Desenvolvimento de Aplicações Distribuídas

Aula 5: Comunicação entre processos remotos/objetos distribuídos

Prof. Dr. Fábio Favarim
favarim@utfpr.edu.br

Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco

20 de Agosto de 2018

Objetivos da Aula

- Entender os princípios da comunicação entre processos remotos
- Sincronização entre processos
- Sockets: UDP e TCP
- Estudo de Caso: API Java para Sockets UDP

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

Princípios da Comunicação Inter-Processos Remotos

Comunicação através da troca de mensagens

Toda comunicação entre dois processos remotos somente ocorre através da **troca de mensagens via rede**. A troca de mensagem entre dois pares de processos pode ser implementado por duas operações básicas: send e receive.

- **send:** processo invoca para enviar mensagem (sequencia de bytes)
- **receive:** processo invoca para receber mensagem

Sincronização

- A comunicação entre dois processos envolve sempre um mecanismo de sincronização.
- Envolve saber quando enviar (send) ou receber (receive) dados
- Dois tipos: comunicação síncrona x comunicação assíncrona

Princípios da Comunicação Inter-Processos Remotos

Comunicação Síncrona

Transmissor e receptor sincronizam a cada mensagem.

- Envio (send): o processo (ou thread) emissor é bloqueado até que a recepção correspondente seja realizada
- Recebimento (receive): o processo (ou thread) receptor fica bloqueado até a recepção correspondente ser realizada

Send e Receive são bloqueantes.

Comunicação Assíncrona

- Envio: o processo envia a mensagem e não fica aguardando após invocar **send**;
- Recebimento (bloqueante): o processo fica **bloqueado** até a recepção ser realizada
- Recebimento (não-bloqueante): se não tem nada para receber (na fila), o processo continua seu processamento

Princípios da Comunicação Inter-Processos Remotos

Para onde enviar a mensagem???

Para saber onde enviar uma mensagem a um processo é preciso saber a localização:

- Endereço IP: permite que a mensagem chegue até o host de destino
- Porta: permite que no host a mensagem seja entregue ao processo correto

Princípios da Comunicação Inter-Processos Remotos

Para que dois processos remotos possam se entender, ambos devem usar o mesmo **protocolo**.

Protocolos

protocolos definem o **formato**, **ordem** das mensagens enviadas e recebidas e **ações** tomadas no envio ou recepção de mensagens.

Sockets

Aula 5: Comunicação entre processos remotos/objetos distribuídos

Prof. Dr. Fábio Favarim
favarim@utfpr.edu.br

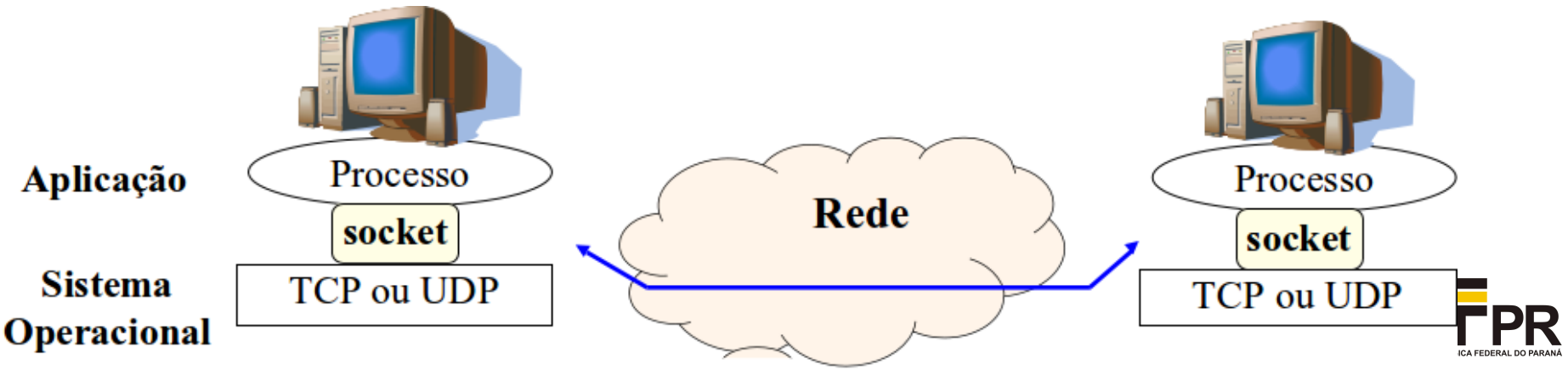
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco

