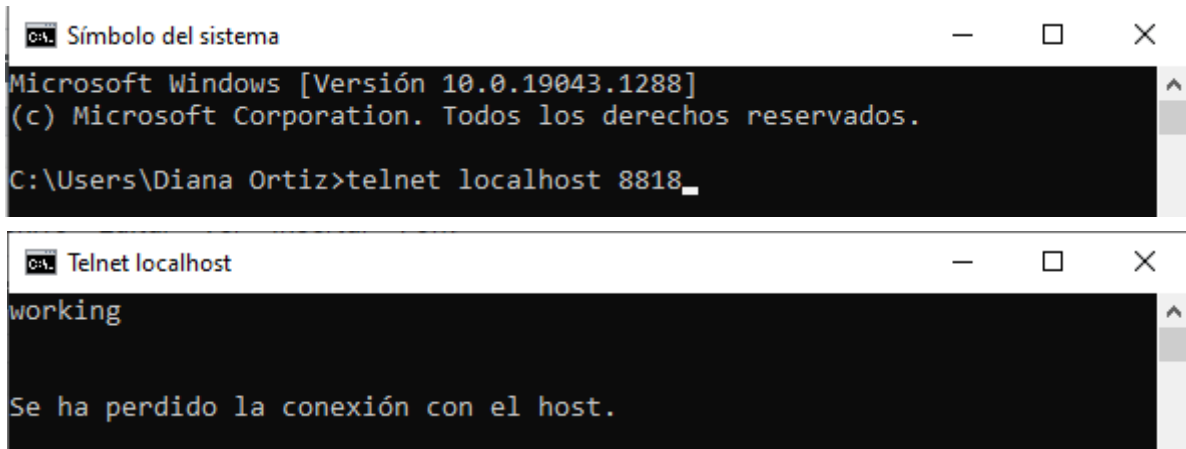
The class server will be running in an infinite loop so it can keep accepting requests from the clients. And once a request comes in, it will assign a new thread to that client to handle the communication. It will also store the client in an ArrayList. This way, they can be used to deliver messages. And when a client logs out, it will remove the instance of that client from the list.

The class client is a thread that handles the client sockets. We're going to instantiate the server so we can pass this server instance to each client and they can get all the other instances to communicate with each other by iterating through the list. We are also going to implement all the protocols for the chat application. In order for us to do this, we need to establish bidirectional communication using the InputStream to read the data and the OutputStream to send the data. Using a buffered reader, we can read line-by-line all the inputs. Since the application works with commands, we need to split the input into different string tokens using `StringUtils.split`, and after this, we can implement all the if cases with the corresponding methods for the commands we are using: `logout`, `login`, `message`, `groupchat`, and `leave`. And the last case where the application doesn't recognize the command. The current socket will close when the user logs out.

For the main class, we only instance the server, define the port, and start the server (main thread).

Results

1. Beginning of the implementation for a chat server where the client connects to the server, shows a message and closes the connection.

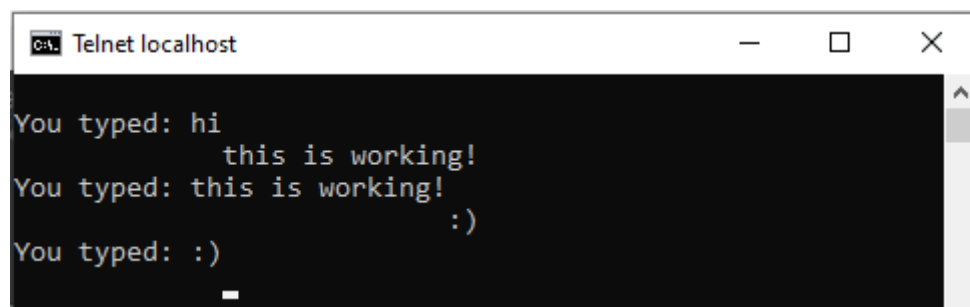


```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19043.1288]
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\Diana Ortiz>telnet localhost 8818_

Telnet localhost
working

Se ha perdido la conexión con el host.
```

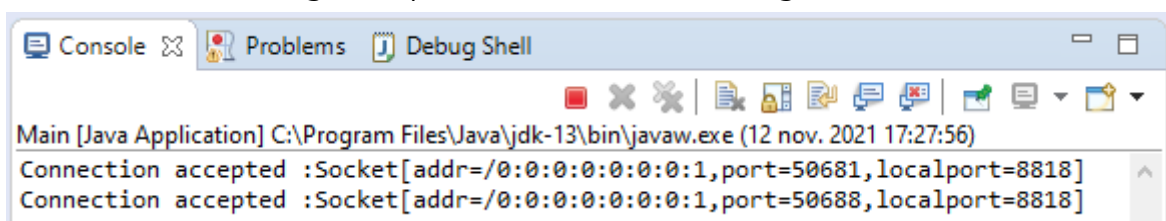
2. Bidirectional communication is implemented in the first version of the chat server.



```
Telnet localhost

You typed: hi
           this is working!
You typed: this is working!
           :)
You typed: :)
           _
```

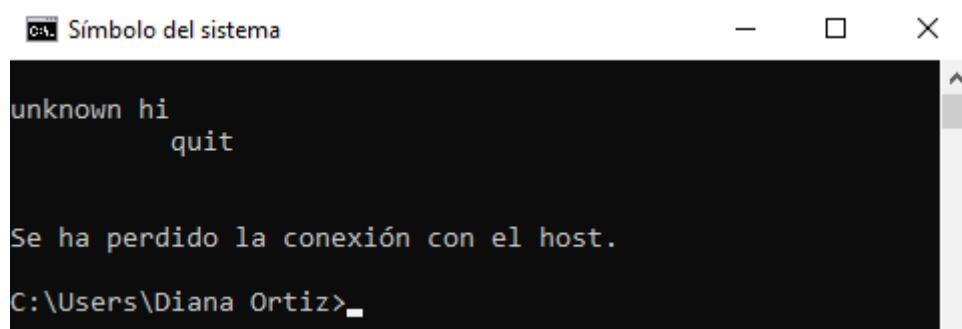
3. Multithreading is implemented and working.



```
Console Problems Debug Shell

Main [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (12 nov. 2021 17:27:56)
Connection accepted :Socket[addr=/0:0:0:0:0:0:0:1,port=50681,localport=8818]
Connection accepted :Socket[addr=/0:0:0:0:0:0:0:1,port=50688,localport=8818]
```

