Java Socket programming is used for communication between the applications running on different JRE.

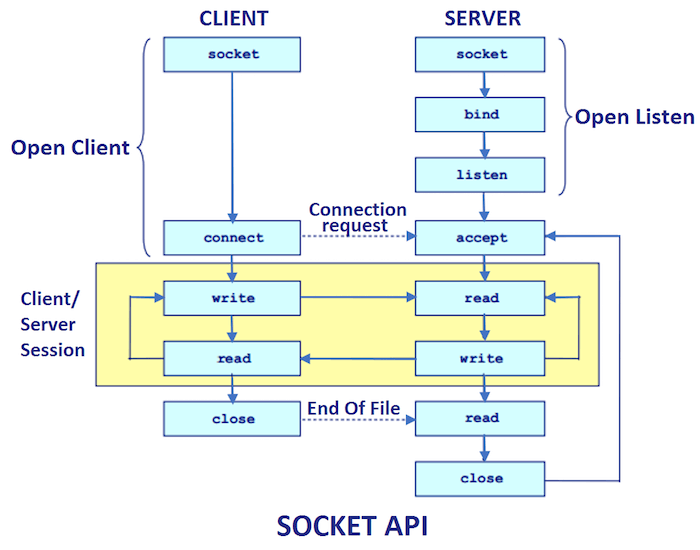
Java Socket programming can be connection-oriented or connection-less.

Socket and ServerSocket classes are used for connection-oriented socket programming and DatagramSocket and DatagramPacket classes are used for connection-less socket programming.

The client in socket programming must know two information:

1. IP Address of Server, and
2. Port number.

Here, we are going to make one-way client and server communication. In this application, client sends a message to the server, server reads the message and prints it. Here, two classes are being used: Socket and ServerSocket. The Socket class is used to communicate client and server. Through this class, we can read and write message. The ServerSocket class is used at server-side. The accept() method of ServerSocket class blocks the console until the client is connected. After the successful connection of client, it returns the instance of Socket at server-side.



Socket class

A socket is simply an endpoint for communications between the machines. The Socket class can be used to create a socket.

Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public InputStream getInputStream() | returns the InputStream attached with this socket. |
| 2) public OutputStream getOutputStream() | returns the OutputStream attached with this socket. |
| 3) public synchronized void close() | closes this socket |

ServerSocket class

The ServerSocket class can be used to create a server socket. This object is used to establish communication with the clients.

Important methods

|  |  |
| --- | --- |
| **Method** | **Description** |
| 1) public Socket accept() | returns the socket and establish a connection between server and client. |
| 2) public synchronized void close() | closes the server socket. |

Example of Java Socket Programming

**Creating Server:**

To create the server application, we need to create the instance of ServerSocket class. Here, we are using 6666 port number for the communication between the client and server. You may also choose any other port number. The accept() method waits for the client. If clients connects with the given port number, it returns an instance of Socket.

1. ServerSocket ss=**new** ServerSocket(6666);
2. Socket s=ss.accept();//establishes connection and waits for the client

**Creating Client:**

To create the client application, we need to create the instance of Socket class. Here, we need to pass the IP address or hostname of the Server and a port number. Here, we are using "localhost" because our server is running on same system.

1. Socket s=**new** Socket("localhost",6666);

Let's see a simple of Java socket programming where client sends a text and server receives and prints it.

*File: MyServer.java*

**import** java.io.\*;

**import** java.net.\*;

**public** **class** MyServer {

**public** **static** **void** main(String[] args) {

**try** {

ServerSocket ss = **new** ServerSocket(6666);

Socket s = ss.accept();// establishes connection

DataInputStream dis = **new** DataInputStream(s.getInputStream());

String str = (String) dis.readUTF();

System.***out***.println("message= " + str);

ss.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

}

*File: MyClient.java*

**package** com.learning.serializable;

**import** java.io.\*;

**import** java.net.\*;

**public** **class** MyClient {

**public** **static** **void** main(String[] args) {

**try** {

Socket s = **new** Socket("localhost", 6666);

DataOutputStream dout = **new** DataOutputStream(s.getOutputStream());

dout.writeUTF("Hello Server");

dout.flush();

dout.close();

s.close();

} **catch** (Exception e) {

System.***out***.println(e);

}

}

}

[download this example](https://www.javatpoint.com/src/networking/socket.zip)

To execute this program open two command prompts and execute each program at each command prompt as displayed in the below figure.

After running the client application, a message will be displayed on the server console.



Example of Java Socket Programming (Read-Write both side)

In this example, client will write first to the server then server will receive and print the text. Then server will write to the client and client will receive and print the text. The step goes on.

*File: MyServer.java*

**import** java.net.\*;

**import** java.io.\*;

**class** MyServer {

**public** **static** **void** main(String args[]) **throws** Exception {

ServerSocket ss = **new** ServerSocket(3333);

Socket s = ss.accept();

DataInputStream din = **new** DataInputStream(s.getInputStream());

DataOutputStream dout = **new** DataOutputStream(s.getOutputStream());

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***));

String str = "", str2 = "";

**while** (!str.equals("stop")) {

str = din.readUTF();

System.***out***.println("client says: " + str);

str2 = br.readLine();

dout.writeUTF(str2);

dout.flush();

}

din.close();

s.close();

ss.close();

}

}

*File: MyClient.java*

**import** java.net.\*;

**import** java.io.\*;

**class** MyClient {

**public** **static** **void** main(String args[]) **throws** Exception {

Socket s = **new** Socket("localhost", 3333);

DataInputStream din = **new** DataInputStream(s.getInputStream());

DataOutputStream dout = **new** DataOutputStream(s.getOutputStream());

BufferedReader br = **new** BufferedReader(**new** InputStreamReader(System.***in***));

String str = "", str2 = "";

**while** (!str.equals("stop")) {

str = br.readLine();

dout.writeUTF(str);

dout.flush();

str2 = din.readUTF();

System.***out***.println("Server says: " + str2);

}

dout.close();

s.close();

}

}