20 de Agosto de 2018

Sockets

Sockets

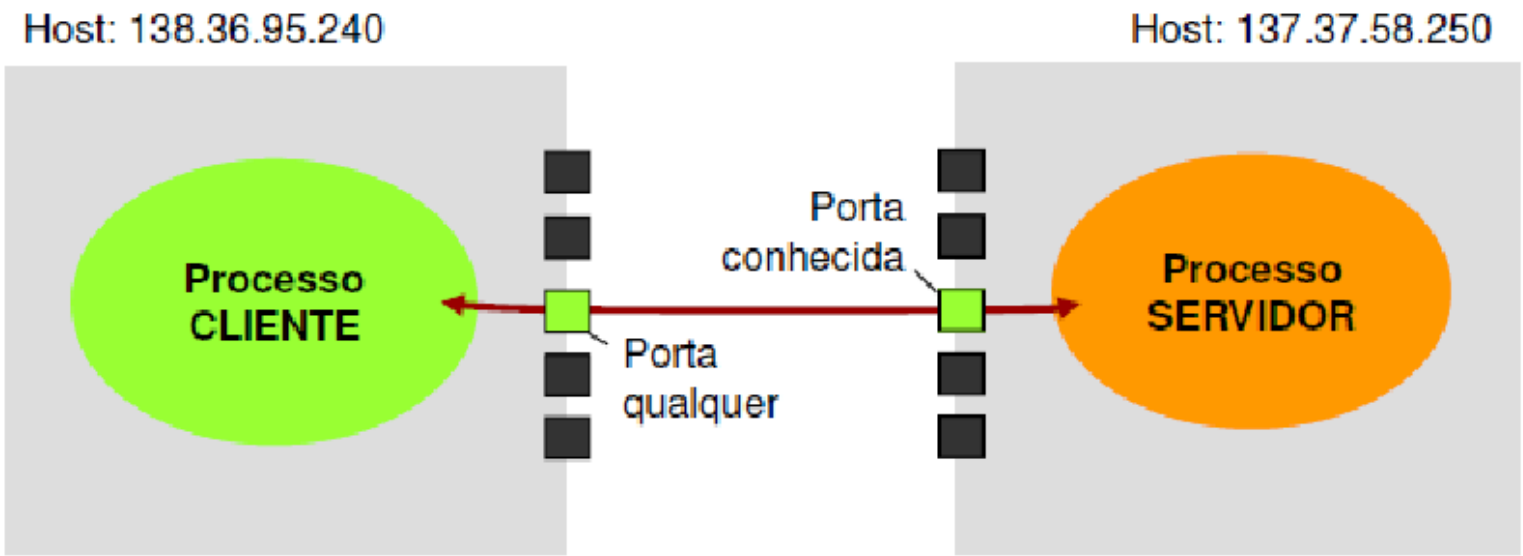
- É a interface provida pela camada de transporte para comunicação via rede.
- **Somente** através de **sockets** é possível enviar uma mensagem via rede.
- **Qualquer tecnologia** que troca mensagens via rede, por baixo **sempre** usa sockets.
- API em C criada em 1983 no 4.2 BSD UNIX, é padrão em todos S.O.



Sockets

Os sockets adotam por padrão o paradigma cliente/servidor.

