Socket em Java

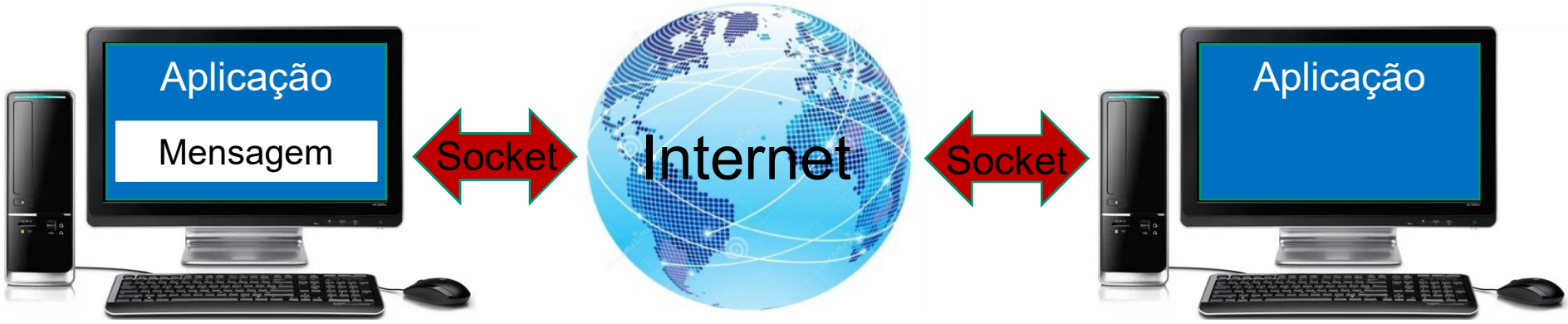
Prof. Rone Ilídio da Silva
DTECH-CAP-UFSJ

O que é socket?

- **Definição**

  Interface entre a Camada de Aplicação e a Camada de Transporte que permite comunicação entre dois processos que podem estar em máquinas distintas.

O que é socket?



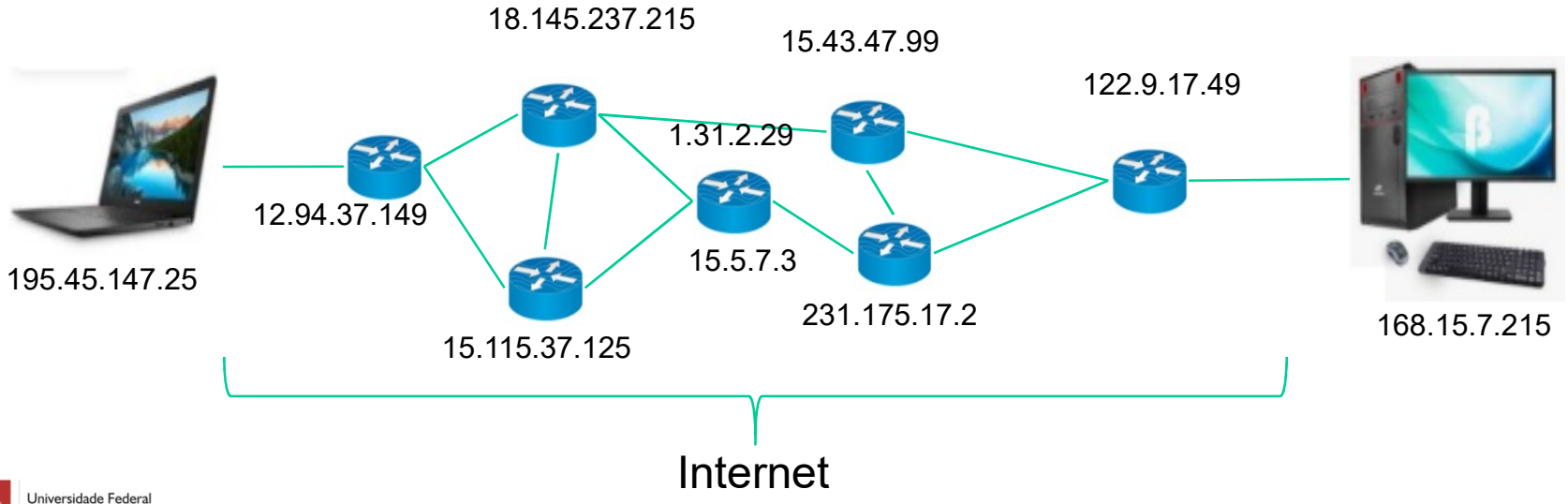
O que é socket?

