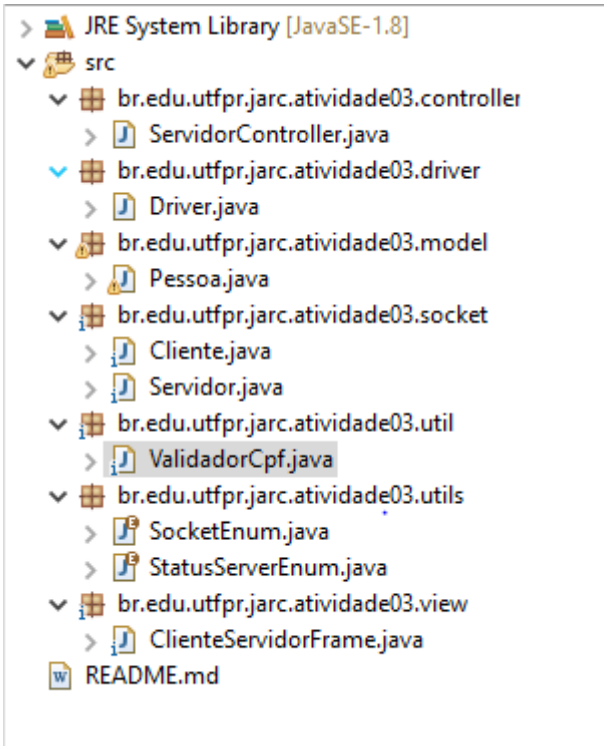
Faculdade: Universidade Tecnológica Federal do Paraná
Curso : Especialização em Tecnologia Java
Matéria : Java Aplicado A Redes De Computadores
Nome : Erik Eduardo Valcezio RA: 02329611
DATA : 31/07/2021

Atividade 3

Desenvolva uma aplicação em que um cliente, utilizando componentes Swing, transmita o objeto Pessoa (nome e idade) para o servidor. O servidor deve exibir os dados recebidos do cliente.
O cliente deve receber um aviso do servidor, informando que os dados foram transmitidos corretamente, e mostrá-lo em um TextArea .
Utilize a figura abaixo como modelo para o cliente.
NOTA: idade foi substituida por CPF para aproveitar a aula anterior

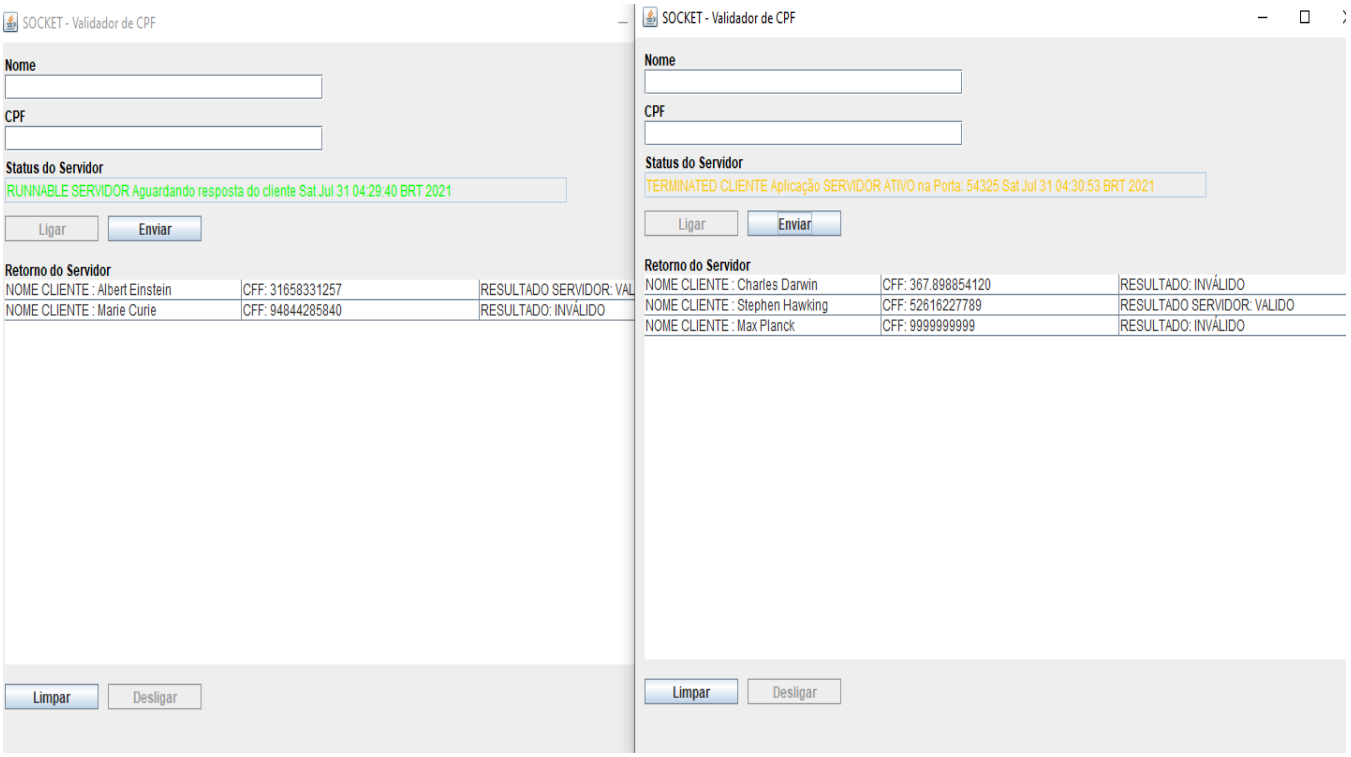
Work Projeto:

