INSTITUTO FEDERAL DE GOIÁS
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Alexandre Alves Trindade
Thiago de Morais Rosa Santos

# Servidor – Cliente TCP

Goiânia
2017

Alexandre Alves Trindade

Thiago de Morais Rosa Santos

# Servidor – Cliente TCP

Trabalho da disciplina Redes Industriais do curso de graduação Engenharia de Controle e Automação apresentado no Instituto Federal de Goiás Campus Goiânia.

Professor: Prof. Me. Enio Prates Vasconcelos Filho

Goiânia

2017

## RESUMO

Este trabalho é uma implementação em java do modelo servidor cliente, utilizando o protocolo TCP. São duas aplicações com interface gráfica, e interação com o usuário, em que é definido uma porta para o servidor, e as configurações do host são o IP e a porta, então é criado um socket (IP mais porta) de conexão entre o servidor e o cliente. Esta comunicação foi testada com Hercules tanto como servidor e cliente, com diferentes IPs além do localhost, com mais de um cliente se conectando ao servidor, o qual vê os clientes em forma de lista na implementação presente neste trabalho.

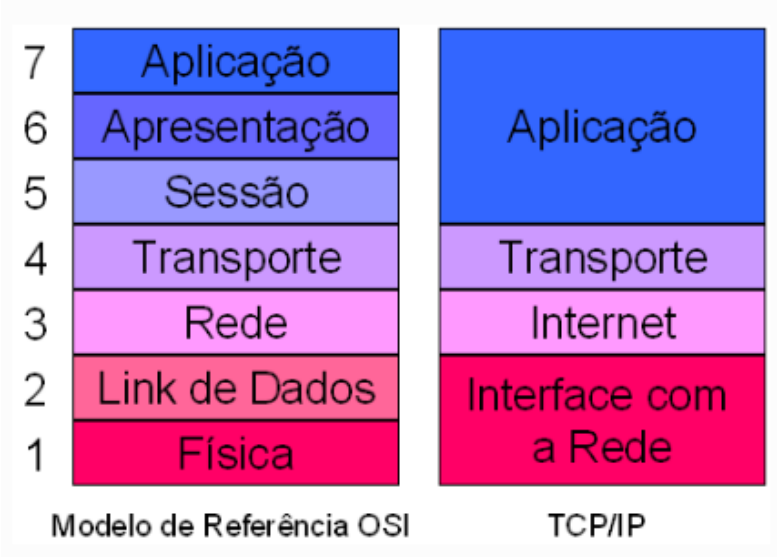


**Palavras-chave:** TCP. Java. Servidor. Cliente.

**SUMÁRIO**

## 1 INTRODUÇÃO COMPREENSIVA

O protocolo TCP/IP é o mais usado atualmente. Ele é na verdade um conjunto de protocolos como FTP, HTTP e o SMTP. A arquitetura TCP/IP pode ser vista como na seguinte figura:




http://www.clubedohardware.com.br/artigos/redes/como-o-protocolo-tcp-ip-funciona-parte-1-r34823/

O desafio descrito neste trabalho foi de implementar numa linguagem de programação de nossa escolha tanto a aplicação servidor quanto o cliente. Inicialmente começamos este trabalho em python, não conseguimos passar de algumas dificuldades como implementar as mensagens do servidor para o cliente. E tivemos mais sucesso na implementação em java, tanto a comunicação TCP, quanto a interface das aplicações utilizando o swing.

## 1.1 JUSTIFICATIVA

Para melhor entender os conteúdos vistos em sala de aula, nos livros dessa disciplina e realizando provas. O aprendizado é completo quando enfrentamos o desafio de implementar, pesquisar e gerar resultados, passando por dúvidas, e algumas dificuldades, com toda certeza iremos passar essa disciplina com mais experiência realizando projetos desse tipo que requerem mais detalhes e profundidade do conteúdo de redes industriais, especificamente comunicação servidor cliente, com a criação de Thread e sockets; do que se não realizássemos essas implementações.

