



**UNIFESSPA**

UNIVERSIDADE FEDERAL DO SUL E SUDESTE DO PARÁ

# Universidade Federal do Sul e Sudeste do Pará

---

## Sistemas Distribuídos

*Prof.: Warley Junior*

[wmvj@unifesspa.edu.br](mailto:wmvj@unifesspa.edu.br)

# Agenda

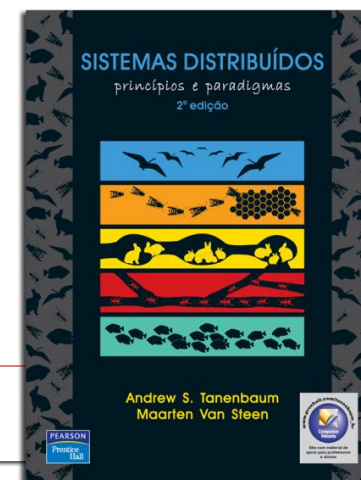
---

- ❑ AULA 4:
- ❑ Comunicação entre Processos
  - ❑ Comunicação UDP
  - ❑ Comunicação TCP
  - ❑ Comunicação Multicast

# Leitura Prévia

---

- ❑ COULOURIS, George. Sistemas distribuídos: conceitos e projetos. 5ª ed. Porto Alegre: Bookman, 2013.
  - Capítulo 4.
- ❑ TANENBAUM, Andrew S. Sistemas distribuídos: princípios e paradigmas. 2ª ed. São Paulo: Pearson Prentice Hall, 2007.
  - Capítulo 4.



# Camadas de *middleware*

---



# Características da Comunicação entre Processos

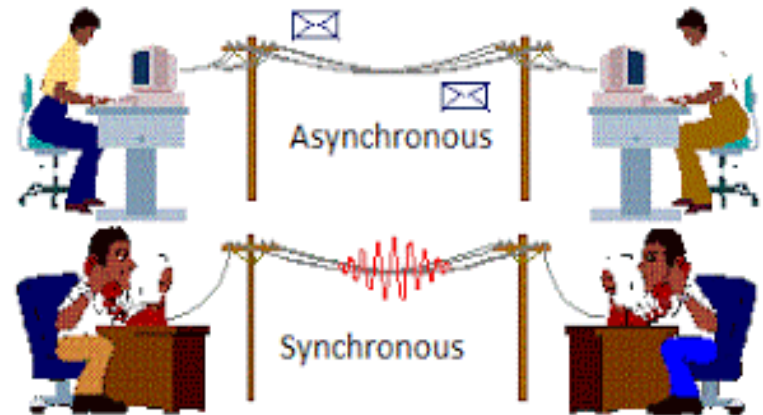
---

- ❑ Comunicação síncrona e assíncrona
- ❑ Destinos de mensagem
- ❑ Confiabilidade
- ❑ Ordenamento

# Tipos de comunicação

---

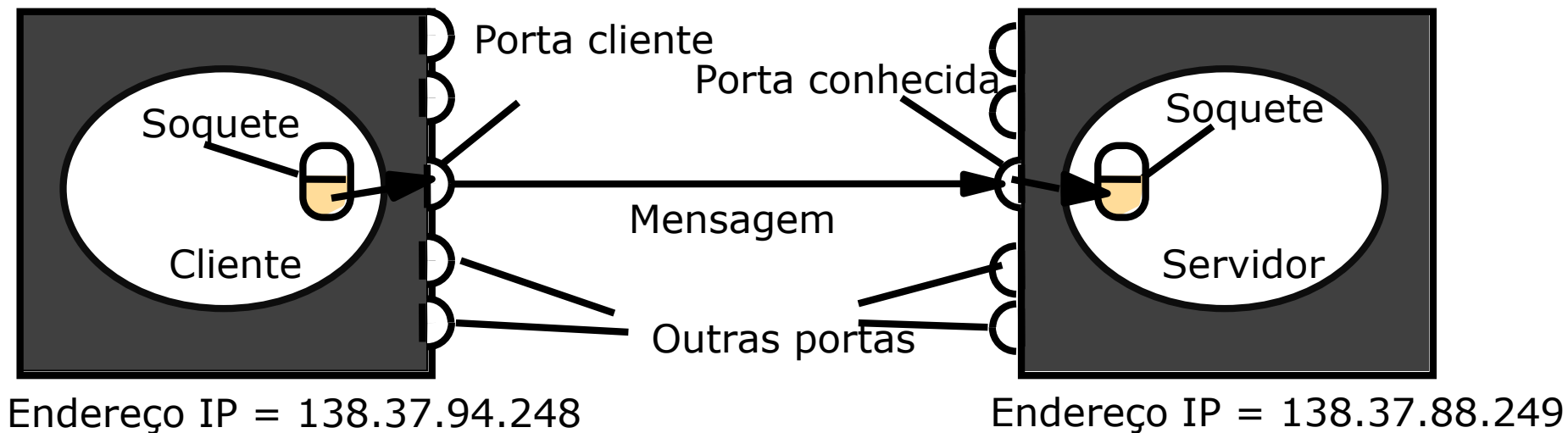
- **Assíncrona** (não bloqueante): processo continua a executar após submeter mensagem para transmissão.
- **Síncrona** (bloqueante): processo bloqueia até que mensagem seja recebida.



