



Faculty of Engineering
The Hashemite University

IP Project: Socket Programming in Java
Instructor: Dr. Mohammad Al-Hammouri

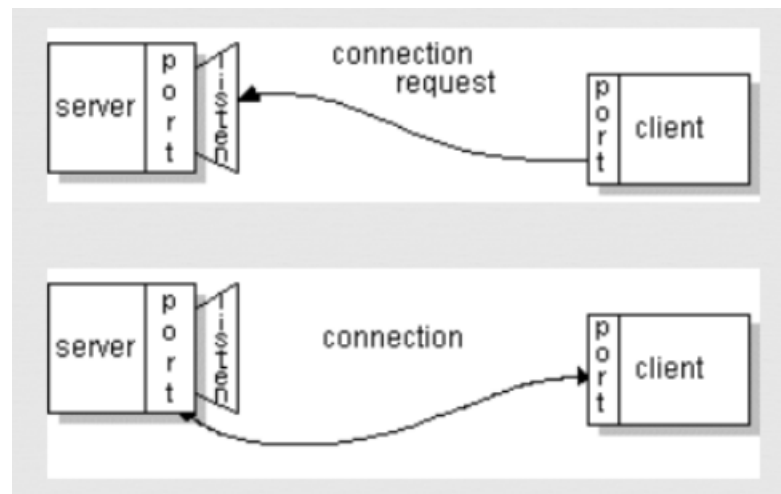
| | |
|----------------------|-------------|
| Name | Noor Al-Ali |
| University ID | 1732975 |
| Due Date | 09/01/2022 |

• Overview

This project presents an introduction to sockets programming over TCP/IP networks, and shows how to write “multiclient”/server applications in Java.

A socket is one endpoint of a two-way communication link between two programs (server and client) running on the network.

In order to communicate over the TCP (transfer control protocol) protocol, a connection must first be established between the server and the client. Server listens for a connection request and client asks for a connection. Once two sockets have been connected, they can be used to transmit data in both or one directions. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to.



• Objectives

1. In this project we need to write a client/server application in Java.
2. Multiple Clients can communicate with the Server.
3. Each Client must provide their ID with a circle radius.
4. The Server replies to the client with the circle area.

• Socket TCP programming in JAVA

In the first phase the program allowed only one client to connect at a time, to allow multiple clients to connect, we used threads in the server.

We need to write two JAVA programs; PacketReceiver.java (server side) and PacketSender.java (client side).

- Client side (PacketSender.java):

1. In the main method we will create a variable to get a random ID for the client.

```
public static void main(String argsv[])
{
    final double senderId=Math.random()*10000;
    final int senderIdInteger = (int) senderId;
```

2. Open a socket on port 3034, open input and output streams and Scanner object to take input from the user.

```
// creating socket object
Socket mySocket = new Socket("localhost", 3034);
// Scanner object to take data from the user
Scanner scanner = new Scanner(System.in);
DataOutputStream dataOutput = new DataOutputStream(mySocket.getOutputStream());
Scanner in = new Scanner(mySocket.getInputStream());
```

3. In a loop we will keep scanning input from the user and write it in using modified UTF-8 encoding with the user ID, to the current output stream while the user input is not “Close”.

```
while (true)
{
    //scanner past the current line and returns the input
    System.out.println("Enter the raduis then press enter... or send \"Close\" to close the connection.... ");
    System.out.print("Raduis ");
    String packet = scanner.nextLine();
    // writes primitive data write of this packet in modified UTF-8 format and send it to the stream
    dataOutput.writeUTF(senderIdInteger+" "+packet);
    if (packet.equalsIgnoreCase("Close")) break;
    System.out.println("The server finished the calculations, the Area is " + in.nextLine());
}
```

4. When the user input is “Close” we should close the stream and the socket.

```
//close stream
dataOutput.close();
// close the connection
mySocket.close();
```

- Server side (PacketReceiver.java):

1. First we need to create a global static variable to use it as a counter to count the number of clients and two methods to increase and decrease the counter once a client connects or disconnects.

```
static int x=0;
static int counterInc() {
    return x++;
}

static int counterDec() {
    return x--;
}
```

2. In the main method we will define the port that we want to connect with in the TCP connection and create a server socket to establish the connection.

```
final int Port = 3034;
ServerSocket myServerSocket = new ServerSocket(Port);
```

3. TCP Echo Server allows only one client to connect at a time, to allow multiple clients to connect, we will use threads for each client connection in the server and define the run() method of thread class which is called if the thread was constructed.

```
Socket mySocket = myServerSocket.accept();
Thread t = new Thread() {
    public void run() {
```

