

UNIVERSIDADE FEDERAL DE OURO PRETO

Innan Plínio Rangel Amorim - 16.2.8416
Vinicius de Souza Fialho - 15.1.8174

Trabalho de Redes de Computadores
Engenharia de Computação

João Monlevade

2016

Introdução

Este trabalho aborda o tema : comunicação em redes , mais especificamente tecnicas de comunicação direta entre 2 máquinas através de uma linguagem especifica, no caso “Java”.

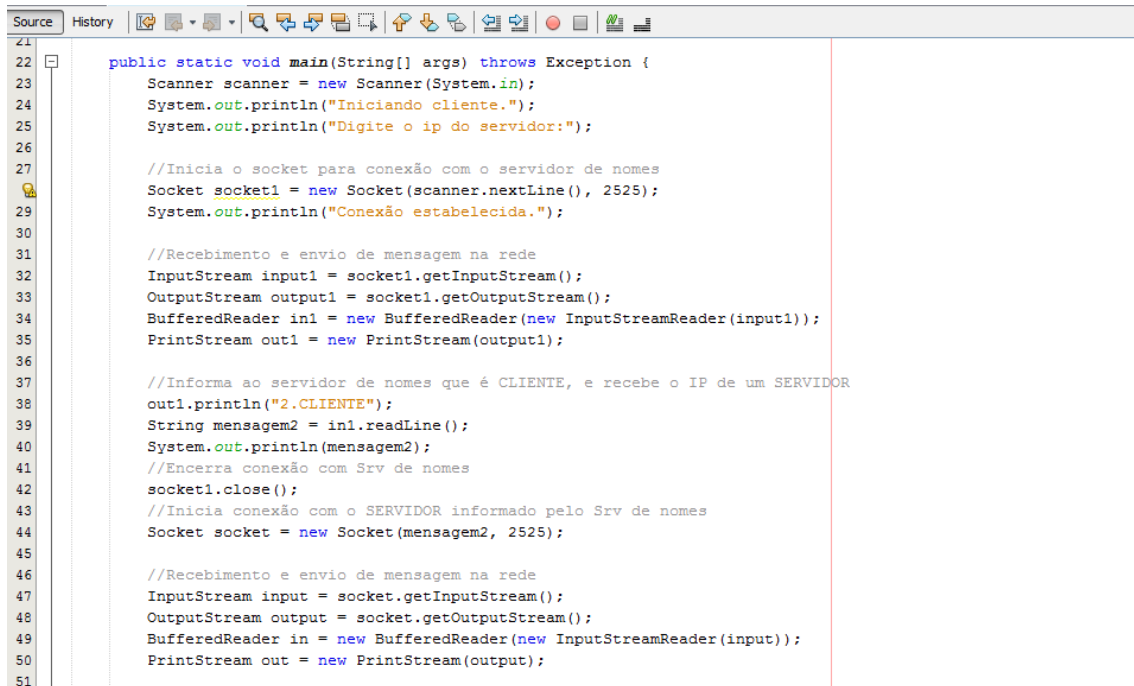
E objetivo deste trabalho aplicar tecnicas de redes para enviar informações entres 2 computadores simulando um cliente – servidor. Basicamente , simulamos um cliente em uma máquina , um servidor em outra maquina, e ainda implementamos um servidor “DNS” em uma terceira máquina , que distribui ip para os clientes, dos servidores cadastrados.

Neste documentos estao os codigos , anotações , e explicações do passo a passo no processo da codificação dos cliente e servidores.

Implementação

A implementação do projeto foi feita em java.

