Chat with multiple users using sockets and threads

Kevin Adriel Woolfolk García (A01251809)
Instituto Tecnológico y de Estudios Superiores de Monterrey
Campus Querétaro

Keywords: Sockets, Threads, Connections, Datagram, Swing, Eclipse.

Abstract: Nowadays, software platforms use sockets to make different connections and threads to increase the performance of their tasks. Sockets are a software endpoint that establishes bidirectional communication between servers and clients. One way to use sockets is by handling requests by threads. A thread is a sequence of instructions that run independently from the program that is executing. In this paper, the steps taken in order to implement sockets and threads using Java are explained, which is shown as a group chat where N users can send and receive all users messages.

1 Introduction

Platforms all over the world use threads in order to have better performance and divide the software task. When a software developer creates an application, a platform or a system, sometimes they don't realize most of the time they are using threads to resolve some functionalities. For this project, a multiple user chat was developed, which connects N number of users and allows them to communicate with each other. This chat is a really good example to explain how threads work and how useful they could be for functionalities. An implementation used by software developers is multi-threading, which is a programmable approach to achieve multi-tasking, where each thread (a single line of instructions) holds control of one program. Any software developer has written a program that displays "Hello World!", or has sorted a list of names, or computed a list of prime numbers. These are examples of sequential programs: each has a beginning, an execution sequence, and an end. At any given time during the runtime of the program there is a single point of execution. A thread is similar to the sequential programs described above: a single thread also has a beginning, an end, a sequence, and at any given point during the runtime of the thread there is a single point of execution.

2 Paradigm

Before introducing the project implementation, it is important to understand which paradigm is being used for this project. Since Java is being used for this project, Object Oriented Programming (OOP) will be the selected paradigm, which is based on the dispatch of messages to objects.

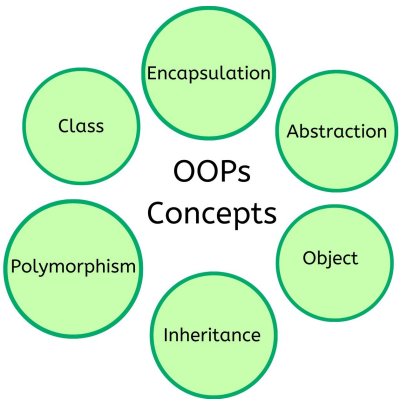


Figure 1.

In a few words, it may be understood that OOP is made by classes from which objects are made by encapsulating data members and functions. An object is a data structure that contains some methods that act upon its attributes. The way objects work is explained on Figure 2.

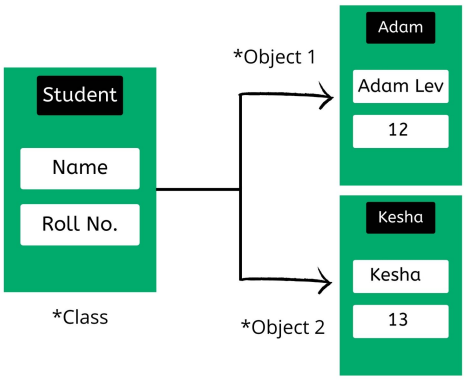


Figure 2.

Since this project makes the users objects, and sends messages with a run method, it may be confirmed the use of this paradigm.

The second paradigm used in this project is concurrency, since the chat is being connected with a socket we have all the users in the same mask and the messages are delivered one by one.

3 Implementation

For this project, Java was used for logic and Swing for GUI design of the interface. First, it is necessary to install Eclipse IDE and install the Swing framework to build the GUI. Once all the eclipse setup is done, the project can be started.

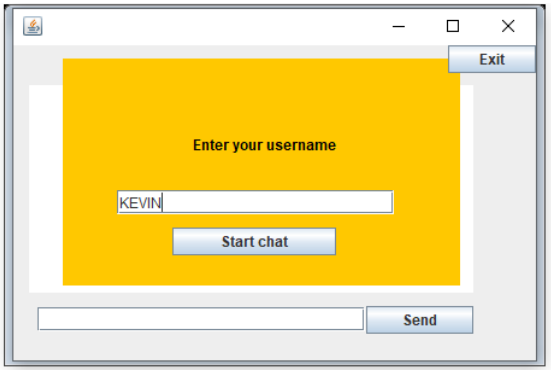
Two classes are needed for this project:

- ChatView.java
- Client.Java

The ChatView will be the user interface and the Client will be the Thread that will be assigned to each of the users every time they join the chat.

