Socket programming

Socket Programming with Java.pptx

```
import
import java.net.*;
import java.io.*;
import java.util.*;//when taking input from user
```

UDP

Server side (to receive)

step 1: Create a DatagramSocket with port number.

step 2:Create a Byte [ ] array to recieve data

step 3:Create an empty Datapacket pass (buf,1024); and recieve ds.recieve(dp)

step4:Type cast to String pass (dp.getData(),0,dp.getLength)

to get address of client dp.getAddress();

to get port of client dp.Port();

Client side (to send)

step 1: Create an empty DatagramSocket object .

step 2:Generate InetAddress by InetAdress.getByName("localhost);

step 3:Take the input

step 3:Create an Datapacket passing bytes ,lenth ,ip and port number;

step4:send data ds.send(dp)

to recieve

```
            DatagramSocket ds = new DatagramSocket(5555);if server
            byte[] buf = new byte[1024];
             DatagramPacket dp = new DatagramPacket(buf, 1024);
             ds.receive(dp);
             String result = new String(dp.getData(), 0, dp.getLength());
             System.out.println("Result from Server: " + result);
```

to send

```
          DatagramSocket ds = new DatagramSocket( );
          InetAddress ip = InetAddress.getByName("localhost");//
          Scanner scan =new Scanner(System.in);
          String data="hello"
          DatagramPacket dp =new DatagramPacket(data.getBytes(),data.length
          ds.send(dp);
        server side:
          DatagramPacket  dp1 = new DatagramPacket(response.getBytes(),resp
```

TCP

Client side

step 1; create Socket object with (”localhost”,port number) ;port number same as server

step 2:sending receiving BufferedReader,Printwriter;

step3:close Socket.

Server

step 1; create ServerSocket object with port number ;

step 2: Socket s=ss.accept();

step 3:sending receiving;BufferedReader,Printwriter

step4:close ServerSocket.

To send data

`PrintWriter writer = new PrintWriter(s.getOutputStream(), true)`

`writer.println(sub);`

DataOutputStream out = new DataOutputStream(socket.getOutputStream());
out.writeUTF("Hello Server!");

To receive data

`BufferedReader reader = new BufferedReader(new InputStreamReader(socket.getInputStream()));`

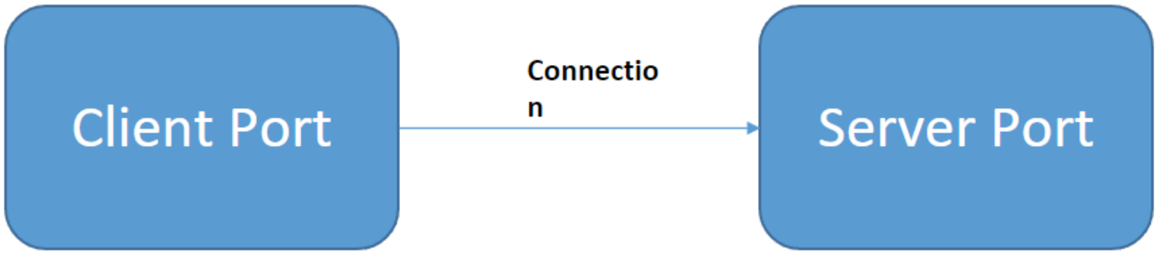
`reader.readLine();`

```
DataInputStream in = new DataInputStream(socket.getInputStream());
String message = in.readUTF();
```

<u>JAVA</u>

Socket

- A Socket is an endpoint of a two way communication link between two programs running on the network.
- The Socket is bound to a port number so that TCP Layer can identify the application that data is destined to be sent.



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.

- Socket and ServerSocket classes are used for connection-oriented socket programming.
- DatagramSocket and DatagramPacket classes are used for connection-less socket programming.
- The client in socket programming must know two information:
- IP address of Server
- Port Number.

Creating Server:

ServerSocket ss=new ServerSocket(6666);

