Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro

***Programming languages***

**Final Project: Broadcast group chat application**

**Teacher:**

Benjamín Valdés Aguirre

**Team Members:**

Luis Claudio Soto Ayala A01205935

Ricardo Rodriguez Garcia A01209849

**Broadcast group chat application**

**(For access the final project, visit:** [**https://github.com/claudious10/ChatGroupApp**](https://github.com/claudious10/ChatGroupApp)**)**

**Introduction**

In the development of this project, the principal owners and developers were *Luis Claudio Soto Ayala* and *Ricardo Rodriguez Garcia*, both students of ***Tecnologico de Monterrey Campus Queretaro*** in the period Agost-December of 2019; the main purpose of this project is to explain our implementation of object oriented programming paradigm with the development of a communication group application in real time using elements studied in class (Threads, objects, concurrency, etc) with the final goal of enriching the community with our work and a specific example of this paradigm to increase the knowledge and for future reference of the developers society. It's important to be aware that the subjects discussed in this report are very specific and related with computer science and IPC mechanism for thread communication. For that reason we highly recommend to have a knowledge base before reading this.

Before starting the explanation of our project, we must say that this project only cover OOP paradigm for the development of this project in combination with concurrency and IPC mechanism to communicate users between terminals (**we will bring every part of the code to explain every part utility to have a better understanding of the project**).

**1.Context of the problem**

As we know, there are two principal types of communication on the devices, the principal categories and most common ones are the Half-duplex communication and Full-duplex communication, in the first one, just one side can send a message and the others will receive that message (Figure 1), it's only a one direction communication and in the second one, both sides can send and receive at the same moment, that means that it is a two direction way communication (Figure 1.1), our solution to this problem uses the second alternative, with the only difference of extending this to a broadcast approach in which all users connected receive all messages from other users and all users can send as well (Figure 1.2).







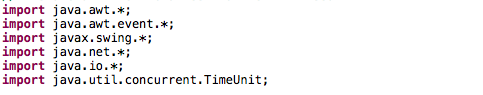
**2.Solution:**

Starting with the project, the essential purpose of this project is to communicate N number of terminals, having and alias in order to have a discussion in the group with multiple users at the same time and, as well, having multiple groups with multiple users working at the same time, using and IP-PORT mechanism to divide the groups and the users depending on which IP-PORT they select to connect to a determined broadcast group as a simulation of a server running and clients connecting to it in a specific group.

For this implementation, we will use Java as a language for developing the application (**So, we assume that it is already installed on your machines**) and it will only contain two files called “Server.java” and “Messages.java”, We will explain the two files and every part of the code to understand better what is happening in each part, then, without further ado, we begin:

**2.1Server.java**

Before starting this file, it's necessary for our readers to know which libraries will be used and the reason of its use, as well we will give a couple of links to see the original documentation for java at the end in the reference section to clarify any doubt you can have, as well, we will give you a detailed step by step to know how to compile and run the entire project for your satisfaction.



**java.awt.\*:** This library contains all of the classes for creating user interfaces and for painting graphics and images. This library contains some functions and components, and complements the javax,swing.\* library. Both libraries work together to build the graphic interface. (**For more information and the benefits of this library visit:** <https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html>)

**java.awt.event.\*:** This library provides interfaces and classes for dealing with different types of events fired by AWT components. An event listener registers with an event source to receive notifications about the events of a particular type. This will allow us to implement a function which will execute when a button is clicked. (**For more information and the benefits of this library visit:** <https://docs.oracle.com/javase/7/docs/api/java/awt/event/package-summary.html>)

**java.net.\*:** This library will be in charge to create the ip configuration network for the group and the multicastSocket, to create and connect to a port to a specific group and start chatting with the people connected to the group.

**java.io.\*:** Imports all the classes that are defined in java.io package to your file. This enables your java program to use those classes and their methods to achieve some tasks. But if you needed more than one file to be imported in your program (and that you're lazy enough to write a line for each such class, you could use import java.io.\* in this case we call this class to manage the exceptions for the creation and use of the sockets in the group (If you are not able to connect for some reason this library will throw an exception. **For more information and the benefits of this library visit:** <https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html> ).

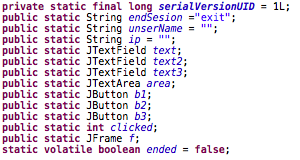
**javax.swing.\*:** This library provides a more sophisticated set of GUI components than the earlier Abstract Window Toolkit (AWT). This will be in charge to create the frame, panels, buttons, text area and text field (different from the text area), and as we said before, work with the java.awt.\* library to build the graphic interface of the chat. **For more information visit:** <https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html>

**java.util.concurrent.TimeUnit:** A TimeUnit represents duration time at a given unit of granularity and provides utility methods to convert across units, and to perform timing and delay operations in these units. This library will help us to stop the main execution when the user types “exit” in the GUI, giving time to the user to read the exit message and before the main stops the execution. **For more information visit:** <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/TimeUnit.html>

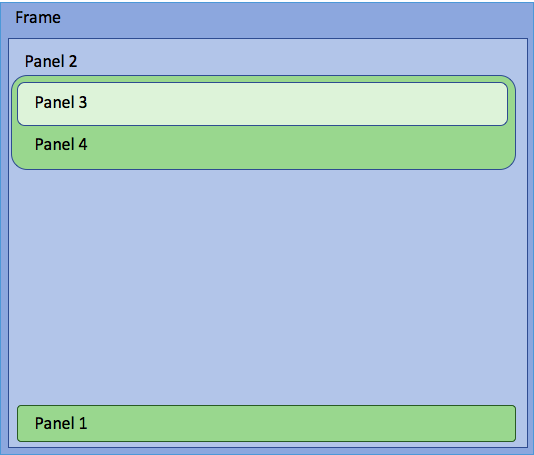
This File of the project have the main purpose of connecting to the specific PORT (We will use port:**1234** as a demonstrative example) and IP (in range of **239.0.0.0 - 239.255.255.255**) to access a group in that range of your preference and select the alias or user name for that group:

In the first part of the class Server, we declared fourteen class variables called serialVersionUID, endSession, userName, ip, text, text2, text3, area, b1, b2, b3, clicked, f and ended.

