
Sockets

Comunicação entre Processos



ROTEIRO

Sockets

Tipos de Sockets

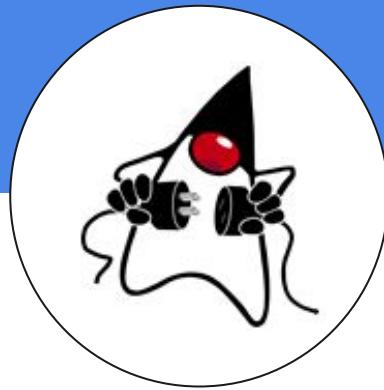
Sockets TCP

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



Cliente



Internet



Servidor

Tipos de Sockets

01 | Comunicações baseadas em fluxo (TCP)

02 | Comunicações baseadas em pacote (UDP)

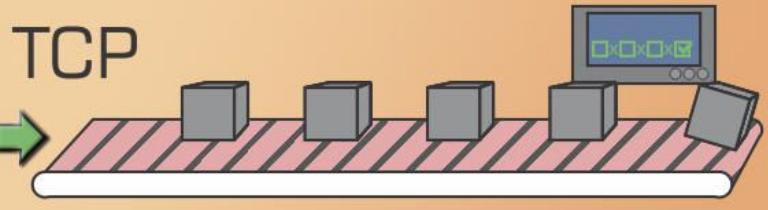
TCP

Orientado à conexão

Confiável (garante a entrega dos dados ao destino)

Adequado para envio de e-mail e download de arquivos

TCP



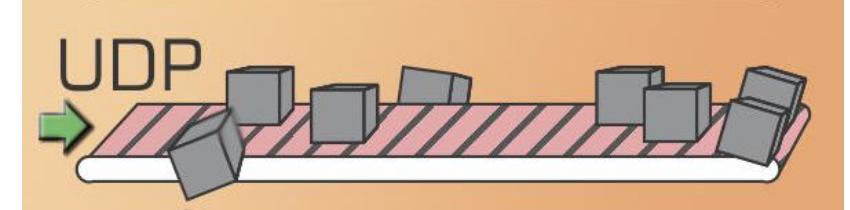
UDP

Não orientado à conexão

Não confiável (não garante a entrega dos dados ao destino)

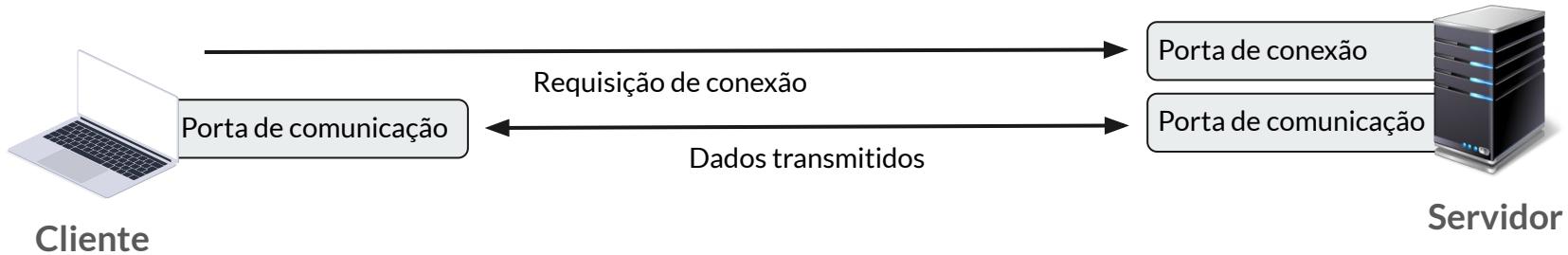
Adequado para fluxo de dados em tempo real (video/voz)