Conceitos

- **Processo**
  - Entenda como um programa rodando em um dispositivo
- **Endereço IP**
  - Identificador associado a cada dispositivo conectado à Internet
- **Porta**
  - Número que identifica um socket em uma máquina

Conceitos

- **Endereço IP**



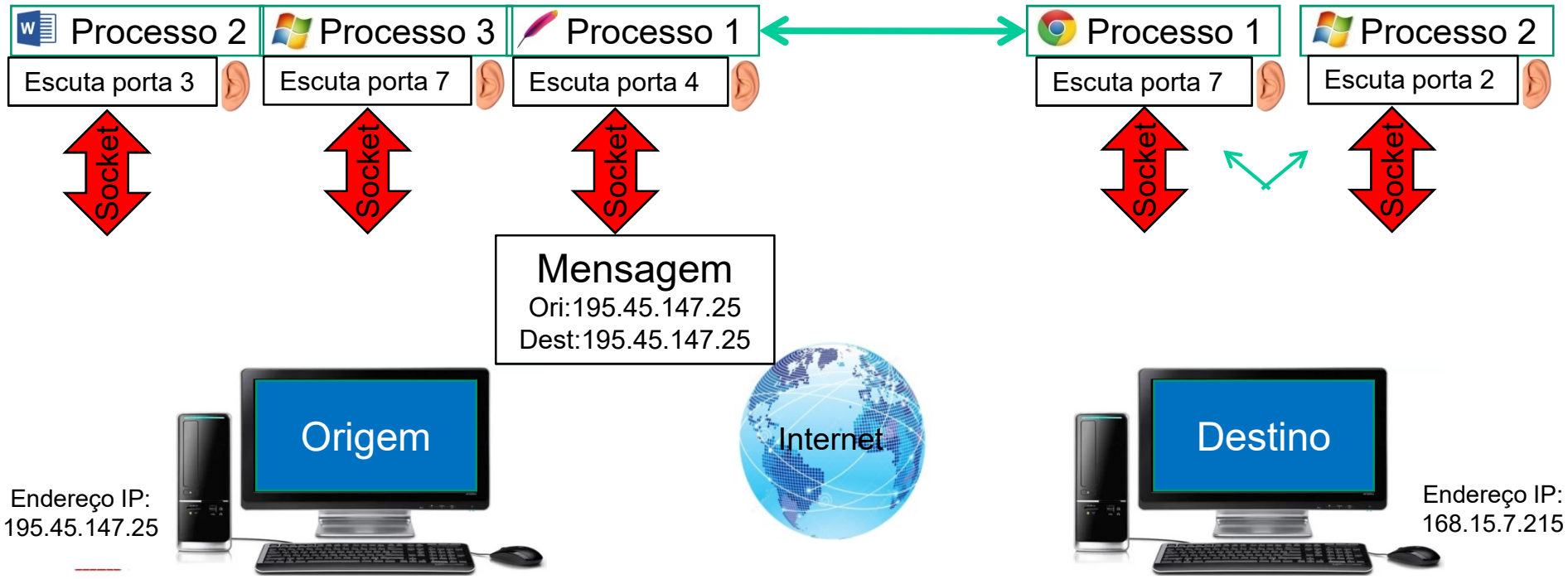
Endereço IP

- Número que identifica todos os computadors conectados à Internet

- Versões:
  - IPv4 → 170.163.13.101
  - IPv6 →2001:0DB8:AD1F:25E2:CADE:CAFE:F0CA:84C1