## 1.2 OBJETIVOS

### 1.2.1 Objetivo geral

Este trabalho tem por objetivo criar uma aplicação servidor e uma aplicação cliente capaz de comunicação em rede.

### 1.2.2 Objetivos específicos

Tratamento de erros, criar mensagens quando houver desconectado o cliente do servidor, criar threads, verificar a porta e o IP do cliente com o servidor, evitar que o programa trave em quaisquer circunstancias durante execução.

# 2 SOCKETS

Socket é um ponto final de comunicação entre duas máquinas. Exemplo em java:

public class Socket

extends Object

implements Closeable

Esta classe implementa o cliente sockets (também conhecido somo “sockets”). O trabalho que realmente realizado é feito por uma instancia do SocketImpl class. Uma aplicação quando muda o socket que cria a implementação socket, pode configurar a si mesma para criar sockets apropriados para o firewall local.

No programa Cliente é encontrado a seguinte linha de código de número 197:

```
conexao = new Socket(ipString, port);
```

Isso cria o socket de comunicação servidor com o cliente, com o IP host e a porta de escuta do servidor.

# DESCRIÇÃO DAS FUNÇÕES

## 2.1 Aplicação Cliente

Demos o nome de NossoCliente para a aplicação e o pacote nossocliente, a classe NossoClietne.java. Importamos as seguintes classes: IOException, PrintStream, Socket, Scanner, Leval, Logger e JOptionPane.

| Classe | Utilidade |
|---|---|
| **import** java.io.IOException; | Esta classe é a classe geral produzida quando há operações input/output com falhas ou interrupções. |
| **import** java.io.PrintStream; | Adiciona a habilidade de representar valores de dados de forma conveniente. |
| **import** java.net.Socket; | Ponto de conexão de rede. Permite aplicações ler e escrever para a rede. A classe Socket representa um socket “cliente”. |

| | |
|---|---|
| **import** java.util.Scanner; | Analisa tipos primitivos e palavras usando expressões regulares. |
| **import** java.util.logging.Level; | A classe Level define um conjunto de níveis de registro padrão que podem ser usados para controlar a saída de log. Os objetos de nível de registro são pedidos e são especificados por números inteiros ordenados. A ativação do log em um determinado nível também permite o registro em todos os níveis mais altos. |
| **import** java.util.logging.Logger; | Um objeto Logger é usado para registrar mensagens para um componente específico do sistema ou aplicativo. Os iniciadores normalmente são nomeados, usando um espaço de nome hierárquico separado por pontos. Os nomes dos loggers podem ser cadeias arbitrárias, mas normalmente devem ser baseados no nome do pacote ou no nome da classe do componente registrado, como java.net ou javax.swing. Além disso, é possível criar loggers "anônimos" que não estão armazenados no namespace Logger. |
| **import** javax.swing.JOptionPane; | JOptionPaneFacilita a criação de uma caixa de diálogo padrão que solicite aos usuários um valor ou os informe de algo. |

Na linha 20 :

public class NossoCliente extends javax.swing.JFrame

Jframe é uma versão extensa do java.awt.Frame que adiciona suporte para a arquitetura do componente JFC / Swing.

A JFrameclasse é ligeiramente incompatível com Frame. Como todos os outros recipientes JFC / Swing de nível superior, um JFramecontém um JRootPanefilho único. O painel de conteúdo fornecido pelo painel raiz deve, como regra, conter todos os componentes não-menu exibidos pelo JFrame. Isso é diferente do Framecaso AWT . Como conveniência adde suas variantes, remove e setLayoutforam substituídas para encaminhá-las contentPaneconforme necessário. Isso significa que você pode escrever:

```
Frame.add (filho);
```

E a criança será adicionada ao contentPane. O painel de conteúdo sempre será não-nulo. A tentativa de configurá-lo como nula fará com que o JFrame lance uma exceção. O painel de conteúdo padrão terá um gerenciador do BorderLayout configurado nela. Consulte para RootPaneContainer obter detalhes sobre como adicionar, remover e configurar o LayoutManager de um JFrame.