Cliente :



```
21
22 public static void main(String[] args) throws Exception {
23     Scanner scanner = new Scanner(System.in);
24     System.out.println("Iniciando cliente.");
25     System.out.println("Digite o ip do servidor:");
26
27     //Inicia o socket para conexão com o servidor de nomes
28     Socket socket1 = new Socket(scanner.nextLine(), 2525);
29     System.out.println("Conexão estabelecida.");
30
31     //Recebimento e envio de mensagem na rede
32     InputStream input1 = socket1.getInputStream();
33     OutputStream output1 = socket1.getOutputStream();
34     BufferedReader in1 = new BufferedReader(new InputStreamReader(input1));
35     PrintStream out1 = new PrintStream(output1);
36
37     //Informa ao servidor de nomes que é CLIENTE, e recebe o IP de um SERVIDOR
38     out1.println("2.CLIENTE");
39     String mensagem2 = in1.readLine();
40     System.out.println(mensagem2);
41     //Encerra conexão com Srv de nomes
42     socket1.close();
43     //Inicia conexão com o SERVIDOR informado pelo Srv de nomes
44     Socket socket = new Socket(mensagem2, 2525);
45
46     //Recebimento e envio de mensagem na rede
47     InputStream input = socket.getInputStream();
48     OutputStream output = socket.getOutputStream();
49     BufferedReader in = new BufferedReader(new InputStreamReader(input));
50     PrintStream out = new PrintStream(output);
51
```

Socket, cria o `socket`, que seria o ponto final de comunicação entre 2 máquinas, ele permite que a comunicação ocorra. Existem 3 tipos de sockets, foi usado o stream socket que envia o pacote com o cuidado de que o servidor recebe e que a conexão está mantida; esta interface é implementada pelo protocolo Transfer Control Protocol (TCP). O socket é iniciado a partir do IP digitado pelo usuário e através da porta 2525.

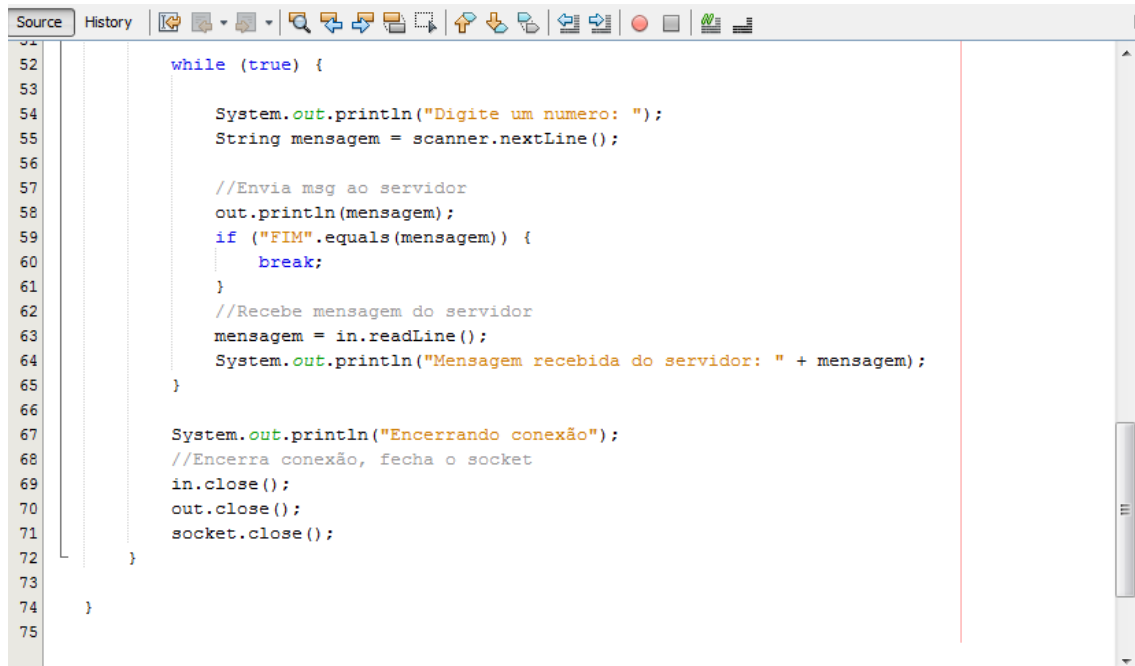
getInputStream faz parte da leitura, ou seja, está conectado a alguma fonte de dados, controla o fluxo de entrada de dados, e retorna um fluxo de entrada para leitura de bytes a partir desse socket.

getOutputStream faz parte da leitura, ou seja, está conectado a alguma fonte de dados, controla o fluxo de saída de dados, e retorna um fluxo de saída para leitura de bytes a partir desse socket.