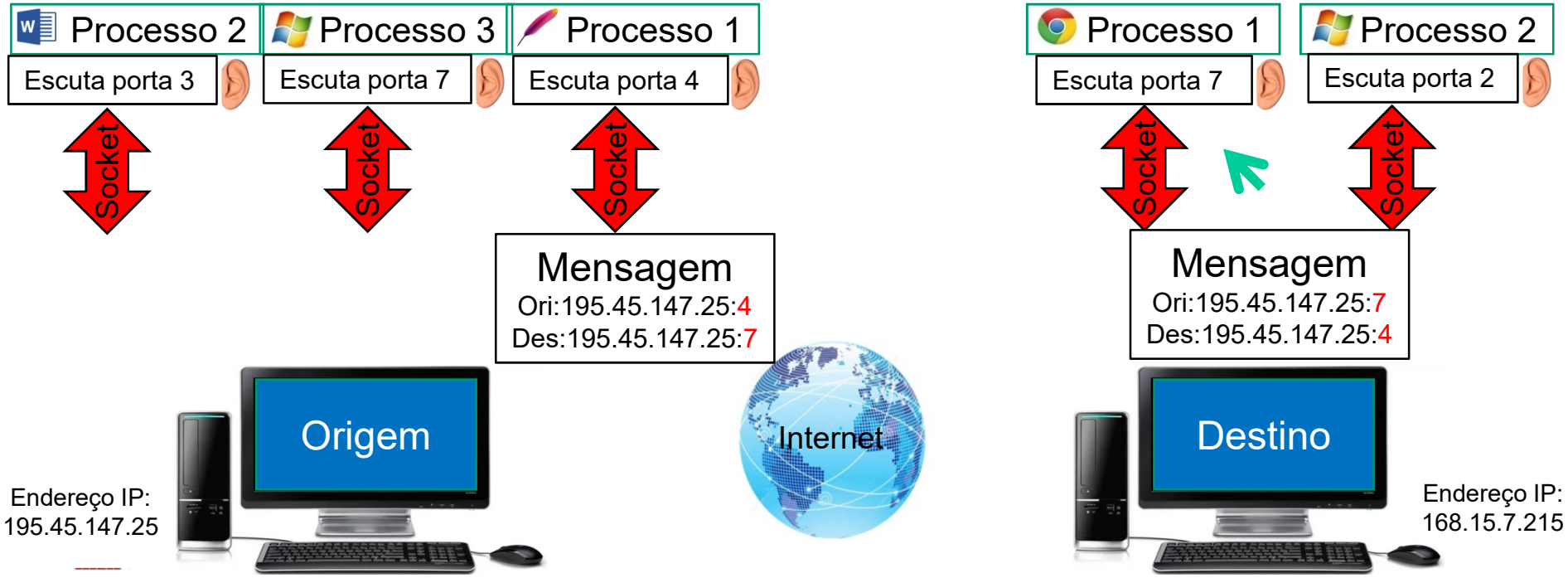
Conceitos

- **Porta**



Conceitos

- **Porta**



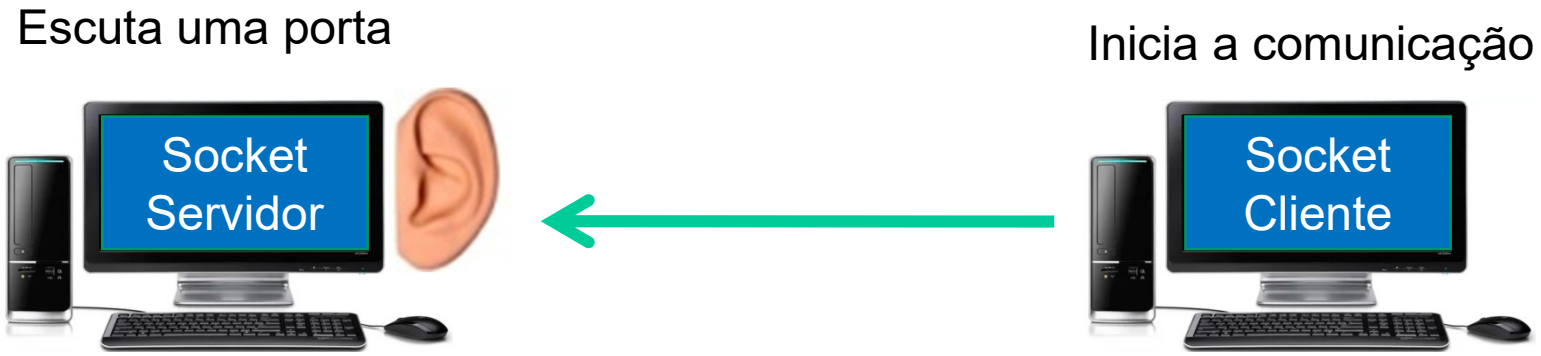
Portas

- Número de16 bits: 0 a 65535
  - 0 a 1023: portas reservadas
  - 1024 a 65535: portas disponíveis

- Exemplos de portas reservadas
  - TCP 80: HTTP
  - TCP 21: FTP

Sockets Clientes e Servidores

- Socket servidor: espera a conexão em uma porta
- Socket cliente: sempre inicia a comunicação



Tipos de Sockets

- Socket TCP
  - Transporte confiável
  - Cria uma conexão entre os processos

- Socket UDP
  - Transporte por melhor esforço
  - Sem conexão

Socket em

Socket UDP

UDPSocket - Classes Importantes

- **DatagramSocket**
  - Representa o socket UDP
  - Objetos dessa classe enviam e recebem mensagens
  - Um objeto dessa classe no cliente e outro no servidor serão responsáveis pela comunicação entre os processos
- **DatagramPacket**
  - Representa a mensagem que será enviada/recebida

- Essas classes estão no pacote java.net.*;

