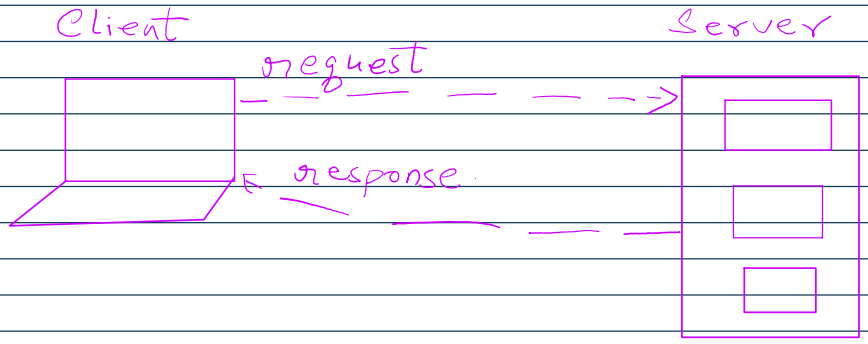
Introduction to Socket Programming

16 April 2025 19:52

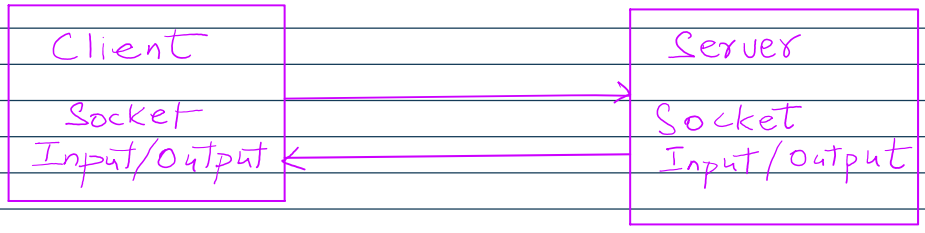
Socket Programming in Java:
Java socket programming is used for communication between the application that is running on different JRE. It can be either connection-oriented or connection-less.



What is socket?
A socket is one endpoint of a 2 way communication between two programs running on the network. The socket is bound to a port number so that TCP layer can layer can identify the application that data. is destined to be sent.

Client Socket: initiates communication.
Server Socket: Weights and listen for incoming requests.

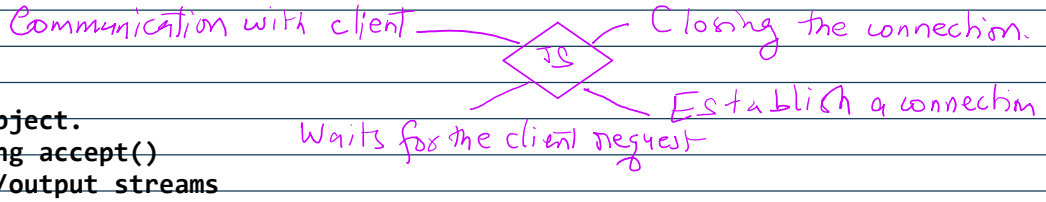
Both client and server use the "java.net" package to establish the connection and communicate.



Methods of socket:

- **public InputStream getInputStream() => returns the InputStream attached with this socket**
- **public OutputStream getOutputStream() => Returns the output stream attached with this socket.**
- **public synchronized void close() => close this socket**
- **public Socket accept() => Returns the socket and establish a connection between client and server.**
- **public synchronized void close() => Closes the server socket.**

Connection Lifecycle:



Server side:

1. **Create a ServerSocket Object.**
2. **Wait for connection using accept()**
3. **Communicate using input/output streams**
4. **Close connection.**

Client side:

1. **Create a socket object specifying server IP address and port number.**

2. Connect to the server.
3. Communicate using input or output streams.
4. Close the connections.

Server code:

```
package SocketProgramming;
import java.io.*;
import java.net.*;
public class Server{
    public static void main(String[] args) {
        try(ServerSocket serverSocket = new ServerSocket(5050)){
            System.out.println("Server is listening on port 5050");
            for(;;){
                Socket clientSocket = serverSocket.accept();
                System.out.println("New client connected");
                new ClientHandler(clientSocket).start();
            }
        }
        catch(IOException ioex){
            ioex.printStackTrace();
        }
    }
}
class ClientHandler extends Thread{
    private Socket socket;
    public ClientHandler(Socket socket){
        this.socket = socket;
    }
    public void run(){
        try(BufferedReader input = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            PrintWriter output = new PrintWriter(socket.getOutputStream(),true)){
                String text;
                while((text = input.readLine())!= null){
                    System.out.println("Received: " + text);
                    output.println("Server echo: "+ text);
                }
            }
        catch(IOException ioe){
            ioe.printStackTrace();
        }
    }
}
```

Client code:

```
package SocketProgramming;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;
public class Client {
    public static void main(String[] args) {
        try (Socket socket = new Socket("localhost", 5050);
        BufferedReader input = new BufferedReader(new InputStreamReader(System.in));
```

```
            PrintWriter output = new PrintWriter(socket.getOutputStream(),true);
            BufferedReader serverInput = new BufferedReader(new
    InputStreamReader(socket.getInputStream())) {
                System.out.println("Connected to server. Type messages: ");
                String msg;
                while(!(msg = input.readLine()).equalsIgnoreCase("exit")){
                    output.println(msg);
                    String reply = serverInput.readLine();
                    System.out.println(reply);
                }
            } catch (IOException e) {
                // TODO: handle exception
                e.printStackTrace();
            }
        }
    }
```

```
Server is listening on port 5050
New client connected
Received: Each client gets its own thread.
Received: this is interesting
```

```
Connected to server. Type messages:
Each client gets its own thread.
Server echo: Each client gets its own thread.
this is interesting
Server echo: this is interesting
exit
```

Advanced Feature:

Multithreaded Server

- **Each client gets its own thread, allowing multiple clients to interact simultaneously.**

Serialization

- **Send objects over sockets using ObjectInputStream and ObjectOutputStream.**

Secure Socket layer(SSL)

- **Use SSLSocket and SSLServer socket for encrypted communication.**