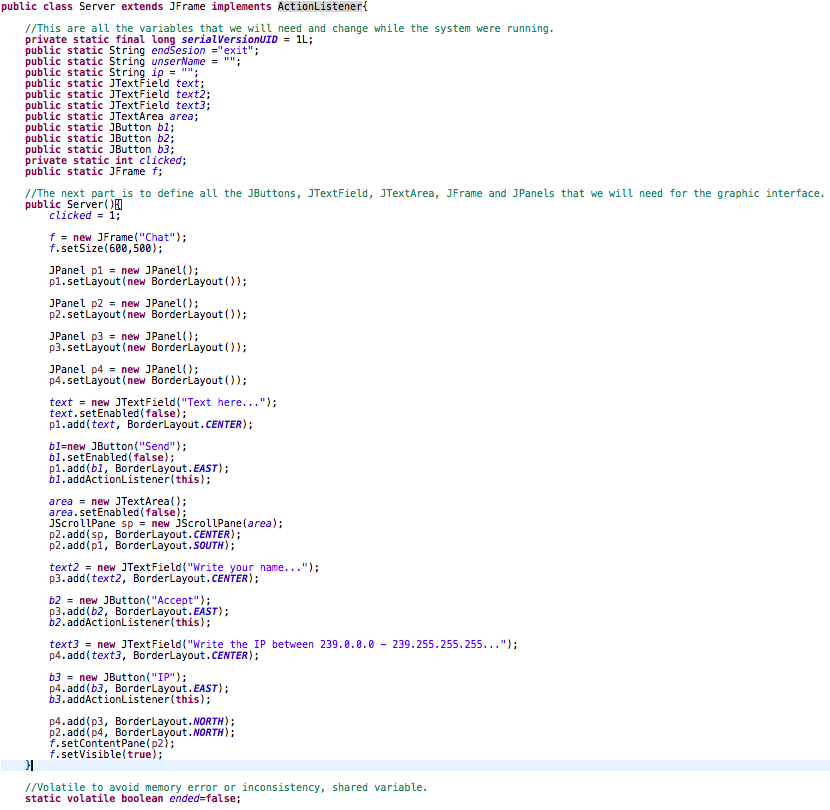
* **serialVersionUID.** The serialization runtime associates with each serializable class, a version number called a serialVersionUID, which is used during deserialization to verify that the sender and receiver of a serialized object have loaded classes for that object that are compatible with respect to serialization.
* **endSession**. This variable is a string that will be reonicized by Messages.java in order to end the connection with the group when the user types in any moment “exit” in the chat.
* **userName:**  This variable is a string which contains the name of the user and will allow the other users to know what user send a specific message.
* **ip.** This variable is a string that contains the ip address of the group the user wants to connect to.
* **text.** This is a JTextField variable which will be deployed in the frame as the space where the user can write the message. This text field at the beginning will be disabled to prohibit the user to start writing something before making the connection.
* **text2.** This is a JTextField variable which will be deployed in the frame as the space where the user must write its name.This text field at the beginning will be enabled to permit the user write the user name before making the connection.
* **text3.** This is a JTextField variable which will be deployed in the frame as the space where the user must write the ip address of the group.This text field at the beginning will be enabled to permit the user write the ip address to do the connection.
* **area.** This is the JTextArea variable where will be deployed all the messages that the user send and receive.
* **b1.** This is the JButton variable with the label “Send” whose function is to send the messages that the user write in the text variable. We will add an actionlistener to this button to implement a function called ActionPerformed which will be executed when the user clicks the button. This button at the beginning will be disabled to prohibit the user to click it and run ActionPerformed function.
* **b2.** This is the JButton variable with the label “Accept” whose function is to define the value of the user name sending the value of the text2 variable. We will add an actionlistener to this button to implement a function called ActionPerformed which will be executed when the user clicks the button. This button at the beginning will be enabled to permit the user to define its username.
* **b3.** This is the JButton variable with the label “IP” whose function is to change the value of the ip variable sending the value of the text3 variable. We will add an actionlistener to this button to implement a function called ActionPerformed which will be executed when the user clicks the button. This button at the beginning will be enabled to permit the user to define the ip address of the group.
* **clicked.** This is an int variable that will have just 1 or 0. This will work as a flag to identify when the user presses the b1 button which is the one that send the messages.
* **f.** This variable is a JFrame variable that will build a window which will contain all the panels, texts field, textarea and buttons for the graphic interface.
* **ended.** This variable is a shared variable in the group for users to being connected or end the connections (**The volatile type allows to share information between threads and avoid memory inconsistency between them, in this case to let all threads know that a user is connected or disconnected**) **(Figure 2.1)**.



The constructor of the class Server defines the variables that we will need to build the graphic interface and their values. The frame variable is for create a window where will be the panels that will contain the text area, text fields and buttons. There are 4 panels in the frame and are distributed as the following image shows:



Then, the constructor will create the text area, the text fields and buttons. The panel 1 contains one button called “Send” at the right side and in the rest of the panel a text field where the user can write a message. The panel 2 contains the panel 1 at the bottom, then a text area above the panel 1 where the messages will be displayed and above the text area the panel 4. The panel 3 contains one button called “Accept” at the right side and in the rest of the panel a text field where user must write its username. Finally, panel 4 contains the panel 3 at the top side and below it one button called “IP” at the right side and in the rest of the panel a text field where user must write the IP address of the group. The final graphic interface is shown in the following images. At the end of the constructor, the panel 2 is added to the frame and set the visible property of the frame to true.



**Main Function**

The first thing to mention about the main() function is that it can throw IOException and InterruptedException that is why the line will be like this:



We have to put IOException because in the next lines we use a function of exec() which will try to execute something and if it has a problem, needs to throw an execution error. Is the same for the InterruptedException, in the line that we implement the TimeUnit.*SECONDS*.sleep(3), this will try to put to sleep the main function and if it has a problem, needs to throw an interrupted error.

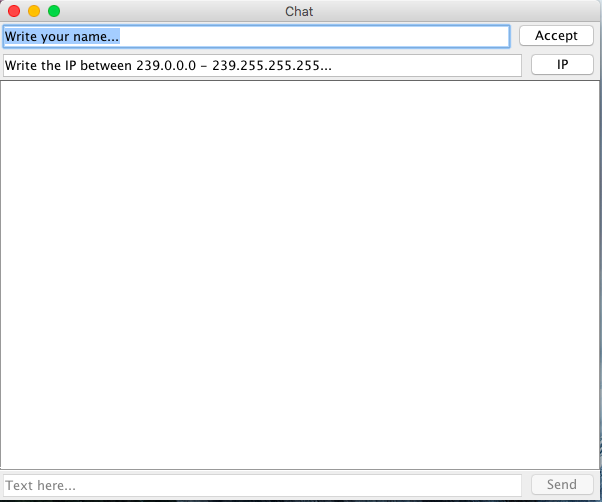
After saying that, we can continue with the content of the Main() function. We need to disable IPv6. For that the java documentation indicates setting jvm property java.net.preferIPv4Stack to true. We do that with the next line:



The next step is to run the Server.java and Messages.java files with the Java Compiler javac. to do that we use these lines of code:



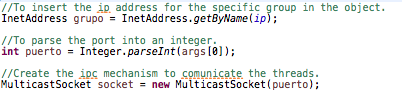
We start the server with the constructor of the class, followed by an if. This class at the execution moment will receive one parameter (**in other case an error message will be displayed and the execution will be ended**), which is the port that the system will use to send and receive the messages and this if will verify that. Once we start the constructor of the class, it creates the graphical interface which is shown in the following image:



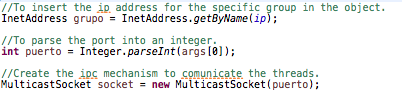
If the execution of the program has no problems with the port, the next step is to try to establish the connection. However, we don’t have the username and the ip address yet, that’s why we need to wait for the user to introduce it’s username and the ip address. For that, we start a while loop that will be cycled until the user write in the text2 the username and press the b2 button and write the ip address, in the text3 and press the b3 button. Once the user do that, the ActionPerformed() function will be executed and the value of the username and ip variables will change. This ActionPerformed() function will be explained better later.

(**The ip must be in the range of 239.0.0.0 to 239.255.255.255 due to its UDP datagram Socket that will use to extend the messages to all connections in our server, for more information visit:** [**https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html**](https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html))

After introducing the Username and IP address, an InetAddress object will be created to verify if the ip has a valid structure and its adequate for UDP and as identifier to start a connection in a specific group, after that the port is stored it in the Port variable to indicate in which port the communication will take place and finally, create the group with the port as a parameter using a MulticastSocket object(**that will be very useful to send and receive Ip multicast packets, in this case the messages from other users.**)



Finally, after creating the group with the specific port, the socket will join to the specific group given by the identifier group, in this case the selected ip for the user to connect to that group, and for the user to properly start a session group, it’s necessary to create a thread for every user instance using the Class Messages, the one that we will talk about later, so now with a multi user group created, the connection established and user instance to participate in the group, we just need a message to send(**Figure 2.2**).

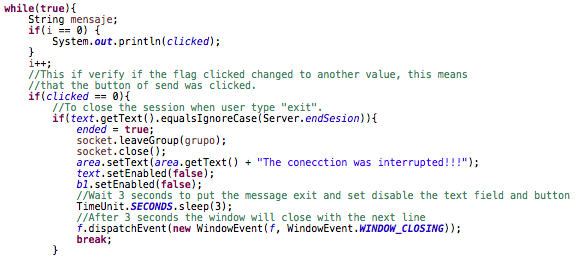


**Figure 2.2**

Now, for the communication we need and endless loop to send and receive messages for other users because we don't know when the user will type “exit”, so the only way to interrupt the connection is typing “exit” to leave the group.

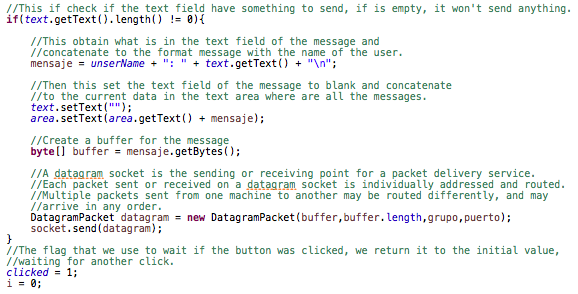
The second if verifies if the value of the clicked variable is 0, which means the user pressed the button and wants to send a message. In the ActionPerformed() function we change the value of clicked, but it will show later.

Once the user presses the button, the next if will get the text of the text field where is the message and verify if it says “exit”. If it said “exit”, the ended variable will change to true, the user will leave the group and the socket will close the communication. The text area, where all the messages are shown, displays a message that the connection ends, the text field where the user writes the message and the button of send changes the property to disable, the Main() function sleeps 3 seconds, the frame closes and we go out of the endless loop and the program finishes.