4. The server will wait for a client to connect, When any client connects we will call the counterInc() method to increase the clients counter, then create a DataInputStream to receive input packet from the client using socket.

```
// waiting for a client to send connection request
counterInc();
System.out.println("New Client Connected... number of clients is " +x);
// Receiving input messages from the client using socket
PrintWriter out = new PrintWriter(mySocket.getOutputStream(), true);
DataInputStream stream = new DataInputStream( mySocket.getInputStream() );
```

5. While the received packet is not “Close” we will read the packet (user input and ID) from the stream, calculate the area and send it to the client.

```
while (!packet.split(" ")[1].equals("Close")){
    try{
        // read from stream
        packet = stream.readUTF();
        // the packet is two parts client id and radius so we need to split them ("id radius") example packet = "1221 2"
        String[] ary = packet.split(" ");
        if (!ary[1].equals("Close")){
            System.out.println( "Server recieved this radius (" +ary[1] +") from Client " +ary[0] );
            double radius = Double.valueOf(ary[1]);
            double area = Math.PI* radius *radius ;
            out.println(area);
        }
        else{
            counterDec();
            System.out.println("Client "+ ary[0] + " Closed the connection ... number of clients is " +x);
        }
    }
}
```

• Result

- How to run the application:
 1. Open a terminal window for the server and a terminal for each client.
 2. In your terminal, navigate to the directory where your files are located.
 3. In the server terminal, to compile the server program run:

```
$ javac PacketReceiver.java
```

Then to start the server run:

```
$ java PacketReceiver
```

This will establish the connection and the server will start waiting for clients.

```
IP$ javac PacketReceiver.java
IP$ java PacketReceiver
Server is waiting for a client to connect ...
```

4. In a new terminal, to compile the client program run:

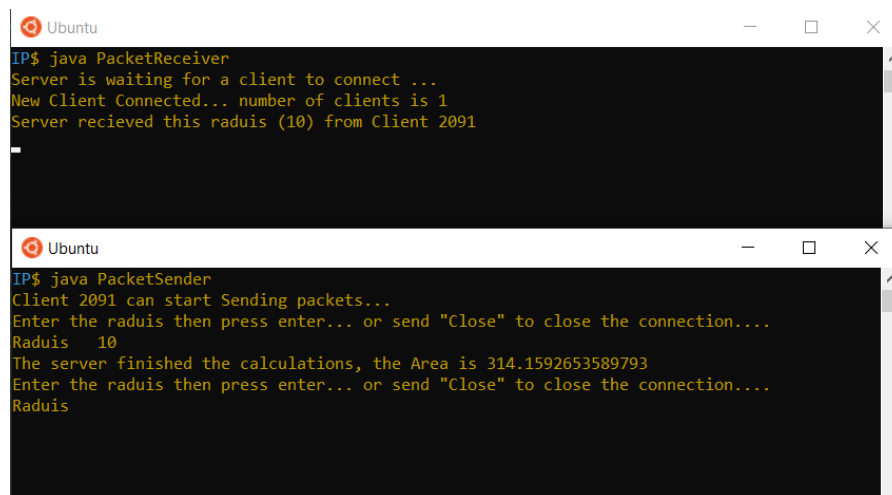
```
$ javac PacketSender.java
```

Then to start the client run:

```
$ java PacketSender
```

This will connect the client with the connection.

5. Now the client is able to enter the radius and send it to the server, the server will receive it, confirm the receiving, calculate the area and send it to the client.



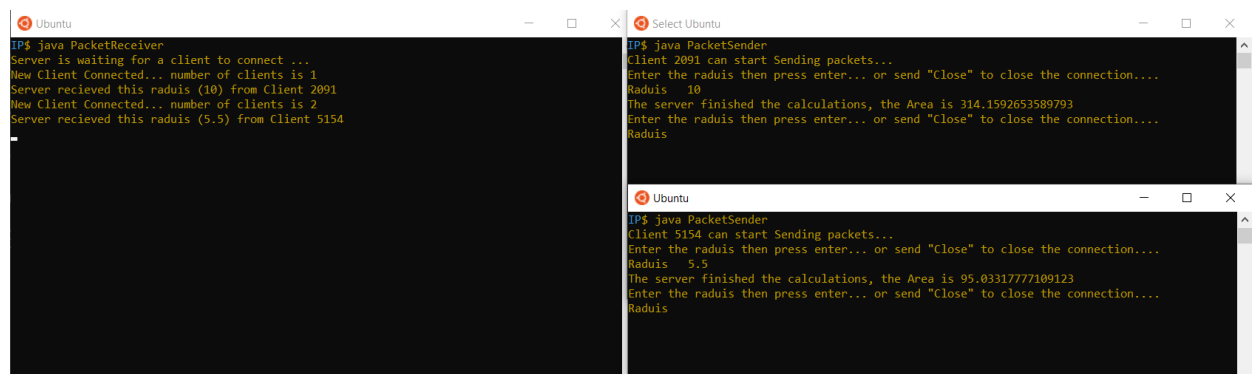
```
Ubuntu
IP$ java PacketReceiver
Server is waiting for a client to connect ...
New Client Connected... number of clients is 1
Server recieved this raduis (10) from Client 2091
-

Ubuntu
IP$ java PacketSender
Client 2091 can start Sending packets...
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis  10
The server finished the calculations, the Area is 314.1592653589793
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis
```

6. If you want to connect with multiple clients at the same time you should open a new terminal for each client and run:

\$ java PacketSender

Then you will be able to send data from the client and server will confirm connected with the new client.