4. First command "quit" is implemented and working.



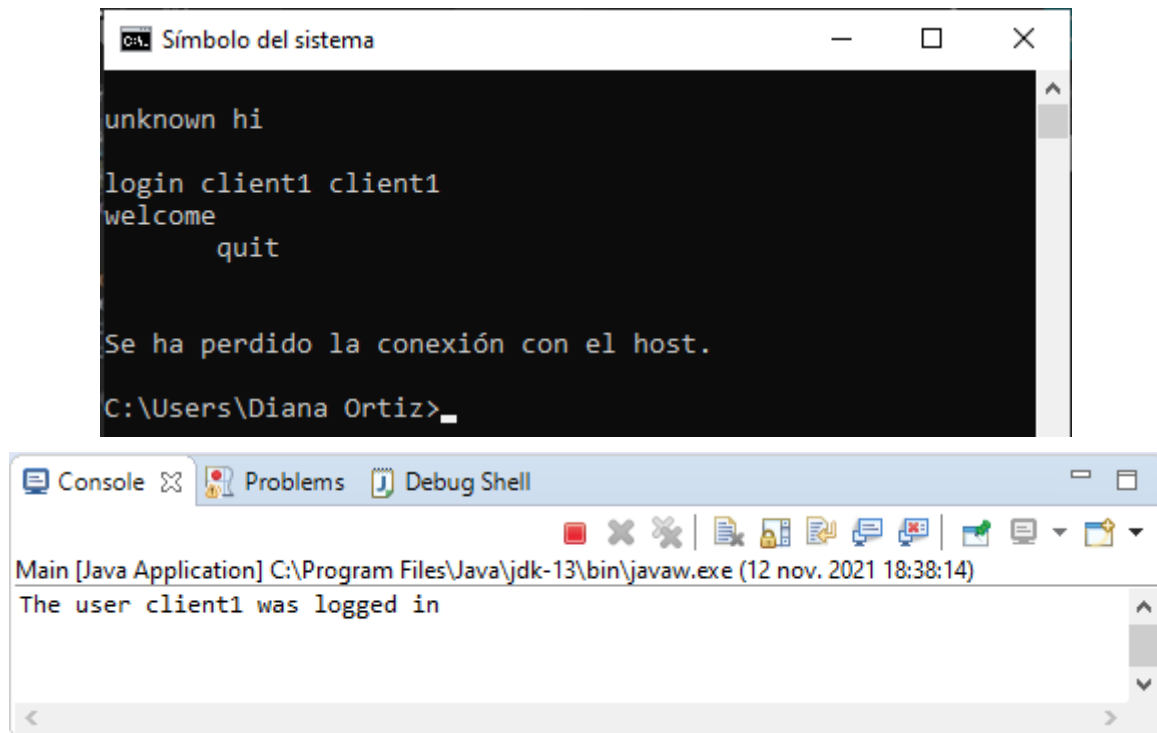
```
Símbolo del sistema

unknown hi
quit

Se ha perdido la conexión con el host.

C:\Users\Diana Ortiz>
```

5. Communication between client and server with a login session, logout and status is implemented and working.

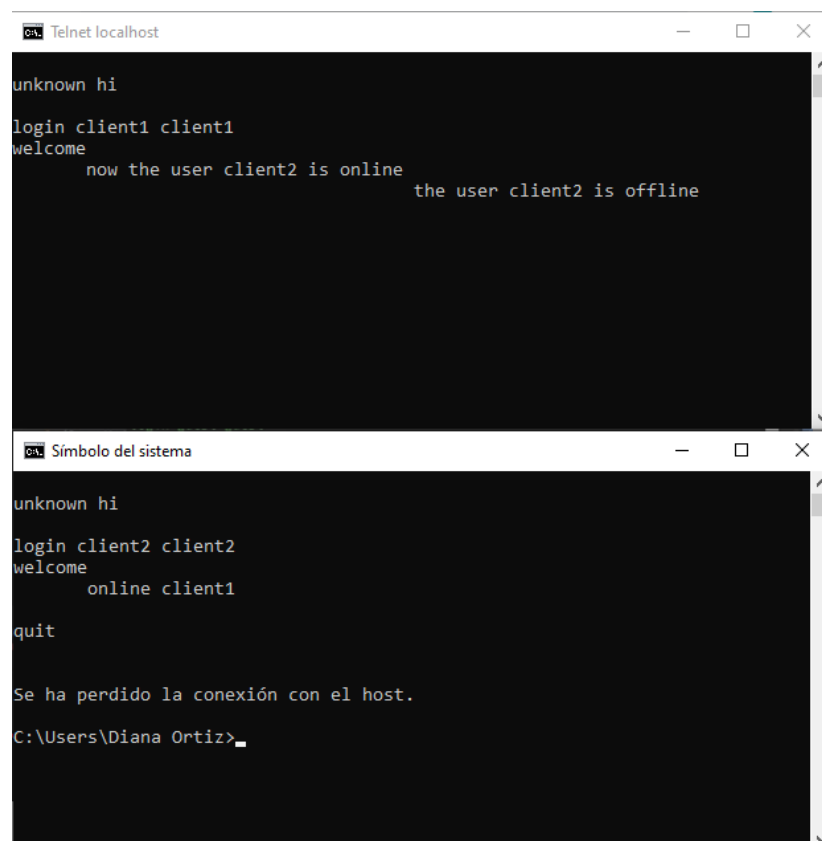


The image shows two windows. The top window is a Windows command prompt titled 'Símbolo del sistema'. It contains the following text: 'unknown hi', 'login client1 client1', 'welcome', 'quit', 'Se ha perdido la conexión con el host.', and 'C:\Users\Diana Ortiz>'. The bottom window is an IDE console titled 'Console'. It shows the output of a Java application: 'Main [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (12 nov. 2021 18:38:14)' and 'The user client1 was logged in'.

```
Símbolo del sistema
unknown hi
login client1 client1
welcome
quit
Se ha perdido la conexión con el host.
C:\Users\Diana Ortiz>

Console
Main [Java Application] C:\Program Files\Java\jdk-13\bin\javaw.exe (12 nov. 2021 18:38:14)
The user client1 was logged in
```