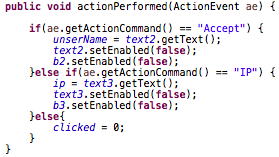
However, if the message is different from “exit”, the next if is for verifying that the message that the user wants to send, contains something to send, if not, it doesn’t send the message.

If the message contains something, the program proceeds to send the message. For sending messages, it's necessary to create a buffer to share with other users in the group, after typing the message by the user and send it, the buffer will store the messages and store the number of bytes for that message, after that, a datagram socket is sending or receiving point for a packet delivery service. Each packet sent or received on a datagram socket is individually addressed and routed. Multiple packets sent from one machine to another may be routed differently, and may arrive in any order to the users connected to the group. At the end, it just modifies the clicked value to 1 again, waiting for the user to press the send button again.



Finally, the ActionPerformed() function, which all the variables that have an actionlistener have, will be executed if the variables with it are actioned, in this case just the buttons.(b1, b2 and b3).

This function verifies which of the buttons were pressed and determines what action it has to do. If the b2 button (“Accept”) is pressed, the username of the user will change to the current text of the text2 variable and changes the property disable to the b2 button and text2 variable. If the b3 button (“IP”) is pressed, the ip variable of it will change to the current text of the text3 variable and changes the property disable to the b3 button and text3 variable. If the b1 button (“Send”) is pressed, it just changes the value of clicked to 0.

****

**Sumary Server.java**

As a result for Server.java, this class is in charge of setup the values and start a connection with the group, as well, it is in charge of creating every instance of the user and connects the user to the selected group catching and sending the messages to the group.

**2.2 Messages.java**

As we said in the Server.java part, it's necessary for our readers to know which libraries will be used and the reason of its use.

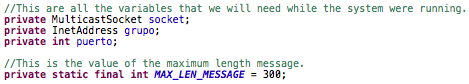
****

**java.net.\*:** This library will be in charge to create the ip configuration network for the group and the multicastSocket, to create and connect to a port to a specific group and start chatting with the people connected to the group.

**java.io.\*:** Imports all the classes that are defined in java.io package to your file. This enables your java program to use those classes and their methods to achieve some tasks. But if you needed more than one file to be imported in your program (and if you're lazy enough to write a line for each class, you could use import java.io.\*) in this case, we call this class to manage the exceptions for the creation and use of the sockets in the group (If you are not able to connect for some reason this library will throw an exception. **For more information and the benefits of this library visit:** <https://docs.oracle.com/javase/7/docs/api/java/io/package-summary.html> ).

This class is the one that the Threads will run to receive and send messages. That is why this class implements Runnable.

****

****

In the first part of the class Messages, we declare four class variables called socket, grupo, puerto y MAX\_LEN\_MESSAGE.

* **socket.** This variable is a MulticastSocket object which will work to create the group.
* **grupo.** This is an InetAddress variable which will be created to verify if the ip has a valid structure and it’s adequate for UDP and as an identifier to start a connection in a specific group.
* **puerto.** This variable is an int variable which is to indicate in which port the communication will take place.
* **MAX\_LEN\_MESSAGE.** This is an int variable that cannot change and defines the maximum length of a message.

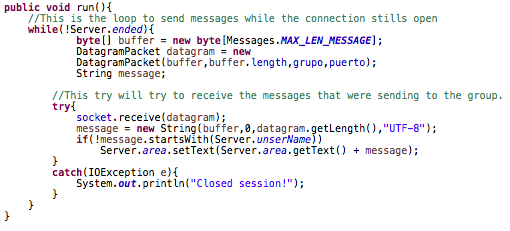
After the variables, we define a constructor of the class.

****

The run() function is the function that the Thread will run when the Thread starts. The function contains a while loop that will be cycled until the Server.ended variable changes to false, and that will just happen, as we saw in the Server.java file, when the user sends “exit” in the chat.

In the function, we create a buffer of bytes with the length of the MAX\_LEN\_MESSAGE that was defined before. The next step is to create a datagram socket sending or receiving node(users) for a packet delivery service, to define to what group we are connected, by which port and the maximum size of the message. We define a string Message that will contain any received message.

The Try will try to receive every message, if there is a problem, it throws an error. So, inside the try, the socket will try to receive the message, the message obtained with the datagram packet and contained in the buffer will be assigned to the message variable and the if will ask that if the message starts with the name of the user, it will be ignored but if not, if will display in the area variable.

****

**3 Setup and instructions for running the application:**

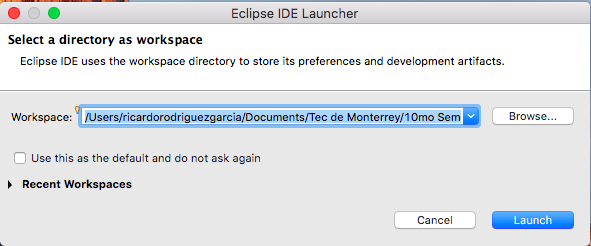
**Setup**

To run this program, you have to have installed in your computer one Integrated Development Environment (IDE). The 2 options that we consider better to use are Eclipse and NetBeans.

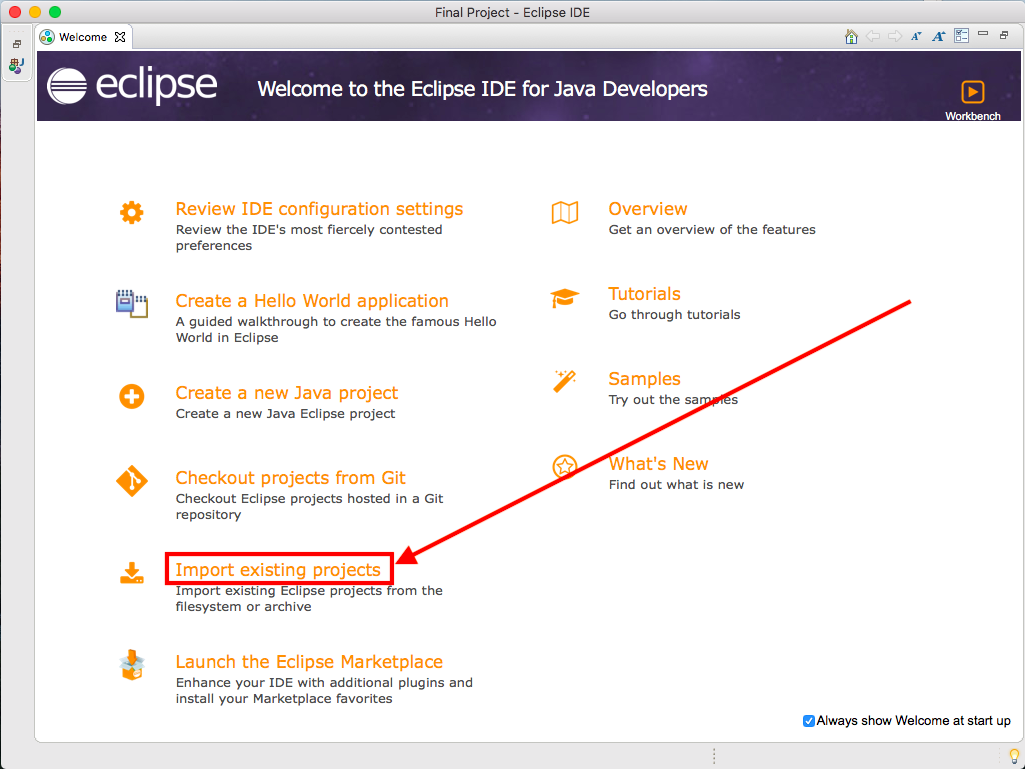
****

Eclipse is the one that we used to develop our program and run it. The instructions will be for Eclipse IDE, so if you don’t know how to install the IDE you can use the next tutorial to install it. (<https://www.youtube.com/watch?v=CHb6HBo2h7U>). Also you need to have installed Java on your computer, that’s why the next link shows you how to do that. (<https://java.com/en/download/help/download_options.xml>)

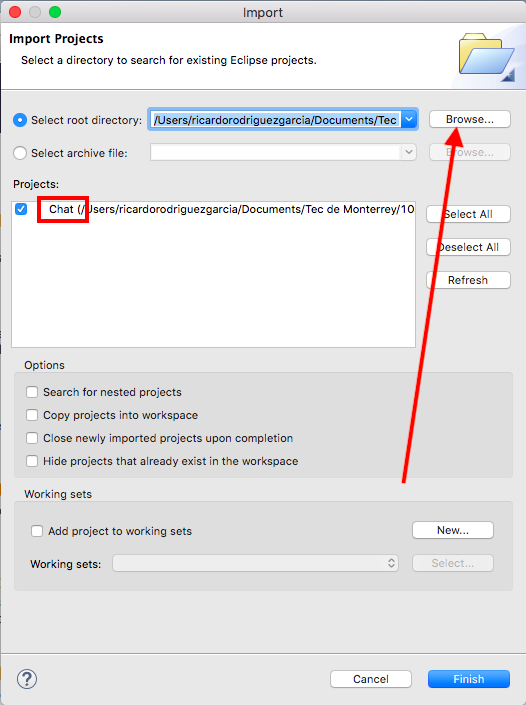
Once you have installed everything in your computer, the first step is to open the Eclipse IDE and it will display a window where you have to put the location where you will be working at.