Linhas 25 – 30:

```
public NossoCliente() {
        initComponents();
        this.status = false;
}
```

O initComponents() inicializa todos os objetos de componentes swing Java que a GUI de front-end usa usando o NetBeans GUI Builder. Esses componentes de swing são gerados automaticamente na classe Java sempre são feitas alterações no projeto do GUI usando o construtor GUI. Como uma regra geral, nunca se deve alterar nenhum aspecto do código nesse método, pois este está inextricavelmente vinculado ao construtor de interface gráfica NetBeans da front-end. Um usuário pode pular rapidamente entre o design front-end clicando no botão DESIGN e também a fonte, clicando no botão SOURCE.

this.status=false; esta linha de código é iniciada como falso, e é usado em outras funções desta aplicação para verificação de True ou False como condição IF.

Linhas 40-152 correspondem à função initComponetns(). Fizemos os botões, os quadros de chat, e o título de cada aplicação utilizando principalmente a ferramenta design do NetBeans.

Linha 166:

private void jConectarDesconectarActionPerformed(java.awt.event.ActionEvent evt)

Dentro desta função é verificado o status da conexão, True or False.

```
if (this.status) {
                jConectarDesconectar.setText("Conectar");
                cAmpoIp.setText("");
                cAmpoIp.setEditable(true);
                cAmpoPorta.setText("");
                cAmpoPorta.setEditable(true);
                this.status = false;
                try {
                        this.conexao.close();
```

```
                } catch (IOException ex) {

        Logger.getLogger(NossoCliente.class.getName()).log(Level.SEVERE, null, ex);

                }       }
```

É feito dentro do if o teste se o cliente está desconectado, então o usuário vê a opção Conectar dentro do botão de conexão. O campo IP é editável e inicialmente em branco, assim como o campo porta. Logger.getLogger() - Encontre ou crie um registrador para um subsistema designado. Se um registrador já foi criado com o nome dado, ele é retornado. Caso contrário, um novo registrador é criado.

Se um novo logger for criado, seu nível de log será configurado com base na configuração do LogManager e ele será configurado para também enviar a saída de log para os Manipuladores do pai. Ele será registrado no namespace global LogManager.

```
else {
                        String ipString = cAmpoIp.getText();
                        String portaString = cAmpoPorta.getText();
                        if (!ipString.equals("") && !portaString.equals("")) {
                                jConectarDesconectar.setText("Desconectar");
                                cAmpoIp.setEditable(false);
                                cAmpoPorta.setEditable(false);

                                int port = Integer.parseInt(portaString);
                                this.status = true;

                                Thread th;

                                th = new Thread(new Runnable() {
                                        public void run() {
                                                try {
                                                        // cria o socket de comunicação com o
servidor e monitora o fluxo de dados
                                conexao = new Socket(ipString, port);
                                Scanner s = new Scanner(conexao.getInputStream());
                                                        // Enquanto o fluxo de comunicação está
aberto e mantém a conexão
                                                        while (s.hasNextLine()) {
                                                                // a cada nova linha de msg ele pega
o texto e adiona a área de mensagens
                                                                String texto =
jTextArea1.getText();
                                                                String msg = s.nextLine();
                                                                texto += ("\n") + ("Server: ") + msg;
                                                                jTextArea1.setText(texto);
                                                        }
```

```
                                                    // Quanto e sai o while significa e que o
fluxo de dados foi fechado e que a
                                                    // conexão
                                                    // com o servidor foi fechada. Então ele
configura o sistema esperando uma nova
                                                    // conexão
                                                    // e dá mensagem que o servidor caiu.

        jConectarDesconectar.setText("Conectar");
                                                    cAmpoIp.setText("");
                                                    cAmpoIp.setEditable(true);
                                                    cAmpoPorta.setText("");
                                                    cAmpoPorta.setEditable(true);
                                                    status = false;

        JOptionPane.showMessageDialog(rootPane, "Você não está mais conectado ao
Servidor!");
                                            } catch (IOException ex) {
                                                    // o servidor está indiponível ou a porta
e/ou ip são inválidos

        jConectarDesconectar.setText("Conectar");
                                                    cAmpoIp.setEditable(true);
                                                    cAmpoPorta.setEditable(true);
                                                    status = false;

        JOptionPane.showMessageDialog(rootPane, "IP e/ou Porta inválidos.");
                                            }
                                        }
                                });
                                th.start();
                        }

                }
        }
```

