Sockets

Comunicação entre Processos

Rafael Vargas Mesquita

ROTEIRO

Sockets

Tipos de Sockets

Sockets TCP

Sockets Java

Exemplos de Sockets com GUI

Códigos de Sockets

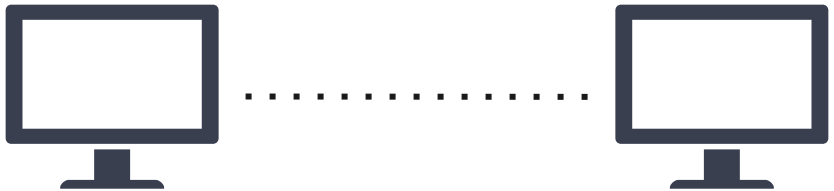
Programação Distribuída

Java Sockets: Introdução

Rafael Mesquita, Prof.

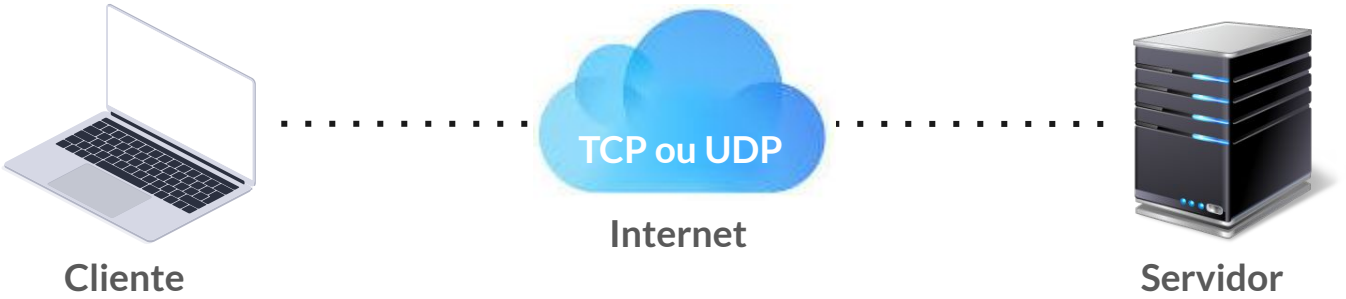
Sockets

- Os sockets oferecem **fluxo bidirecional**, capacidade de trafegar **objetos**, valores **inteiros**, **caracteres** e a opção de expandir o escopo de atuação para além do próprio sistema operacional local.
- Esta última característica traz a principal funcionalidade dos sockets: o poder de estabelecer uma comunicação entre dois pontos ligados pela rede.
- Os sockets permitem que essa comunicação aconteça localmente.
- Os sockets são utilizados para implementar protocolos de serviços de rede: HTTP e DNS



Cliente / Servidor

- Basicamente os sockets são utilizados para criar aplicações que funcionam no modelo cliente / servidor
- O servidor tem a função de ficar aguardando as requisições, e o cliente fica a cargo de solicitar ou enviar informações ao servidor

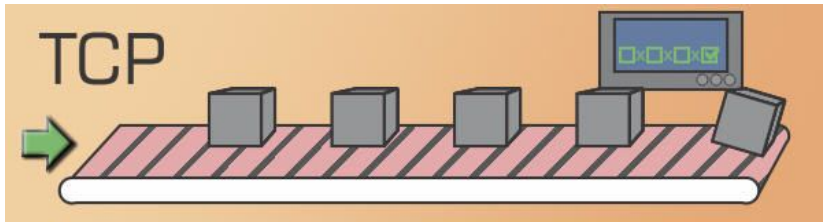


Tipos de Sockets

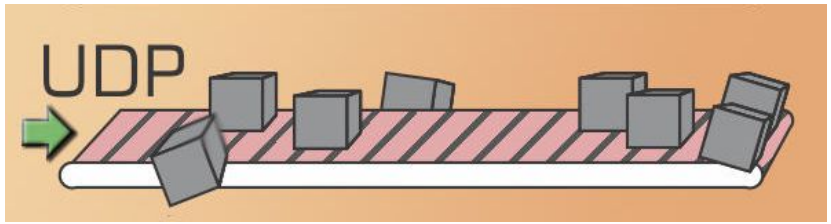
01 | Comunicações baseadas em fluxo (TCP)

02 | Comunicações baseadas em pacote (UDP)

TCP
Orientado à conexão
Confiável (garante a entrega dos dados ao destino)
Adequado para envio de e-mail e download de arquivos