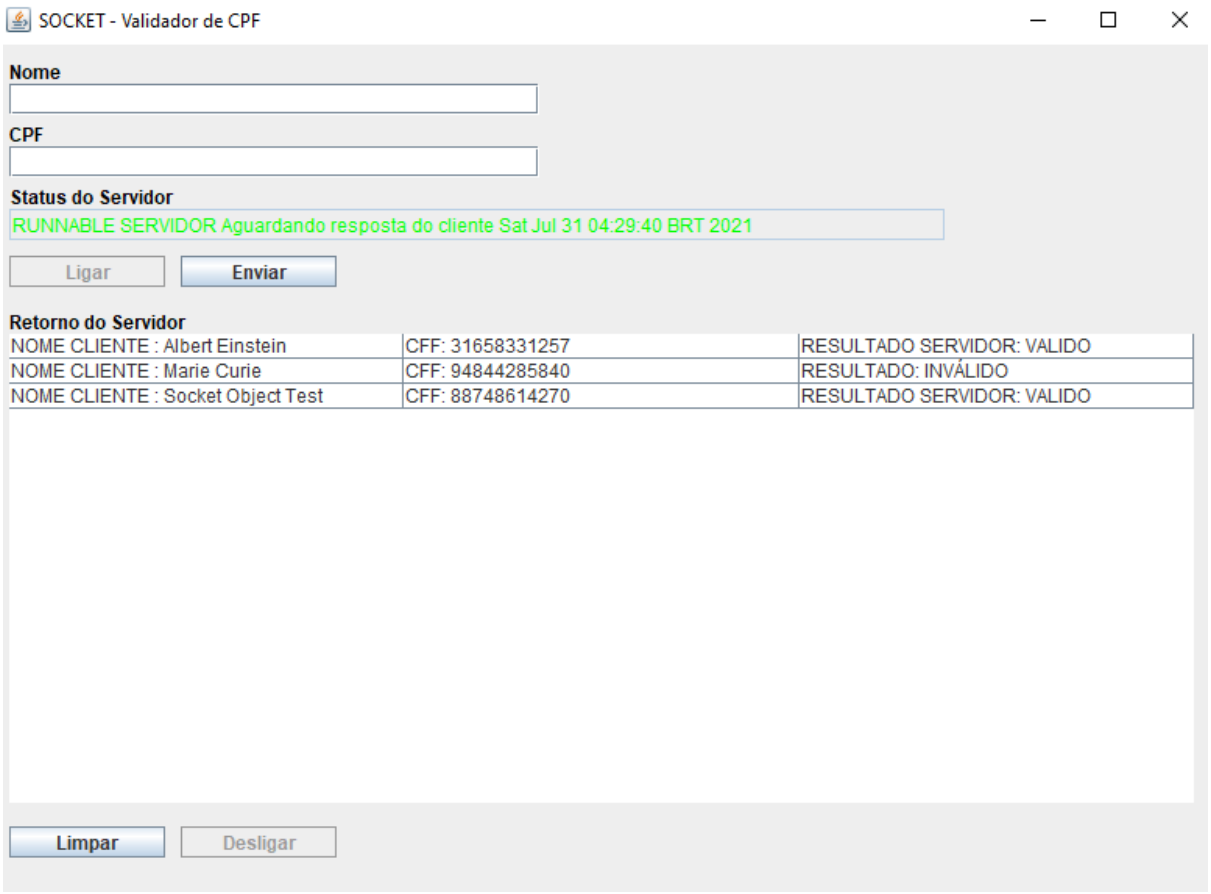


GIT:
https://github.com/erikvalcezio/java-projects-utfpr/tree/develop/Java-Aplicado-A-Redes-De-Computadores/socket-serversocket-thread-check-cpf

As que representam a solicitação dessa atividade são ClienteServidorFrame.java (Swing), ServidorController.java (Thread), Cliente.java e Servidor.java (Socket) e Pessoa (Objeto modelo).
Abaixo o código completo:

Telas da mesma aplicação (testado até 8 telas simultâneas, a primeira execução ficou como host do Servidor na porta 54325 e as demais clientes)





```java
package br.edu.utfpr.jarc.atividade03.driver;

import javax.swing.JFrame;

public class Driver {
    /**
     * @author Erik Valcezio data: 11/07/2021 - Aplicação orientada a comunicação
     *         Comunicação entre Cliente e Servidor da atividade01 Aplicação já foi
     *         desenvolvida para trabalhar com multithead para esse exemplo.
     *  @version 1.0.0
     *  @date 29/07/2021
     *
     *  @author Erik Valcezio data: 31/07/2021 - Aplicação orientada a comunicação Multipla Threads
     *
     *  Inclusão SWING, pode ser instaciada várias conexões em simultaneas ao mesmo tempo.
     *  Thread Servidor socket fica ativa em quem ligou a aplicação via tela do SWING.
     *  O retorno é enviada em uma Jtable via tela "ClienteServidorFrame"
     *
     *  Usado alguns Design patterns como singtlon e outros.
     *
     *  @version 2.0.0
     *  @date 29/07/2021
     *
     *
     */

    public static void main(String[] args) {

        ClienteServidorFrame clienteSerivdor = new ClienteServidorFrame();
        clienteSerivdor.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);


    }

}