UDP

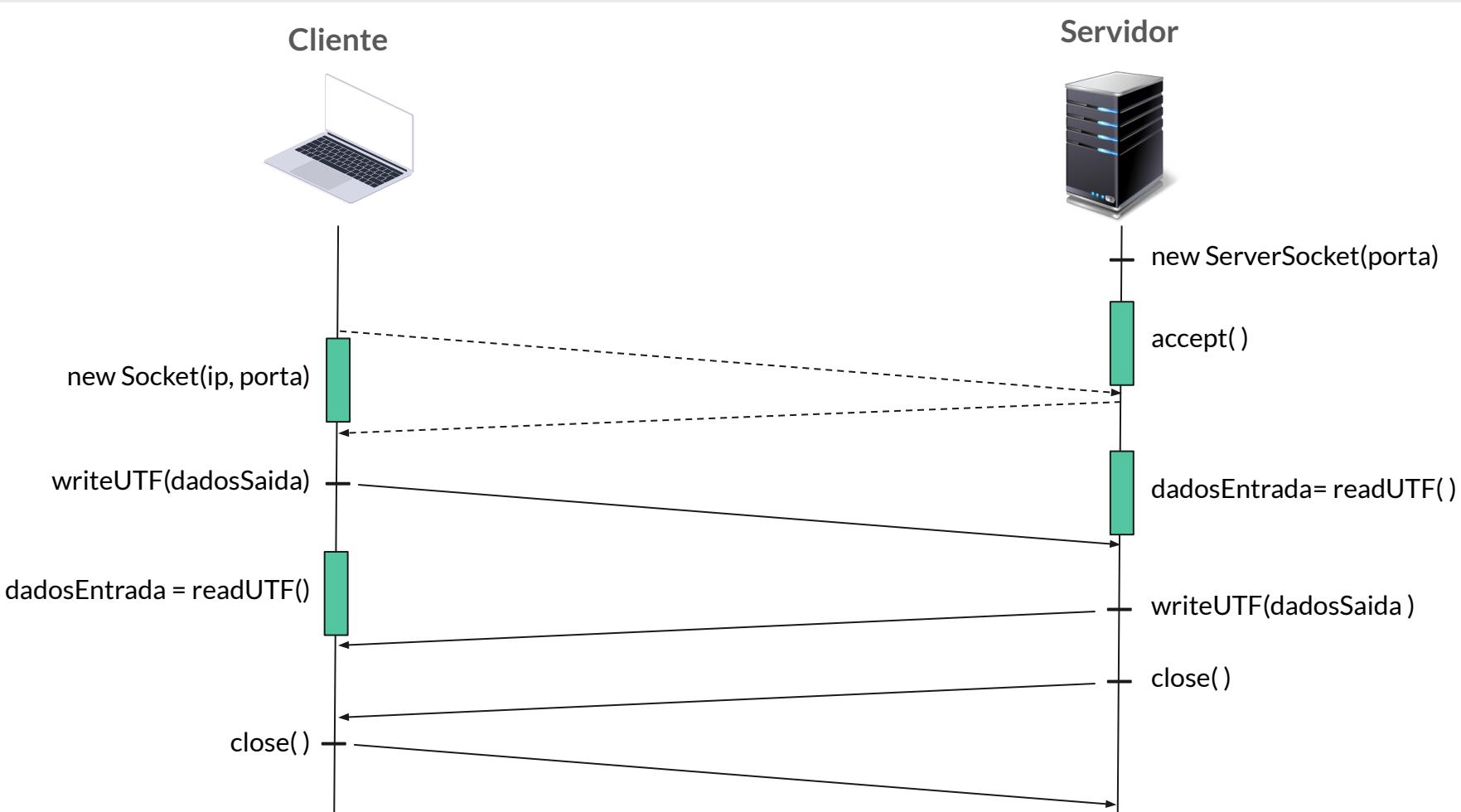


Sockets (TCP)

- 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.



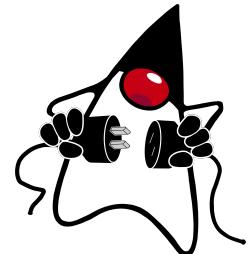
Sockets (TCP)



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;`





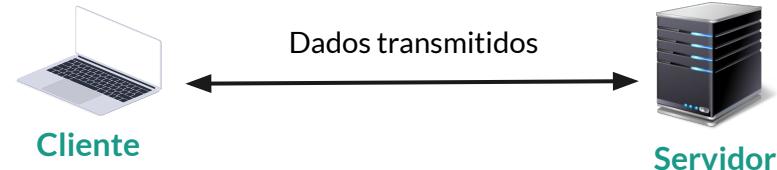
Sockets (TCP): Exemplos com GUI

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)



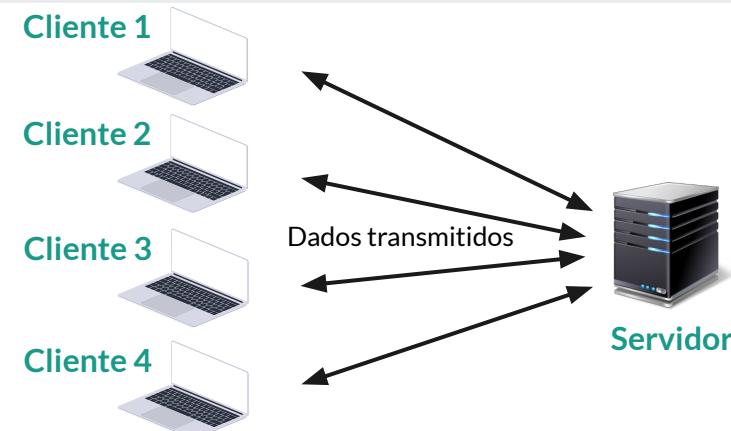
- **Cliente:** conecta-se ao servidor (Jogador 0)
- **Servidor:** inicia o servidor (Jogador X)



