Socket Programming in Java

Difficulty Level : Medium Last Updated : 08 Nov, 2021

This article describes a very basic one-way Client and Server setup where a Client connects, sends messages to the server and the server shows them using a socket connection. There's a lot of low-level stuff that needs to happen for these things to work but the Java API networking package (java.net) takes care of all of that, making network programming very easy for programmers.

Client-Side Programming

Establish a Socket Connection

To connect to another machine we need a socket connection. A socket connection means the two machines have information about each other's network location (IP Address) and TCP port. The java.net.Socket class represents a Socket. To open a socket:

```
Socket socket = new Socket("127.0.0.1", 5000)
```

- The first argument – **IP address of Server**. (127.0.0.1 is the IP address of localhost, where code will run on the single stand-alone machine).
- The second argument – **TCP Port**. (Just a number representing which application to run on a server. For example, HTTP runs on port 80. Port number can be from 0 to 65535)

Communication

To communicate over a socket connection, streams are used to both input and output the data.

Closing the connection

The socket connection is closed explicitly once the message to the server is sent.

In the program, the Client keeps reading input from a user and sends it to the server until "Over" is typed.

Java Implementation

```
// A Java program for a Client
import java.net.*;
import java.io.*;
```

```
public class Client
{
    // initialize socket and input output streams
    private Socket socket            = null;
    private DataInputStream  input   = null;
    private DataOutputStream out     = null;

    // constructor to put ip address and port
    public Client(String address, int port)
    {
        // establish a connection
        try
        {
            socket = new Socket(address, port);
            System.out.println("Connected");

            // takes input from terminal
            input  = new DataInputStream(System.in);

            // sends output to the socket
            out    = new DataOutputStream(socket.getOutputStream());
        }
        catch(UnknownHostException u)
        {
            System.out.println(u);
        }
        catch(IOException i)
        {
            System.out.println(i);
        }

        // string to read message from input
        String line = "";

        // keep reading until "Over" is input
        while (!line.equals("Over"))
        {
            try
            {
                line = input.readLine();
                out.writeUTF(line);
            }
            catch(IOException i)
            {
                System.out.println(i);
            }
        }

        // close the connection
        try
        {
            input.close();
            out.close();
            socket.close();
        }
        catch(IOException i)
```

```
            {
                System.out.println(i);
            }
        }

        public static void main(String args[])
        {
            Client client = new Client("127.0.0.1", 5000);
        }
    }
```

Server Programming

Establish a Socket Connection

To write a server application two sockets are needed.

- A ServerSocket which waits for the client requests (when a client makes a new Socket())
- A plain old Socket socket to use for communication with the client.

Communication

getOutputStream() method is used to send the output through the socket.

Close the Connection

After finishing, it is important to close the connection by closing the socket as well as input/output streams.

```
// A Java program for a Server
import java.net.*;
import java.io.*;

public class Server
{
    //initialize socket and input stream
    private Socket          socket   = null;
    private ServerSocket    server   = null;
    private DataInputStream in       =  null;

    // constructor with port
    public Server(int port)
    {
        // starts server and waits for a connection
```

```
        try
        {
            server = new ServerSocket(port);
            System.out.println("Server started");

            System.out.println("Waiting for a client ...");

            socket = server.accept();
            System.out.println("Client accepted");

            // takes input from the client socket
            in = new DataInputStream(
                new BufferedInputStream(socket.getInputStream()));

            String line = "";

            // reads message from client until "Over" is sent
            while (!line.equals("Over"))
            {
                try
                {
                    line = in.readUTF();
                    System.out.println(line);

                }
                catch(IOException i)
                {
                    System.out.println(i);
                }
            }
            System.out.println("Closing connection");

            // close connection
            socket.close();
            in.close();
        }
        catch(IOException i)
        {
            System.out.println(i);
        }
    }

    public static void main(String args[])
    {
        Server server = new Server(5000);
    }
}
```

Important Points

- Server application makes a ServerSocket on a specific port which is 5000. This starts our

Server listening for client requests coming in for port 5000.

- Then Server makes a new Socket to communicate with the client.

```
socket = server.accept()
```

- The accept() method blocks(just sits there) until a client connects to the server.
- Then we take input from the socket using getInputStream() method. Our Server keeps receiving messages until the Client sends “Over”.
- After we’re done we close the connection by closing the socket and the input stream.
- To run the Client and Server application on your machine, compile both of them. Then first run the server application and then run the Client application.

To run on Terminal or Command Prompt

Open two windows one for Server and another for Client

1. First run the Server application as,

```
$ java Server
```

Server started
Waiting for a client …

2. Then run the Client application on another terminal as,

```
$ java Client
```

It will show – Connected and the server accepts the client and shows,
Client accepted

3. Then you can start typing messages in the Client window. Here is a sample input to the Client

```
Hello
I made my first socket connection
Over
```

Which the Server simultaneously receives and shows,

```
Hello
I made my first socket connection
```

```
Over
Closing connection
```

Notice that sending “Over” closes the connection between the Client and the Server just like said before.

If you’re using Eclipse or likes of such-

1. Compile both of them on two different terminals or tabs
2. Run the Server program first
3. Then run the Client program
4. Type messages in the Client Window which will be received and shown by the Server Window simultaneously.
5. Type Over to end.

This article is contributed by **Souradeep Barua**. If you like GeeksforGeeks and would like to contribute, you can also write an article and mail your article to review-team@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.