Caso o status da conexão retorne True, os campos de modificação do IP e porta não estão disponíveis para conexão e o usuário lê Desconectar no botão de conexão. Então é criado um Thread e a aplicação tenta criar um socket de comunicação servidor-cliente. O Scanner recebe o input, e cada nova linha o texto é enviado ao chat do sevidor-cliente. Ao desconectar do servidor é enviado uma mensagem de desconexão com o servidor. É também previsto erros de IP inválido.

Linhas 235 – 255:

```
private void jEnviarActionPerformed(java.awt.event.ActionEvent evt) {// GEN-FIRST:event_jEnviarActionPerformed

        if (conexao.isConnected() && !cAmpoMensagem.getText().equals("")) {
            try {
                // abre o fluxo de saída da conexão e envia a mensagem digitada
                PrintStream saida = new PrintStream(conexao.getOutputStream());
                saida.println(cAmpoMensagem.getText());
                String area = jTextArea1.getText();// adquirindo o texto na campo das mensagens recebidas e enviadas
                String messe = cAmpoMensagem.getText();// adquirindo mensagm enviada
                area += ("\n") + ("Me: ") + messe;// concatenando as mensagens enviadas no campo das mensagens recebidas

    // e enviadas anteriormente
                jTextArea1.setText(area);
                // limpa o campo de mensagem
                cAmpoMensagem.setText("");
            } catch (IOException ex) {

    Logger.getLogger(NossoCliente.class.getName()).log(Level.SEVERE, null, ex);

            }

        }
    }
```

A função do botão enviar. Se houver conexão entre servidor cliente e o campo de mensagem vazio, a aplicação envia a mensagem enviada mantendo as mensagens anteriores, enviadas e recebidas, de modo que se possa manter um histórico da comunicação.

```
public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        // <editor-fold defaultstate="collapsed" desc=" Look and feel setting code
        // (optional) ">
        /*
         * If Nimbus (introduced in Java SE 6) is not available, stay with the default
         * look and feel. For details see
         * http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {

    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
```

```
                        }
                    }
                } catch (ClassNotFoundException ex) {

        java.util.logging.Logger.getLogger(NossoCliente.class.getName()).log(java.util.loggin
g.Level.SEVERE, null,
                                        ex);
                } catch (InstantiationException ex) {

        java.util.logging.Logger.getLogger(NossoCliente.class.getName()).log(java.util.loggin
g.Level.SEVERE, null,
                                        ex);
                } catch (IllegalAccessException ex) {

        java.util.logging.Logger.getLogger(NossoCliente.class.getName()).log(java.util.loggin
g.Level.SEVERE, null,
                                        ex);
                } catch (javax.swing.UnsupportedLookAndFeelException ex) {

        java.util.logging.Logger.getLogger(NossoCliente.class.getName()).log(java.util.loggin
g.Level.SEVERE, null,
                                        ex);
                }
                // </editor-fold>

                /* Create and display the form */
                java.awt.EventQueue.invokeLater(new Runnable() {
                    public void run() {
                        new NossoCliente().setVisible(true);
                    }
                });
        }
```

Fornece um pouco de informação sobre uma instalação LookAndFeel para melhor configurar um menu ou para a configuração inicial da aplicação. EventQueue é uma classe independente de plataforma que faz fila de eventos, tanto das classes de pares subjacentes quanto de classes de aplicativos confiáveis.