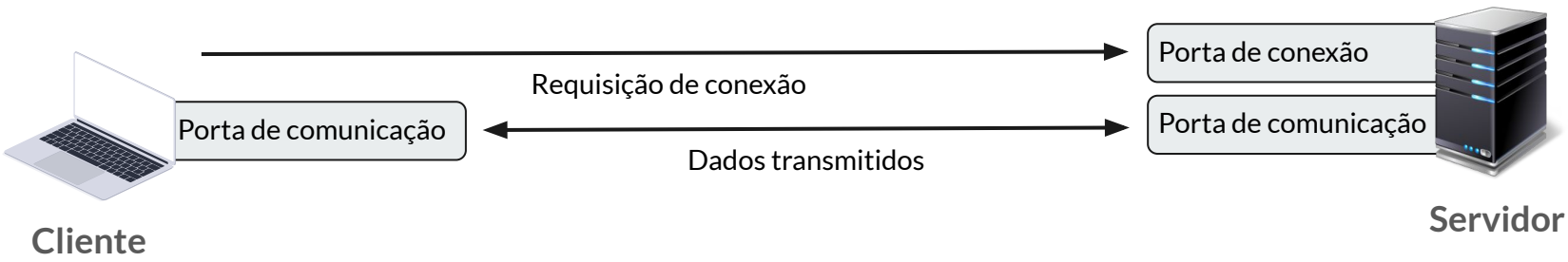


UDP
Não orientado à conexão
Não confiável (não garante a entrega dos dados ao destino)
Adequado para fluxo de dados em tempo real (vídeo/voz)

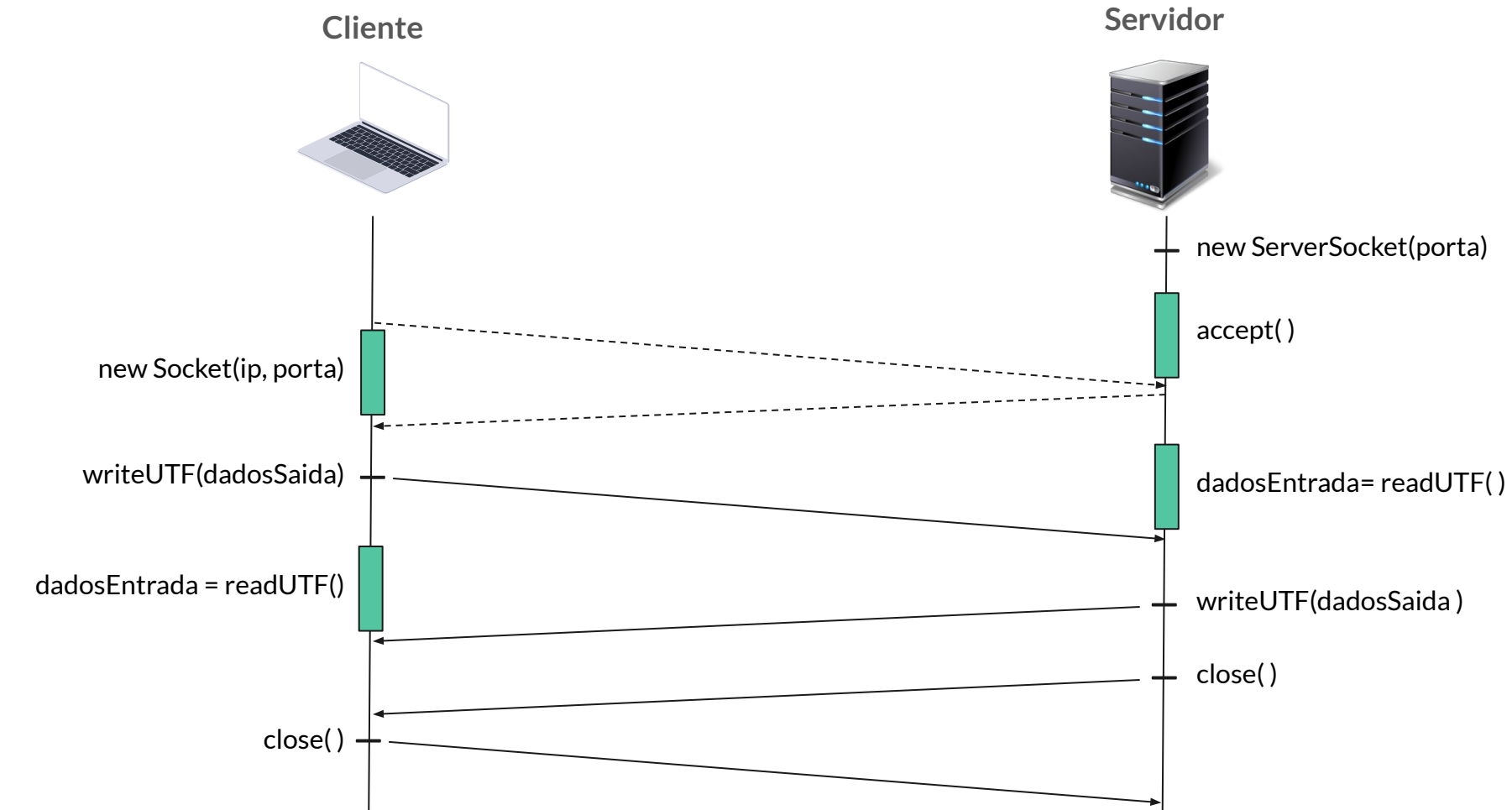


Sockets (TCP)