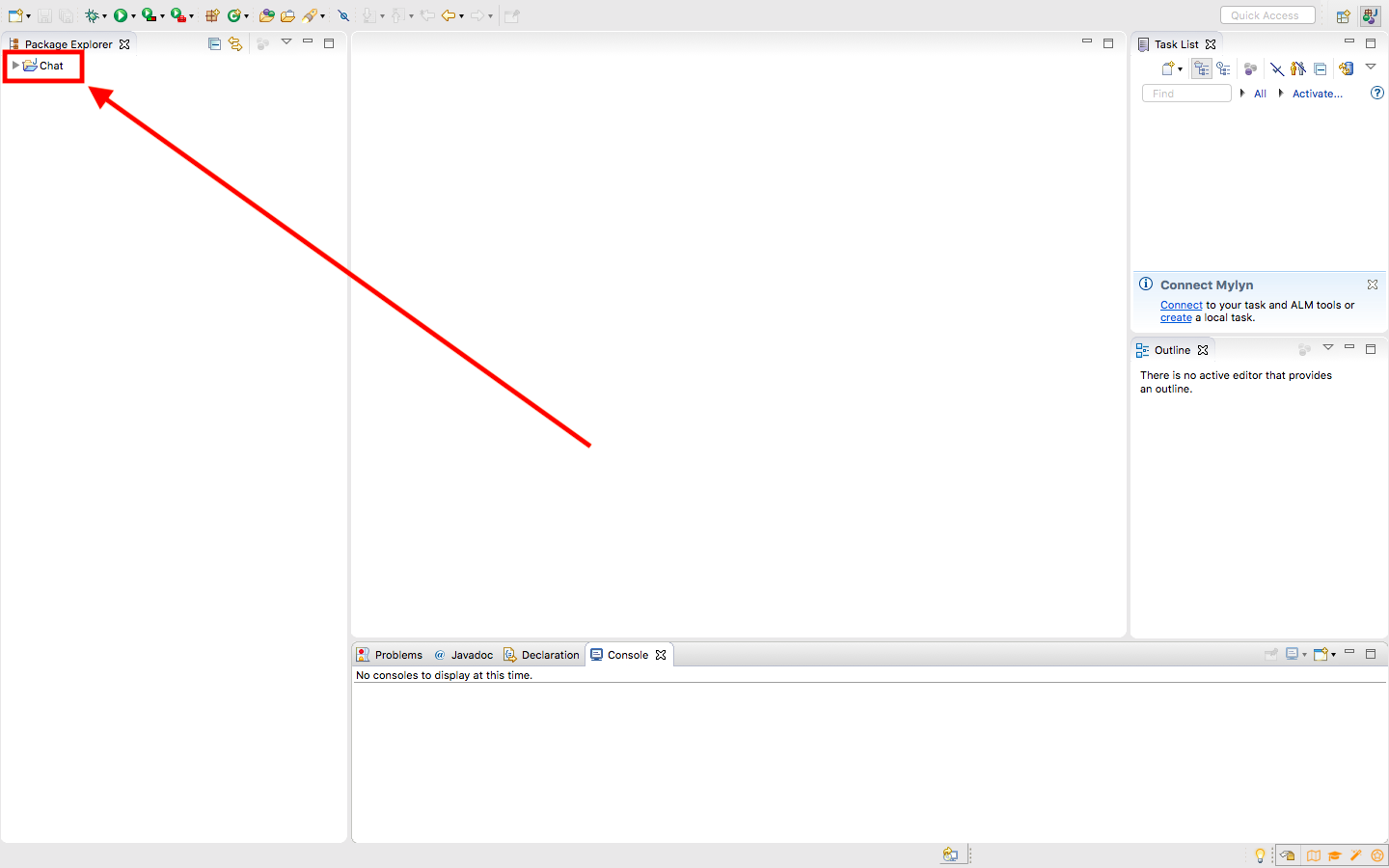
After you introduce the location of your workspace, you must press Launch button and it will display the next interface where you have to click on the “Import existing projects”.



Then, it will display a window where you have to put the location of the folder project. In the big blank space will show the folder project that will be imported to the workspace and it has to be selected. Then you have to press the “Finish” button.

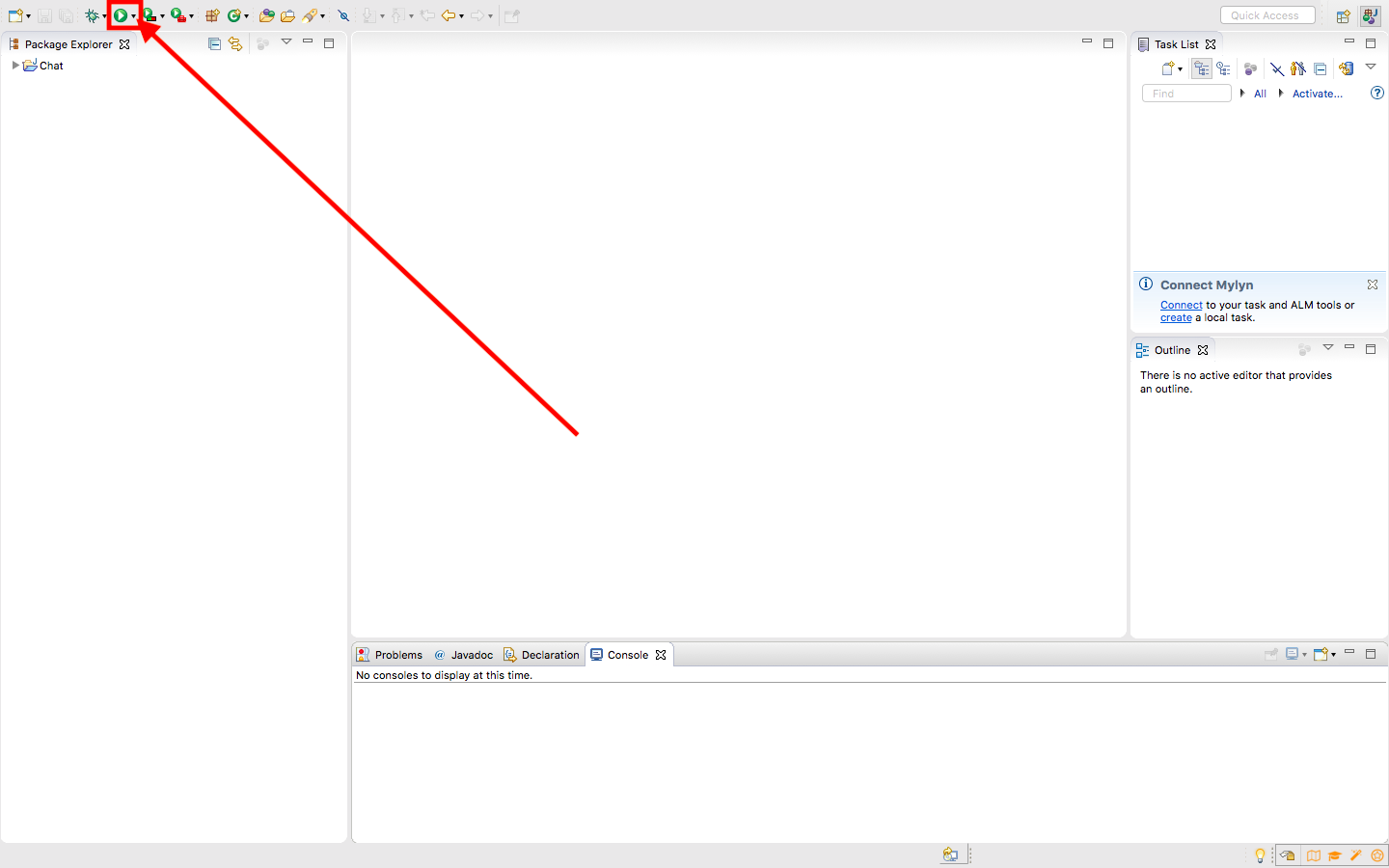


Then, Eclipse will initiate and the project will be at the left side of the interface.