Classes do Exemplo

- **UDPServer**
  - Cria um servidor UDP que recebe as mensagens e exibe na tela
- **UDPClient**
  - Envia para o servidor mensagens fornecidas pelo usuário

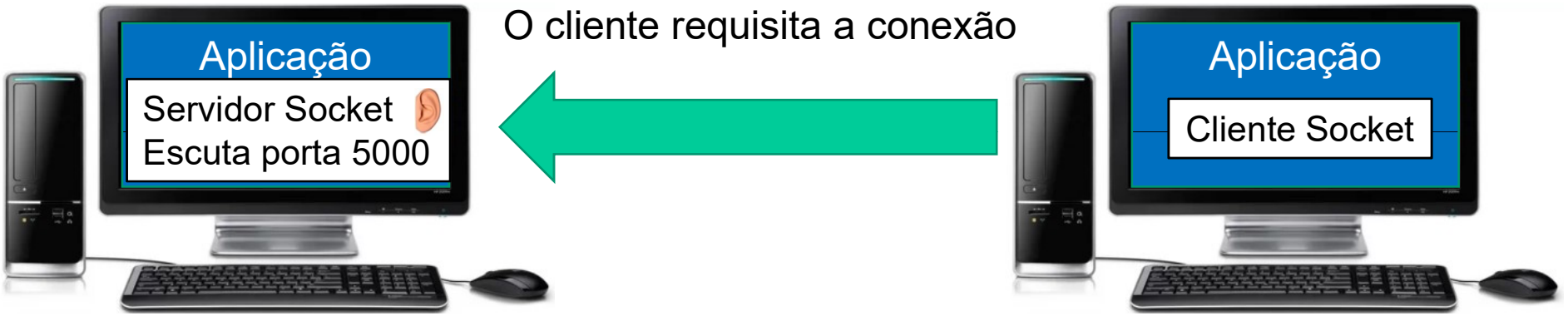
- Se o usuário digitar fim, tanto o servidor quanto o cliente são finalizados

```
import java.net.*;
class UDPServer {
  public static void main(String args[]) throws Exception {
    System.out.println("Servidor Iniciado...");
    byte[] receiveData = new byte[1024];
    DatagramSocket serverSocket = new DatagramSocket(5000);
    DatagramPacket receivePacket = new DatagramPacket(receiveData, receiveData.length);
    while (true) {
      serverSocket.receive(receivePacket);
      String sentence = new String(receivePacket.getData());
      if (sentence.substring(0,3).equals("fim")) break;
      InetAddress IPAddress = receivePacket.getAddress();
      System.out.println("Recebido de " + IPAddress.getHostAddress()+":" + sentence);
    }
    serverSocket.close();
  }
}
```