1. O **servidor** disponibiliza um socket e aguarda o recebimento de uma solicitação de conexão
2. O **cliente** executa um socket para se comunicar à máquina servidora
3. Caso não ocorra problemas, o servidor aceita a conexão gerando um novo socket em uma porta qualquer do seu lado, criando um **canal de comunicação** entre o cliente e servidor.



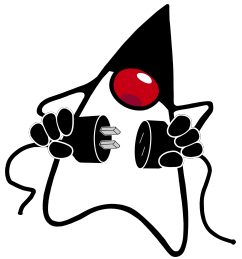
Sockets (TCP)


Cliente
Servidor
new ServerSocket(porta)
accept( )
new Socket(ip, porta)
writeUTF(dadosSaida)
dadosEntrada= readUTF( )
dadosEntrada = readUTF()
writeUTF(dadosSaida)
close( )
close( )

Sockets (Java)

Os recursos fundamentais de Sockets em Java são declarados pelas classes e interfaces do pacote **java.net**

- java.net.Socket;
- java.net.ServerSocket;
- java.io.DataInputStream;
- java.io.DataOutputStream;
- java.io.ObjectOutputStream;
- java.io.ObjectInputStream;

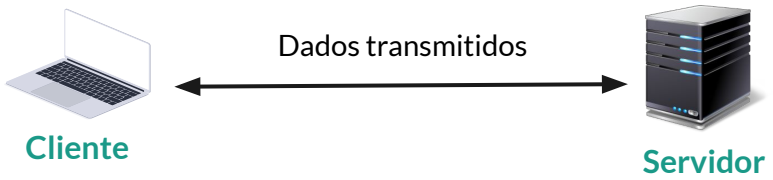


Sockets (TCP): Exemplos com GUI

1. Jogo da velha (1 cliente e 1 servidor)
2. Cálculo dos números primos (4 clientes e 1 servidor)

Sockets (TCP): Exemplo com GUI - Jogo da Velha (aplicação com 1 cliente e 1 servidor)

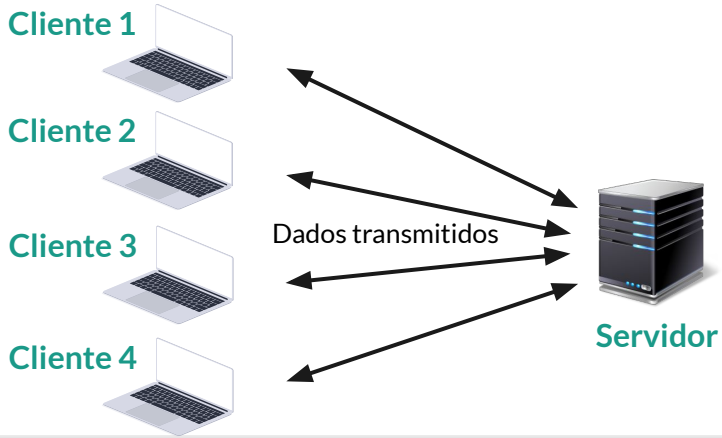
- **Cliente**: conecta-se ao servidor (Jogador 0)
- **Servidor**: inicia o servidor (Jogador X)



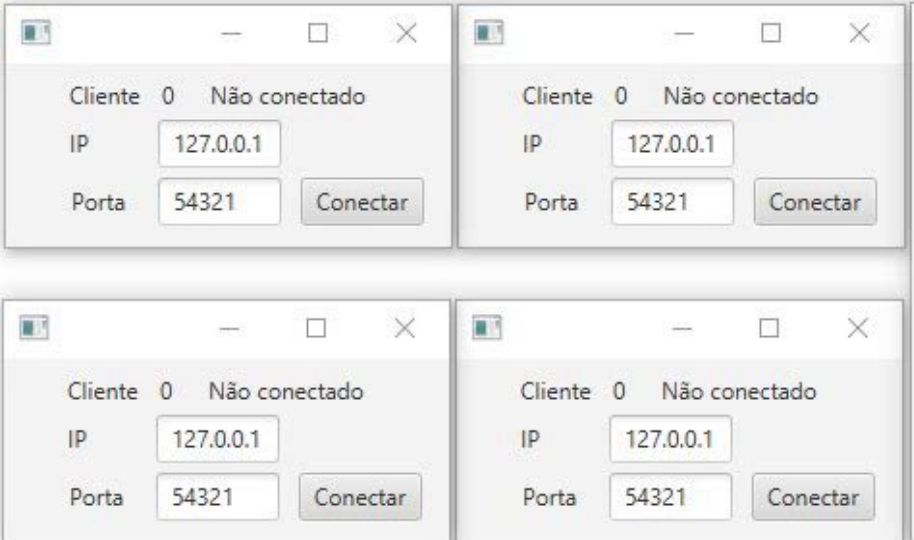
https://github.com/ravarmes/sockets-jogodavelha-javafx