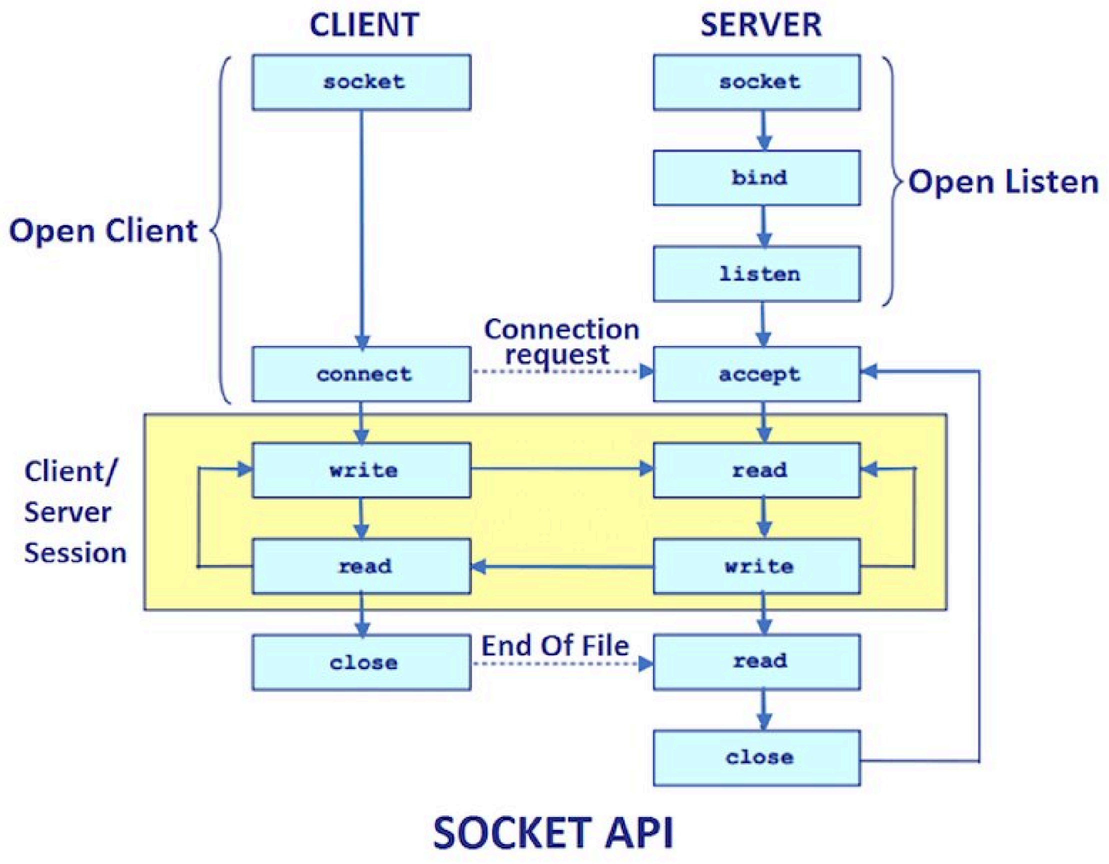
Socket s=ss.accept();//establishes connection and waits for the client

Creating Client:

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

ServerSocket class

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



The image shows the Socket API process flow between a client and server. Here's how they communicate:

Server Side (Open Listen):

1. Creates a socket
2. Binds to a specific port
3. Listens for incoming connections

Client Side (Open Client):

1. Creates a socket
2. Initiates a connection request to the server

Communication Session:

- Once connected, both client and server enter a read/write cycle where they can:
- Send data using write operations
- Receive data using read operations
- Communicate bi-directionally

Connection Termination:

1. Client sends End of File and closes connection
2. Server reads final data and closes its end
3. Server returns to listening state for new connections

For this communication to work, the client needs two crucial pieces of information:

1. The IP address of the server
2. The port number: unique number assigned to different application.0-65,535 0 TCP-reversed not used ,UDP-not available

IANA (Internet Assigned Numbers Authority) is a standard body that assigns the port numbers.

TCP Socket (Client and Server Side)

```
Creating Server:



ServerSocket ss=new ServerSocket(6666);

Socket s=ss.accept();//establishes connection and waits for the client



Creating Client:

```

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

 BufferedReader reader = new BufferedReader(new InputStreamReader(socket.g
  String msg = reader.readLine();
 PrintWriter writer = new PrintWriter(socket.getOutputStream(), true);
  writer.println(message);
```

Here is a tabular representation of the differences between TCP and UDP with at least 8 points:

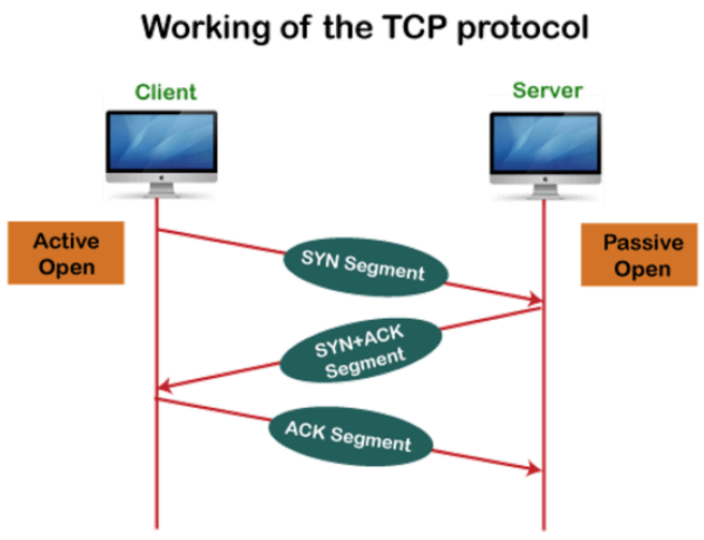
Feature	**TCP (Transmission Control Protocol)**	**UDP (User Datagram Protocol)**
Type	Connection-oriented protocol.	Connectionless protocol.
Speed	Slower due to reliable transmission and acknowledgment.	Faster as it does not ensure reliability or acknowledgment.
Handshake Process	Uses handshake protocols like SYN, SYN-ACK, ACK to establish a connection.	Does not use any handshake protocols.
Error Handling	Performs error checking and makes error recovery.	Performs error checking but discards corrupted packets.
Acknowledgment	Has acknowledgment segments for data delivery confirmation.	Does not have acknowledgment segments.
Data Transmission	Ensures reliable delivery of data in order.	Does not ensure reliable or ordered delivery of data.
Overhead	Heavy-weight protocol due to additional features like reliability checks.	Lightweight protocol as it has minimal overhead.
Use Cases	Suitable for applications requiring reliability (e.g., file transfer, email).	Suitable for applications requiring speed (e.g., video streaming, gaming).

This comparison highlights the key distinctions between TCP and UDP in a concise format.

Transmission Control Protocol

- TCP stands for **Transmission Control Protocol**.
- It is a transport layer protocol that facilitates the transmission of packets from source to destination.
- It is a connection-oriented protocol that means it establishes the connection prior to the communication that occurs between the computing devices in a network.
- This protocol is used with an IP protocol, so together, they are referred to as a TCP/IP.
- TCP take the data from the application layer, divides the data into a several packets, provides numbering to these packets, and finally transmits these packets to the destination.
- The TCP, on the other side, will reassemble the packets and transmits them to the application layer.
- Since, TCP is a connection-oriented protocol, so the connection will remain established until the communication is not completed between the sender and the receiver.

> Transport layer protocol, connection -oriented used with IP
> takes data from application
> breaks down the data to packets and numbers and sends the other side reassemble connection
> is established through out the process



Based on the image, the TCP protocol follows a three-way handshake process between client and server:

1. First, the client (in Active Open state) sends a SYN (synchronize) segment to the server
2. The server (in Passive Open state) responds with a SYN+ACK segment, acknowledging the client's request
3. Finally, the client sends back an ACK (acknowledgment) segment, completing the connection establishment

This three-way handshake ensures a reliable connection is established before any data transfer begins. The process demonstrates TCP's connection-oriented nature, where both parties confirm they're ready to communicate before actual data transmission starts.

UDP

UDP (User Datagram Protocol) is a lightweight, connectionless communication protocol used for sending and receiving data over a network. Unlike TCP, it does not establish a connection or guarantee reliable delivery, making it faster but less dependable. It sends data in packets called datagrams, which may arrive out of order, be duplicated, or get lost without retransmission. UDP is commonly used for applications where speed is crucial, such as video streaming, gaming, or voice-over-IP, where occasional data loss is acceptable. Its simplicity makes it efficient but best suited for scenarios where reliability can be handled at the application level.

What is the task of Socket class?How socket class can be used in socket program? What are the use of TCP and UDP socket?

Task of the `Socket` Class:

The `Socket` class in Java is responsible for establishing a connection between a client and a server over a network. It allows data to be sent and received through a network communication channel.

Using `Socket` Class in a Program:

In a socket program, the `Socket` class is used to create a connection to a remote server. For example:

```
Socket socket = new Socket("localhost", 8080);
```

This line creates a socket and connects it to a server running on `localhost` at port `8080`.

Use of TCP Sockets:

- **TCP (Transmission Control Protocol)** sockets provide reliable, connection-oriented communication. They guarantee that data will be delivered in the correct order without errors.
- They are used in applications requiring accurate and secure data transfer, like HTTP, file transfer, or email protocols.

Use of UDP Sockets:

- **UDP (User Datagram Protocol)** sockets provide connectionless, faster communication with no guarantee of delivery, order, or error checking.
- They are used in applications where speed is critical and occasional data loss is acceptable, like video streaming or online gaming.