InputStreamReader: lê um texto a partir de um fluxo de caractere – entrada. Ele funciona como uma ponte de fluxo de bytes para caractere, como um tradutor, ou seja, ele recebe informação do outputStream, e traduz em caracteres.

PrintStream: permite que as informações que foram enviadas do servidor para o cliente sejam “*printadas*”, ou seja, exibidas na tela.

Nessa parte do código o cliente cria um socket que se comunica com o servidor "dns" recebe o IP do servidor, e cria um novo socket para se comunicar com ele.

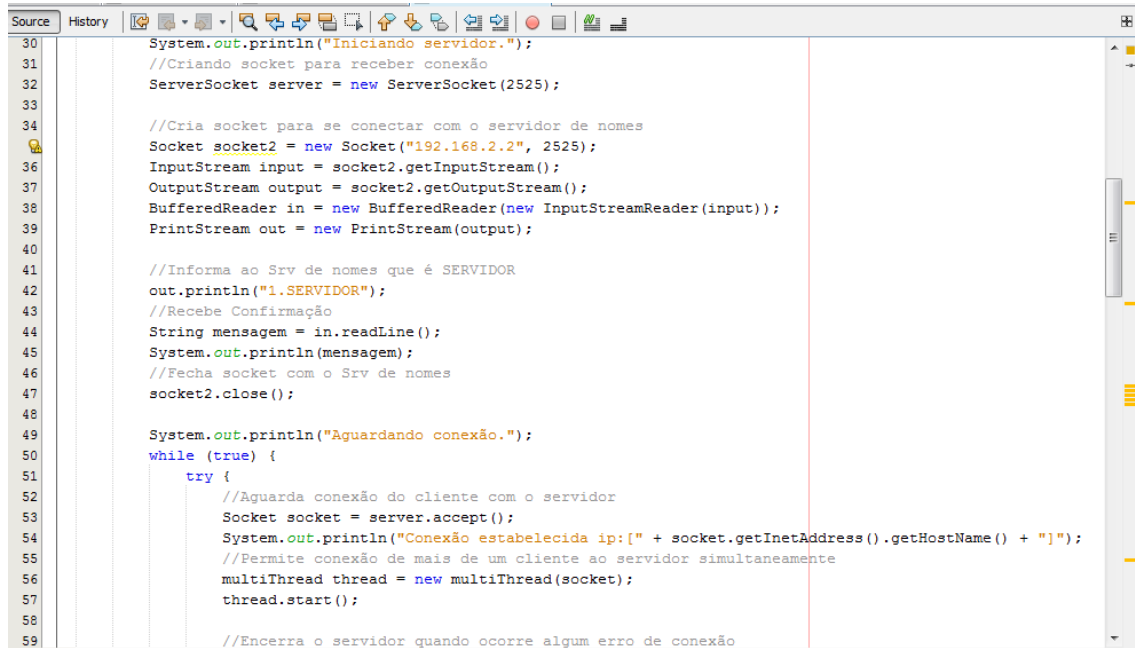
A screenshot of an IDE window showing a Java source file. The code is a while loop that handles communication with a server. It prompts the user to enter a number, reads a message, sends it to the server, and receives a response. The loop breaks when the message is "FIM". After the loop, it prints a closing message and closes the input stream, output stream, and the socket.

```
52     while (true) {
53
54         System.out.println("Digite um numero: ");
55         String mensagem = scanner.nextLine();
56
57         //Envia msg ao servidor
58         out.println(mensagem);
59         if ("FIM".equals(mensagem)) {
60             break;
61         }
62         //Recebe mensagem do servidor
63         mensagem = in.readLine();
64         System.out.println("Mensagem recebida do servidor: " + mensagem);
65     }
66
67     System.out.println("Encerrando conexão");
68     //Encerra conexão, fecha o socket
69     in.close();
70     out.close();
71     socket.close();
72 }
73
74 }
75
```

Os comandos *in.close* , *out.close* , *socket.close* , fecham o fluxo do inputstream e outputstream, e fecham o *socket* , finalizando a conexão com o servidor.