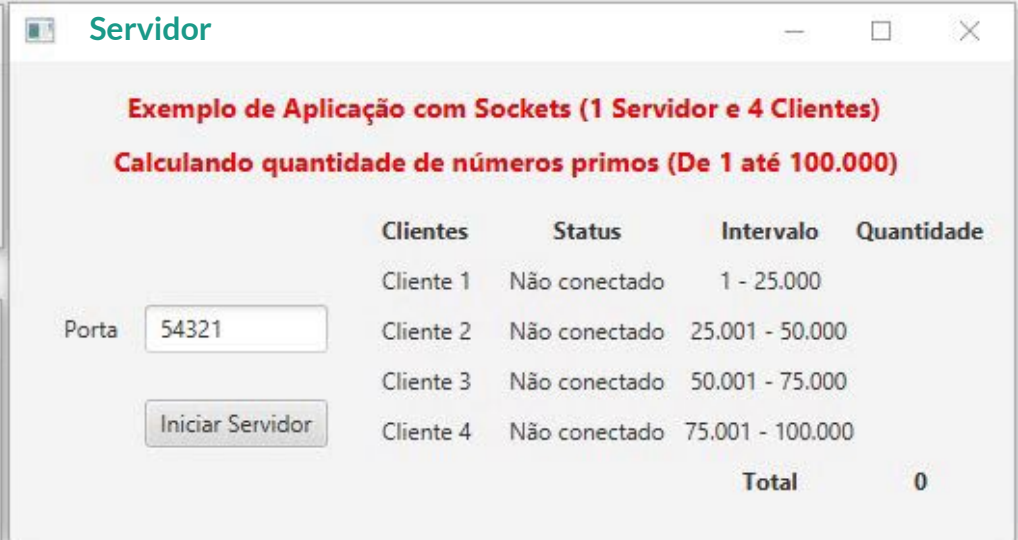
Sockets (TCP): Exemplo com GUI - Cálculo dos Primos (aplicação com 4 clientes e 1 servidor)

- **Clientes**: calculam a quantidade de números primos
- **Servidor**: distribui os intervalos para os clientes



https://github.com/ravarmes/sockets-primos-javafx





Programação Distribuída

Java Sockets: Códigos

Rafael Mesquita, Prof.

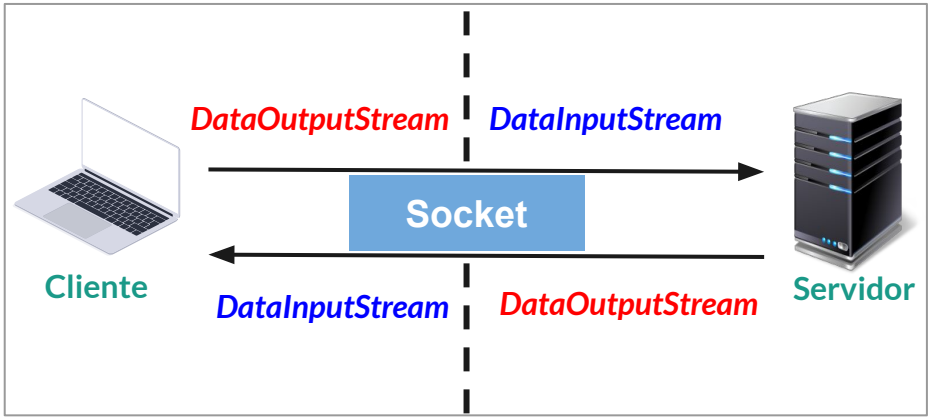
Sockets (TCP): Códigos

1. Sockets (TCP): Código - Envio de texto
2. Sockets (TCP): Código - Envio de texto com Thread
3. Sockets (TCP): Código - Envio de objeto