```

```
 1 package br.edu.utfpr.jarc.atividade03.view;
 2
 3 import java.awt.Color;
23
24 public class ClienteServidorFrame extends JFrame {
25     /**
26      * Classe para tela do usuário
27      */
28
29     private static final long serialVersionUID = 1L;
30
31     private JLabel labelNome, labelCPF, labelStatusServidor,
   labelRetornoServidor;
32     private JTextField textoNome, textoCPF, textStatus;
33     private JButton botaoDesligar, botaoLimpar, botaoEnviar, botaoLigar;
34     private JTable tabela;
35     private DefaultTableModel modelo;
36
37     private ServidorController servidorControl;
38     private Cliente cliente;
39     private Thread thread;
40
41     public ClienteServidorFrame() {
42         super("SOCKET - Validador de CPF");
43         Container container = getContentPane();
44         setLayout(null);
45
46         labelNome = new JLabel("Nome");
47         labelCPF = new JLabel("CPF");
48         labelStatusServidor = new JLabel("Status do Servidor");
49         labelRetornoServidor = new JLabel("Retorno do Servidor");
50
51         labelNome.setBounds(10, 10, 280, 15);
52         labelCPF.setBounds(10, 50, 280, 15);
53         labelStatusServidor.setBounds(10, 90, 280, 15);
54         labelRetornoServidor.setBounds(10, 170, 280, 15);
55
56         labelNome.setForeground(Color.BLACK);
57         labelCPF.setForeground(Color.BLACK);
58         labelStatusServidor.setForeground(Color.BLACK);
59         labelRetornoServidor.setForeground(Color.BLACK);
60
61         container.add(labelNome);
62         container.add(labelCPF);
63         container.add(labelStatusServidor);
64         container.add(labelRetornoServidor);
65
66         textoNome = new JTextField();
67         textoCPF = new JTextField();
68         textStatus = new JTextField();
```

```
 69
 70          textoNome.setBounds(10, 25, 340, 20);
 71          textoCPF.setBounds(10, 65, 340, 20);
 72          textStatus.setBounds(10, 105, 600, 20);
 73
 74          textStatus.setEditable(false);
 75          textStatus.setForeground(Color.RED);
 76
 77          container.add(textoNome);
 78          container.add(textoCPF);
 79          container.add(textStatus);
 80          textStatus.setText(StatusServerEnum.STATUS_OFF.getStatus());
 81
 82          botaoLigar = new JButton("Ligar");
 83          botaoEnviar = new JButton("Enviar");
 84          botaoEnviar.setEnabled(false);
 85
 86          botaoLigar.setBounds(10, 135, 100, 20);
 87          botaoEnviar.setBounds(120, 135, 100, 20);
 88
 89          container.add(botaoLigar);
 90          container.add(botaoEnviar);
 91
 92          tabela = new JTable();
 93
 94          tabela.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
 95
 96          modelo = (DefaultTableModel) tabela.getModel();
 97
 98          modelo.addColumn("NOME CLIENTE");
 99          modelo.addColumn("CFF");
100          modelo.addColumn("RESULTADO");
101
102          tabela.setBounds(10, 185, 760, 300);
103          container.add(tabela);
104
105          botaoLimpar = new JButton("Limpar");
106          botaoDesligar = new JButton("Desligar");
107
108          botaoLimpar.setBounds(10, 500, 100, 20);
109          botaoDesligar.setBounds(120, 500, 100, 20);
110          botaoDesligar.setEnabled(false);
111
112
113          container.add(botaoLimpar);
114          container.add(botaoDesligar);
115
116          setSize(800, 600);
117          setVisible(true);
118          setLocationRelativeTo(null);
```

```
119
120         if (this.servidorControl == null) {
121             servidorControl = new ServidorController();
122         }
123
124         botaoEnviar.addActionListener(new ActionListener() {
125             @Override
126             public void actionPerformed(ActionEvent e) {
127                 enviar();
128             }
129         });
130
131         botaoLimpar.addActionListener(new ActionListener() {
