Java Sockets

Súmula

- Diferenças entre TCP e UDP
- Comunicação utilizando *Streams* – TCP
- Comunicação utilizando *Datagramas* - UDP

Diferença entre TCP e UDP

TCP (Transmission Control Protocol)

- Orientado a conexão
- Confiável
- Stream
- Controle de fluxo
- Mensagens Ordenadas
- Mais lento

Diferença entre TCP e UDP

UDP (User DatagramProtocol)

- Orientado a datagrama
- Não é confiável
- Datagramas (pacotes)
- Sem controle de fluxo
- Sem garantia de ordem ou de chegada
- Menor *Overhead*
- Mais apropriado a *broadcast*

Sockets TCP

Conceitos básicos de sockets em TCP/IP:

CLIENTE
- inicia a conexão, ativo
- conhece servidor e seu endereço/nome

SERVIDOR
- atende diversos clientes
- espera um pedido de conexão de um cliente, passivo

CONEXÃO
- um cliente e servidor devem estabelecer um canal próprio

Sockets TCP

COMUNICAÇÃO

- após a conexão, qualquer um pode inicia-la
- canal é bidirecional
- assíncrona bloqueante em geral
  - send:
    não espera receive
    espera passagem dos dados para subsistema de comunicação
  - receive:
    bloqueia até que haja dados a serem lidos

Sockets TCP

STREAM

- receive (leitura dos dados recebidos)

qualquer parte dos dados já recebidos pelo subsistema na máquina destino

CONTROLE DE FLUXO

- mensagens enviadas a diversas conexões de um processo ficam em fluxos distintos

CONFIÁVEL

- mensagens não são perdidas, nem duplicadas
- integridade do conteúdo da mensagem é preservado

Sockets TCP

SERVIDOR

- possui (cria) um socket associado a uma porta
- espera pedidos de conexões de clientes
- conexão aceita
  - novo socket é criado para a conexão em nova porta
  - permite aceitar outras conexões na mesma porta enquanto conexões anteriores estejam abertas

Sockets TCP

CLIENTE

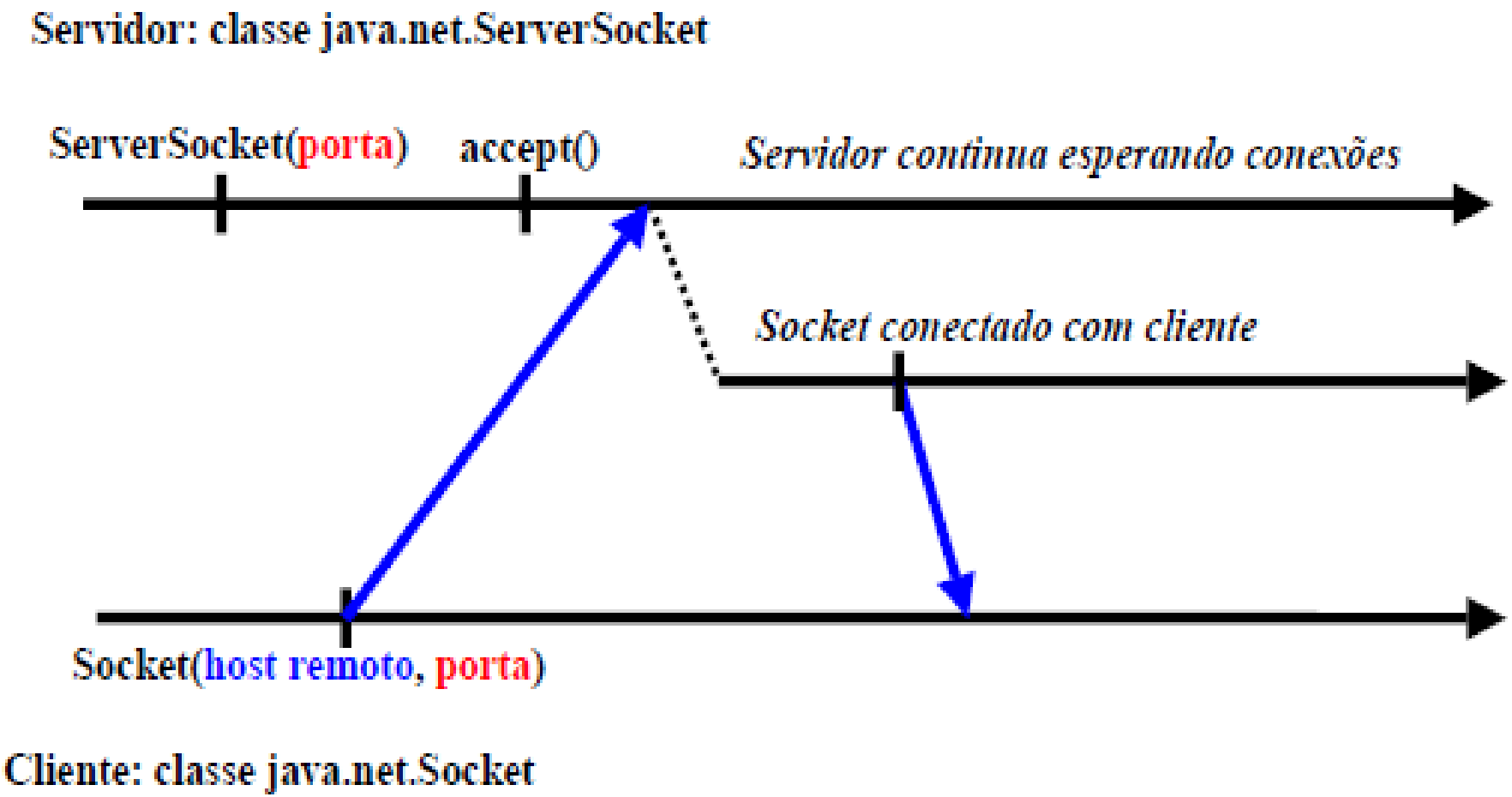
- conhece hostname (IP) da máquina servidora
- conhece porta do programa servidor
- pede conexão
- se conexão aceita
  - um socket é criado
  - associado a uma porta na máquina cliente