3.1 Interface

Before starting to develop the code, it was necessary to design an attractive welcome interface for users to place their username and start the chat. For this project, the JFrame library was imported in order to have all kinds of elements as labels, text fields and buttons. It's really important to add an action listener to all the buttons.



The screen should look like this or with a similar functionality where the user must place their username first, and then enter the chat.

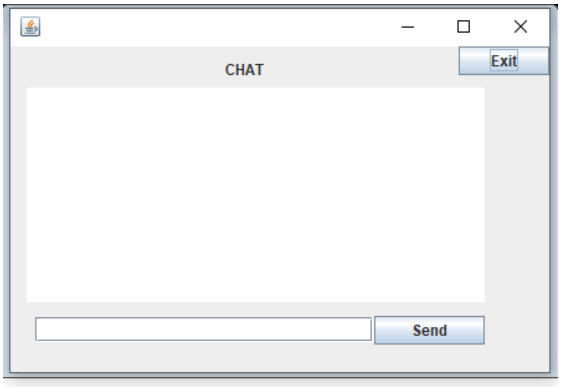


Figure 3.

Once the user has entered the chat, he or she will be able to send messages to all the users that are connected in the socket.

3.2 Creating thread

Since this class will be the thread of the client it is important to add an implement Runnable:

```
public class Client implements Runnable{
```

and have our run method to start the thread.

```
@Override
public void run() {
```

In order to start the thread, it is necessary to create the object in chat view.

```
Client client = new
Client(userName,port,model,socket,false,group);
Thread clientThread = new Thread(client);
clientThread.start();
```

For our object we need 6 parameters:

- User name
- Port
- Chat area
- Socket
- Exit
- Group

The user name will be the name of the client, the port is the number where the connection is taking place, the chat area is the model of JList where we send our new message and display it, the socket is the multicast socket we are joining with the address, the exit boolean is the flag which will tell us when the user leaves the chat and group is the Inet Address of the group chat.

In the class client we have two methods, one is the setter for user name; since we run ChatView we create the object with user name as an empty name,

```
public void setUserName(String userName) {
    this.userName = userName;
}
```

the other one is to exit from the window application and socket in a proper way.

```
public void exit(){
    this.exit = true;
}
```

On the run method, it is important to start a while loop that will be done when exit equals true, so that the thread stops to read the messages.

3.3 Set socket and Datagram

When the client chat view starts, we need to start a socket with a port number so we can

connect to the other clients. Since an Inet address is being added to each of the users to make a connection, we need to use a Multicast socket that helps us send and receive IP packets. Once we declare our group and socket, we join them and set time to live to 0 for being in localhost, in case the project was inside another network it would be necessary to change the number.

```
group = InetAddress.getByName("239.0.0.0");
socket = new MulticastSocket(port);
socket.setTimeToLive(0);
socket.joinGroup(group);
```

At the moment the "send" button is clicked, and call the action listener we start the buffer and send to the socket the message in a datagram packet.

```
byte[] buffer = newMessage.getBytes();
DatagramPacket datagram = new
DatagramPacket(buffer,buffer.length,group,port);
try {
    //Send data to socket
    socket.send(datagram);
} catch (IOException e1) {
    e1.printStackTrace();
}
```

On the other side of the run method in client class on the while loop we declare our buffer with the datagram packet.

```
byte[] buffer = new byte[1000];
DatagramPacket datagram =
new DatagramPacket(buffer,buffer.length,group,port);
String message;
```

When the socket receives a message it runs into the try-catch and each thread will send the message into their JList and display it.

```
try{
    socket.receive(datagram);
    message = new
     String(buffer,0,datagram.getLength(),"UTF-8");
    chatArea.addElement(message);

}catch(IOException e)
{
    System.out.println("Socket closed!");
}
```

4 Test

For this project there are two important parameters for testing the chat, that are to send a massive amount of messages to all users at the same time and the other one is to connect a big amount of user connections at the same time. In this section we will look at three test cases that were applied and how the difficulty through each case increased to show chat performance.