132             @Override
133             public void actionPerformed(ActionEvent e) {
134                 limparTabela();
135             }
136         });
137
138         botaoDesligar.addActionListener(new ActionListener() {
139             @Override
140             public void actionPerformed(ActionEvent e) {
141                 desligar();
142                 limparTabela();
143             }
144         });
145
146         botaoLigar.addActionListener(new ActionListener() {
147             @Override
148             public void actionPerformed(ActionEvent e) {
149                 ligar();
150             }
151         });
152     }
153
154     private void limparTabela() {
155         modelo.setRowCount(0);
156         modelo.getDataVector().clear();
157     }
158
159     /**
160      * thread.start();  Inicia o Servidor por methodo run() implicido
    usando recursos do lambda
161      * na classe ControllerServidor
162      */
163     private void ligar() {
164
165         this.servidorControl.turnOnServer(true);
166
167         this.thread = this.servidorControl.excutarThreadServidor();
```

```
168         this.thread.start();
169 
170         while (this.servidorControl.getStatusServidor() == null) {
171             // awaits customer response SERVIDOR
172         }
173 
174         if (this.servidorControl.getStatusServidor().equals
    (StatusServerEnum.STATUS_WAIT.getStatus())) {
175             textStatus.setForeground(Color.GREEN);
176             textStatus.setText(this.thread.getState() + " SERVIDOR " +
    this.servidorControl.getStatusServidor() + " "
177                     + Calendar.getInstance().getTime());
178         }
179         if (this.servidorControl.getStatusServidor().equals
    (StatusServerEnum.STATUS_RUNNABLE.getStatus())) {
180             textStatus.setForeground(Color.ORANGE);
181             textStatus.setText(this.thread.getState() + " CLIENTE "  +
    this.servidorControl.getStatusServidor() + " "
182                     + Calendar.getInstance().getTime());
183         }
184 
185         //botaoDesligar.setEnabled(true); //em desenvolvimento
186         botaoLigar.setEnabled(false);
187         botaoEnviar.setEnabled(true);
188 
189     }
190 
191     private void addLine(String status, String validador, String string) {
192         modelo.addRow(new Object[] { status, validador, string });
193 
194     }
195 
196     /**
197      * Semântica para Botao desligar em desenvolvimento, não está ativo na
    tela da API
198      */
199 
200     private void desligar() {
201         if (this.servidorControl.getStatusServidor().equals
    (StatusServerEnum.STATUS_WAIT.getStatus())) {
202             if (this.servidorControl == null) {
203 
204                 JOptionPane.showMessageDialog(this, "Servidor Não está
    ligado!");
205             }
206 
207             this.servidorControl.turnOnServer(false);
208             this.textStatus.setText(StatusServerEnum.STATUS_OFF.getStatus
    ());
209             this.textStatus.setForeground(Color.RED);
```

```
210             botaoDesligar.setEnabled(false);
211             botaoLigar.setEnabled(true);
212             botaoEnviar.setEnabled(false);
213             JOptionPane.showMessageDialog(this, "Servidor Desligado!");
214             this.thread.interrupt();
215 
216         } else {
217             JOptionPane.showMessageDialog(this, "Aplicação Servidor ativa
    em outra execução!");
218         }
219     }
220 
221     /**
222      * Remover validação do CPF e colocar do lado Servidor
223      * nesse momento da matéria apenas apresenta comunicação por thread +
    socket com objeto
224      */
225 
226     private void enviar() {
227         if (!textoNome.getText().equals("") && !textoCPF.getText().equals
    ("")) {
228 
229             if (cliente == null) {
230                 cliente = new Cliente();
231             }
232 
233 