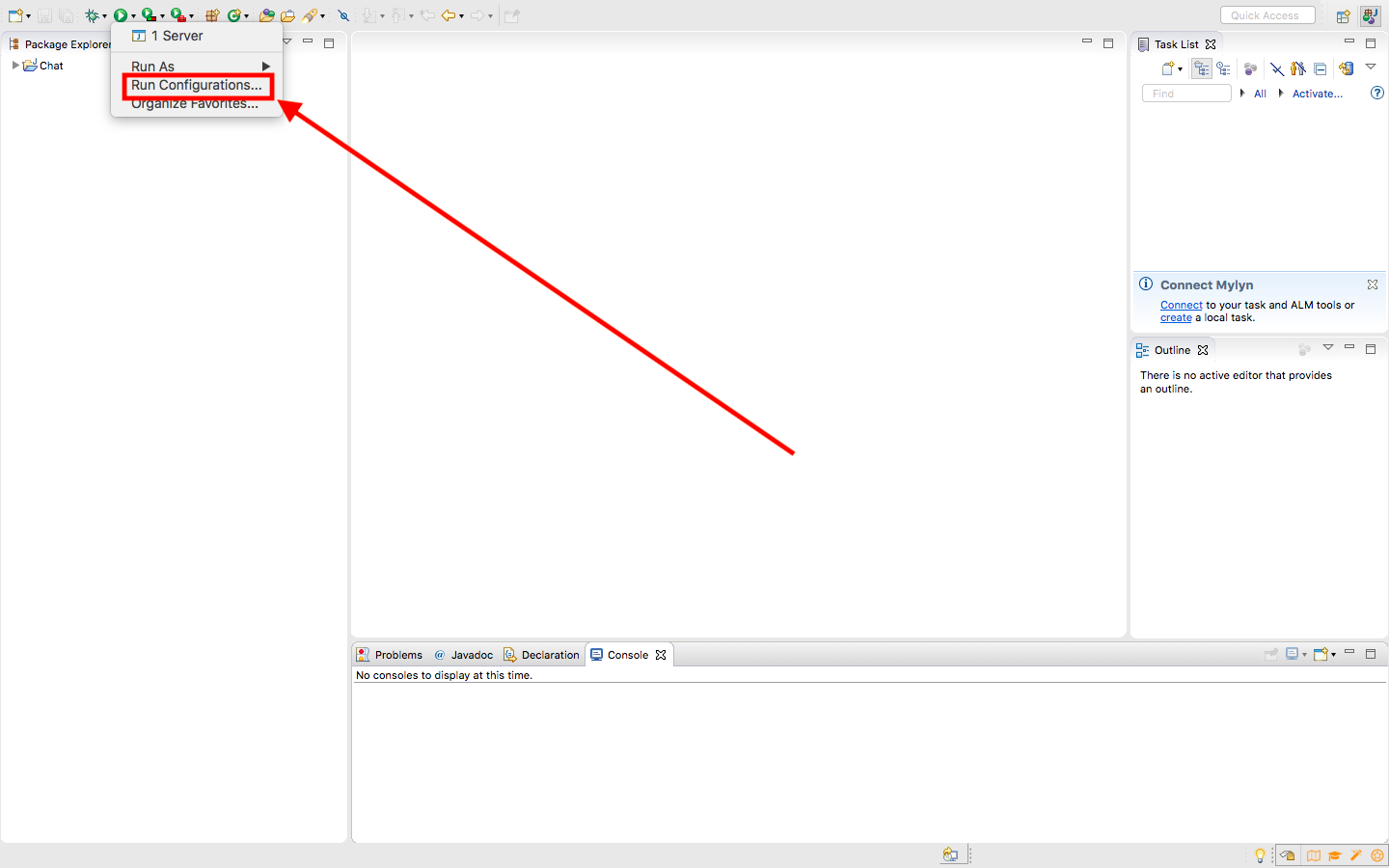


**Running instructions**

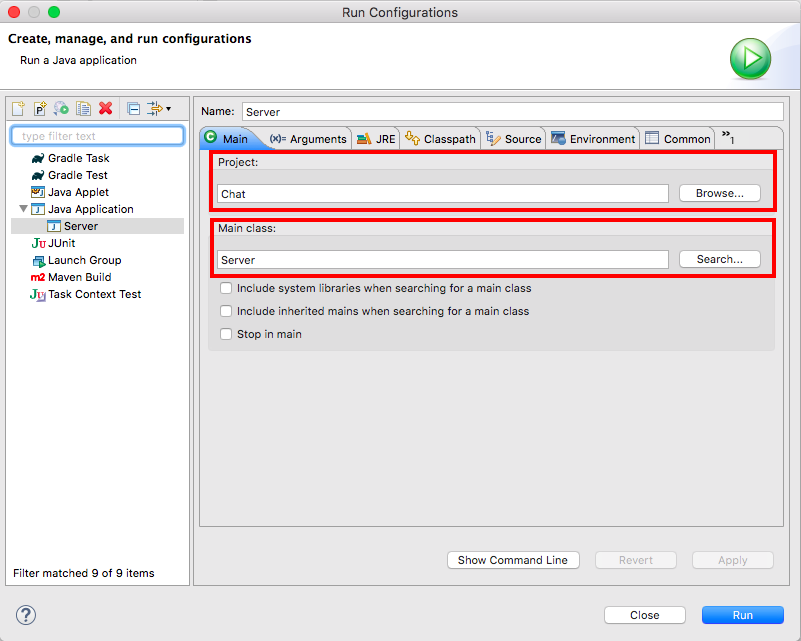
To run the program, at the top will be a “play” icon and beside it, a small arrow. Click the small arrow.

****

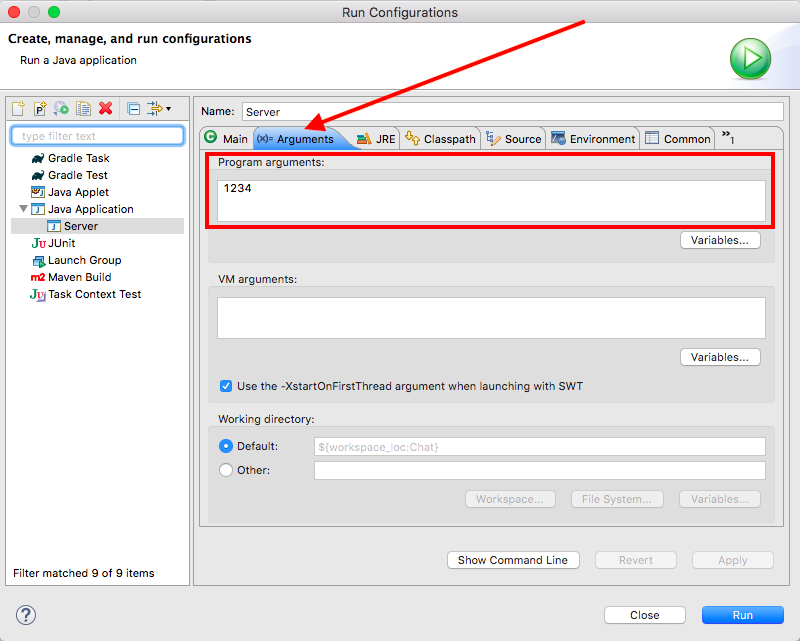
Then, go to “Run Configurations”.

****

It will display a window with the configuration of the execution. You have to verify that in the space of the project name said “Chat”, if not, you have to click on the “Browse” button and select the corresponding folder of the project. Below this, you have to verify that in the space of the main class called “Server”, if not, you have to click on the “Search” button and select the Server class.