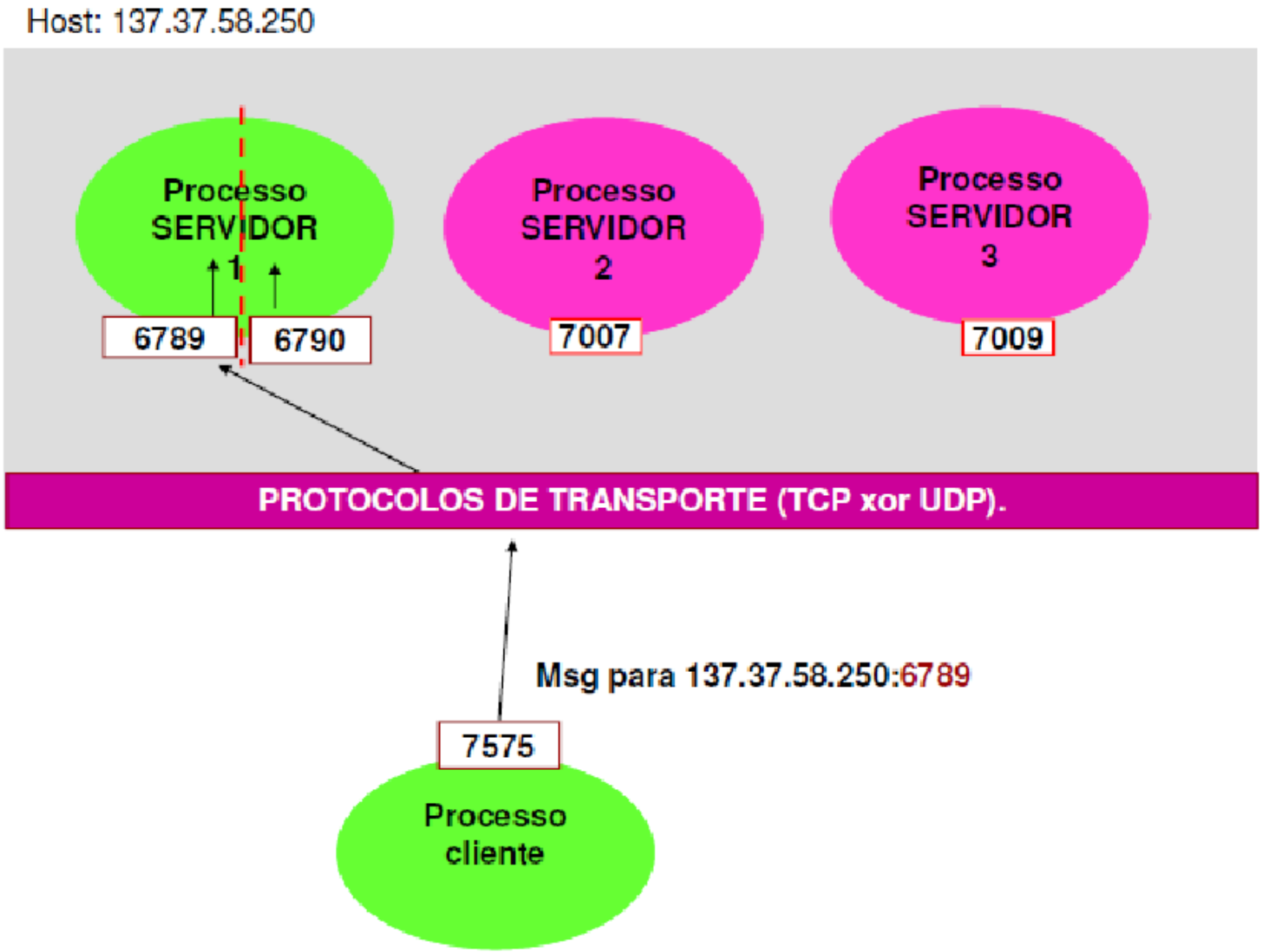
- **Servidor:** cria o socket em um **porta conhecida** e fica na escuta a espera de mensagens dos clientes
- **Cliente:** cria o socket em um **porta qualquer** e envia mensagens através de seu socket para o servidor



Socket = IP + Porta

Sockets

Um socket é identificado pelo par - **IP:PORTA**



Sockets: Modos de Operação

TCP (*Transfer Control Protocol*): Orientado a Conexão

- As mensagens são enviadas através de um canal de comunicação.
- Fluxo contínuo – são trocadas diversas mensagens consecutivas e estão relacionadas entre si – ex: vídeo

UDP (*User Datagram Protocol*): Não orientado a Conexão

- Mensagens de tamanho fixo (datagramas) são transmitidas **individualmente** para destinos especificos.
- Cada mensagem é tratada como uma unidade completa de informação

Sockets: UDP - User Datagram Protocol

UDP: Sem conexão – Não confiável

- Mensagens de tamanho fixo (datagramas) são transmitidas **individualmente** para destinos especificos.
- **não garante a entrega** dos datagramas (**confiabilidade**)
- **não garante a ordem da entrega** (**ordenamento**)
- endereço destino é especificado em cada datagrama
- não há estabelecimento da conexão
- Ex: Correspondência via serviço postal, VoIP, DNS

Sockets UDP: Sem conexão – Não confiável

- **Analogia aos Correios**
- Envio de cartas destinadas a um endereço;
- O endereço da origem/destinho é adicionado **a cada carta** enviada;
- A maioria das cartas chega mas algumas podem ser perdidas no caminho;
- As cartas provavelmente chegarão na ordem em que foram enviadas mas não há garantias;
  - Quanto mais distante se estiver do destinatário,maior a probabilidade das cartas chegarem fora de ordem ou serem perdidas;
- É possível numerar as cartas e o destinatário lhe escrever solicitando aquelas que não recebeu.