234             if (this.servidorControl.getStatusServidor().equals
    (StatusServerEnum.STATUS_RUNNABLE.getStatus())
235                     || this.servidorControl.getStatusServidor().equals
    (StatusServerEnum.STATUS_WAIT.getStatus())) {
236 
237                 Pessoa pessoa = cliente.comunicarServidor
    (SocketEnum.HOST1.getHost(),
238                         Integer.parseInt(SocketEnum.HOST1.getPort()),
    textoNome.getText().trim(), textoCPF.getText().trim());
239 
240                 JOptionPane.showMessageDialog(this, "Dados enviados para
    servidor!");
241 
242                 boolean result = new ValidadorCpf(pessoa.getCpf
    ()).verificarCPF();
243 
244                     addLine("NOME CLIENTE : " + pessoa.getNome(), "CFF: " +
    pessoa.getCpf(),
245                             result ? "RESULTADO SERVIDOR: VALIDO" :
    "RESULTADO: INVÁLIDO");
246 
247                     this.limpar();
248 
```

```
249             } else {
250 
251                 JOptionPane.showMessageDialog(this, "Atenção: Ligar o
    servidor!");
252             }
253 
254         } else {
255             JOptionPane.showMessageDialog(this, "Nome e CPF devem ser
    informados.");
256         }
257     }
258 
259     private void limpar() {
260         this.textoNome.setText("");
261         this.textoCPF.setText("");
262     }
263 }
264 
```

```java
package br.edu.utfpr.jarc.atividade03.socket;

import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.Socket;

import br.edu.utfpr.jarc.atividade03.model.Pessoa;

public class Cliente {
    /**
     * @author Erik Valcezio data: 11/07/2021 - Aplicação orientada a comunicação
     *         Classe com as dependencias necessaria para instanciar "Cliente"
     *         Socket que faz requisição com a class Servidor, para atender
     *         atividade01 Java Aplicado A Redes De Computadores na UTFPR
     * @parm host é endereço de ip para requisição no servidor
     * @parm porta é o número da porta
     *
     * @version 2.0.0
     *
     * classe recebe dados via tela de nome e cpf para e envia por Objeto Pessoa para
     * o servidor
     */

    private ObjectInputStream entrada;
    private Pessoa pessoa;

    public Pessoa comunicarServidor(String host, int porta, String nome, String cpf) {

        try (Socket socket = new Socket(host, porta)) {

             //Envia objeto Pessoa do cliente para o servidor
            new ObjectOutputStream(socket.getOutputStream()).writeObject(new Pessoa(nome, cpf));

            //Recebe o retorno do servidor
            this.entrada = new ObjectInputStream(socket.getInputStream());

            return (Pessoa) this.entrada.readObject();

        } catch (Exception e) {
            //montar classe LOGGER da aplicação, remover Sysout e subir a exception para tela do usuário "throw"
            System.out.println("Erro no envio de dados para o Servidor");
        }
        return pessoa;
    }
```

```
45 }
46
```

```java
package br.edu.utfpr.jarc.atividade03.controller;

import br.edu.utfpr.jarc.atividade03.socket.Servidor;

public class ServidorController  {

    private Servidor servidor;

    public ServidorController() {

        if (this.servidor == null) {
            this.servidor = new Servidor();
        }
    }

    public Thread excutarThreadServidor() throws UnsupportedOperationException{

        return new Thread(() -> {
            try {
                servidor.iniciarServidor(Integer.parseInt(SocketEnum.HOST1.getPort()));
            } catch (Exception e) {
                throw new UnsupportedOperationException(e.getMessage());
            }
        }, "Thread_Servidor");
    }

    public String getStatusServidor() {
        return this.servidor.getStep();
    }

    public void turnOnServer(boolean onOff) {
        this.servidor.setTurnOn(onOff);
    }

    public boolean getturnOn() {
        return this.servidor.getTurnOn();
    }

}

