Title : Write a program for creating mini chat application using socket programming

Socket programming in Java allows different programs to communicate with each other over a network, whether they are running on the same machine or different ones.

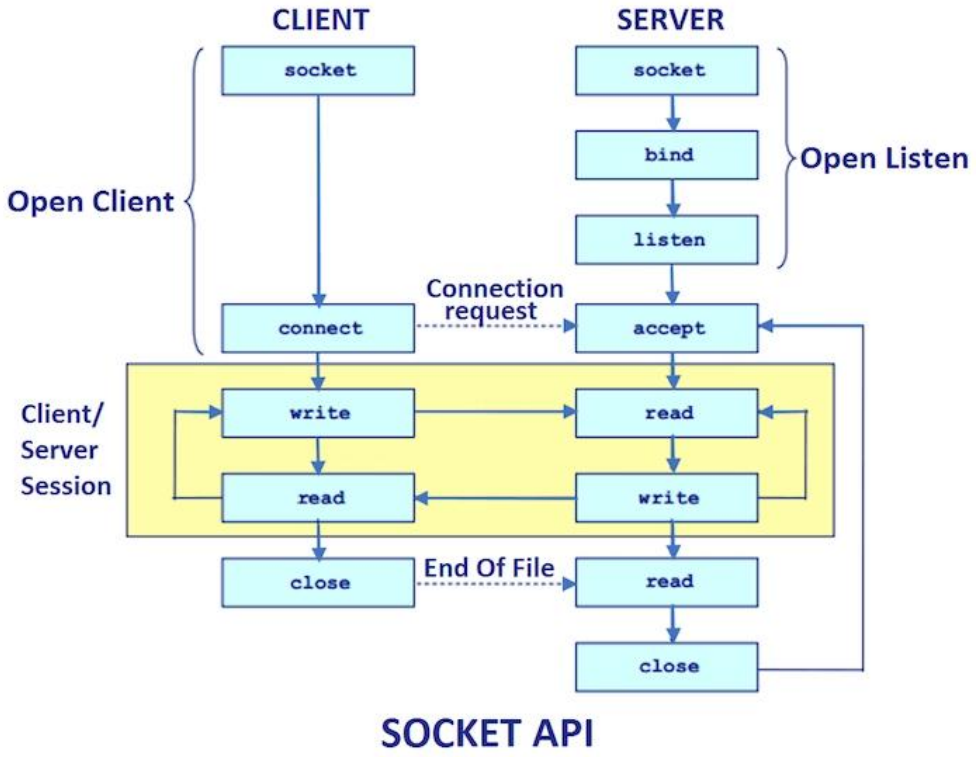
A socket is an endpoint of a two-way communication link between two programs running on the network. Socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent. Java provides a set of classes, defined in a package called java.net, to enable the rapid development of network applications.

The two key classes from the java.net package used in creation of server and client programs are:

- ServerSocket
- Socket

A server program creates a specific type of socket that is used to listen for client requests (server socket), In the case of a connection request, the program creates a new socket through which it will exchange data with the client using input and output streams. The socket abstraction is very similar to the file concept:

1. developers have to open a socket
2. perform I/O and
3. close it.



The steps for creating a simple server program are:

1. Open the Server Socket:

   ServerSocket server = new ServerSocket(PORT);

2. Wait for the Client Request:

   Socket client = server.accept();

3. Create I/O streams for communicating to the client

   DataInputStream is = new DataInputStream(client.getInputStream());
   DataOutputStream os = new DataOutputStream(client.getOutputStream());

4. Perform communication with client

   Receive from client: **String line = is.readLine();**

   Send to client: **os.writeBytes("Hello\n");**

5. Close socket:

   client.close();

The steps for creating a simple client program are:

1. Create a Socket Object:

   Socket client = new Socket(server, port_id);

2. Create I/O streams for communicating with the server.

   is = new DataInputStream(client.getInputStream());

   os = new DataOutputStream(client.getOutputStream());

3. Perform I/O or communication with the server:

   Receive data from the server: **String line = is.readLine();**

   Send data to the server: **os.writeBytes("Hello\n");**

4. Close the socket when done:

   client.close();

Practical Use Cases of Socket Programming :

- **Real-Time Messaging Apps:**

  Applications like WhatsApp and Slack use socket programming to establish real-time communication channels between users. When a user sends a message, the socket ensures instant delivery to the recipient without delays, enabling seamless chat experiences.

- **Multiplayer Games:**

  Online multiplayer games rely on sockets to keep all players connected in real-time. For example, when a player moves or performs an action, the update is immediately sent to other players using socket-based communication, ensuring synchronized gameplay.

- **File Transfer Protocols:**

  Applications like FTP clients and peer-to-peer file-sharing platforms use sockets to enable direct file transfer between machines over a network. This allows users to upload or download files efficiently across the internet.