```java
import java.io.*;
import java.net.*;

public class ChatServer {
    private static final int PORT = 1234;

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server started, waiting for clients to connect...");

            Socket socket = serverSocket.accept();
            System.out.println("Client connected");

            BufferedReader input = new BufferedReader(new InputStreamReader(soc
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

            BufferedReader consoleInput = new BufferedReader(new InputStreamRea
            String received, sent;

            while (true) {
                // Read message from client
                received = input.readLine();
                if (received.equalsIgnoreCase("exit")) {
                    System.out.println("Client disconnected");
                    break;
                }
                System.out.println("Client: " + received);

                // Server response
                System.out.print("Server: ");
                sent = consoleInput.readLine();
                output.println(sent);

                if (sent.equalsIgnoreCase("exit")) {
                    System.out.println("Server shutting down");
                    break;
                }
            }

            socket.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
import java.net.*;

public class ChatClient {
    private static final String SERVER_ADDRESS = "localhost";
    private static final int SERVER_PORT = 1234;

    public static void main(String[] args) {
        try (Socket socket = new Socket(SERVER_ADDRESS, SERVER_PORT)) {
            System.out.println("Connected to server");

            BufferedReader input = new BufferedReader(new InputStreamReader(soc
            PrintWriter output = new PrintWriter(socket.getOutputStream(), true);

            BufferedReader consoleInput = new BufferedReader(new InputStreamRea
            String received, sent;

            while (true) {
                // Send message to server
                System.out.print("Client: ");
                sent = consoleInput.readLine();
                output.println(sent);

                if (sent.equalsIgnoreCase("exit")) {
```

```
                System.out.println("Client disconnecting");
                break;
            }

            // Read response from server
            received = input.readLine();
            System.out.println("Server: " + received);

            if (received.equalsIgnoreCase("exit")) {
                System.out.println("Server has closed the connection");
                break;
            }
        }

        socket.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
  }
}
```

How it works:

1. **ChatServer**: The server waits for a client connection on a specific port (`1234`). Once a client connects, it starts a loop where it receives messages from the client and sends responses back.
2. **ChatClient**: The client connects to the server using the server's IP address (`localhost` in this case) and the same port number (`1234`). The client sends messages to the server and waits for a response.

To run the program:

1. Start the `ChatServer` first.
2. Run the `ChatClient` to connect to the server.

Write a TCP socket program to send "Hello server" text to the remote machine listening on 9999 port and having 192.168.10.20 IP address

```java
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) {
        String serverAddress = "192.168.10.20";  // Server IP
        int port = 9999;  // Server port

        try (Socket socket = new Socket(serverAddress, port);
             PrintWriter out = new PrintWriter(socket.getOutputStream(), true)) {

            // Send the "Hello server" message to the server
            String message = "Hello server";
            out.println(message);
            System.out.println("Message sent to the server: " + message);

        } catch (IOException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

Write program using socket program to send the string "hello" to the client.

```java
import java.io.*;
import java.net.*;

public class TCPServer {
    private static final int PORT = 9999;  // Server will listen on this port

    public static void main(String[] args) {
        try (ServerSocket serverSocket = new ServerSocket(PORT)) {
            System.out.println("Server is running and waiting for client to connect...")
            // Accept a client connection
            Socket clientSocket = serverSocket.accept();
            System.out.println("Client connected");

            // Create output stream to send data to the client
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);

            // Send the message "Hello" to the client
            String message = "Hello";
            out.println(message);
            System.out.println("Message sent to the client: " + message);

            // Close the connection with the client
            clientSocket.close();
            System.out.println("Client connection closed.");

        } catch (IOException e) {
            System.out.println("Server exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

```
import java.io.*;
import java.net.*;

public class TCPClient {
    public static void main(String[] args) {
        String serverAddress = "localhost";  // Server IP (use IP if server is on anoth
        int port = 9999;  // Server port

        try (Socket socket = new Socket(serverAddress, port);
             BufferedReader in = new BufferedReader(new InputStreamReader(socke
            // Read the message from the server
            String message = in.readLine();
            System.out.println("Message from server: " + message);

        } catch (IOException e) {
            System.out.println("Client exception: " + e.getMessage());
            e.printStackTrace();
        }
    }
}
```

- **DatagramSocket**: A Java class used for sending and receiving datagram packets over a network using connectionless UDP communication.
- **DatagramPacket**: A Java class that represents data in the form of a packet to be sent or received via a DatagramSocket over a network.