```

```java
package br.edu.utfpr.jarc.atividade03.model;

import java.io.Serializable;

public class Pessoa implements Serializable{
    /**
     * Classe representa Objeto para aula
     */

    private static final long serialVersionUID = 33L;
    private String nome;
    private String cpf;

    public Pessoa() {
    }

    public Pessoa(String nome, String cpf) {
        super();
        this.nome = nome;
        this.cpf = cpf;
    }

    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public String getCpf() {
        return cpf;
    }
    public void setCpf(String cpf) {
        this.cpf = cpf;
    }

}

```

```java
package br.edu.utfpr.jarc.atividade03.socket;

import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.net.ServerSocket;
import java.net.Socket;

import br.edu.utfpr.jarc.atividade03.model.Pessoa;
import br.edu.utfpr.jarc.atividade03.utils.StatusServerEnum;

public class Servidor {
    /**
     * @author Erik Valcezio data: 11/07/2021 - Aplicação orientada a comunicação
     *         Classe com as dependencias necessaria para instanciar "Servidor"
     *         ServerSocket e Socket para atender atividade01 Java Aplicado A Redes
     *         De Computadores na UTFPR  *
     *
     *         Classe é iniciada por thread atravez da ServidorController
     *         Recebe dados do cliente e retorna objeto, nesse momento não
     *         está sendo aplicada nenhuma regra de validação
     */

    private ObjectInputStream entrada;
    private ObjectOutputStream saida;
    private String step;
    private boolean turnOn = false;

    public void iniciarServidor(int porta) {

        do {
            getTurnOn();

            try (ServerSocket server = new ServerSocket(porta)) {

                this.step = StatusServerEnum.STATUS_WAIT.getStatus();
                Socket socket = server.accept();

                entrada = new ObjectInputStream(socket.getInputStream());

                Pessoa p = (Pessoa) entrada.readObject();

                saida = new ObjectOutputStream(socket.getOutputStream());

                saida.writeObject(p);

                this.step = StatusServerEnum.STATUS_SEND.getStatus();
```

```
48
49              } catch (IOException | ClassNotFoundException e) {
50                  this.step = StatusServerEnum.STATUS_RUNNABLE.getStatus();
51                  this.setTurnOn(false);
52              }
53
54          } while (turnOn);
55      }
56
57      public String getStep() {
58          return this.step;
59      }
60
61      public void setTurnOn(boolean turnOn) {
62          this.turnOn = turnOn;
63      }
64
65      public boolean getTurnOn() {
66          return this.turnOn;
67
68      }
69
70  }
71
```

```java
package br.edu.utfpr.jarc.atividade03.utils;

public enum SocketEnum {

    HOST1("127.0.0.1", "54325");

    private String host;
    private String port;


    SocketEnum(String host,String port) {
        this.host = host;
        this.port = port;

    }

    public String getPort() {
        return port;
    }

    public String getHost() {
        return host;
    }

    public static SocketEnum getServer(String host) {
        if (host == null) {
            return null;
        }

        for (SocketEnum socket : SocketEnum.values()) {
            if(socket.getHost().equalsIgnoreCase(host)) {
                return socket;
            }
        }

        return null;
    }

}