6. Communication between server and client for showing messages online and offline is implemented and working.

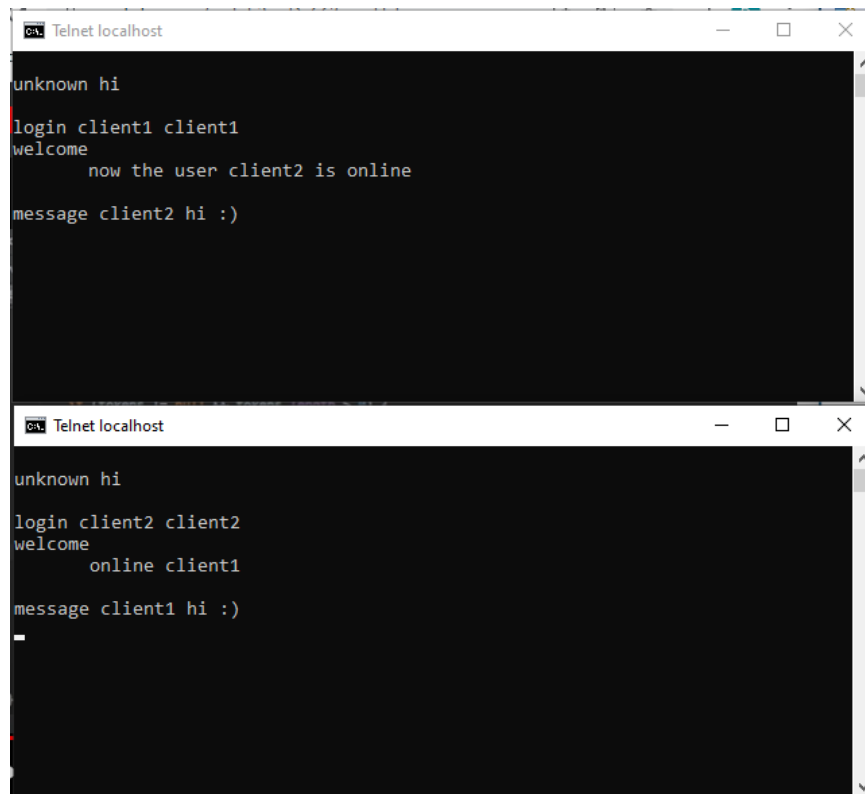


The image shows two Telnet sessions. The top session is titled 'Telnet localhost' and contains the following text: 'unknown hi', 'login client1 client1', 'welcome', 'now the user client2 is online', and 'the user client2 is offline'. The bottom session is titled 'Símbolo del sistema' and contains the following text: 'unknown hi', 'login client2 client2', 'welcome', 'online client1', 'quit', 'Se ha perdido la conexión con el host.', and 'C:\Users\Diana Ortiz>'. The text 'online client1' is indented.

```
Telnet localhost
unknown hi
login client1 client1
welcome
now the user client2 is online
the user client2 is offline

Símbolo del sistema
unknown hi
login client2 client2
welcome
online client1
quit
Se ha perdido la conexión con el host.
C:\Users\Diana Ortiz>
```

7. Communication between clients with direct messages is implemented and working.

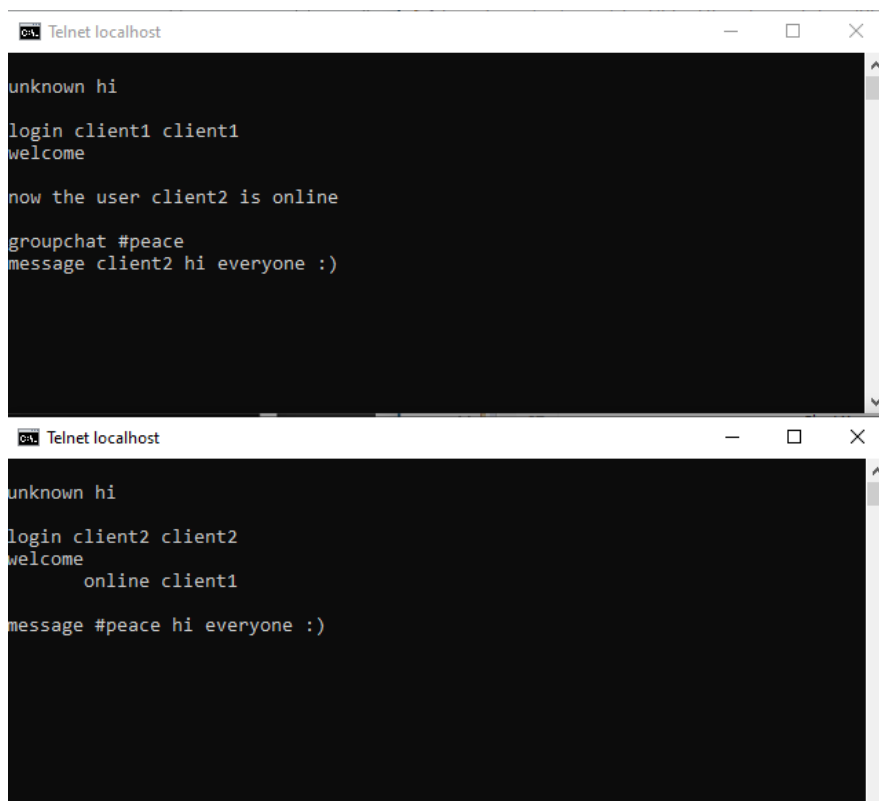


The image shows two terminal windows, both titled 'Telnet localhost'. The top window shows the following sequence of commands and responses: 'unknown hi', 'login client1 client1', 'welcome', 'now the user client2 is online', and 'message client2 hi :)'. The bottom window shows: 'unknown hi', 'login client2 client2', 'welcome', 'online client1', and 'message client1 hi :)'. This demonstrates a direct message being sent from client2 to client1.

```
unknown hi
login client1 client1
welcome
now the user client2 is online
message client2 hi :)

unknown hi
login client2 client2
welcome
online client1
message client1 hi :)
```