Before showing the test cases iIt is important to know the next information :

1) To create more than one user we have to change the variable user_number in ChatView.java on line 46 for the number of users that we want to create.

```
//change value to create more users
int user_number = 1;
for(int i=0;i<user_number;i++) {
    ChatView window = new ChatView();
    window.frame.setVisible(true);
}
```

2) In the chat there are 3 special words that send massive message to all users:

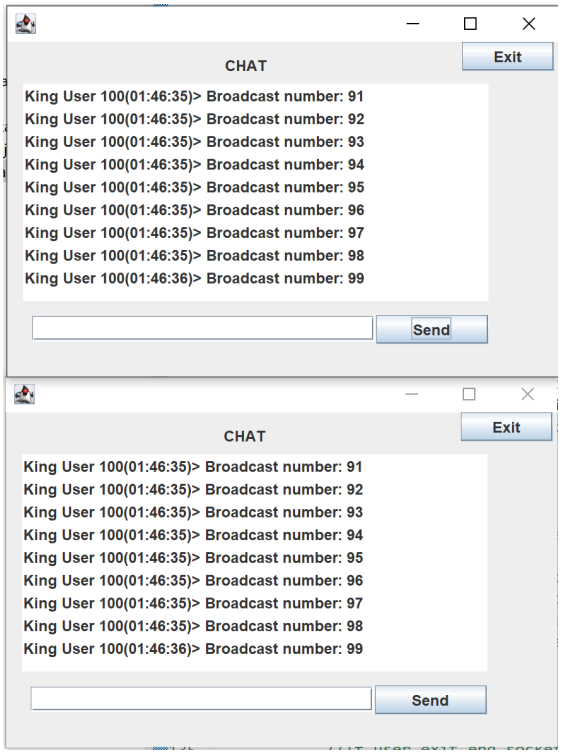
- 100broadcast: send 100 messages and Sleep thread for 100ms
- 1000broadcast: send 1,000 messages and Sleep thread for 10ms

- 10000broadcast: send 10,000 messages and Sleep thread for 1ms

Test Case 1

Users = 2

Messages = 100

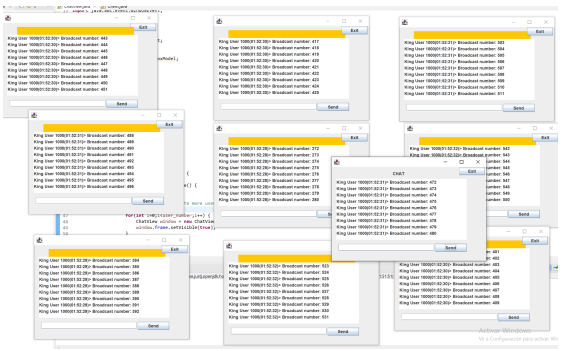


The program ran correctly and didn't have any complications.

Test Case 2

Users= 10

Messages = 1,000

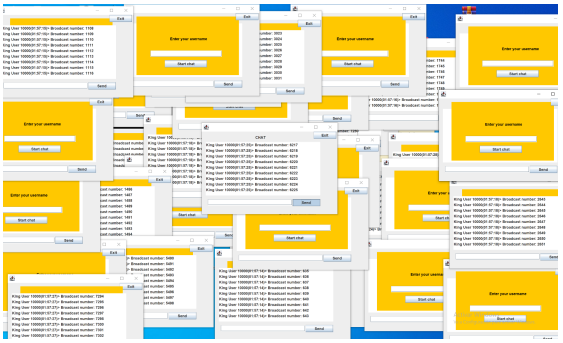


In this case the messages have a lot of delay and each chat seems to freeze sometimes.

Test Case 3

Users= 100

Messages = 10,000



The chat was delayed a lot of time and the users weren't able to receive the messages at the same time.

To understand how these test cases are working and why they work or fail we can watch a thread diagram shown on Figure 5, where the running of the threads of the project is exemplified, while also showing how it sends and receives data as if there were N clients.