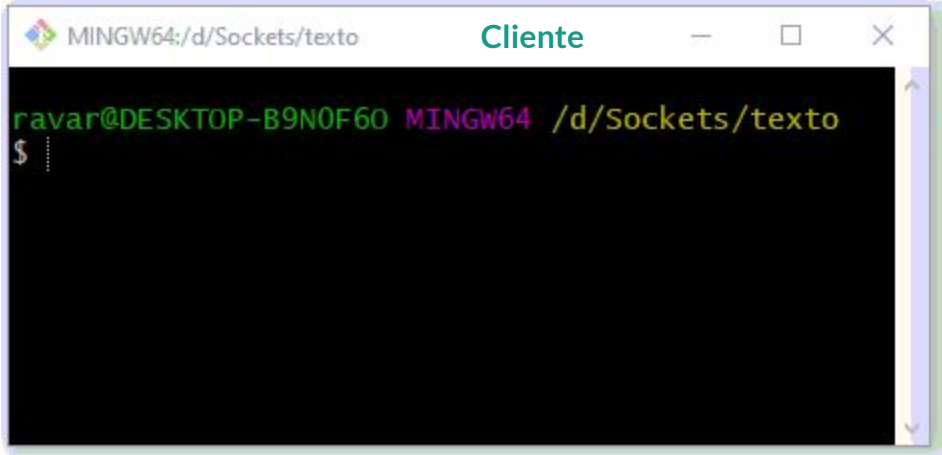
https://github.com/ravarmes/sockets-clienteservidor-java

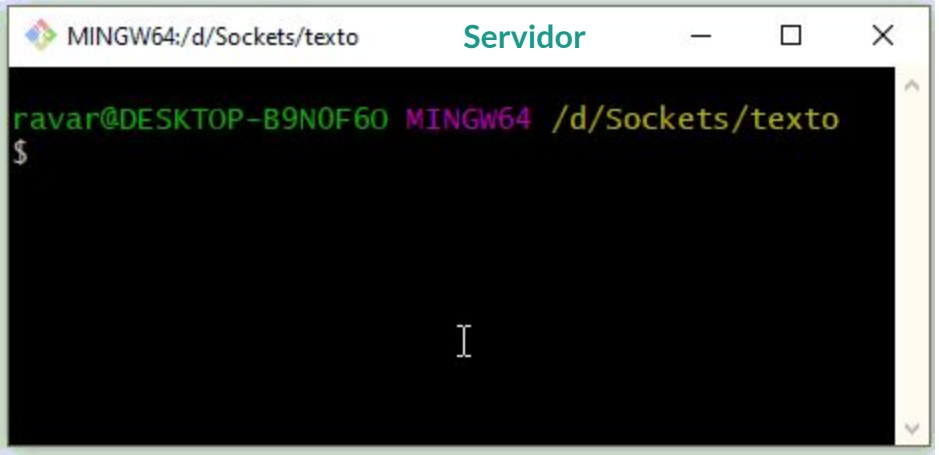
Sockets (TCP): Código - Envio de texto

- **Cliente**: envia mensagem em minúsculo
- **Servidor**: devolve a mensagem em maiúsculo



Cliente envia "rafael" e Servidor envia "RAFAEL"





Sockets (TCP): Código - Envio de texto | Cliente.java

```java
public class Cliente {

    public static void main(String[] args) throws IOException {
        //1 - Abrir conexão
        Socket socket = new Socket("127.0.0.1", 54321);

        //2 - Definir stream de saída de dados do cliente
        DataOutputStream saida = new DataOutputStream(socket.getOutputStream());
        saida.writeUTF("rafael"); //Enviar  mensagem em minúsculo para o servidor

        //3 - Definir stream de entrada de dados no cliente
        DataInputStream entrada = new DataInputStream(socket.getInputStream());
        String novaMensagem = entrada.readUTF();//Receber mensagem em maiúsculo do servidor
        System.out.println(novaMensagem); //Mostrar mensagem em maiúsculo no cliente

        //4 - Fechar streams de entrada e saída de dados
        entrada.close();
        saida.close();

        //5 - Fechar o socket
        socket.close();
    }
}
```

1 - Abrir a conexão;
2 - Definir stream de saída;
3 - Definir stream de entrada;
4 - Fechar os streams de entrada e saída;
5 - Fechar o socket.