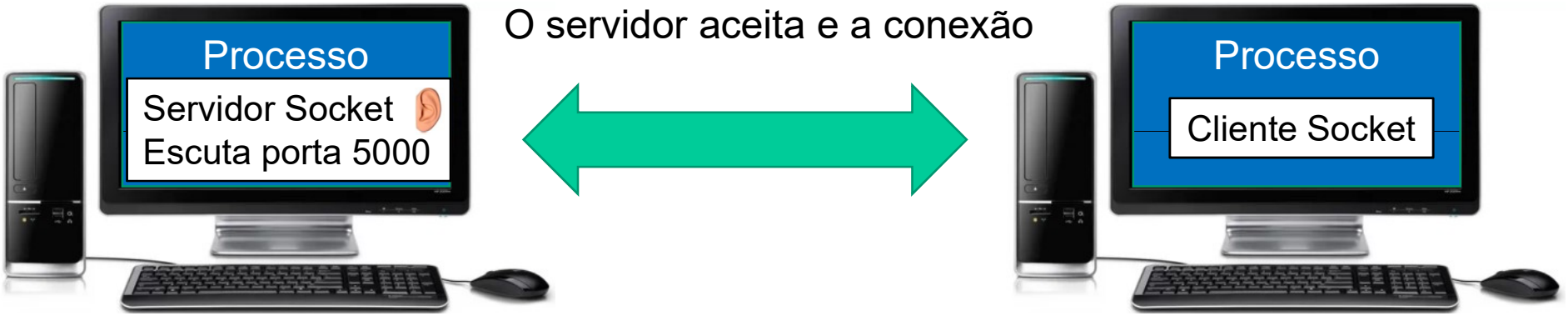
```
import java.net.*;
import javax.swing.*;
class UDPClient {
   public static void main(String args[]) throws Exception {
      DatagramSocket clientSocket = new DatagramSocket();
      byte[] sendData = new byte[1024];
      InetAddress IPAddress = InetAddress.getByName("localhost");
      while (true) {
         String sentence = JOptionPane.showInputDialog("Digite:");
         sendData = sentence.getBytes();
         DatagramPacket sendPacket = new DatagramPacket(sendData, sentence.length(),
IPAddress, 5000);
         clientSocket.send(sendPacket);
         if (sentence.equals("fim")) break;
      }
      clientSocket.close();
   }
}
```

Socket TCP

Exemplo de conexão



Exemplo de conexão



Class Importantes

- **ServerSocket**
  - Define objetos servidores, os quais abrirão um canal de comunicação e esperarão conexões externas
- **Socket**:
  - define objetos que formarão o link (um no cliente outro no servidor)
- Essas classes estão no pacote java.net.*;

Classes do Exemplo

- **TCPServer**
- Cria um servidor TCP
- Escuta a porta 5000
- Recebe as mensagens
- Exibe na tela
- Envia de volta para o cliente

- **TCPClient**
  - Inicia a conexão
  - Envia para o servidor mensagens fornecidas pelo usuário
  - Exibe mensagens recebidas

- Se o usuário digitar fim, tanto o servidor quanto o cliente são finalizados

```
import java.net.*;import java.io.*;
public class TCPServer {
        public static void main(String args[]) throws IOException {
                System.out.println("Servidor iniciado ...");
                ServerSocket socketServidor = new ServerSocket(5000);
                Socket socketCliente = socketServidor.accept();
                PrintWriter out = new PrintWriter(socketCliente.getOutputStream(), true);
                BufferedReader in = new BufferedReader(
                                        new InputStreamReader(socketCliente.getInputStream()));
                String entrada;
                while ((entrada = in.readLine()) != null) {
                        System.out.println("Recebido:" + entrada);
                        out.println(entrada);
                        if (entrada.equals("fim"))
                                break;
                }
                out.close(); in.close();
                socketServidor.close(); socketCliente.close();
        }
}
```

```
import java.io.*;import java.net.*;import javax.swing.*;
public class TCPClient {
  public static void main(String args[]) throws IOException {
    Socket socket = new Socket("localhost", 5000);
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    String entradaservidor = "";
    String entradausuario =  JOptionPane.showInputDialog("Digite:");
    while (!entradausuario.equals("fim")) {
      out.println(entradausuario);
      entradaservidor = in.readLine();
      entradausuario = JOptionPane.showInputDialog("Chegou: " +
                    entradaservidor + "\nDigite para enviar :");
    }
    out.close(); in.close(); socket.close();
  }
}
```