Sockets: TCP - Transfer Control Protocol

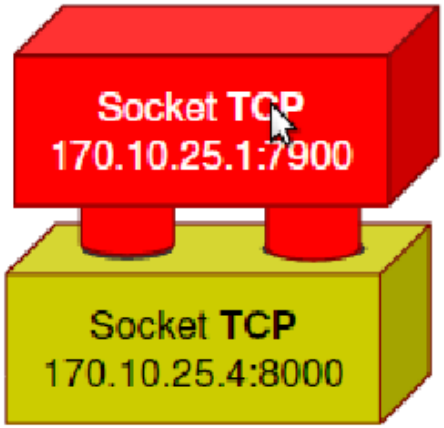
TCP: Orientado a conexão – Confiável

- Uma conexão deve ser estabelecida antes da transmissão dos dados;
- A conexão deve ser encerrada após a transmissão dos dados;
- Dados são enviados em fluxo contínuo (stream) na conexão (canal)
- Não são enviados datagrama a datagrama, mas segmentados pelo TCP automaticamente em segmentos
  - Basta gravar no stream para enviar uma mensagem
  - Basta ler do stream para receber uma mensagem
- É **garantida a entrega** dos segmentos
- É **garantida a ordem de entrega** dos segmentos

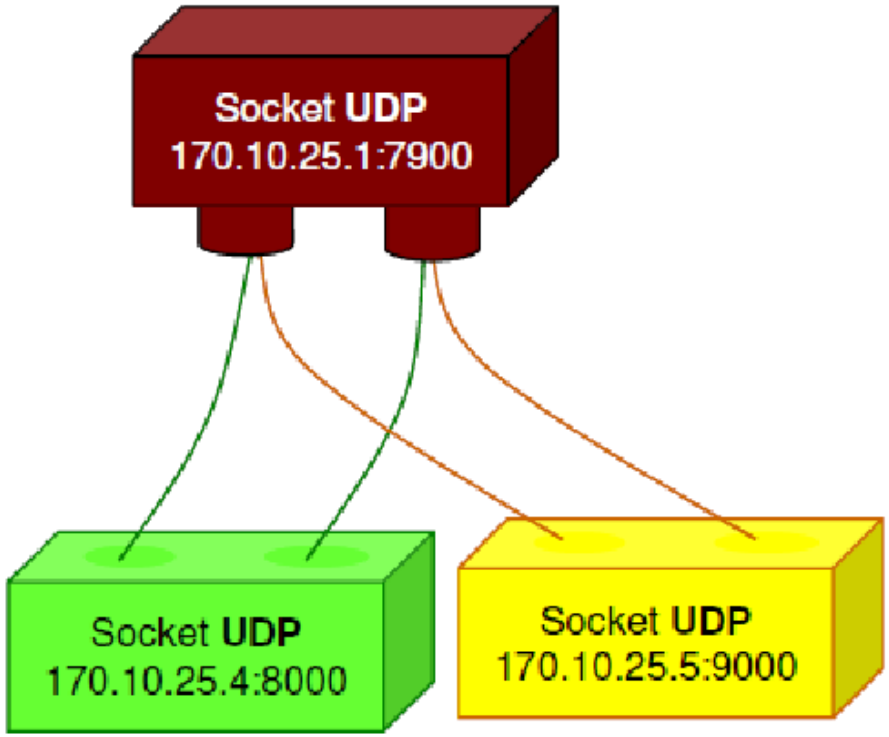
Sockets TCP: Orientado a conexão – Confiável

- **Analogia ao Sistema Telefônico**
- É necessário discar um número, o outro lado atende e uma **conexão é estabelecida**;
- Cada lado da conversa escuta as palavras na ordem em que foram emitidas (confiabilidade/ordenamento);
- Se não há resposta ou o telefone está ocupado é rapidamente detectado.
- O endereço (número do telefone) do destino somente é preciso ser especificado uma única vez.