Nessa parte do código o cliente troca informações com o servidor ,e fecha a conexão se requisitado.

Servidor:



```
30 System.out.println("Iniciando servidor.");
31 //Criando socket para receber conexão
32 ServerSocket server = new ServerSocket(2525);
33
34 //Cria socket para se conectar com o servidor de nomes
35 Socket socket2 = new Socket("192.168.2.2", 2525);
36 InputStream input = socket2.getInputStream();
37 OutputStream output = socket2.getOutputStream();
38 BufferedReader in = new BufferedReader(new InputStreamReader(input));
39 PrintStream out = new PrintStream(output);
40
41 //Informa ao Srv de nomes que é SERVIDOR
42 out.println("1.SERVIDOR");
43 //Recebe Confirmação
44 String mensagem = in.readLine();
45 System.out.println(mensagem);
46 //Fecha socket com o Srv de nomes
47 socket2.close();
48
49 System.out.println("Aguardando conexão.");
50 while (true) {
51     try {
52         //Aguarda conexão do cliente com o servidor
53         Socket socket = server.accept();
54         System.out.println("Conexão estabelecida ip:[" + socket.getInetAddress().getHostName() + "]);
55         //Permite conexão de mais de um cliente ao servidor simultaneamente
56         MultiThread thread = new MultiThread(socket);
57         thread.start();
58
59         //Encerra o servidor quando ocorre algum erro de conexão
```

ServerSocket: Cria um socket servidor, vinculado à porta especificada para a comunicação com o cliente.

Server.accept: Aguarda uma conexão ser solicitada no socket, e estabelece conexão.

Server.close: Encerra o socket.

Observações: Os comandos de entrada e saída de dados utilizados no cliente, também foram utilizados no servidor. Além disso, foi utilizado o conceito de Threads, permitindo que o servidor se conecte com vários clientes simultaneamente.

Também foi criado um socket para o servidor se comunicar com o DNS, fazendo com que seja possível armazenar o IP do servidor, ao efetuar a comunicação. Após isso, esse socket é fechado, e o servidor fica aguardando algum cliente se conectar.

Servidor DNS

```
Source History
31 //Inicialização do Servidor
32 public static void main(String[] args) throws IOException {
33     System.out.println("Iniciando servidor.");
34     //Criando socket para receber conexão
35     ServerSocket server = new ServerSocket(2525);
36     //Lista para armazenar os IP's dos Servidores
37     ArrayList<String> lista = new ArrayList<>();
38
39     System.out.println("Aguardando conexão.");
40
41     cont = 0;
42     while (true) {
43         try {
44             //Aguarda conexão do cliente com o servidor
45             Socket socket = server.accept();
46             System.out.println("Conexão estabelecida ip:[" + socket.getInetAddress().getHostAddress() + "]");
47             //Permite conexão de mais de um cliente ao servidor simultaneamente
48             multiThread thread = new multiThread(socket, lista, cont);
49             thread.start();
50
51             //Encerra o servidor quando ocorre algum erro de conexão
52         } catch (Exception except) {
53             System.out.println("Encerrando servidor.");
54             server.close();
55             break;
56         }
57     }
58 }
59
60
```

```
Source History
87 //Recebe a mensagem enviada pelo cliente/servidor e armazena em "mensagem"
88 String mensagem = in.readLine().trim();
89
90 //Verifica se é servidor. Armazena o IP na ArrayList
91 if (mensagem.contains("1.SERVIDOR")) {
92     /*Verifica se o IP do servidor já está na Lista
93     Caso esteja, não faz nada.*/
94     if (lista.contains(socket.getInetAddress().getHostAddress())) {
95         break;
96     }
97     out.println(socket.getInetAddress().getHostAddress());
98     lista.add(socket.getInetAddress().getHostAddress());
99     System.out.println("Servidor salvo!");
100     break;
101 } /*Verifica se é Cliente. Envia o IP de um servidor para conexão,
102 de forma sequencial. O primeiro servidor a entrar na lista, será
103 o primeiro a receber cliente, e assim por diante.*/ else if (mensagem.contains("2.CLIENTE")) {
104
105     if (TCPServernome.cont == lista.size()) {
106         TCPServernome.cont = 0;
107     }
108     out.println(lista.get(TCPServernome.cont));
109     System.out.println("Cliente Conectado!");
110     TCPServernome.cont++;
111     break;
112 }
113
114 System.out.println("Encerrando conexão");
115 System.out.println("Cliente [" + socket.getInetAddress().getHostName() + "] finalizado. ");
116 //Encerra conexão com o cliente/servidor
```