Introdução à Programação Multithread

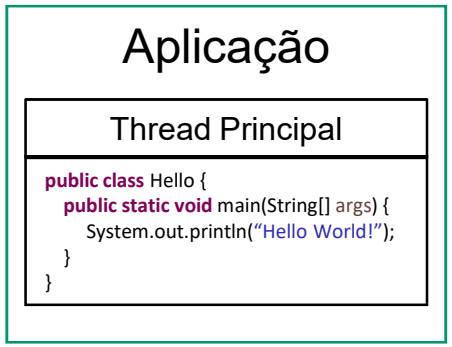
UFSJ Universidade Federal de São João del-Rei

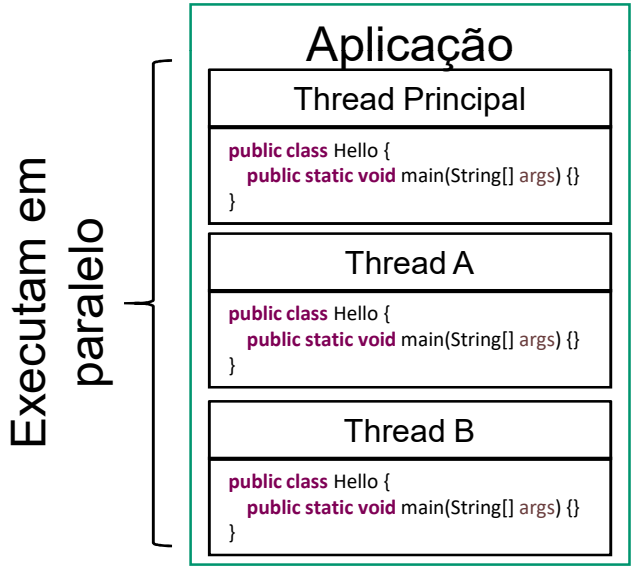
Introdução à Programação Multithread

- Processo
  - Contem um conjunto de execução básico completo com uma área privada de memória
  - A grosso modo: um programa em execução
- Thread
  - Partes de um processo que executam de forma paralela
  - Exemplo:
    - Navegador baixa uma página enquanto exibe o resultado do que já foi baixado e os gifs animados

Introdução à Programação Multithread

- Todo programa possui pelo menos uma thread: Thread Principal
- A Thread Principal pode chamar quantas threads forem necessárias

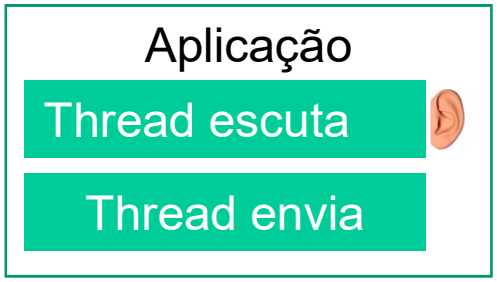




Introdução à Programação Multithread

- Qual a utilidade de multithread com sockets?