Ele encapsula a máquina de despacho de eventos assíncronos que extrai eventos da fila e os despacha chamando dispatchEvent(AWTEvent) método para isso EventQueuecom o evento a ser despachado como um argumento. O comportamento específico desta maquinação depende da implementação. Os únicos requisitos são que os eventos que realmente foram colados nesta fila (observe que os eventos que estão sendo publicados no EventQueuepodem ser agrupados) são despachados:

Sequencialmente.

Ou seja, não é permitido que vários eventos dessa fila sejam despachados simultaneamente.
Na mesma ordem em que são entram na fila.
Ou seja, se AWTEvent A é despachado para o EventQueue antes AWTEvent B, o evento B não será despachado antes do evento A.
Linhas 300 – 310:
Declarações também não devem ser modificadas, relacionadas com a função initComponents().

## 2.2 Aplicação Servidor

Na aplicação do servidor as seguintes classes não são importadas no cliente.

| Classe | Utilidade |
|---|---|
| import java.net.ServerSocket; | Esta classe implementa sockets de servidor. Um soquete do servidor espera que os pedidos entrem na rede. Ele realiza alguma operação com base nesse pedido e, em seguida, retorna um resultado ao solicitante.<br><br>O trabalho real do soquete do servidor é executado por uma instância da SocketImplclasse. Um aplicativo pode alterar a fábrica de soquetes que cria a implementação do soquete para se configurar para criar sockets apropriados para o firewall local. |
| import java.util.LinkedList; | Implementação de lista vinculada de forma dupla das interfaces Liste Deque. Implementa todas as operações de lista opcionais e permite todos os elementos (incluindo null).<br><br>Todas as operações funcionam como seria de esperar para uma lista duplamente vinculada. As operações que indexam na lista irão percorrer a lista desde o início ou o final, o que for mais próximo do índice especificado. |
| import java.awt.EventQueue; | EventQueue É uma classe independente de plataforma que faz fila de eventos, tanto das classes de pares subjacentes quanto de classes de aplicativos confiáveis. |
| import javax.swing.JFrame; | Uma versão extensa do java.awt.Frameque adiciona suporte para a arquitetura do componente JFC / Swing. |

Linhas 23-38:

public class NossoServidor extends javax.swing.JFrame {

```
private ServerSocket servidor;
   private LinkedList<Socket> clientes;
   private boolean status;

   public NossoServidor() {
      initComponents();
      this.clientes = new LinkedList<>();
      this.status = false;
}
```

Declarações do servidor, cliente, e  do status booleano. Os clientes serão organizados em forma de lista no servidor.

Linhas 48 – 190 : código gerado como padrão para a função initComponents().

Linhas 193 – 202 : função para mostrar os clientes com conexão ativa.

Linhas 206 – 334  função :

private void btConectarActionPerformed(java.awt.event.ActionEvent evt)