8. Communication between clients in group chats is implemented and working.



The image shows two terminal windows, both titled 'Telnet localhost'. The top window shows the following sequence of commands and responses: 'unknown hi', 'login client1 client1', 'welcome', 'now the user client2 is online', 'groupchat #peace', and 'message client2 hi everyone :)'. The bottom window shows: 'unknown hi', 'login client2 client2', 'welcome', 'online client1', and 'message #peace hi everyone :)'. This demonstrates a group chat message being sent from client2 to client1.

```
unknown hi
login client1 client1
welcome
now the user client2 is online
groupchat #peace
message client2 hi everyone :)

unknown hi
login client2 client2
welcome
online client1
message #peace hi everyone :)
```

9. The server is tested with ten connections from different clients and is still working.

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57409,localport=8818]
The user Clau was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57414,localport=8818]
The user Gerardo was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57421,localport=8818]
The user Lalo was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57426,localport=8818]
The user Paola was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57435,localport=8818]
The user Azul was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57457,localport=8818]
The user Xime was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57458,localport=8818]
The user Marce was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57463,localport=8818]
The user Jaime was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57468,localport=8818]
The user Vicky was logged in
```

```
Connection accepted :Socket[addr=/0:0:0:0:0:0:1,port=57474,localport=8818]
The user Isidro was logged in
```

Conclusions

I want to start saying that I think implementing a basic multi-user chat application is a great way to learn and understand the concepts you need to use for this project. But unfortunately this application has some limitations that we need to have in consideration for future improvements.

I think one problem with this implementation is that if we have a really long list of clients connected to the server, the searching time can increase. Maybe we could actually avoid this problem using two hashmaps but for this version of the application I think it could be an issue. Another problem is that if the server fails, the whole system will fail. Also there's no graphical user interface, so you can't use a chat window. Using telnet as a client works fine for the purpose of the project but it doesn't look pretty.

However, concurrency is really important for this implementation because, if you think of this approach like in a real application, you can realize a network program that could be a chat, a web server, or a peer-to-peer file sharing software. But this program will be connected to tens of computers or clients at the same time. And yeah, there are ways to use a single-thread strategy to handle these scenarios, but they normally choose to use multithreading.