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>

Four screenshots of the JavaFX client application are shown side-by-side. Each window has a title bar with minimize, maximize, and close buttons. The main content area displays a table with two columns: "Cliente" and "Status". Under "Cliente", it says "0" and under "Status", it says "Não conectado" (Not connected). Below the table are two input fields: "IP" containing "127.0.0.1" and "Porta" containing "54321". To the right of these fields is a "Conectar" (Connect) button.

A screenshot of the JavaFX server application titled "Servidor". It contains a section with red text:
Exemplo de Aplicação com Sockets (1 Servidor e 4 Clientes)
Calculando quantidade de números primos (De 1 até 100.000)

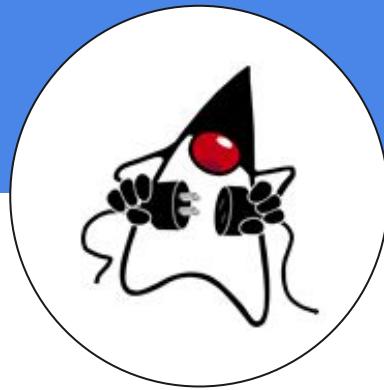
Below this, there is a table with the following columns: "Clientes", "Status", "Intervalo", and "Quantidade". The table currently shows the following data:

Clientes	Status	Intervalo	Quantidade
Cliente 1	Não conectado	1 - 25.000	
Cliente 2	Não conectado	25.001 - 50.000	
Cliente 3	Não conectado	50.001 - 75.000	
Cliente 4	Não conectado	75.001 - 100.000	

At the bottom of the table, there is a "Total" column with a value of "0".

On the left side of the server window, there is a "Porta" input field containing "54321" and a "Iniciar Servidor" (Start Server) button.

Programação Distribuída



Java Sockets: Códigos

Rafael Mesquita, Prof.



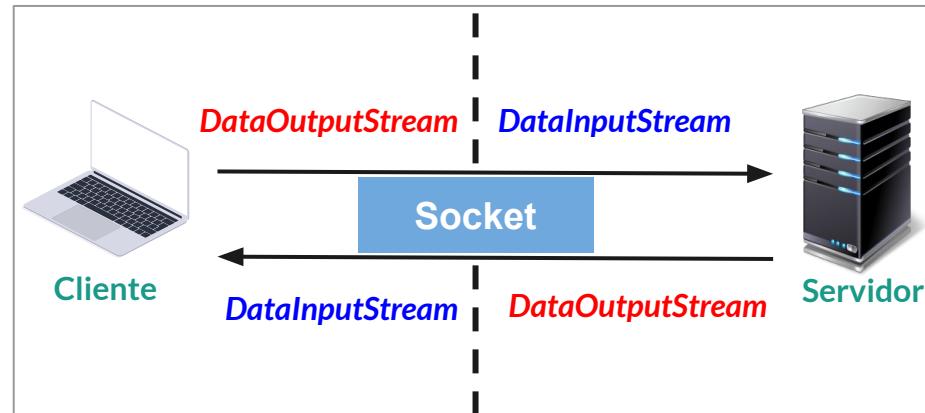
Sockets (TCP): Códigos

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”

The image shows two terminal windows side-by-side. Both windows have a title bar with the text "MINGW64:/d/Sockets/texto" and a window title of "Cliente" or "Servidor". The left window is titled "Cliente" and the right window is titled "Servidor". Both windows show a command-line interface with a prompt starting with "\$". In the "Cliente" window, the user has typed "rafael" and pressed Enter. In the "Servidor" window, the user has typed "RAFAEL" and is currently pressing Enter again. This illustrates the client sending lowercase text and the server returning uppercase text.

```
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