# Sockets

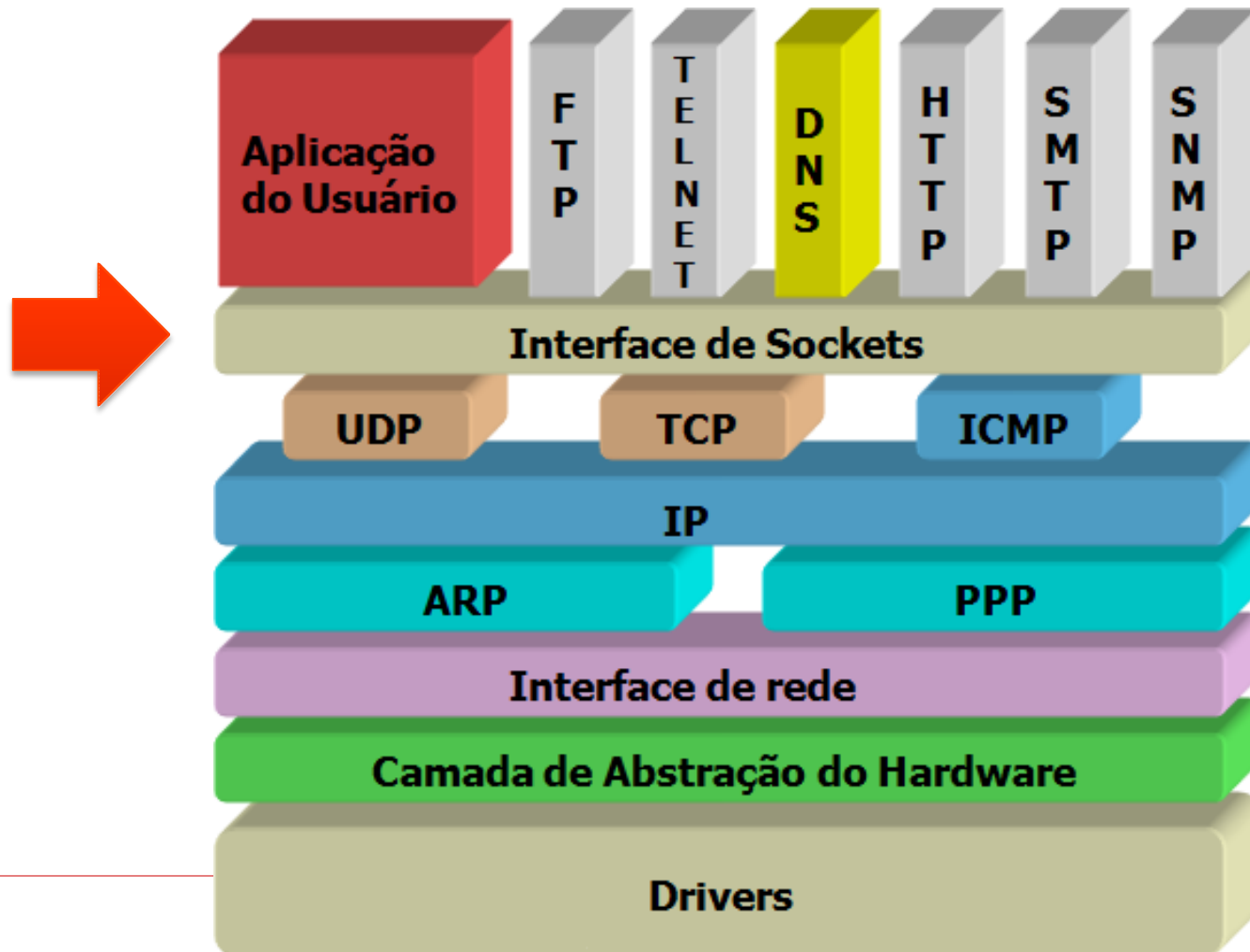
---

- ❑ Abstração que representa uma porta de comunicação bidirecional associada a um processo.