And that's because of the thousands of requests per second. If a client had to wait for the previous client to make a request and get a response before it came to completion, the server would become hopelessly bogged down. So, with multithreading, the server can run more smoothly and actually serve the clients better.

Now you can try to implement a multi-user chat application by yourself and make all the improvements possible for this basic version. Chat applications are really useful tools. So don't wait anymore to build your own.

References

Mehner, M. (2021, November 15). WhatsApp, WeChat and Facebook Messenger Apps - Global usage of Messaging Apps, Penetration and Statistics. Messenger People by Sinch. Retrieved November 17, 2021, from <https://www.messengerpeople.com/global-messenger-usage-statistics/>

Cressler, C. (2021, June 4). Understanding the Architecture & System Design of a Chat Application. CometChat. Retrieved November 17, 2021, from <https://www.cometchat.com/blog/chat-application-architecture-and-system-design>

GeeksforGeeks. (2019, November 15). Client-Server Model. Retrieved November 17, 2021, from <https://www.geeksforgeeks.org/client-server-model/>

Jenkov, J. (2020, November 17). Concurrency vs. Parallelism. Jenkov. Retrieved November 17, 2021, from <http://tutorials.jenkov.com/java-concurrency/concurrency-vs-parallelism.html>

tutorialspoint.com. (2018, October 10). Message Passing vs Shared Memory Process communication Models. Tutorialspoint. Retrieved November 17, 2021, from <https://www.tutorialspoint.com/cache3.com/message-passing-vs-shared-memory-process-communication-models.html>

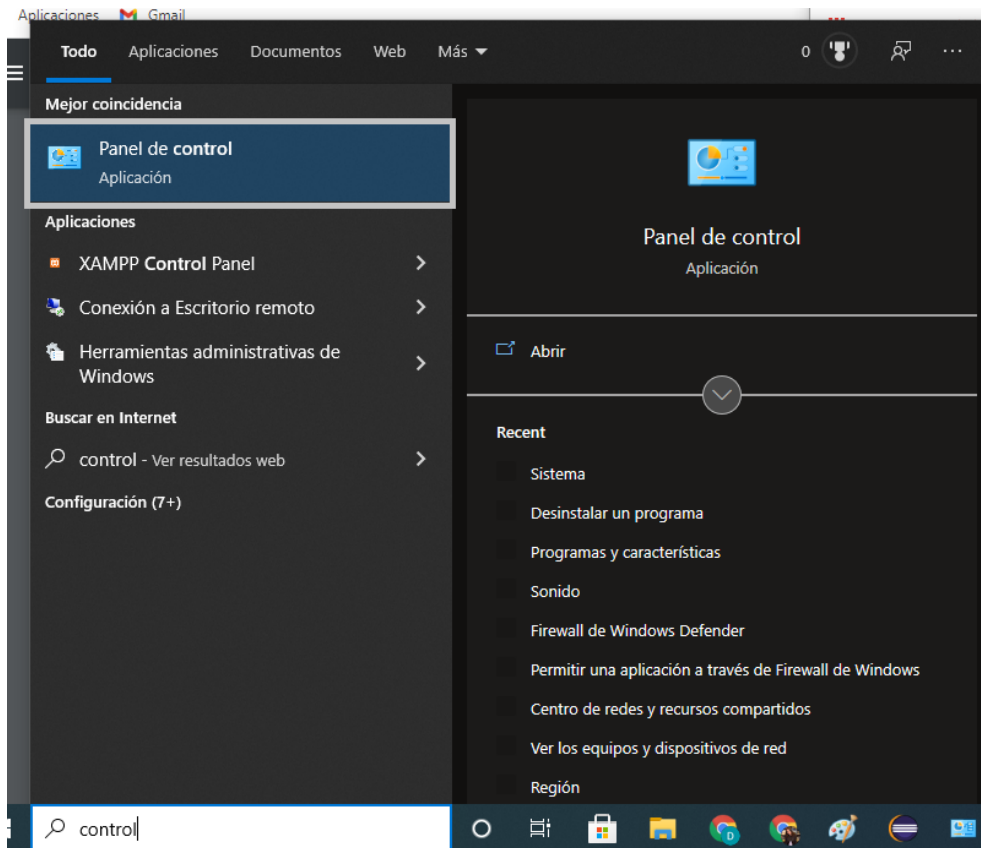
Chlipala, A. (2014, October). Reading 17: Concurrency. Software Construction. Retrieved November 17, 2021, from <https://web.mit.edu/6.005/www/fa14/classes/17-concurrency/>

Jenkov, J. (2015, October 1). Thread Safety and Shared Resources. Jenkov. Retrieved November 17, 2021, from <http://tutorials.jenkov.com/java-concurrency/thread-safety.html>

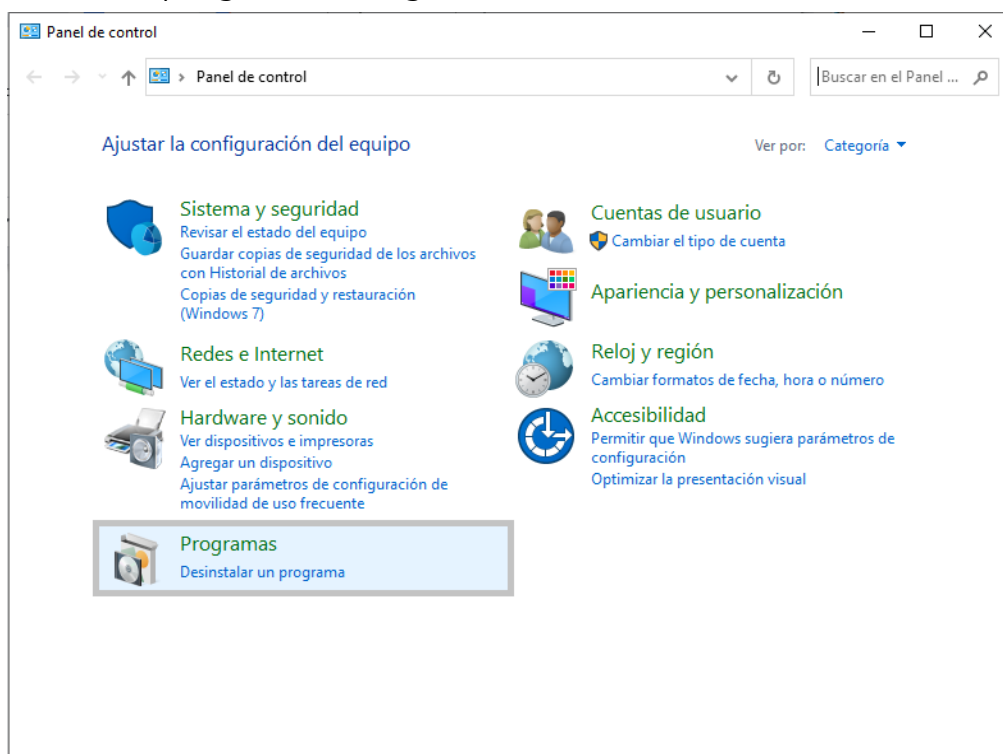
Setup instructions

Enable the Telnet Client

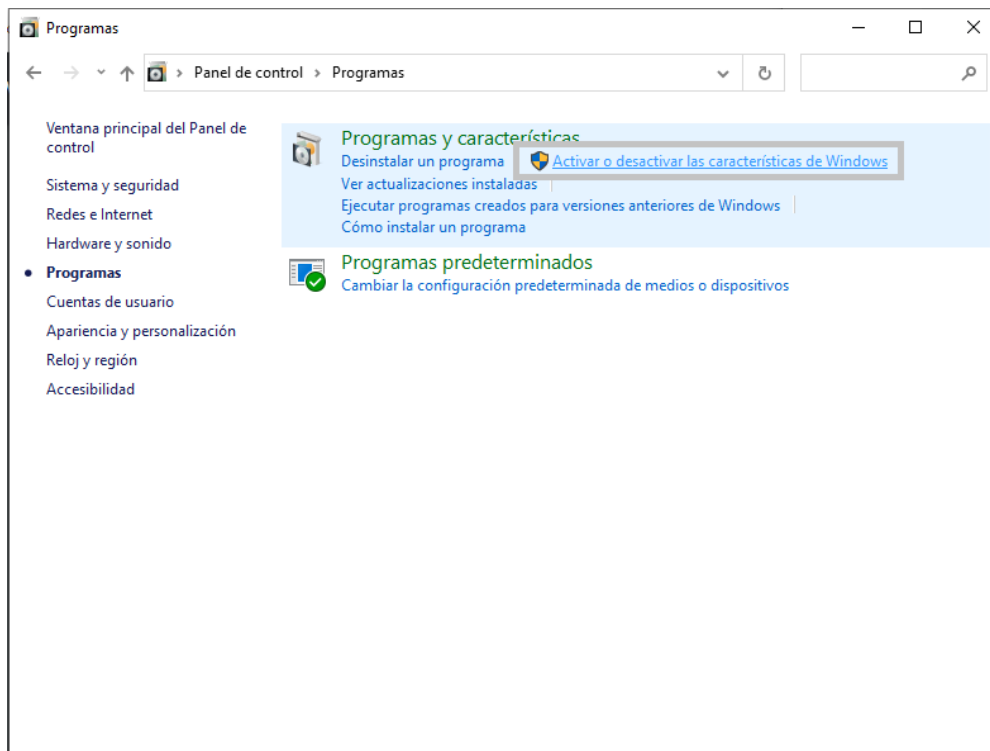
1. Open the Control Panel



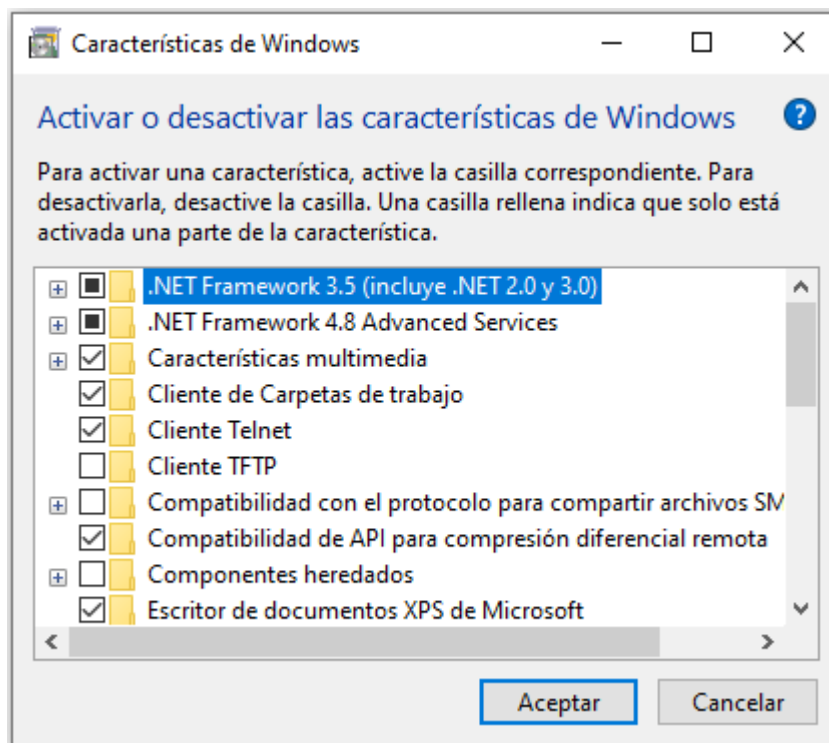
2. Enter the program settings menu



3. Open the Windows features menu and choose the sub-heading “Turn Windows features on or off”.

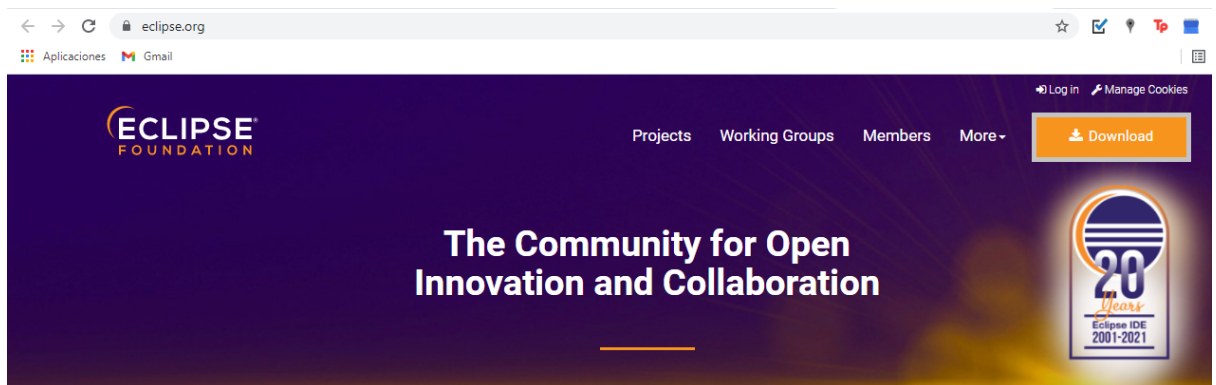


4. Enable Telnet in Windows 10 Features, scroll down the list of features until you find the folder named “Telnet Client”. Tick it, then press “OK” to enable Telnet in Windows 10.

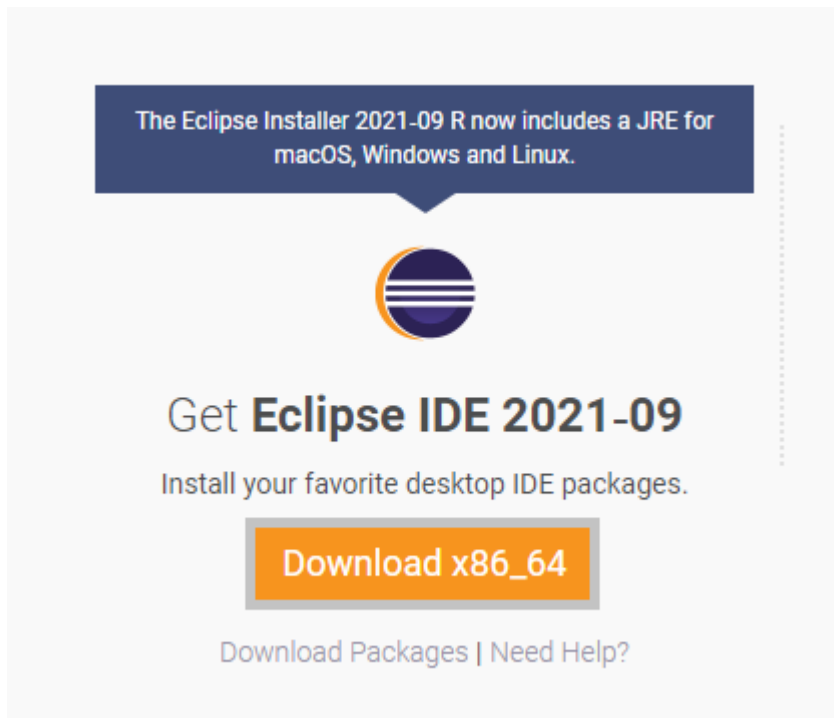


Download and install eclipse to run java

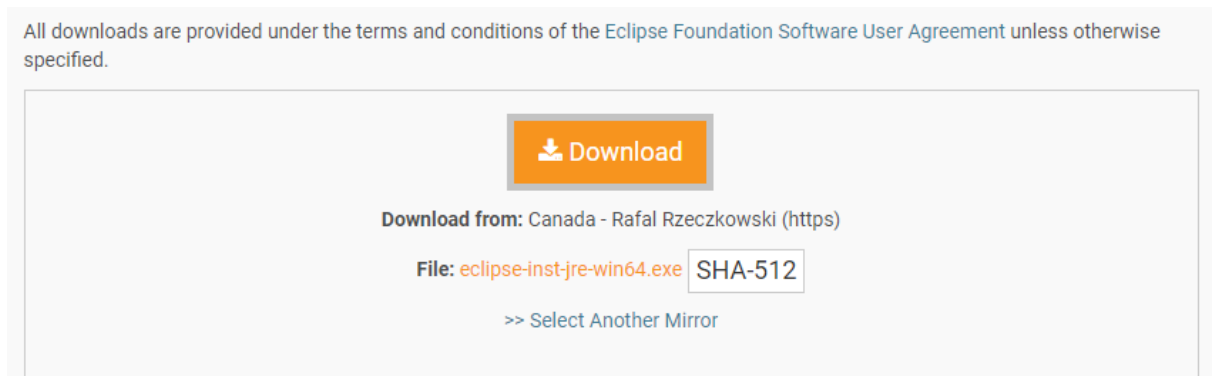
1. Open your browser and type <https://www.eclipse.org/>