- **Remote Server Communication:**

  Socket programming is fundamental in developing client-server models, such as web servers and database servers. For instance, when a browser requests a webpage, a socket is used to establish a connection and transfer the requested data from the server to the client.

Server.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.ServerSocket;
import java.net.Socket;

class Server {

    ServerSocket server;
    Socket socket;
    BufferedReader br;
    PrintWriter out;

    public Server() {
        try {
            server = new ServerSocket(7777);
            System.out.println("Server started. Listening for a client...");
            System.out.println("Waiting......");
            socket = server.accept();

            br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream());

            startReading();
            startWriting();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void startReading() {
        Runnable r1 = () -> {
            System.out.println("Reader Started....");

            while (true) {
                try {
                    String msg = br.readLine();
                    if (msg.equals("exits")) {
                        System.out.println("Clinet Terminated the chat");
                        break;
                    }
                    System.out.println("Client: " + msg);
                } catch (Exception e) {
                    e.printStackTrace();
```

```
                }
            }
        };

        new Thread(r1).start();

    }

    public void startWriting() {
        Runnable r2 = () -> {
            System.out.println("Writer Started....");
            while (true) {
                try {
                    BufferedReader br1 = new BufferedReader(new
InputStreamReader(System.in));
                    String content = br1.readLine();
                    out.println(content);
                    out.flush();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
        new Thread(r2).start();
    }

    public static void main(String[] args) {

        System.out.println("This Is Server ..... going to start the server");
        new Server();

    }
}
```

Client.java

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.Socket;

class Client {

    Socket socket;
    BufferedReader br;
    PrintWriter out;

    public Client() {
        try {
            System.out.println("Sending Request to server");
            socket = new Socket("127.0.0.1", 7777);

            System.out.println("Connection done");

            br = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream());

            startReading();
            startWriting();

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    public void startReading() {
        Runnable r1 = () -> {
            System.out.println("Reader Started....");

            while (true) {
                try {
                    String msg = br.readLine();
                    if (msg.equals("exits")) {
                        System.out.println("Clinet Terminated the chat");
                        break;
                    }
                    System.out.println("Client: " + msg);
                } catch (Exception e) {
                    e.printStackTrace();
                }
```

```
            }
        };

        new Thread(r1).start();

    }

    public void startWriting() {
        Runnable r2 = () -> {
            System.out.println("Writer Started....");
            while (true) {
                try {
                    BufferedReader br1 = new BufferedReader(new
InputStreamReader(System.in));
                    String content = br1.readLine();
                    out.println(content);
                    out.flush();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        };
        new Thread(r2).start();
    }

    public static void main(String[] args) {

        System.out.println("This Is Client ..... going to connect to the
server");
        new Client();

    }
}
```

Server Program – Important Methods

Method / Constructor: Server() (Constructor)
Description: Called when creating a new Server object. Creates ServerSocket on port 7777, waits for client connection, sets up input/output streams, starts reading and writing threads.

Method: server.accept()
Description: Blocking method that waits until a client connects and returns a Socket object.

Method: new BufferedReader(new InputStreamReader(socket.getInputStream()))
Description: Reads text data from the client. Converts byte stream to character stream, then buffers for efficient reading.

Method: new PrintWriter(socket.getOutputStream())
Description: Sends text data to the client.

Method: startReading()
Description: Starts a thread that continuously reads messages from the client using br.readLine(). Stops if "exits" is received.

Method: br.readLine()
Description: Reads one line of text from the client. Blocks until a message is received.

Method: startWriting()
Description: Starts a thread that reads input from the server console (System.in) and sends it to the client.

Method: out.println(content)
Description: Sends the message to the client.

Method: out.flush()
Description: Forces sending of any buffered data immediately.

Method: new Thread(r1).start() / new Thread(r2).start()
Description: Starts two threads for reading and writing simultaneously.

Method: main(String[] args)
Description: Entry point of the program. Creates a Server object to start the server.

Client Program – Important Methods

Method / Constructor: Client() (Constructor)
Description: Called when creating a new Client object. Connects to the server, sets up input/output streams, starts reading and writing threads.

Method: new Socket("127.0.0.1", 7777)
Description: Creates a socket connection to the server running on localhost at port 7777.

Method: new BufferedReader(new InputStreamReader(socket.getInputStream()))
Description: Reads text data sent by the server.

Method: new PrintWriter(socket.getOutputStream())
Description: Sends text data to the server.

Method: startReading()
Description: Starts a thread that continuously reads messages from the server. Stops if "exits" is received.

Method: br.readLine()
Description: Reads one line of text from the server.

Method: startWriting()
Description: Starts a thread that reads input from the client console and sends it to the server.

Method: out.println(content)
Description: Sends the message to the server.

Method: out.flush()
Description: Ensures the message is sent immediately.

Method: new Thread(r1).start() / new Thread(r2).start()
Description: Starts two threads for reading and writing simultaneously.

Method: main(String[] args)
Description: Entry point of the program. Creates a Client object to start the connection.