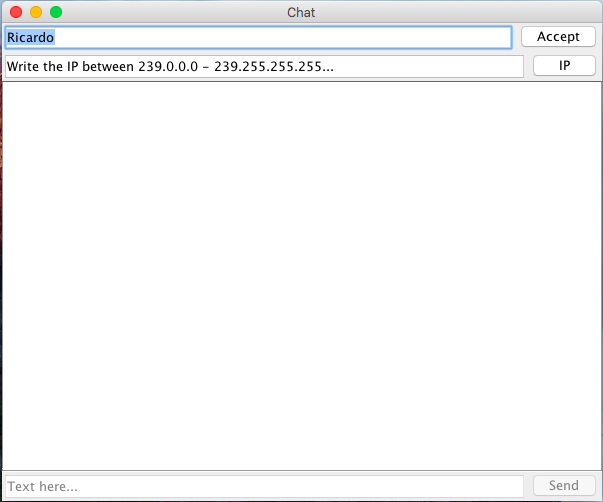
The next step is moving to the “Arguments” tab. Below the tab says “Program arguments” and below it, there is a big blank space where you must write the port that the program will use for the communication. We will use the “1234” port for this example.



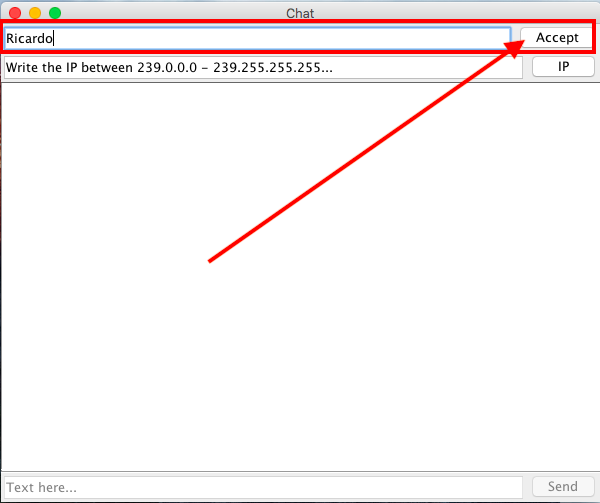
Once you put the port number, you just have to click the “Run” button at the bottom of the window and the program will start to run.

**4. Test and usage of the application:**

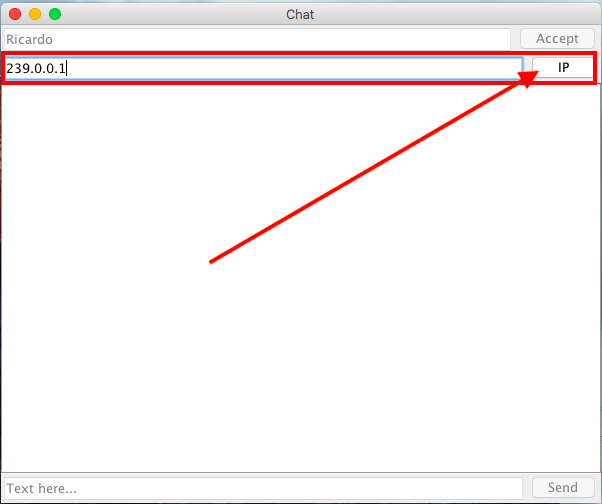
Once you run the program, it will display the graphic interface.



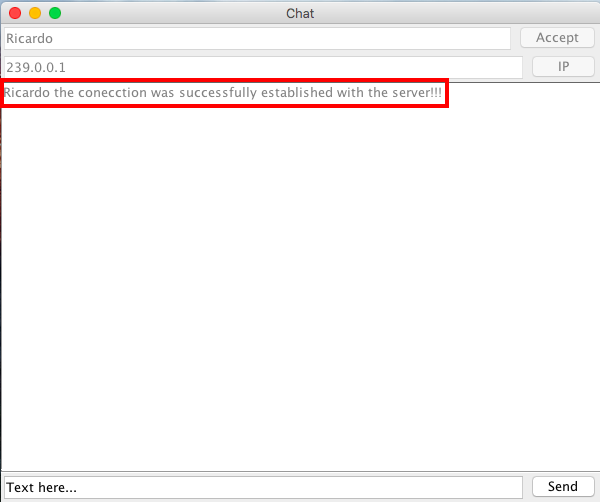
The first thing to do is to write your username in the first blank space at the top side and then, click the “Accept” button.



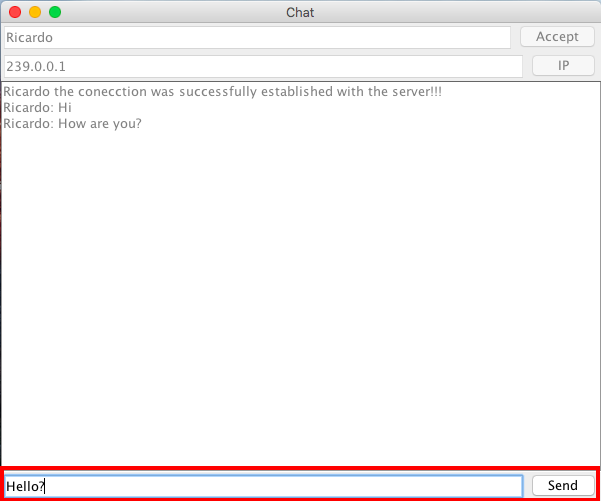
The next step is to write the ip address in the second blank space at the top side and click the “IP” button.

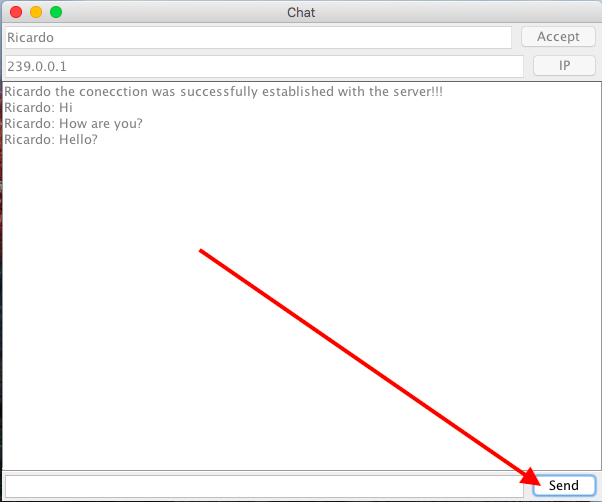


If the connection was successful, the program will display “UserName the connection was successfully established with the server!!!”the text fields where you put the username and the ip address change to disable and the corresponding buttons too. The text field and “Send” button which are at the bottom of the window, will change to enable and you can start sending messages.

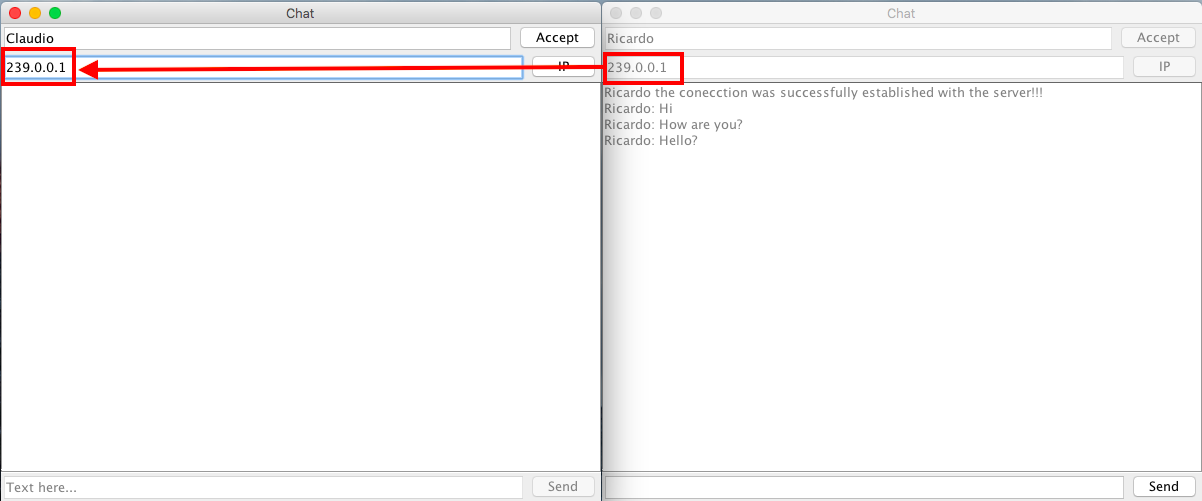


From now on, you can start sending messages, you just have to write the message in the text field at the bottom of the window and then click the “Send” button. And it will display the message in the text area in the center of the window.