Sockets (TCP): Código - Envio de texto | Servidor.java

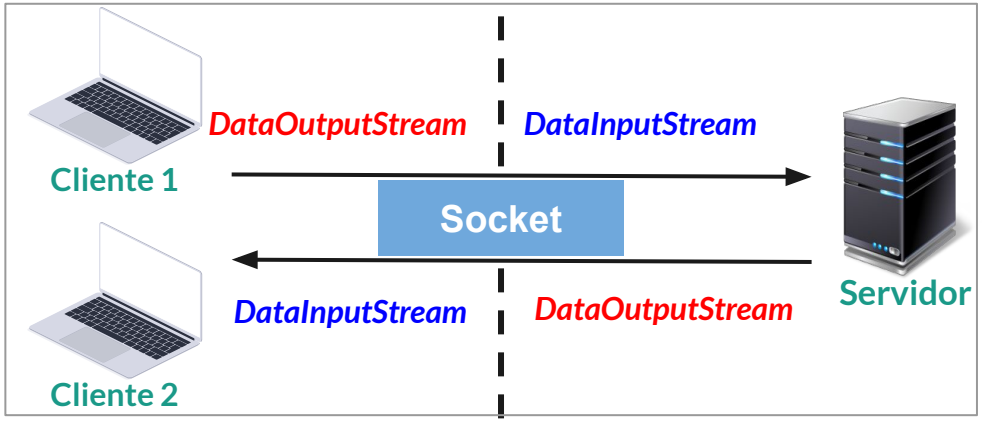
```java
public class Servidor {
    public static void main(String[] args) throws IOException {
        //1 - Definir o serverSocket (abrir porta de conexão)
        ServerSocket serverSocket = new ServerSocket(54321);
        System.out.println("A porta 54321 foi aberta!");
        System.out.println("Servidor esperando receber mensagem de cliente...");
        //2 - Aguardar solicitação de conexão de cliente
        Socket socket = serverSocket.accept();
        //Mostrar endereço IP do cliente conectado
        System.out.println("Cliente " + socket.getInetAddress().getHostAddress() + " conectado");

        //3 - Definir stream de entrada de dados no servidor
        DataInputStream entrada = new DataInputStream(socket.getInputStream());
        String mensagem = entrada.readUTF();//receber mensagem em minúsculo do cliente
        String novaMensagem = mensagem.toUpperCase(); //converter mensagem em maiúsculo

        //4 - Definir stream de saída de dados do servidor
        DataOutputStream saida = new DataOutputStream(socket.getOutputStream());
        saida.writeUTF(novaMensagem); //Enviar mensagem em maiúsculo para cliente

        //5 - Fechar streams de entrada e saída de dados
        entrada.close();
        saida.close();

        //6 - Fechar sockets de comunicação e conexão
        socket.close();
        serverSocket.close();
    }
}
```

1 - Definir o server socket;
2 - Aguardar solicitação de conexão de cliente;
3 - Definir streams de entrada de dados:
3 - Definir streams de saída de dados:
5 - Fechar streams;
6 - Fechar sockets de conexão e comunicação.

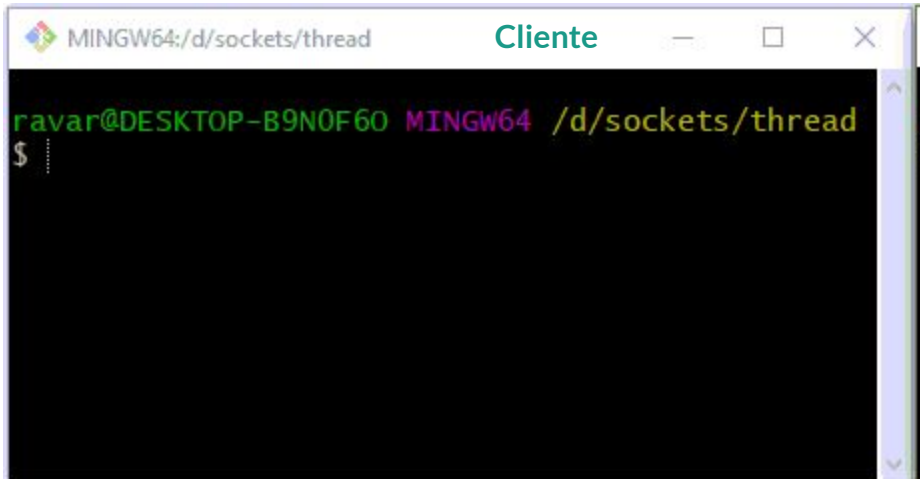
Sockets (TCP): Código - Envio de texto com Thread

- **Cliente**: envia mensagem em minúsculo
- **Servidor**: devolve a mensagem em maiúsculo
- **ThreadSockets**: linha de execução por cliente

Por meio da utilização de threads, são permitidas várias conexões com o servidor ao mesmo tempo.