Sockets: TCP x UDP



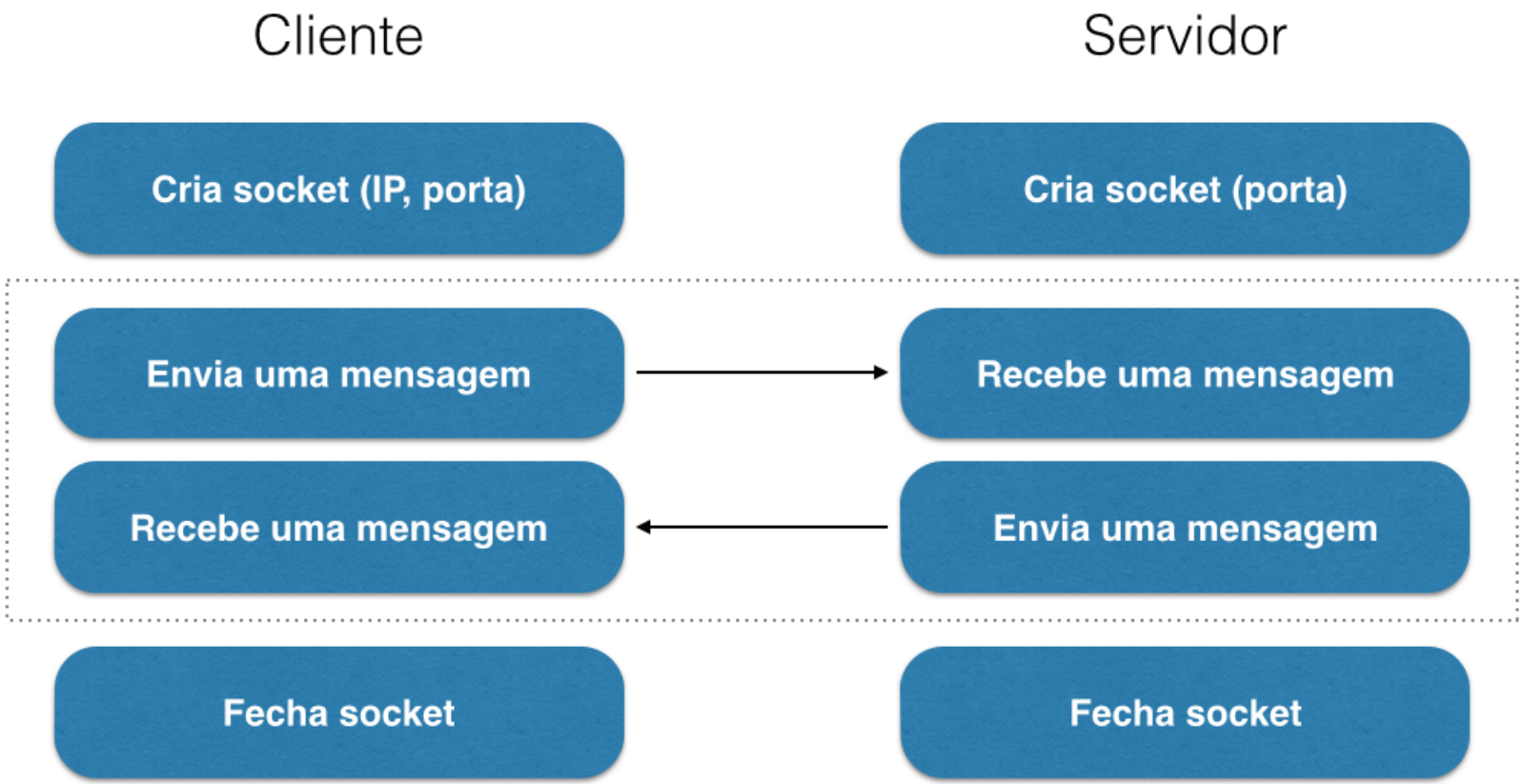
TCP:CONECTADO
Uma origem e um destino



UDP:Promíscuo
Várias origens e vários destinos

Passos para comunicação usando Socket UDP

- Passo 1: criar socket
- Passos 2/3: realizar a comunicação (enviando e recebendo datagramas usando send e receive)
- Passo 4: fechar socket



Programação com Sockets UDP em Java

Pacote **java.net**, principais classes:

- **UDP:** DatagramPacket e DatagramSocket;
- Outras: obtenção endereço IP e porta de um pacote, entre outras coisas.

Para enviar dados, insere-se os mesmos em um **DatagramPacket**, enviando-o através do **DatagramSocket**.