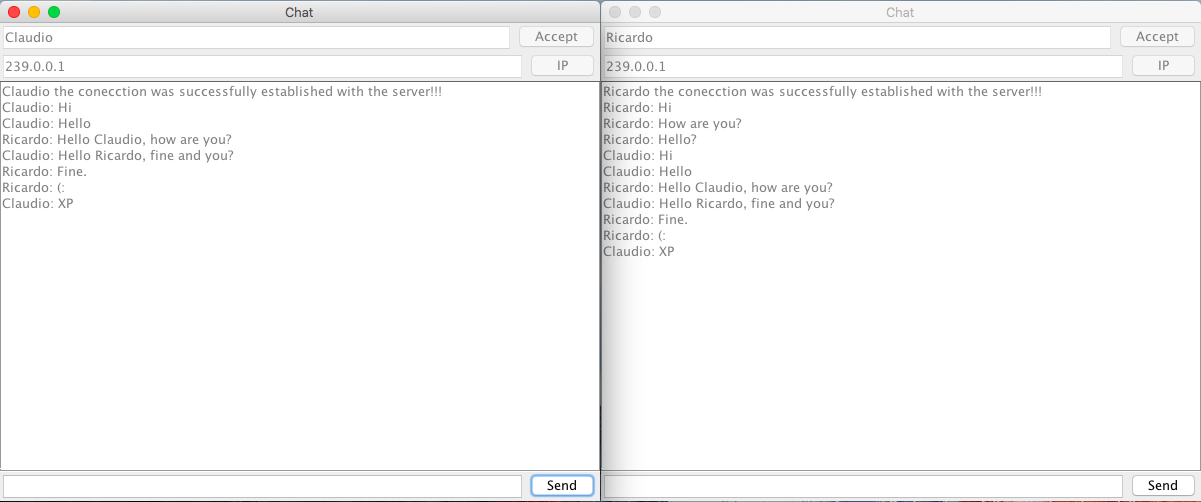


****

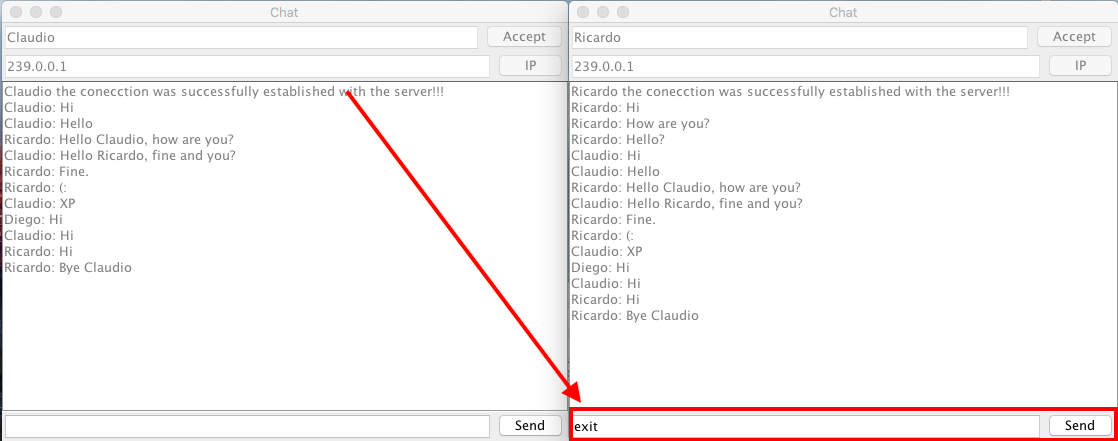
Now you can send messages but nobody is going to receive your messages, that’s why you have to run the program again, in order to create another thread that will run the graphical interface and the communication services, and basically you do the same steps that we said before to establish the communication. In the next image, we are running two different executions of the program. If the user “Claudio” wants to text the user “Ricardo”, he must write the same ip address of “Ricardo” to join the same group of “Ricardo”.Then, you proceed with the process of the connection.

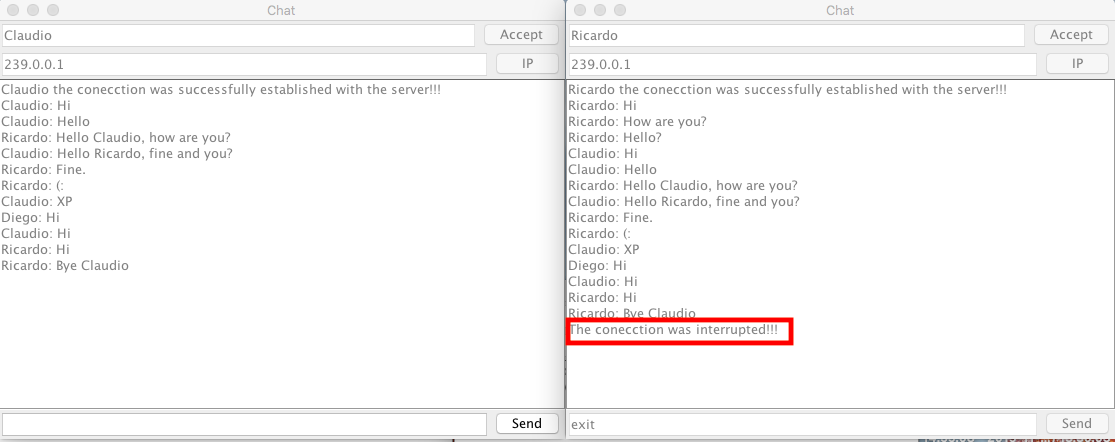


Now there are two users connected to the same group. You can start sending messages between the users and each one must receive all the messages.



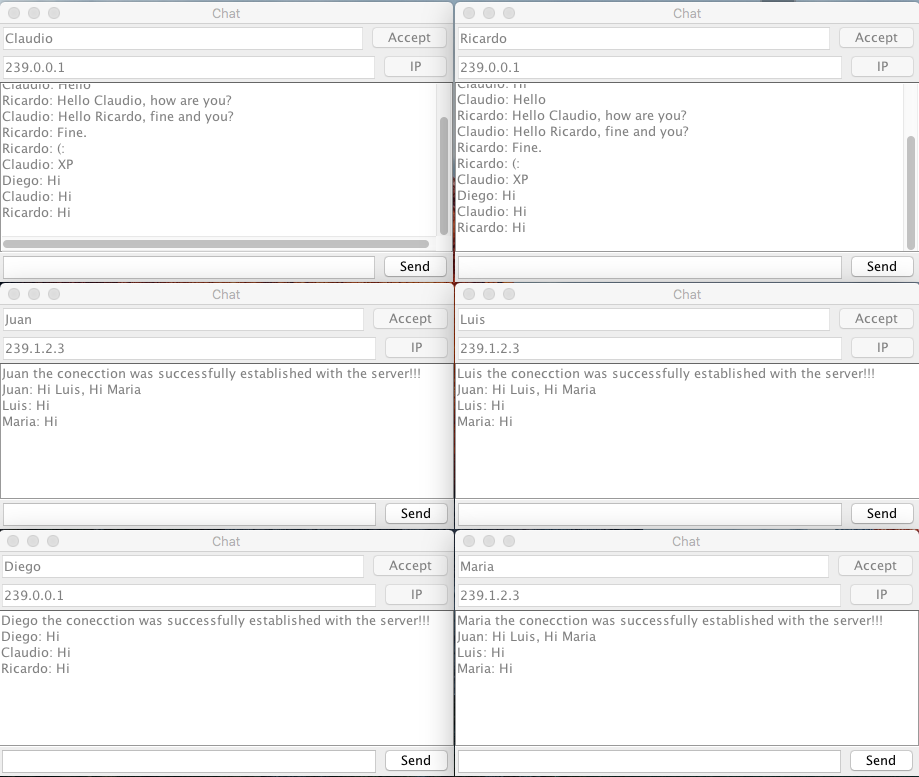
You can do this with a lot of users, there's no limit. And if you don’t want to speak with “Ricardo”, you can use another IP to start another group with other users. And finally, if you don’t want to continue texting, you can stop the program writing “exit” in the text field for the message and pressing the “Send” button, so the text field and “Send” button change to disable, the program displays a message “” and the execution finishes, and the window will be closed.