No servidor DNS é criado uma ArrayList() com intuito de armazenar os IP's dos servidores que se conectarem. Quando o cliente se conecta, o servidor dns envia o ip contido na primeira posição da lista, para os proximos clientes ele envia os proximos servidores na lista, de forma sequencial.

Codigo completo

TCP.Server-nome

```
/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package tcpserver;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import java.util.ArrayList;

/**
 *
 * @author Innan
 */
public class TCPServernome {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    //Variavel global para contagem de servidores na Lista
    public static int cont;

    //Inicialização do Servidor
    public static void main(String[] args) throws IOException {
        System.out.println("Iniciando servidor.");
        //Criando socket para receber conexão
        ServerSocket server = new ServerSocket(2525);
        //Lista para armazenar os IP's dos Servidores
        ArrayList<String> lista = new ArrayList<>();

        System.out.println("Aguardando conexão.");

        cont = 0;
        while (true) {
            try {
                //Aguarda conexão do cliente com o servidor
                Socket socket = server.accept();
                System.out.println("Conexão estabelecida ip:[" +
socket.getInetAddress().getHostAddress() + "]);
                //Permite conexão de mais de um cliente ao servidor
                simultaneamente
                multiThread thread = new multiThread(socket, lista,
cont);
                thread.start();

                //Encerra o servidor quando ocorre algum erro de
conexão
            } catch (Exception except) {
                System.out.println("Encerrando servidor.");
                server.close();
                break;
            }
        }
    }
}
```

```

    }
    }
}

class multiThread extends Thread {

    private ArrayList<String> lista;
    private Socket socket;
    private InputStream input;
    private OutputStream output;
    private BufferedReader in;
    private PrintStream out;

    //Iniciliação das threads para conexões multi-cliente
    public multiThread(Socket socket, ArrayList lista, int cont)
    throws IOException {

        this.lista = lista;
        this.socket = socket;
        input = socket.getInputStream();
        output = socket.getOutputStream();
        in = new BufferedReader(new InputStreamReader(input));
        out = new PrintStream(output);

    }

    @Override
    public void run() {

        try {
            while (true) {
                //Recebe a mensagem enviada pelo cliente/servidor e
                //armazena em "mensagem"
                String mensagem = in.readLine().trim();

                //Verifica se é servidor. Armazena o IP na ArrayList
                if (mensagem.contains("1.SERVIDOR")) {
                    /*Verifica se o IP do servidor já está na Lista
                    Caso esteja, não faz nada.*/
                    if
                    (lista.contains(socket.getInetAddress().getHostAddress())) {
                        break;
                    }

                    out.println(socket.getInetAddress().getHostAddress());
                    lista.add(socket.getInetAddress().getHostAddress());
                    System.out.println("Servidor salvo!");
                    break;
                } /*Verifica se é Cliente. Envia o IP de um servidor
                para conexão,
                de forma sequencial. O primeiro servidor a entrar na
                lista, será
                o primeiro a receber cliente, e assim por diante.*/
                else if (mensagem.contains("2.CLIENTE")) {

                    if (TCPServernome.cont == lista.size()) {
                        TCPServernome.cont = 0;
                    }
                    out.println(lista.get(TCPServernome.cont));
                    System.out.println("Cliente Conectado!");
                    TCPServernome.cont++;
                    break;
                }
            }
            System.out.println("Encerrando conexão");
            System.out.println("Cliente [" +
            socket.getInetAddress().getHostName() + "] finalizado. ");
            //Encerra conexão com o cliente/servidor