```

```java
package br.edu.utfpr.jarc.atividade03.utils;

public enum StatusServerEnum {

    STATUS_WAIT("Aguardando resposta do cliente"),
    STATUS_RUNNABLE("Aplicação SERVIDOR ATIVO na Porta: " +
 SocketEnum.HOST1.getPort()),
    STATUS_SEND("Resposta enviada para o cliente"),
    STATUS_ON("Ligado"),
    STATUS_OFF("Desligado");

    private String status;

    StatusServerEnum(String status) {
        this.status = status;
    }

    public String getStatus() {
        return this.status;
    }

}

```

```java
package br.edu.utfpr.jarc.atividade03.util;

public class ValidadorCpf {
    /**
     * @author Erik Valcezio
     * data: 11/07/2021
     * Criado com base do autor djJoe no link https://www.hardware.com.br/comunidade/tutorial-cpf/1261362/
     * Porém foi modificado com melhorias em todos metódos que estão comentados
     *
     * data: 29/07/2021
     * Usado na atividade de SWING, para demostrar a funcionalidades
     *
     */

    private String cpf;

    public ValidadorCpf(String cpf) {
        this.cpf = cpf.trim();
    }
    /**
     * @return faz todas validações para documento CPF
     */
    public boolean verificarCPF() {
        System.out.println("Fazendo a validação do CPF");
        removerCaracteres();
        return (verificarSeTamanhoInvalido() && verificarSeDigIguais() && calculoComCpf());
    }

    private void removerCaracteres() {
        this.cpf = this.cpf.replace("-", "");
        this.cpf = this.cpf.replace(".", "");
        System.out.println("removerCaracteres : " + this.cpf);
    }

    public boolean verificarSeTamanhoInvalido() {

        if (this.cpf.length() == 11)
            return true;
        return false;
    }

    /**
     * @return valida o sequencial se está igual 0
     * http://www.receita.fazenda.gov.br/aplicacoes/atcta/cpf/consultapublica.asp
     */

```

```
47      private boolean verificarSeDigIguais() {
48          char primDig = '0';
49          char[] charCpf = this.cpf.toCharArray();
50
51          for (char c : charCpf)
52
53              if (c != primDig)
54                  return true;
55          return false;
56
57      }
58
59      /**
60       * @return valida primeiro e segundo digito do CPF para verificar
  integridade "true = OK"
61       */
62      private boolean calculoComCpf() {
63          return validaPrimeiroDigCpf() && validaSegundoDigCpf();
64      }
65
66      /**
67       * @return valida o primeiro digito do CPF na base do calculo com resto
  para comparar
68       */
69      private boolean validaPrimeiroDigCpf() {
70          Integer total = 0;
71          String validaDig = cpf.substring(0, 9);
72          char[] charCpf = validaDig.toCharArray();
73          for (int i = 0, x = 10; x >= 2; x--, i++) {
74              total += ((Integer.parseInt(String.valueOf(charCpf[i]))) * x);
75          }
76          total = ((total * 10) % 11);
77          return (total.equals(Integer.parseInt(cpf.substring(9, 10))));
78      }
79
80      /**
81       * @return valida o segundo digito do CPF na base do calculo com resto
  para comparar
82       */
83      private boolean validaSegundoDigCpf() {
84          Integer total = 0;
85          String validaDig = cpf.substring(0, 10);
86          char[] charCpf = validaDig.toCharArray();
87          for (int i = 0, x = 11; x >= 2; x--, i++) {
88              total += ((Integer.parseInt(String.valueOf(charCpf[i]))) * x);
89          }
90          total = ((total * 10) % 11);
91          return (total.equals(Integer.parseInt(cpf.substring(10, 11)))); //
  compara sobra com o digito
92      }
```

```
93 }
94
95
```