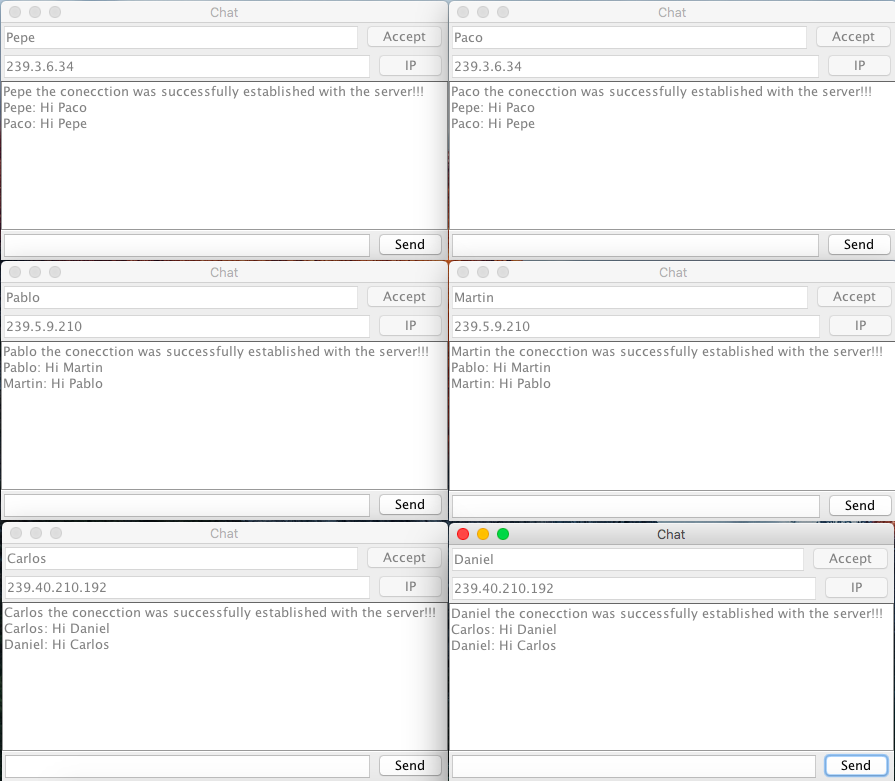




In that moment, the running process finishes.

**Test Cases**

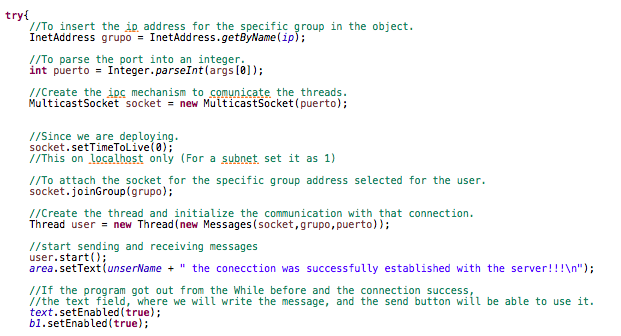
****

****

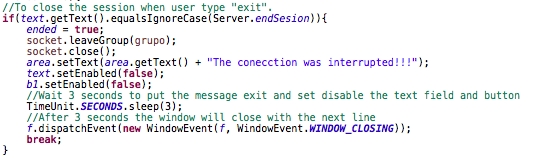
**5. Work of each member:**

**Claudio implementation**

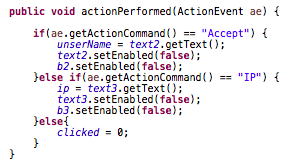
Claudio implements the part of code in Server where the InetAddress, Port, MulticastSocket is initialized, establish a connection with the IP and creates a Thread that will run the Messages class, the implementation for the ip and multicast socket will have the final function to establish the connection between users (threads) and the message buffer to send and receive in broadcast way to other users connected to the port, IP and selected group.

****

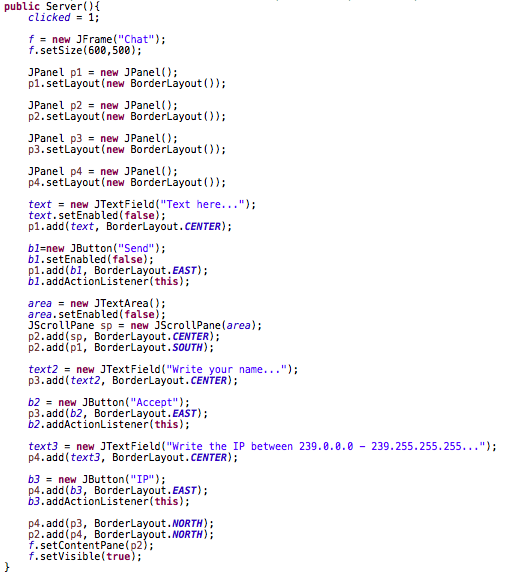
Furthermore, Claudio worked in the logic and action that will occur if the user types “exit” to finish the connection when the user types the word “exit” in the terminal.

****

Moreover, he implements the function of ActionPerformed that will run when a button is clicked.

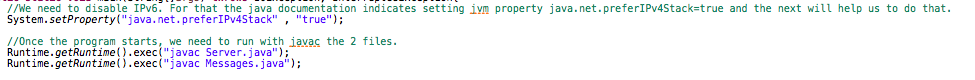


Finally, Claudio worked together with Ricardo to implement the graphic interface.

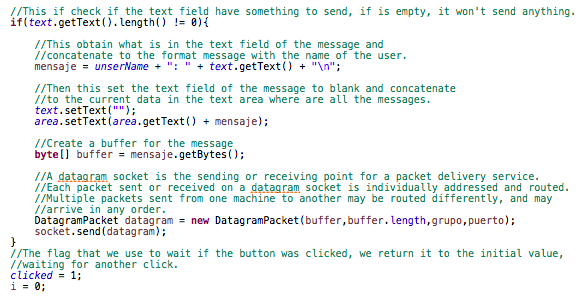
****

**Ricardo implementation**

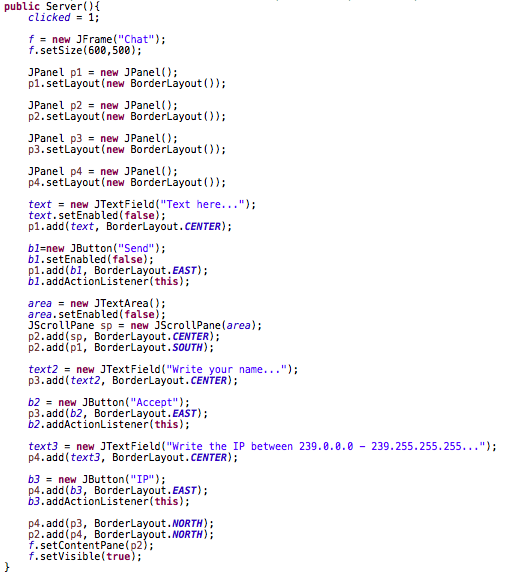
Ricardo implements the part of the code Server where the IPv6 sets to disable and modifies the JVM property and the lines of code where the Server and Messages files are running.

****

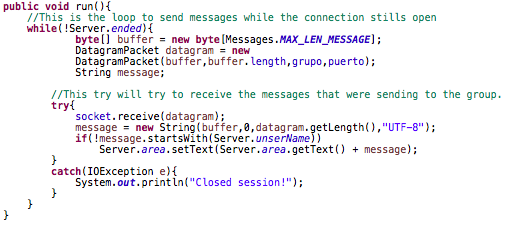
Moreover, he implements the logic part where if the text message is different from empty, the text where the messages were written in the graphic interface will be obtained and concatenated to the message variable, introduce it to a bytes array and send it in a Datagram Packet by a socket.

****

Furthermore, he worked with Claudio to implement the graphic interface.

****

Finally, Ricardo implements the main part of the Messages file where the Thread will be running until the user types “exit”, if not, it will define variables to try to receive every time messages that will be sent to the group.

****

**References:**

* Castelnovo, P. (2019). Variables volátiles en Java. Retrieved 17 November 2019, from <https://medium.com/@pablocastelnovo/variables-vol%C3%A1tiles-en-java-f5ae078bf8b9>
* InetAddress (Java Platform SE 7 ). (2019). Retrieved 17 November 2019, from <https://docs.oracle.com/javase/7/docs/api/java/net/InetAddress.html>
* MulticastSocket (Java Platform SE 7 ). (2019). Retrieved 17 November 2019, from https://docs.oracle.com/javase/7/docs/api/java/net/MulticastSocket.html