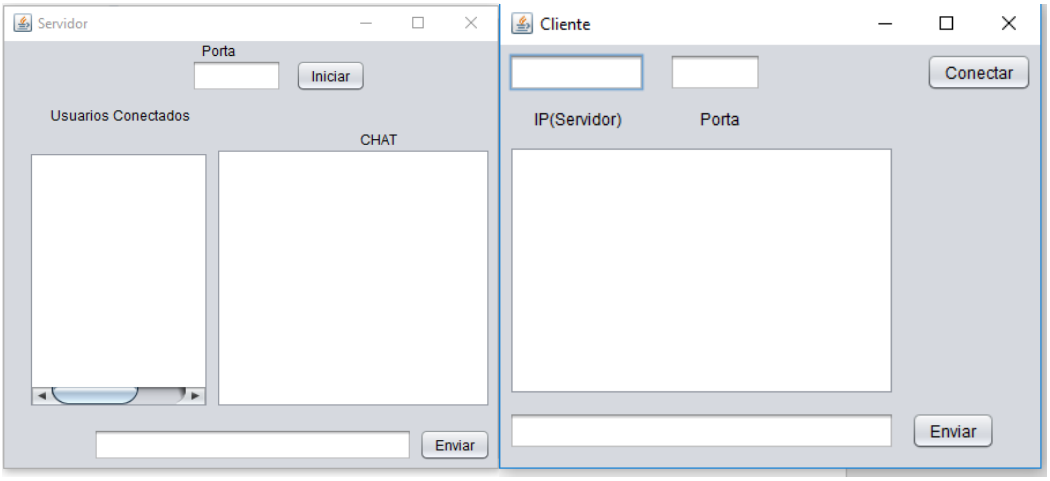
Se o servidor não está conectado, o campo porta é editável e o usuário lê no botão “Conectar”. E o programa realiza a ação de fechar a conexão se estiver aberta com algum cliente. Se o servidor estiver conectado, o campo porta não é editável, o botão de conexão “Desconectar”. Esta função muito similar à descrita na aplicação cliente, é criado um Thread onde um socket entre o servidor e o cliente é executado. E o Scanner verifica o input do usuário e assim é enviado a mensagem para o cliente. E quando a conexão é desfeita, os campos são novamente editáveis e aparece na tela um aviso de desconexão. É criado um Thread para a comunicação para cada cliente. Os clientes são ordenados numa lista.

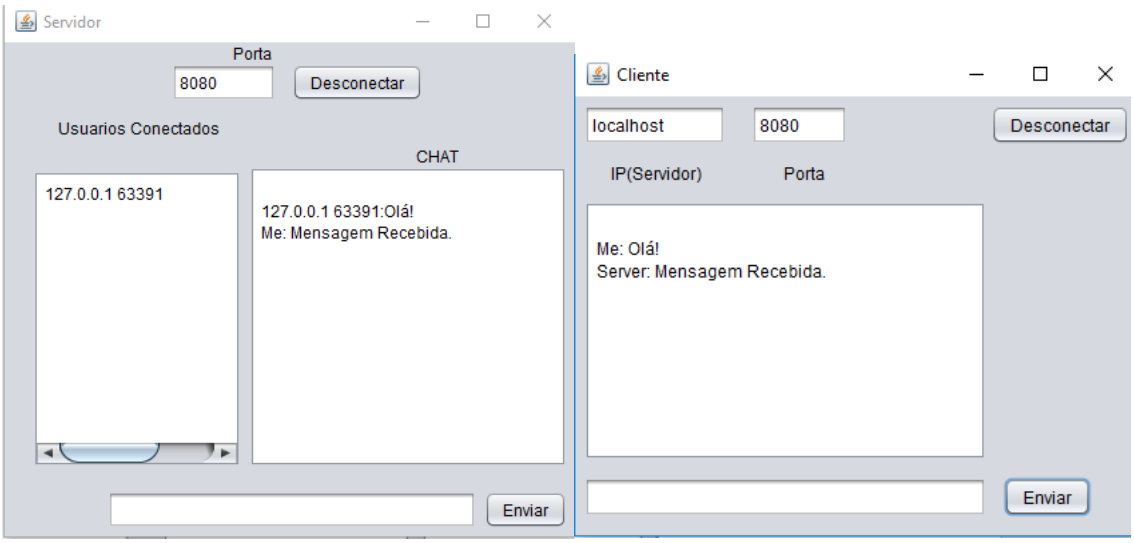
Linha 336 – 372: Varre a lista de clientes e envia a mensagem para todos.

O código main é a classe LookandFeel e no final do programa encontram-se as variáveis utilizadas como javax.swing.