Clientes enviam "rafael" e Servidor envia "RAFAEL"



Sockets (TCP): Código - Envio de texto com Thread | Cliente.java

```java
public class Cliente {

    public static void main(String[] args) throws IOException {
        //1 - Abrir conexão
        Socket socket = new Socket("127.0.0.1", 54321);

        //2 - Definir stream de saída de dados do cliente
        DataOutputStream saida = new DataOutputStream(socket.getOutputStream());
        saida.writeUTF("rafael"); //Enviar  mensagem em minúsculo para o servidor

        //3 - Definir stream de entrada de dados no cliente
        DataInputStream entrada = new DataInputStream(socket.getInputStream());
        String novaMensagem = entrada.readUTF();//Receber mensagem em maiúsculo do servidor
        System.out.println(novaMensagem); //Mostrar mensagem em maiúsculo no cliente

        //4 - Fechar streams de entrada e saída de dados
        entrada.close();
        saida.close();

        //5 - Fechar o socket
        socket.close();
    }
}
```

1 - Abrir a conexão;
2 - Definir stream de saída;
3 - Definir stream de entrada;
4 - Fechar os streams de entrada e saída;
5 - Fechar o socket.

Sockets (TCP): Código - Envio de texto com Thread | Servidor.java

```java
public class Servidor {
    public static void main(String[] args) throws IOException {
        //1 - Definir o serverSocket (abrir porta de conexão)
        ServerSocket servidorSocket = new ServerSocket(54322);
        System.out.println("A porta 54322 foi aberta!");
        System.out.println("Servidor esperando receber mensagens de clientes...");
        while (true) {
            //2 - Aguardar solicitações de conexão de clientes
            Socket socket = servidorSocket.accept();
            //Mostrar endereço IP do cliente conectado
            System.out.println("Cliente " + socket.getInetAddress().getHostAddress() + " conectado");

            //3 - Definir uma thread para cada cliente conectado
            ThreadSockets thread = new ThreadSockets(socket);
            thread.start();
        }
    }
}
```

1 - Definir o server socket (abrir porta de conexão);
2 - Aguardar solicitações de conexão de clientes;
3 - Criar thread para cada cliente conectado:

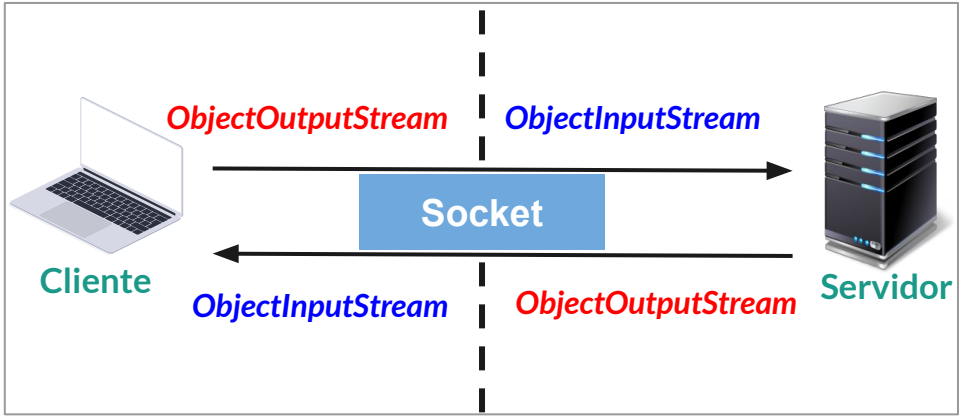
Sockets (TCP): Código - Envio de texto com Thread | ThreadSockets.java

```java
public class ThreadSockets extends Thread {
    private Socket socket;
    public ThreadSockets(Socket s) {
        this.socket = s;
    }

    public void run() {
        System.out.println(Thread.currentThread().getName());//Imprimir o nome da Thread
        try {
            //1 - Definir stream de entrada de dados no servidor
            DataInputStream entrada = new DataInputStream(socket.getInputStream());
            String mensagem = entrada.readUTF();//Recebendo mensagem em Minúsculo do Cliente
            String novaMensagem = mensagem.toUpperCase(); //Convertendo em Maiúsculo

            //2 - Definir stream de saída de dados do servidor
            DataOutputStream saida = new DataOutputStream(socket.getOutputStream());
            saida.writeUTF(novaMensagem); //Enviando mensagem em Maiúsculo para Cliente

            //3 - Fechar streams de entrada e saída de dados
            entrada.close();
            saida.close();

            //4 - Fechar socket de comunicação
            socket.close();
        } catch (IOException ioe) {
            System.out.println("Erro: " + ioe.toString());
        }
    }
}
```