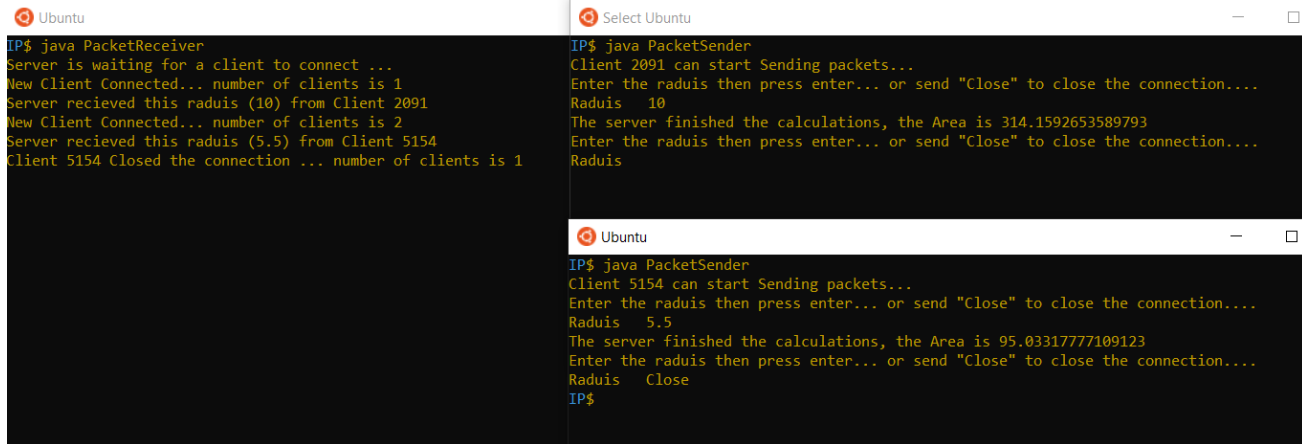


```
Ubuntu
IP$ java PacketReceiver
Server is waiting for a client to connect ...
New Client Connected... number of clients is 1
Server recieved this raduis (10) from Client 2091
New Client Connected... number of clients is 2
Server recieved this raduis (5.5) from Client 5154
-

Select Ubuntu
IP$ java PacketSender
Client 2091 can start Sending packets...
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis  10
The server finished the calculations, the Area is 314.1592653589793
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis

Ubuntu
IP$ java PacketSender
Client 5154 can start Sending packets...
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis  5.5
The server finished the calculations, the Area is 95.03317777109123
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis
```

7. To close the connection, send “Close” from the client side and the server will confirm closing the connection and decrease the clients counter.



The image shows three terminal windows from an Ubuntu environment. The top-left window, titled 'Ubuntu', shows the execution of 'java PacketReceiver'. It displays messages for two clients: Client 2091 (radius 10) and Client 5154 (radius 5.5), and shows the number of clients decreasing as they close connections. The top-right window, titled 'Select Ubuntu', shows the execution of 'java PacketSender'. It prompts for a radius, calculates the area (314.1592653589793 for radius 10), and prompts for a close command. The bottom window, also titled 'Ubuntu', shows another execution of 'java PacketSender' for Client 5154, calculating the area (95.03317777109123 for radius 5.5) and accepting the 'Close' command.

```
IP$ java PacketReceiver
Server is waiting for a client to connect ...
New Client Connected... number of clients is 1
Server recieved this raduis (10) from Client 2091
New Client Connected... number of clients is 2
Server recieved this raduis (5.5) from Client 5154
Client 5154 Closed the connection ... number of clients is 1

IP$ java PacketSender
Client 2091 can start Sending packets...
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis  10
The server finished the calculations, the Area is 314.1592653589793
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis

IP$ java PacketSender
Client 5154 can start Sending packets...
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis  5.5
The server finished the calculations, the Area is 95.03317777109123
Enter the raduis then press enter... or send "Close" to close the connection....
Raduis  Close
IP$
```

- **Conclusion**

In this project, we learnt about socket application over TCP/IP, and wrote a simple multiClient/Server application in Java.