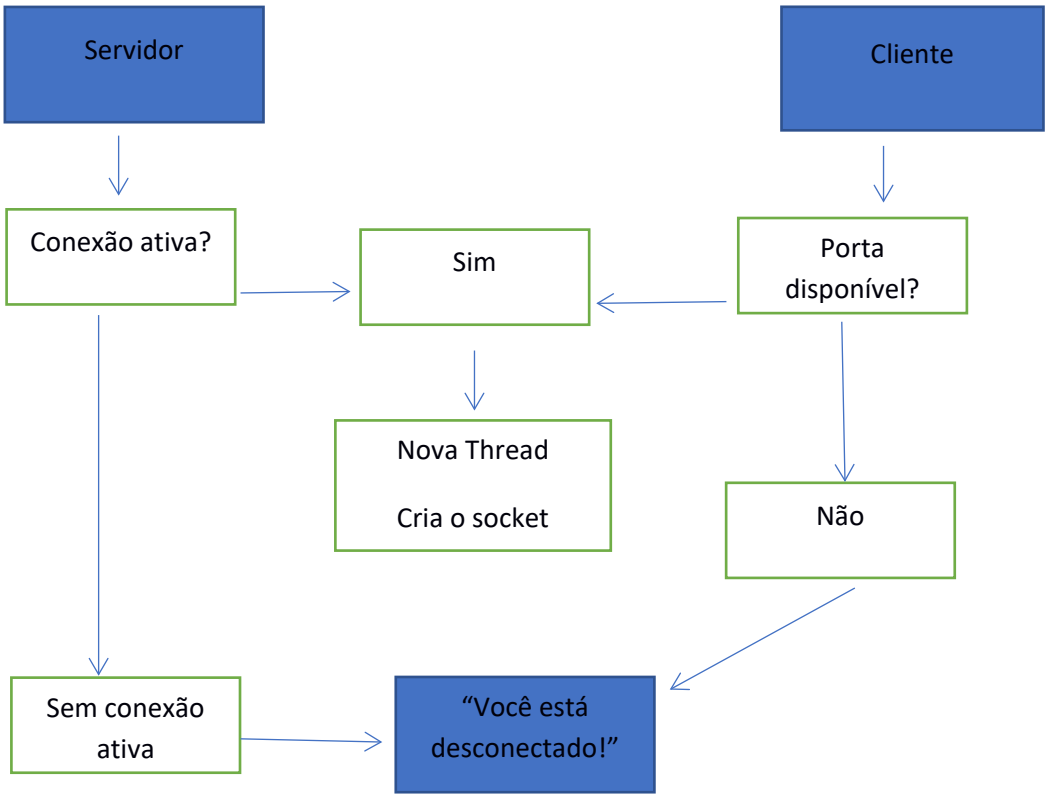
## 3 Como funciona

Segue respectivamente as aplicações Servidor e Cliente. Para utilizá-las, definimos uma porta para o servidor e clicamos em iniciar. O servidor agora está aguardando novas conexões, então configuramos a porta do cliente para 8080 – a mesma porta do servidor - neste exemplo e IP como localhost (127.0.0.1). As mensagens envidas e recebidas podem ser vistas nas figuras a seguir.



## 4 Fluxograma



# 5 CONCLUSÕES E PERSPECTIVAS

Concluímos que os objetivos definidos para este trabalho foram atingidos com sucesso, a comunicação foi realizada, mensagens enviadas tanto do servidor para o cliente e vice-versa. É possível a conexão de mais de um cliente ao mesmo tempo, o servidor trata os clientes em uma lista, as mensagens envidas do servidor chega em todos os clientes.

Deixamos as seguintes observações para melhorias: melhor tratamento de erros, existem detalhes ainda, principalmente na aplicação cliente que podem ser melhorados, como por exemplo, adicionar o botão Enter para enviar mensagens, e melhorar a experiência para o usuário. Outra questão é relacionada com tratamento de erros, mensagens claras e definidas e imediatas, quando houver envio de IP inválido, e indicação de porta não disponível.

# REFERÊNCIAS


https://docs.oracle.com/javase/7/docs/

www.clubedohardware.com.br/artigos/redes/como-o-protocolo-tcp-ip-funciona-parte-1-r34823/

http://www.java2s.com/Tutorial/Java/0320__Network/javanetSocket.htm

https://stackoverflow.com

Acesso 3 de agosto de 2017.