# Sockets

---





# Sockets

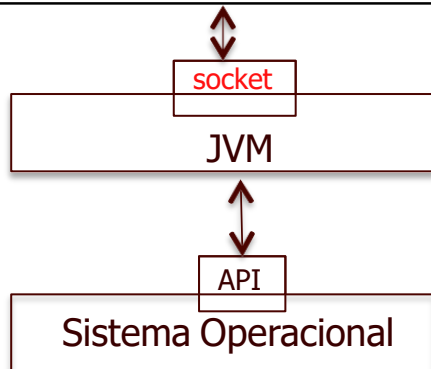
---



# Sockets

## Host 1

```
public class CalculatorClient {  
}
```



## Method Summary

All Methods	Static Methods	Instance Methods	Concrete Methods
Modifier and Type		Method and Description	
void		<code>bind(SocketAddress bindpoint)</code>	Binds the socket to a local address.
void		<code>close()</code>	Closes this socket.
void		<code>connect(SocketAddress endpoint)</code>	Connects this socket to the server.
void		<code>connect(SocketAddress endpoint, int timeout)</code>	Connects this socket to the server with a specified timeout value.
<code>SocketChannel</code>		<code>getChannel()</code>	Returns the unique <code>SocketChannel</code> object associated with this socket, if any.
<code>InetAddress</code>		<code>getInetAddress()</code>	Returns the address to which the socket is connected.
<code>InputStream</code>		<code>getInputStream()</code>	Returns an input stream for this socket.
boolean		<code>getKeepAlive()</code>	Tests if <code>SO_KEEPALIVE</code> is enabled.
<code>InetAddress</code>		<code>getLocalAddress()</code>	Gets the local address to which the socket is bound.

- ❑ A classe Socket tem 42 métodos.
- ❑ Variam de SO para SO.

# API Java para endereços Internet

---

- Classe InetAddress

- Obtém o endereço IP invocando o método `getByName()` e fornecendo o hostname.

- Exceção: `UnknownHostException`.

```
InetAddress host =  
InetAddress.getByName("localhost");
```

# Comunicação UDP

---

- ❑ Protocolo **não orientado a conexão**.
- ❑ Transporte de dados **não confiável**.
- ❑ Emprega *send* **não-bloqueante** e *receive* **bloqueante**.
- ❑ Mensagem de entrada é inserida em uma fila.
- ❑ Mensagens são retiradas da fila por invocações de *receive* no socket.

# Comunicação por datagrama UDP

---

## ☐ Problemas

- ☐ Tamanho da mensagem, Bloqueio, Timeouts, Recepção anônima.

## ☐ Modelo de falhas

- ☐ Falhas por omissão
- ☐ Ordenamento

## ☐ Emprego de UDP

- ☐ *Streaming de Vídeo, VoIP, DNS...*

## ☐ API Java para datagramas UDP

# Java API para Comunicação UDP

---

- ❑ Duas classes: DatagramPacket, DatagramSocket
- ❑ Classe DatagramPacket
  - ❑ Fornece dois construtores:
    - Requisitando (request)
    - Recebendo (reply)

# Java API para Comunicação UDP

---

## □ Classe DatagramPacket

- Mensagem pode ser recuperada do DatagramPacket através do método *getData*.
- Porta - método **getPort**.
- Endereço IP - método **getAddress**.

vetor de bytes contendo a mensagem	comprimento da mensagem	endereço IP	número da porta
--	----------------------------	----------------	--------------------

# Java API para Comunicação UDP

---

## □ Classe DatagramSocket

- Suporta sockets para enviar e receber datagramas.
- Construtores:
  - Argumento: número da porta a ser utilizada pelo processo.
  - Um construtor sem argumentos permite ao sistema escolher qualquer porta local disponível.



# Java API para Comunicação UDP

---

## □ Classe DatagramSocket

### ■ Métodos:

#### □ send:

- Argumento: instância da classe DatagramPacket contendo uma mensagem e seu destino.