```



```

        in.close();
        out.close();
        socket.close();
    } //Caso ocorra algum erro, exibe mensagem abaixo
    catch (Exception except) {
        System.out.println("FATAL ERROR, FAVOR SEARCH NO
GOOGLE.");
    }
}
}

```

TCPServer

```

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package tcpserver;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.ServerSocket;
import java.net.Socket;
import static tcpserver.TCPServer.fibo;

/**
 *
 * @author Innan
 */
public class TCPServer {

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    //Inicialização do servidor
    public static void main(String[] args) throws IOException {
        System.out.println("Iniciando servidor.");
        //Criando socket para receber conexão
        ServerSocket server = new ServerSocket(2525);

        //Cria socket para se conectar com o servidor de nomes
        Socket socket2 = new Socket("192.168.2.2", 2525);
        InputStream input = socket2.getInputStream();
        OutputStream output = socket2.getOutputStream();
        BufferedReader in = new BufferedReader(new
InputStreamReader(input));
        PrintStream out = new PrintStream(output);

        //Informa ao Srv de nomes que é SERVIDOR
        out.println("1.SERVIDOR");
        //Recebe Confirmação
        String mensagem = in.readLine();
        System.out.println(mensagem);
        //Fecha socket com o Srv de nomes
        socket2.close();

        System.out.println("Aguardando conexão.");
        while (true) {
            try {
                //Aguarda conexão do cliente com o servidor
                Socket socket = server.accept();
            }

```

```

        System.out.println("Conexão estabelecida ip:[" +
socket.getInetAddress().getHostName() + "]);
        //Permite conexão de mais de um cliente ao servidor
simultaneamente
        multiThread thread = new multiThread(socket);
        thread.start();

        //Encerra o servidor quando ocorre algum erro de
conexão
    } catch (Exception except) {
        System.out.println("Encerrando servidor.");
        server.close();
        break;
    }
}

//Função fibonacci
public static int fibo(int n) {
    if (n < 2) {
        return n;
    } else {
        return fibo(n - 1) + fibo(n - 2);
    }
}

}

class multiThread extends Thread {

    private Socket socket;
    private InputStream input;
    private OutputStream output;
    private BufferedReader in;
    private PrintStream out;

    //Inicilização das threads para conexões multi-cliente
    public multiThread(Socket socket) throws IOException {
        this.socket = socket;
        input = socket.getInputStream();
        output = socket.getOutputStream();

        in = new BufferedReader(new InputStreamReader(input));
        out = new PrintStream(output);
    }

    @Override
    public void run() {
        try {
            //Mantém conexão com o cliente até ser digitado "FIM"
            while (true) {
                //Recebe a mensagem enviada pelo cliente e armazena em
"mensagem"
                String mensagem = in.readLine().trim();

                System.out.println("Msg recebida do cliente [" +
socket.getInetAddress().getHostName() + "]: " + mensagem);
                //Verifica o que foi digitado, caso seja "FIM" encerra
conexão do cliente
                if ("FIM".equals(mensagem.toUpperCase())) {
                    break;
                }
                //Retorna o resultado da função fibonacci
                out.println(String.valueOf(fibo(Integer.valueOf(mensagem))));
                System.out.println("Msg enviada ao cliente [" +
socket.getInetAddress().getHostName() + "]: " +
fibo(Integer.valueOf(mensagem)));
            }
        }
    }
}

```

```

    }

    System.out.println("Encerrando conexão");
    System.out.println("Cliente [" +
socket.getInetAddress().getHostName() + "] finalizado. ");
    //Encerra conexão com o cliente
    in.close();
    out.close();
    socket.close();
} //Caso a mensagem digitada não seja um numero ou "FIM" exibe
uma mensagem de erro
catch (Exception except) {
    System.out.println("FATAL ERROR, FAVOR SEARCH NO
GOOGLE.");
}
}
}