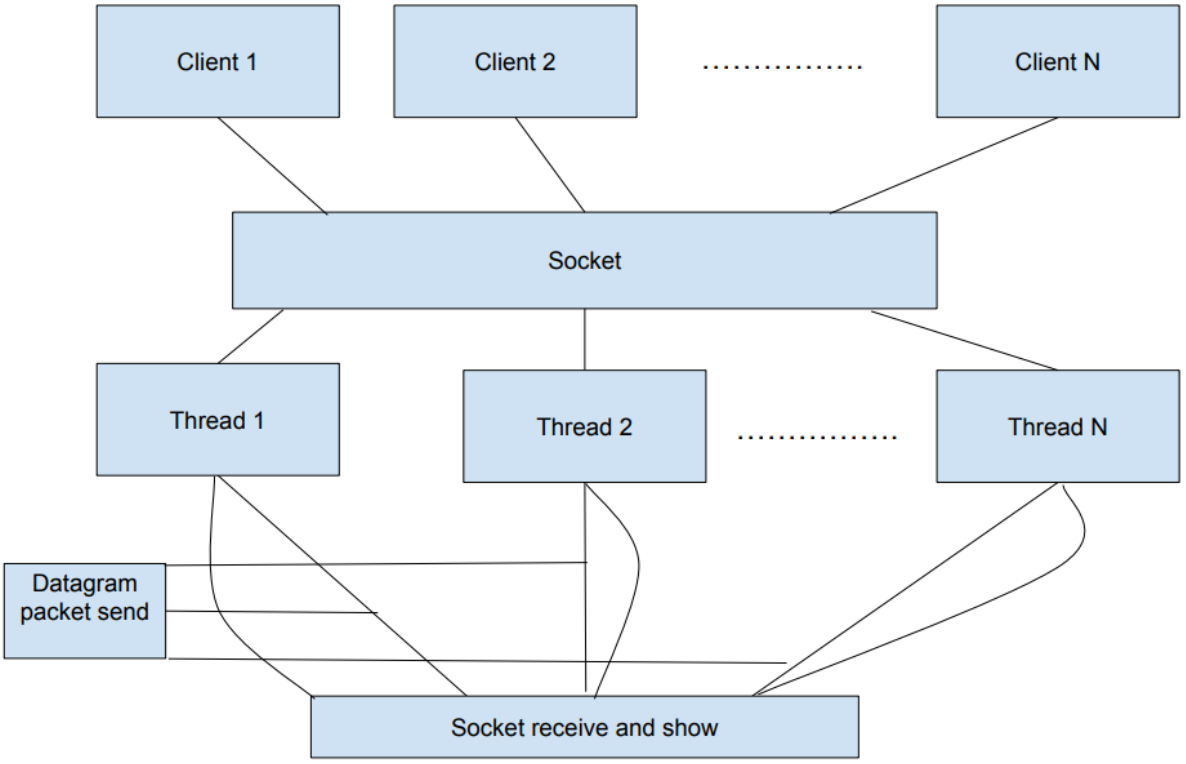


Figure 5. Thread diagram

5 Conclusion

Due to the overall research and outcome of the project, I realized how threads actually work and how important they are in real-life projects, allowing us to increase performance. While developing the chat, I noticed that threads are an important thing developers must be able to do, and the relevance of making a connection with a socket, in order to communicate with each other and use the threads as a reader for new upcoming messages.

6 References

Carel. (2017). What are Sockets and Threads?. november, de steemit Sitio web: https://steemit.com/programming/@carel111/what-are-sockets-and-threads

University of Pennsylvania. (NA). What Is a Thread?. 18/11/2021, de University of Pennsylvania Sitio web: https://www.cis.upenn.edu/~bcpierce/courses/629/papers/Java-tutorial/java/threads/definition.html

Oracle. (NA). java.net Class MulticastSocket. 18/11/2021, de Oracle Sitio web: https://docs.oracle.com/cd/E17802_01/j2se/j2se/1.5.0/jcp/beta2/apidiffs/java/net/MulticastSocket.html

NA. (NA). Programming paradigms. 18/11/2021, de Java Programming Sitio web: https://java-programming.mooc.fi/part-7/1-programming-paradigms

NA. (2013). Socket programming with a thread pool server model in c++. 18/11/2021,

de Stackoverflow Sitio web: https://stackoverflow.com/questions/13535543/socket-programming-with-a-thread-pool-server-model-in-c

NA. (2020). R – Object Oriented Programming. 15/11/2021, de Geeks for geeks Sitio web: https://www.geeksforgeeks.org/r-object-oriented-programming/