2. Click on the “Download” button.



3. Click on the “Download 64 bit” button.



4. Click on the “Download” button.

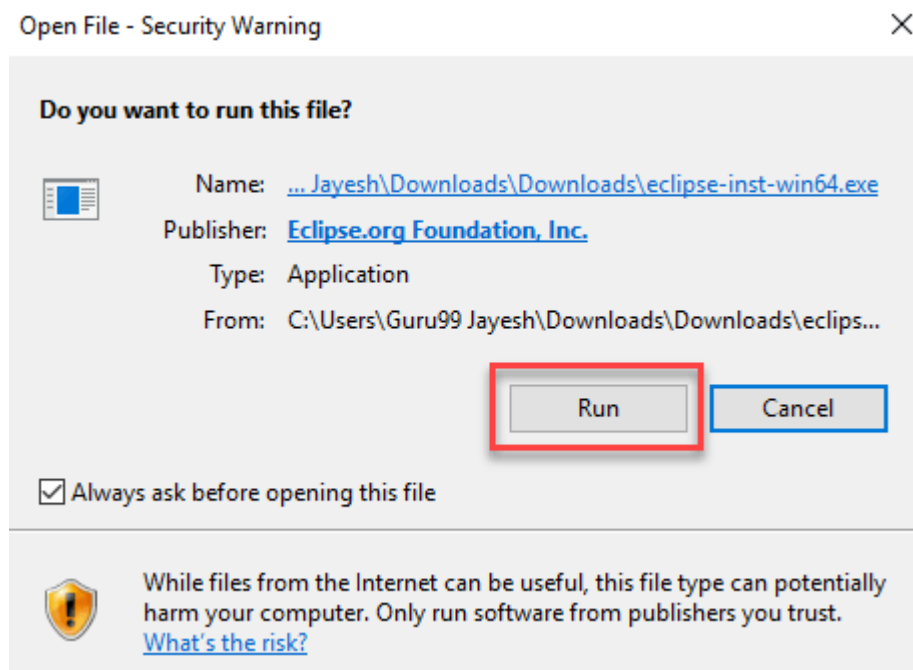


5. Install Eclipse.

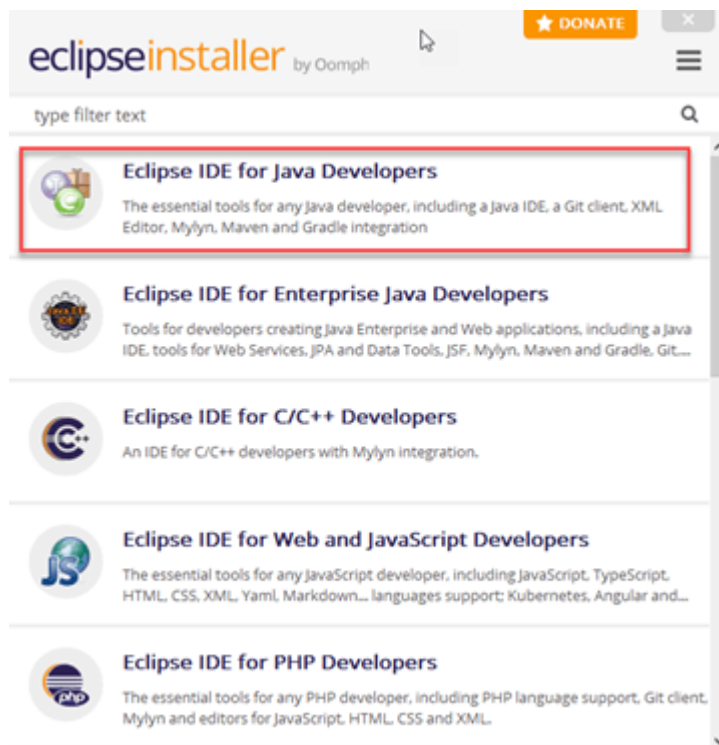
Click on “downloads” in Windows file explorer.

Click on “eclipse-inst-jre-win64.exe”

Click on the Run button.



Click on “Eclipse IDE for Java Developers”.

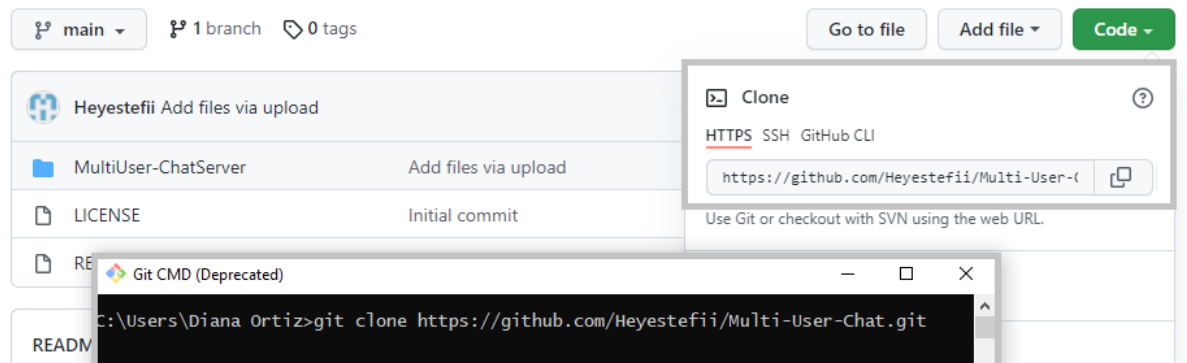


Click on the “INSTALL” button.

Click on the “LAUNCH” button.

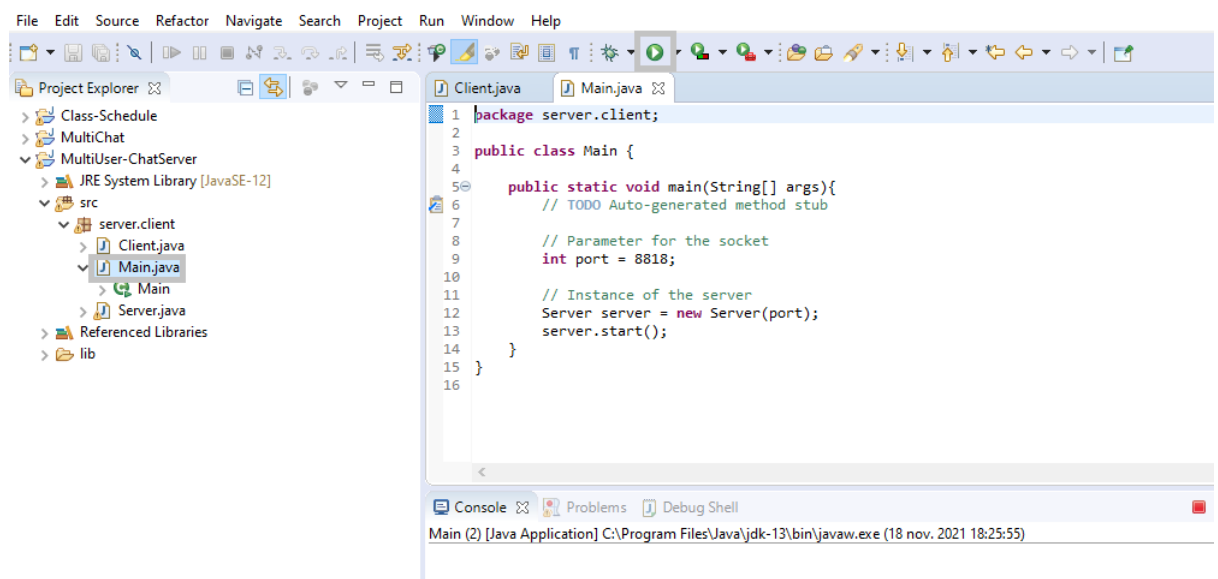
Click on the “Launch” button.

6. Clone the repository from here
<https://github.com/Heyestefii/Multi-User-Chat> with this link

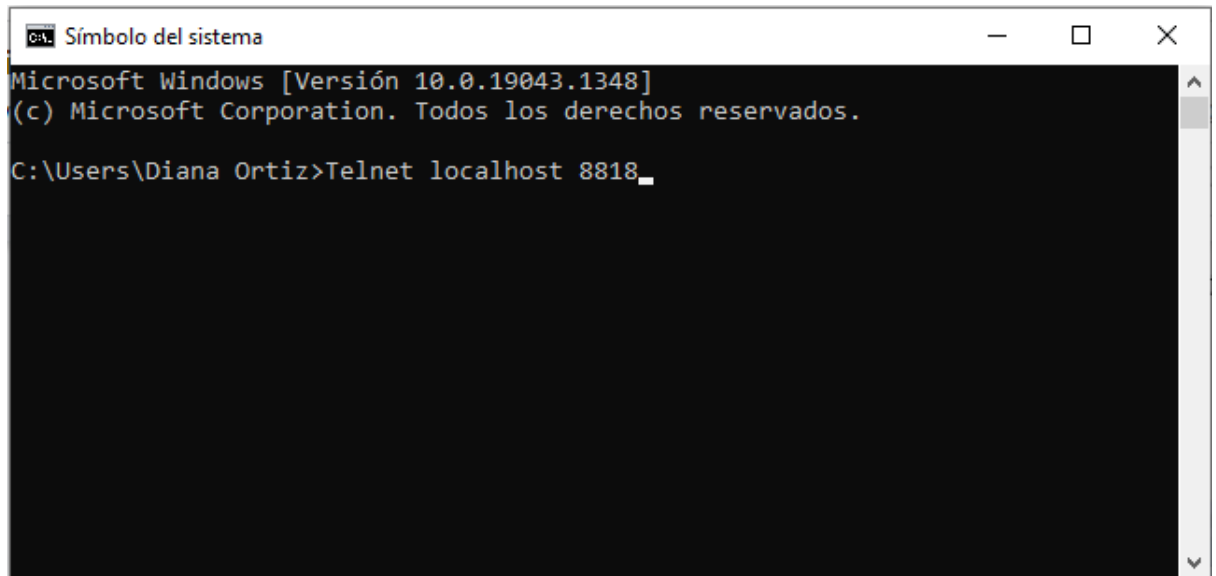


7. Import the project in Eclipse
Open Eclipse.
Go to Open File -> Import
Select "Existing Projects into Workspace" from the Selection Wizard.
Select Next.
Browse to find the location of the project.

8. Run the main class of the project.



9. Open your CMD and use telnet to connect to the project as a client with the port 8818.



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.19043.1348]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Diana Ortiz>Telnet localhost 8818_
```

10. Now you can use the commands to interact between clients and the server. You can find the different users who can access the server in the class Client.

- login <user> <password>
- logout
- message <user> body
- groupchat #topic
- message #topic body
- leave #groupchat