```

TCPClient

```

/*
 * To change this license header, choose License Headers in Project
Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package tcpclient;

import java.io.BufferedReader;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.io.OutputStream;
import java.io.PrintStream;
import java.net.Socket;
import java.util.Scanner;

/**
 *
 * @author Innan
 */
public class TCPClient {

    public static void main(String[] args) throws Exception {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Iniciando cliente.");
        System.out.println("Digite o ip do servidor:");

        //Inicia o socket para conexão com o servidor de nomes
        Socket socket1 = new Socket(scanner.nextLine(), 2525);
        System.out.println("Conexão estabelecida.");

        //Recebimento e envio de mensagem na rede
        InputStream input1 = socket1.getInputStream();
        OutputStream output1 = socket1.getOutputStream();
        BufferedReader in1 = new BufferedReader(new
InputStreamReader(input1));
        PrintStream out1 = new PrintStream(output1);

        //Informa ao servidor de nomes que é CLIENTE, e recebe o IP de
um SERVIDOR
        out1.println("2.CLIENTE");
        String mensagem2 = in1.readLine();
        System.out.println(mensagem2);
        //Encerra conexão com Srv de nomes
        socket1.close();
        //Inicia conexão com o SERVIDOR informado pelo Srv de nomes
        Socket socket = new Socket(mensagem2, 2525);
    }
}

```

```

//Recebimento e envio de mensagem na rede
InputStream input = socket.getInputStream();
OutputStream output = socket.getOutputStream();
BufferedReader in = new BufferedReader(new
InputStreamReader(input));
PrintStream out = new PrintStream(output);

while (true) {

    System.out.println("Digite um numero: ");
    String mensagem = scanner.nextLine();

    //Envia msg ao servidor
    out.println(mensagem);
    if ("FIM".equals(mensagem)) {
        break;
    }
    //Recebe mensagem do servidor
    mensagem = in.readLine();
    System.out.println("Mensagem recebida do servidor: " +
mensagem);
}

    System.out.println("Encerrando conexão");
    //Encerra conexão, fecha o socket
    in.close();
    out.close();
    socket.close();
}
}

```

Testes

- Conexão entre servidor DNS e Servidor, os IP's foram guardados corretamente e a conexão estabelecida sem maiores erros
- Conexão entre cliente e servidor dns, cliente recebeu IP corretamente, e estabeleceu conexão com o servidor.
- Varios clientes solicitando conexão ao servidor DNS, e recebendo corretamente os IP's dos servidores, seguindo a ordem de preferencia
- Comunicação entre cliente servidor ocorrendo normalmente.

Conclusao

Neste presente trabalho abordamos a comunicação entre computadores através de uma rede , implementamos um servidor dns(servidor de nomes) que verifica os servidores disponiveis e estabelece a comunicação entre eles e o cliente .Aprendemos a estabelecer uma comunicação usando socket e ampliamos o nosso conhecimento sobre a ideia de servidor “DNS”.

Cumprimos o objetivo do trabalho, estabelecendo uma comunicação entre o cliente e o servidor , através do servidor “DNS”,e implementamos o uso de Multithreading , que permite que o servidor se comunique com vários clientes.

Este trabalho foi extremamente importante para o esclarecimento de duvidas em relação a comunicação em rede , alem de ampliar o nosso conhecimento em relação ao uso do servidor “DNS”,e o uso de Multithreading.

Bibliografias

- Oracle, Class ServerSocket. Disponível em - <https://docs.oracle.com/javase/7/docs/api/java/net/ServerSocket.html> - Acesso em 25/10/2016.
- Oracle, Socket. Disponível em - <https://docs.oracle.com/javase/7/docs/api/java/net/Socket.html> - Acesso em 25/10/16.
- Oracle, Class InputStreamReader. Disponível em - <https://docs.oracle.com/javase/7/docs/api/java/io/InputStreamReader.html> - Acesso em 25/10/16.
- Oracle, Class PrintStream. Disponível em - <https://docs.oracle.com/javase/7/docs/api/java/io/PrintStream.html> - Acesso em 25/10/16.
- Oracle, Class Thread. Disponível em - <https://docs.oracle.com/javase/7/docs/api/java/lang/Thread.html> - Acesso em 25/10/16.
- Oracle, Class ArrayList. Disponível em - <https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html> - Acesso em 13/02/2017