```
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();  
    }  
}
```

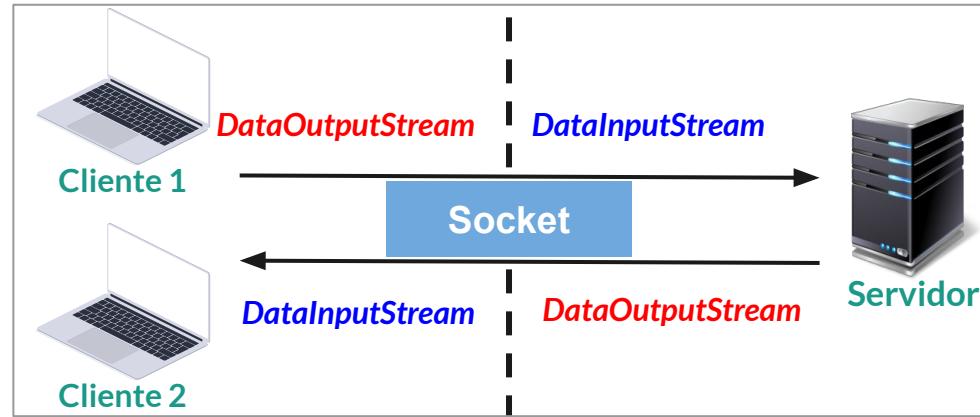
- 1 - Definir o server socket;
- 2 - Aguardar solicitação de conexão de cliente;
- 3 - Definir streams de entrada de dados:
- 3 - Definir streams de saída de dados:
- 5 - Fechar streams;
- 6 - Fechar sockets de conexão e comunicação.

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.

Cientes enviam “rafael” e Servidor envia “RAFAEL”



MINGW64:/d/sockets/thread Cliente

```
ravar@DESKTOP-B9N0F60 MINGW64 /d/sockets/thread$
```

MINGW64:/d/sockets/thread Servidor

```
ravar@DESKTOP-B9N0F60 MINGW64 /d/sockets/thread$ |
```

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

```
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

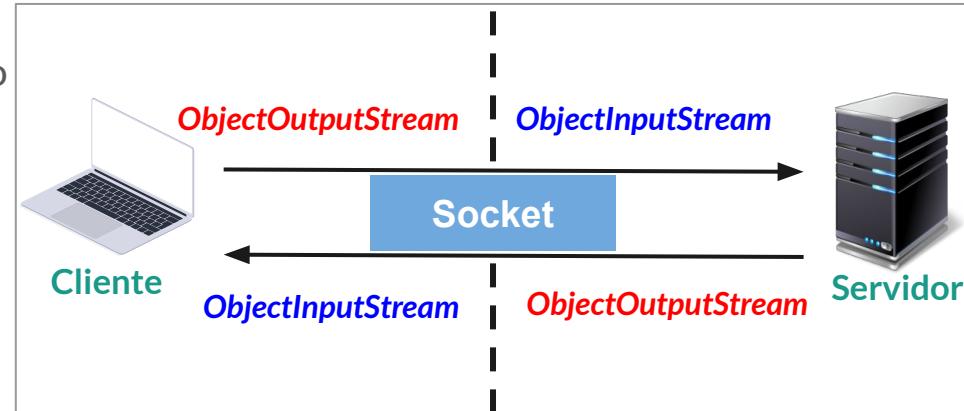
```
public class ThreadSockets extends Thread {  
    private Socket socket;  
    public ThreadSockets(Socket s) {  
        this.socket = s;  
    }  
  
    public void run() {  
        System.out.println(Thread.currentThread().getName()); //Imprimir o nome da Thread  
        try {  
            //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



MINGW64:/d/sockets/objeto Cliente

```
ravar@DESKTOP-B9N0F60 MINGW64 /d/sockets/objeto
$
```

MINGW64:/d/sockets/objeto Servidor

```
ravar@DESKTOP-B9N0F60 MINGW64 /d/sockets/objeto
$ 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

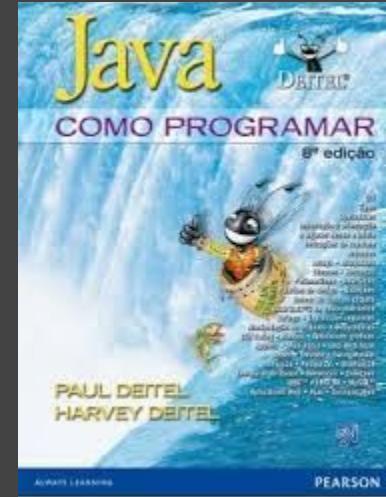
```
public class Servidor {  
  
    public static void main(String[] args) throws IOException, ClassNotFoundException {  
        //1 - Definir o serverSocket (abrir porta de conexão)  
        ServerSocket serverSocket = new ServerSocket(54323);  
        System.out.println("A porta 54323 foi aberta!");  
        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.

```
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, 8^a Edição.
Pearson, 2010.





Obrigado.



Sobre mim



Rafael Mesquita, Prof.

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