#### □ receive:

- Argumento: DatagramPacket vazio no qual serão inseridos a mensagem, seu tamanho e sua origem.
  - Exceção: IOExceptions

# Java API para Comunicação UDP

---

## □ Classe DatagramSocket

### ■ Métodos (cont.):

#### □ `setSoTimeout`

- Configuração de *timeout*.
- Método *receive* ficará bloqueado durante o tempo especificado e depois lançará uma exceção `InterruptedException`.

# Java API para Comunicação UDP

**Passo 1:** Cria um socket.

**Passo 2:** Envia uma mensagem a um servidor na porta 6789.

**Passo 3:** Espera resposta.

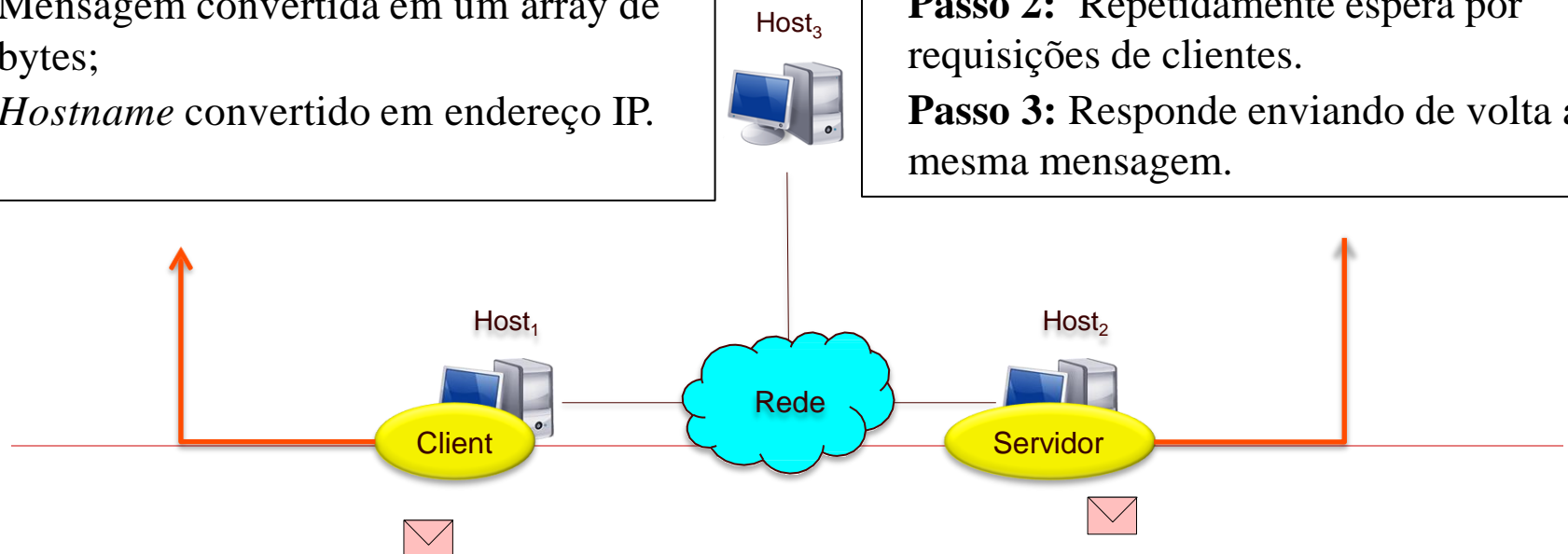
**Passo 4:** Os argumentos de entrada de *main* são o *hostname* do servidor e a mensagem.

- Mensagem convertida em um array de bytes;
- *Hostname* convertido em endereço IP.

**Passo 1:** Cria um socket associado a porta de servidor 6789

**Passo 2:** Repetidamente espera por requisições de clientes.

**Passo 3:** Responde enviando de volta a mesma mensagem.



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

public class UDPClient {

    public static void main(String args[]) {
        // args fornece o conteúdo da mensagem e o nome de host do servidor
        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket();
            byte[] m = args[0].getBytes();
            InetAddress aHost = InetAddress.getByName(args[1]);
            int serverPort = 6789;
            DatagramPacket request
                = new DatagramPacket(m, args[0].length(), aHost, serverPort);
            aSocket.send(request);
            byte[] buffer = new byte[1000];
            DatagramPacket reply = new DatagramPacket(buffer, buffer.length);
            aSocket.receive(reply);
            System.out.println("Reply: " + new String(reply.getData()));
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        } finally {
            if (aSocket != null) {
                aSocket.close();
            }
        }
    }
}
```



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

public class UDPServer {

    public static void main(String args[]) {
        DatagramSocket aSocket = null;
        try {
            aSocket = new DatagramSocket(6789);
            byte[] buffer = new byte[1000];
            while (true) {
                DatagramPacket request = new DatagramPacket(buffer, buffer.length);
                aSocket.receive(request);
                DatagramPacket reply = new DatagramPacket(request.getData(),
                    request.getLength(), request.getAddress(), request.getPort());
                aSocket.send(reply);
            }
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        } finally {
            if (aSocket != null) {
                aSocket.close();
            }
        }
    }
}
```