Sockets UDP (Java) - Classe DatagramSocket

- **DatagramSocket(int port, InetAddress addr)** - cria um socket datagrama, ligado a um endereço local (addr, port) específico.
- **Métodos:**
  - **void receive(DatagramPacket p)**: recebe um pacote de datagrama deste socket.
  - **void send(DatagramPacket p)**: envia um pacote de datagrama deste socket.
  - **InetAddress getLocalAddress()**: retorna o endereço **local** do socket
  - **int getLocalPort()**: retorna o número de porta **local** do socket
  - **void close()**: fecha o socket deste datagrama.

Sockets UDP (Java) - classe DatagramPacket

- **DatagramPacket(byte[] dados, int tamanho, InetAddress endereco, int porta)**: constrói um datagrama para enviar **dados** de **tamanho** para uma máquina em um **endereço** e **porta** específicos.
- **DatagramPacket(byte[] buf, int tamanho)**: constrói um datagrama para receber **dados** com determinado **tamanho**.
- **Métodos:**
  - **InetAddress getAddress()**: retorna o endereço da origem do datagrama;
  - **int getPort()**: retorna o número de porta da origem do datagrama;
  - **byte[] getData()**: retorna os dados contidos no datagrama;
  - **int getLength()**: retorna o tamanho dos dados do datagrama.

Sockets UDP (Java) - Comandos Básicos

Criar Socket

```
1    DatagramSocket socket = new DatagramSocket(porta);
```

Fechar Socket

```
2    socket.close();
```

Enviar Datagrama

```
3    socket.send(dgEnvio);
```

Receber Datagrama

```
4    socket.receive(dgRec);
```

Sockets UDP (Java) - Comandos Básicos

Criar um datagrama para enviar dados

```
5       dgEnvio = new DatagramPacket(msg, msg.length, endereco, porta);
```

Criar um datagrama para receber dados

```
6       dgRec = new DatagramPacket(buffer, buffer.length);
```

Importante

```
7           endereco = InetAddress.getByName(``172.29.100.100'')
8           msg e buffer = array de bytes(byte[])
```

Comando Extras

```
9           datagrama.getAddress(); //obtem IP de origem
10          datagrama.getPort());  //obtem porta de origem
```

Sockets UDP (Java) - Programa Cliente

```
11      //Dados do servidor
12      InetAddress endereco = InetAddress.getByName("127.0.0.1");
13      int porta = 4321;
14
15      //Passo 1: criar socket
16      DatagramSocket socket = new DatagramSocket();
17
18      //Passo 2: realizar a comunicacao com o servidor (ficar em loop???)
19      String mensagem = "Alo galera maneira";
20      byte[] msg = mensagem.getBytes();
21      DatagramPacket dgEnvio= new DatagramPacket(msg,msg.length,endereco,
            porta);
22      socket.send(dgEnvio);
23
24      //recebimento dos dados do servidor - fica bloqueado no receive
25      DatagramPacket dgRec = new DatagramPacket(new byte[1024],1024);
26      socket.receive(dgRec);
27
28      String msgRecebida = new String(dgRec.getData());
29
30      //Passo 3: fecha socket
31      socket.close();
```

Sockets UDP (Java) - Programa Servidor

```
32      //Passo 1: criar socket em porta especifica.
33      DatagramSocket socket = new DatagramSocket(4321);
34
35      //Passo 2: realizar a comunicacao com o cliente (ficar em loop??)
36      // cria datagrama para receber requisicao do cliente
37      DatagramPacket dgRec = new DatagramPacket(new byte[1024], 1024);
38      socket.receive(dgRec);
39
40      String mensagem = new String(dgRec.getData()).trim();
41      System.out.println("Recebeu " + mensagem + " de "+
42                         dgRec.getAddress() + ":" + dgRec.getPort());
43
44      // envia resposta
45      DatagramPacket dgEnvio = new DatagramPacket(dgRec.getData(), gdRec.
            getLength(),
46      dgRec.getAddress(), dgRec.getPort());
47      socket.send(dgEnvio);
48
49      // Passo 3: fecha o socket (somente quando terminar o servidor)
50      socket.close();
```

Exercícios

- Resolução da Prática 2 (exercícios UDP)
- Códigos apresentados nos slides estão disponíveis no Moodle
- Agora: Alo mundo com Sockets UDP!!!

Próxima aula

- Estudo de Caso: Sockets UDP em C