  → Uma thread envia enquanto outra escuta



Classe Thread

- Uma classe filha de Thread deve possuir o método run()
- O método run() será executado em outra thread
- Para executar uma thread bastar chamar o método star() o objeto filho de Thread

- Exemplo:
  - Classe ThreadPrincipal → exibe a palavra “Principal”
  - Classe ThreadParalelo → exibe a palavra “Paralelo”

Classe ThreadPrincipal

```
public class ThreadPrincipal{
      public static void main(String args[]){
         ThreadParalelo a = new ThreadParalelo("#Paralelo#");
         a.start();
         for(int i = 0; i<1000; i++){
            System.out.println(i+":" +"&Principal&");
         }
   }
}
```

Classe ThreadParalelo

```
public class ThreadParalelo extends Thread{
    String texto;
    public ThreadParalelo(String texto){
        this.texto = texto;
    }
    public void run(){
        for(int i = 0; i<1000; i++){
            System.out.println(i+":" +texto);
        }
    }
}
```

Socket e Thread

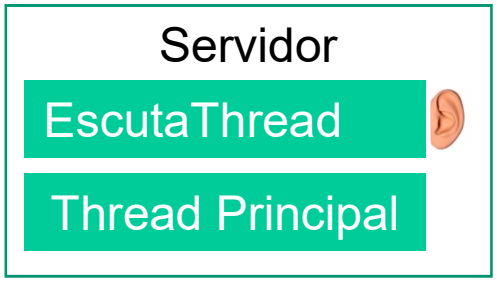
Socket e Thread

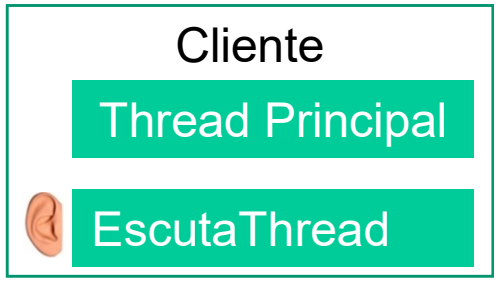
- Exemplo: chat

- Formado por três classes:
  - Servidor: possui o serverSocket
  - Cliente: possui o cliente socket
  - EscutaThread: thread que escuta as mensagens recebidas e exibe na tela

Socket e Thread

- Tanto o Servidor quanto o cliente possuem um objeto do tipo EscutaThread, com isso ambos podem enviar e receber mensagens





Servidor

```
import java.io.*; import javax.swing.*;import java.net.*;
public class Servidor {
  public static void main(String args[]) throws IOException {
     System.out.println("Servidor iniciado ...");
     ServerSocket socketServidor = new ServerSocket(5000);
     Socket socketCliente = socketServidor.accept();
     PrintWriter out = new PrintWriter(socketCliente.getOutputStream(), true);
     BufferedReader in = new BufferedReader(new InputStreamReader(socketCliente.getInputStream()));
     EscutaThread escuta = new EscutaThread(in);
     escuta.start();
      String entrada;
      while  (true){
        entrada = JOptionPane.showInputDialog(null, "Mensagem para o cliente", "Servidor",
                                                          JOptionPane.INFORMATION_MESSAGE);
        System.out.println("Servidor:" + entrada);
         out.println(entrada);
     }
   }
}
```

Cliente

```
import java.io.*; import java.net.Socket; import javax.swing.JOptionPane;
public class Cliente {
  public static void main(String args[]) throws IOException {
    System.out.println("Cliente iniciado ...");
    Socket socket = new Socket("localhost", 5000);
    PrintWriter out = new PrintWriter(socket.getOutputStream(), true);
    BufferedReader in = new BufferedReader(new InputStreamReader(socket.getInputStream()));
    EscutaThread escuta = new EscutaThread(in);
    escuta.start();
    String entrada = "";
    while (true) {
      entrada = JOptionPane.showInputDialog(null, "Mensagem para o servidor", "Cliente",
          JOptionPane.INFORMATION_MESSAGE);
      out.println(entrada);
      System.out.println("Cliente: " + entrada);
    }
  }
}
```

EscutaThread

```
import java.io.*;
public class EscutaThread extends Thread{
  BufferedReader in;
  public EscutaThread(BufferedReader in) {
    this.in = in;
  }
  public void run(){
    String entrada;
    try {
      while ((entrada = in.readLine()) != null) {
        System.out.println("Recebi:" + entrada);
      }
    } catch (IOException e) {}
  }
}
```