# Comunicação TCP

---

## □ Características:

- *Acknowledgement* e retransmissões de mensagens.
- Controle de fluxo.
- Ordenação de mensagens.
- Verifica mensagens duplicadas.
- Estabelece a conexão antes de enviar o *stream*.

# Comunicação TCP

---

## □ Processo de conexão

- Cliente: cria um socket para o *stream* e o associa a uma porta qualquer, então envia um *connect request* ao servidor indicando a porta.
- Servidor: cria um *listening* socket, o associa a uma porta local conhecida e espera pedidos de conexão de clientes.
- O *listening socket* possui uma fila para colocar pedidos de conexão.

# Java API para Comunicação TCP

---

- Duas classes: **ServerSocket, Socket**
- **Classe ServerSocket**
  - Cria um socket na porta do servidor para escutar pedidos de conexão de clientes.
  - Método *accept*
    - Retira um pedido de conexão da fila ou bloqueia até que uma requisição chegue.




# Java API para Comunicação TCP

---

## □ Classe Socket

- Cliente: usa um construtor para criar um socket associado a uma porta local e conecta o processo à porta do computador remoto.
- Argumentos: *hostname* e porta do processo servidor.
- Métodos para acessar os dois fluxos associados ao socket:
  - `getInputStream()`.
  - `getOutputStream()`.



```

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

public class TCPClient {

    public static void main(String args[]) {
        // os argumentos fornecem a mensagem e o nome de host do destino
        Socket s = null;
        try {
            int serverPort = 7896;
            s = new Socket(args[1], serverPort);
            DataInputStream in = new DataInputStream(s.getInputStream());
            DataOutputStream out
                = new DataOutputStream(s.getOutputStream());
            out.writeUTF(args[0]); // UTF é uma codificação de string
            String data = in.readUTF();
            System.out.println("Received: " + data);
        } catch (UnknownHostException e) {
            System.out.println("Sock:" + e.getMessage());
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } finally {
            if (s != null) {
                try {
                    s.close();
                } catch (IOException e) { /*close falhou*/
                }
            }
        }
    }
}

```



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

public class TCPServer {

    public static void main(String args[]) {
        try {
            int serverPort = 7896;
            ServerSocket listenSocket = new ServerSocket(serverPort);
            while (true) {
                Socket clientSocket = listenSocket.accept();
                Connection c = new Connection(clientSocket);
            }
        } catch (IOException e) {
            System.out.println("Listen:" + e.getMessage());
        }
    }
}
```



```

class Connection extends Thread {

    DataInputStream in;
    DataOutputStream out;
    Socket clientSocket;

    public Connection(Socket aClientSocket) {
        try {
            clientSocket = aClientSocket;
            in = new DataInputStream(clientSocket.getInputStream());
            out = new DataOutputStream(clientSocket.getOutputStream());
            this.start();
        } catch (IOException e) {
            System.out.println("Connection:" + e.getMessage());
        }
    }

    public void run() {
        try { // Servidor de eco
            String data = in.readUTF();
            out.writeUTF(data);
        } catch (EOFException e) {
            System.out.println("EOF:" + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO:" + e.getMessage());
        } finally {
            try {
                clientSocket.close();
            } catch (IOException e) { /*close falhou*/
            }
        }
    }
}

```

# Comunicação Multicast

---


- ❑ Permite ao emissor transmitir um único pacote IP para um conjunto de computadores que formam o *grupo multicast*.
- ❑ Emissor não toma conhecimento das identidades dos receptores individuais e do tamanho do grupo.
- ❑ Especificado por um IP da Classe D
  - 1º octeto = 1110xxxx em IPv4;
  - 224.0.0.0 a 239.255.255.255.

# Java API para Comunicação Multicast

---

## □ Classe *MulticastSocket*

- Subclasse da DatagramSocket com habilidade adicional de juntar-se a grupos *multicast*.
- Dois construtores
- Métodos:
  - `joinGroup;`
  - `leaveGroup;`
  - `setTimeToLive.`



