

Socket Programming in Java

Last Updated: 26-02-2018

This article describes a very basic one-way Client and Server setup where a Client connects, sends messages to server and the server shows them using 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 other 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)
```

- First argument – **IP address of Server**. (127.0.0.1 is the IP address of localhost, where code will run on single stand-alone machine).
- 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 server is sent.

In the program, Client keeps reading input from user and sends 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)
```

```
{  
    socket = new Socket(address, port);
```



```
        // 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();  
    }
```

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

```
        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

```
// 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.readLine();
                }
                catch (IOException e)
                {
                    System.out.println("IOException: " + e);
                }
            }
        }
        catch (Exception e)
        {
            System.out.println("Exception: " + e);
        }
    }
}
```

```
catch(IOException i)
{
```



```
        // 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.

Open two windows one for Server and another for Client



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
```

Which the Server simultaneously receives and shows,

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 showed 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 contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Attention reader! Don't stop learning now. Get hold of all the important Java and Collections concepts with the **Fundamentals of Java and Java Collections Course** at a student-friendly price and become industry ready.

[Socket Programming in Python](#)

[Socket Programming with Multi-threading in Python](#)

[Explicitly assigning port number to client in Socket](#)

[Difference between Secure Socket Layer \(SSL\) and Transport Layer Security \(TLS\)](#)

[Secure Socket Layer \(SSL\)](#)

[Difference between Secure Socket Layer \(SSL\) and Secure Electronic Transaction \(SET\)](#)

[Socket in Computer Network](#)

[Difference between Rest API and Web Socket API](#)

[Java tricks for competitive programming \(for Java 8\)](#)

[Fast I/O in Java in Competitive Programming](#)

[Beginning Java programming with Hello World Example](#)

[Commonly Asked Java Programming Interview Questions | Set 1](#)

[Commonly Asked Java Programming Interview Questions | Set 2](#)

[Multi-threaded chat Application in Java | Set 1 \(Server Side Programming\)](#)

[Multi-threaded Chat Application in Java | Set 2 \(Client Side Programming\)](#)

[Comparison of Java with other programming languages](#)

[Object Oriented Programming \(OOPs\) Concept in Java](#)

[Java Programming Basics](#)

[Different Approaches to Concurrent Programming in Java](#)

Article Tags : [Computer Networks](#) [Java](#)

Practice Tags : [Java](#) [Computer Networks](#)

☐ To-do ☐ DoneBased on **24** vote(s)[Improve Article](#)

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

[Load Comments](#)

 5th Floor, A-118,
Sector-136, Noida, Uttar Pradesh - 201305

 feedback@geeksforgeeks.org

Company

[About Us](#)

Learn

[Algorithms](#)

Practice

[Courses](#)

Contribute

[Write an Article](#)



@geeksforgeeks, some rights reserved