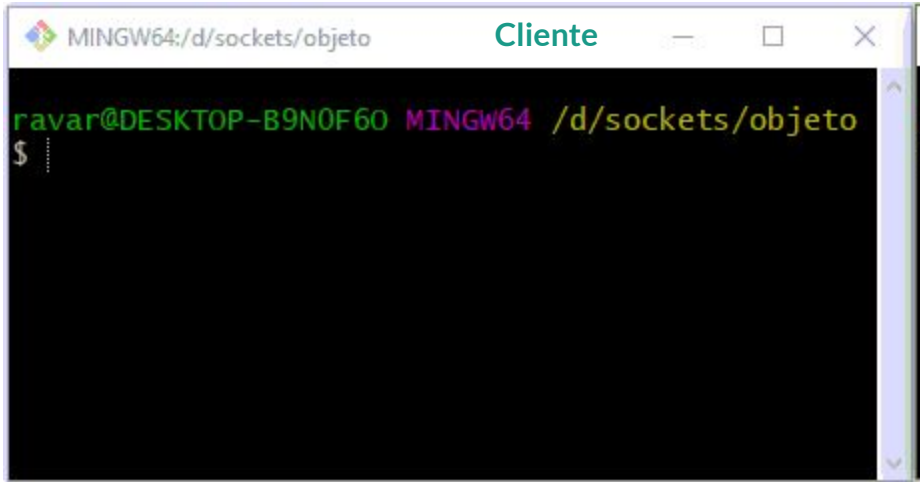
1 - Definir stream de entrada de dados;
2 - Definir stream de saída de dados:
3 - Fechar streams de entrada e saída de dados;
4 - Fechar socket de comunicação.

Sockets (TCP): Código - Envio de objeto

- **Cliente**: envia objeto do tipo pessoa preenchido
- **Servidor**: mostra dados do objeto na tela
- **Pessoa**: classe com os atributos nome e idade



Cliente envia objeto new Pessoa("Rafael Vargas", 38)
Servidor mostra dados do objeto



Sockets (TCP): Código - Envio de objeto | Cliente.java

```java
public class Cliente {

    public static void main(String[] args) throws IOException, ClassNotFoundException {
        //1 - Abrir conexão
        Socket socket = new Socket("127.0.0.1", 54323);

        //2 - Definir stream de saída de dados do cliente
        ObjectOutputStream saida = new ObjectOutputStream(socket.getOutputStream());
        Pessoa p = new Pessoa("Rafael Vargas", 38);
        saida.writeObject(p);

        //4 - Fechar streams de saída de dados
        saida.close();

        //5 - Fechar o socket
        socket.close();
    }
}
```

1 - Abrir a conexão;
2 - Definir stream de saída de dados;
3 - Fechar os stream de saída de dados;
4 - Fechar o socket.

Sockets (TCP): Código - Envio de objeto | Servidor.java

```java
public class Servidor {

    public static void main(String[] args) throws IOException, ClassNotFoundException {
        //1 - Definir o serverSocket (abrir porta de conexão)
        ServerSocket serverSocket = new ServerSocket(54323);
        System.out.println("A porta 54323 foi aberta!");
        System.out.println("Servidor esperando receber objeto de cliente...");
        //2 - Aguardar solicitação de conexão de cliente
        Socket socket = serverSocket.accept();
        //Mostrar endereço IP do cliente conectado
        System.out.println("Cliente " + socket.getInetAddress().getHostAddress() + " conectado");

        //3 - Definir stream de entrada de dados no servidor
        ObjectInputStream entrada = new ObjectInputStream(socket.getInputStream());
        Pessoa p = (Pessoa) entrada.readObject();
        System.out.println("Nome: " + p.getNome() + "\nIdade: " + p.getIdade());

        //5 - Fechar streams de entrada de dados
        entrada.close();

        //6 - Fechar sockets de comunicação e conexão
        socket.close();
        serverSocket.close();
    }
}
```

1 - Definir o server socket;
2 - Aguardar solicitação de conexão de cliente;
3 - Definir streams de entrada de dados:
4 - Fechar streams de entrada de dados;
5 - Fechar sockets de conexão e comunicação.

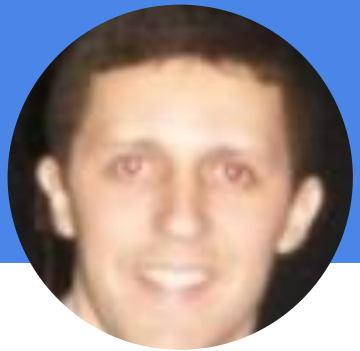
```java
public class Pessoa  implements Serializable{

    private String nome;
    private int idade;

    public Pessoa(String nome, int idade) {
        this.nome = nome;
        this.idade = idade;
    }

    public void setNome(String nome) { this.nome = nome; }
    public void setIdade(int idade) { this.idade = idade; }
    public String getNome() { return this.nome; }
    public int getIdade() { return this.idade; }
}
```

Referências Bibliográficas

H. M. Deitel, P. J. Deitel. Java: Como Programar, **Capítulo 27 – Redes**, 8ª Edição. Pearson, 2010.

Obrigado.

Sobre mim



Rafael Mesquita, Prof.

Prof. Dr. Formado em Ciência da Computação pela Universidade Federal de Lavras