```

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

public class MulticastPeer {

    public static void main(String args[]) {
        // args fornece o conteúdo da mensagem e o grupo multicast de destino
        // (por exemplo, "228.5.6.7")
        MulticastSocket s = null;
        try {
            InetAddress group = InetAddress.getByName(args[1]);
            s = new MulticastSocket(6789);
            s.joinGroup(group);
            byte[] m = args[0].getBytes();
            DatagramPacket messageOut
                = new DatagramPacket(m, m.length, group, 6789);
            s.send(messageOut);
            byte[] buffer = new byte[1000];
            for (int i = 0; i < 3; i++) { // obtém mensagens de outros participantes do grupo
                DatagramPacket messageIn
                    = new DatagramPacket(buffer, buffer.length);
                s.receive(messageIn);
                System.out.println("Received:" + new String(messageIn.getData()));
            }
            s.leaveGroup(group);
        } catch (SocketException e) {
            System.out.println("Socket: " + e.getMessage());
        } catch (IOException e) {
            System.out.println("IO: " + e.getMessage());
        } finally {
            if (s != null) {
                s.close();
            }
        }
    }
}

```

# Sockets

---

## ❑ Principais Tipos de Socket

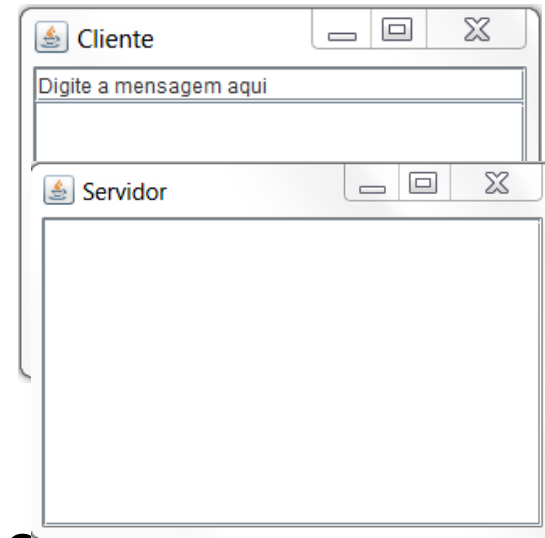
- ❑ Socket Datagrama: envia/recebe datagramas sem criar conexão; usa protocolo UDP.
- ❑ Socket Multicast: recebe as mensagens endereçadas a um grupo; usa UDP multicast.
- ❑ Socket Stream: estabelece uma conexão com outro socket; usa protocolo TCP.



# Praticando Sockets Datagrama

---

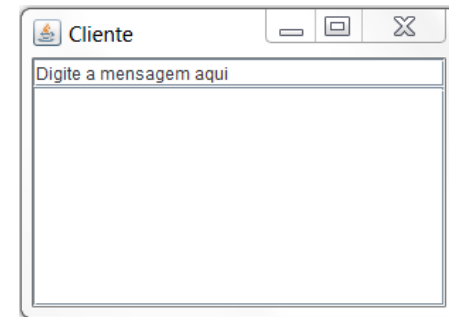
1. Abra o Netbeans;
2. Clique em Arquivo>Abrir Projeto
3. Selecione o projeto **DatagramaSocket**
4. Execute o servidor **Server.java**
5. Execute o cliente **Client.java**
6. Digite uma mensagem na parte superior da janela do cliente e tecle ENTER para enviá-la para o servidor em um datagrama UDP; a mensagem será recebida pelo servidor e enviada de volta ao cliente.



# Praticando Sockets Multicast

---

1. Abra o Netbeans;
2. Clique em Arquivo>Abrir Projeto
3. Selecione o projeto **MulticastSockets**
4. Execute 3 vezes o cliente **Client.java**
5. Digite uma mensagem na parte superior da janela de qualquer um dos clientes e tecle ENTER para enviá-la a todos os clientes usando UDP *multicast*.



# Praticando Sockets Stream

1. Clique em Arquivo>Abrir Projeto
2. Abra o Netbeans;
3. Selecione o projeto **SocketsStream**
4. Execute o servidor **Server.java**
5. Execute o cliente **Client.java**
6. A conexão TCP entre o cliente e o servidor será estabelecida automaticamente;
7. Digite mensagens na parte superior da janela do cliente e tecle ENTER para enviá-las ao servidor; faça o mesmo no servidor para enviar mensagens ao cliente.