Sockets TCP e Classes Java

Classes sockets TCP em Java

- no pacote java.net
- escondem detalhes dependentes de plataforma
- API mais simples para fase de conexão
- API com muitas alternativas para fase de send/receive
- código (mais?) portável
- classes
  - ServerSocket
    - usada por servidores
  - Socket
    - usada por clientes e servidores

Sockets TCP e Classes Java



Sockets TCP e Classes Java

Primeiro

- servidor cria *ServerSocket*
- espera por mensagens em uma determinada porta (método *accept( )*)

Segundo

- cliente cria *Socket*, conectando com o servidor

Servidor

- pode criar uma nova *Thread* para atender cliente
- continua aceitando novas conexõ es na mesma porta

Código para Criar Conexão

SERVIDOR

```
(...) ServerSocket s = new
   ServerSocket(8189);

while (true){

 Socket conexao =
   s.accept();

 /* Disparar uma thread que
   faç  a algo, passando
   conexã o como parâ  metro
   */

 (...)

}
```

CLIENTE

```
(...)Socket s;

try{

s =new
   Socket(“  poncho”  ,8189);

}catch(Exception e)
   {/*Erro*/

   System.exit(0);

}

/* Socket conectado */
```

Como enviar e receber mensagens

- A classe *Socket* não tem *send( )* e *receive( )*

- Os métodos *getInputStream( )* e *getOutputStream( )*
  - retornam objetos “fluxos de bytes” (*streams*)
  - que podem ser manipulados como se viessem de arquivos
  - esses métodos pertencem às classes InputStream e OutputStream, e suas derivadas
- Várias classes e métodos para leitura e escrita em *streams*
  - podendo transmitir desde bytes até objetos
- Para fechar uma conexã o, utilizar *close( )*

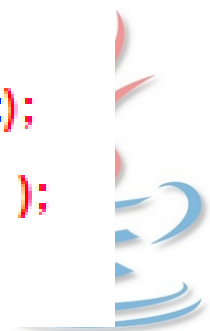
Como enviar e receber mensagens

RECEBER

```
(...)InputStream input;
try {  input = s.getInputStream();
} catch (IOException e) {(...)}
ObjectInputStream objInput;
try {
objInput = new
   ObjectInputStream(input);
String line = (String)
   objInput.readObject( );
}catch (Exception e){(...)}
```

ENVIAR

```
(...) OutputStream output;
try {  output =
   s.getOutputStream();
} catch (IOException e) {(...)}
ObjectOutputStream objOutput;
try {
objOutput = new
   ObjectOutputStream(output);
objOutput.writeObject( " Olá" );
}catch (Exception e){(...)}
```



Uso de Streams

- **Um socket (conexão) pode ser usado ao mesmo tempo para**
  - input stream
  - output stream

- **Mas os streams são ou de input ou de output**
- **Após a conexão**
  - tanto cliente quanto servidor podem tomar a iniciativa de
  - trocar mensagens
  - evitar somente deadlocks
    - dois em receive inicialmente

TCP: limite de conexões

- **Quantidade de conexões**
  - limite da fila de pedidos de conexão em espera
    - na versão 1.2: 50 é o default
  - limite de conexões abertas
    - na versão 1.2: não encontrado

Exemplo Echo

- **Descrição**

- le string da standard input
- envia o string ao servidor Echo
- recebe resposta do servidor Echo
- imprime resposta

Exemplo Echo

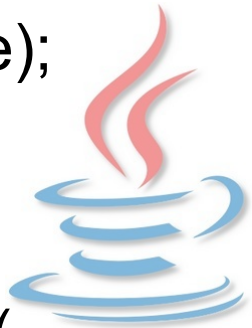
- **Exemplo Código Echo**

```
import java.io.*;
import java.net.*;

    public class EchoClient {
    public static void main(String[] args)
            throws IOException {

    Socket echoSocket = null;
    PrintWriter out = null;
    BufferedReader in = null;
```

Exemplo Echo

- **Exemplo Código Echo**

```
try {
    // cria socket local e conecta ao servidor
    echoSocket = new Socket("taranis", 7);
    // PrintWriter: 1o arg: OutputStream
    //                  2o arg: println com aç ã o flush
    out = new
        PrintWriter(echoSocket.getOutputStream(), true);
    // BufferedReader: arg: Reader
    // InputStreamReader: arg: InputStream
    //                              subclasse de Reader
    in = new BufferedReader(new InputStreamReader(
          echoSocket.getInputStream()));
}
```



Exemplo Echo

- **Exemplo Código Echo**

```
catch (UnknownHostException e) {
   System.err.println("Don't know about host: taranis.");
   System.exit(1);
} catch (IOException e) {
   System.err.println("Couldn't get I/O for ”
                      + "the connection to: taranis.");
   System.exit(1);
}
```

Exemplo Echo

- **Exemplo Código Echo**

```
// objeto para I/O do teclado
BufferedReader stdIn = new BufferedReader(
    new InputStreamReader(System.in));
String userInput;

// le do teclado, envia para servidor e imprime resposta
// até que linha lida seja “nula”
while ((userInput = stdIn.readLine()) != null) {
    out.println(userInput);
    System.out.println("echo: " + in.readLine());
}
```

Exemplo Echo

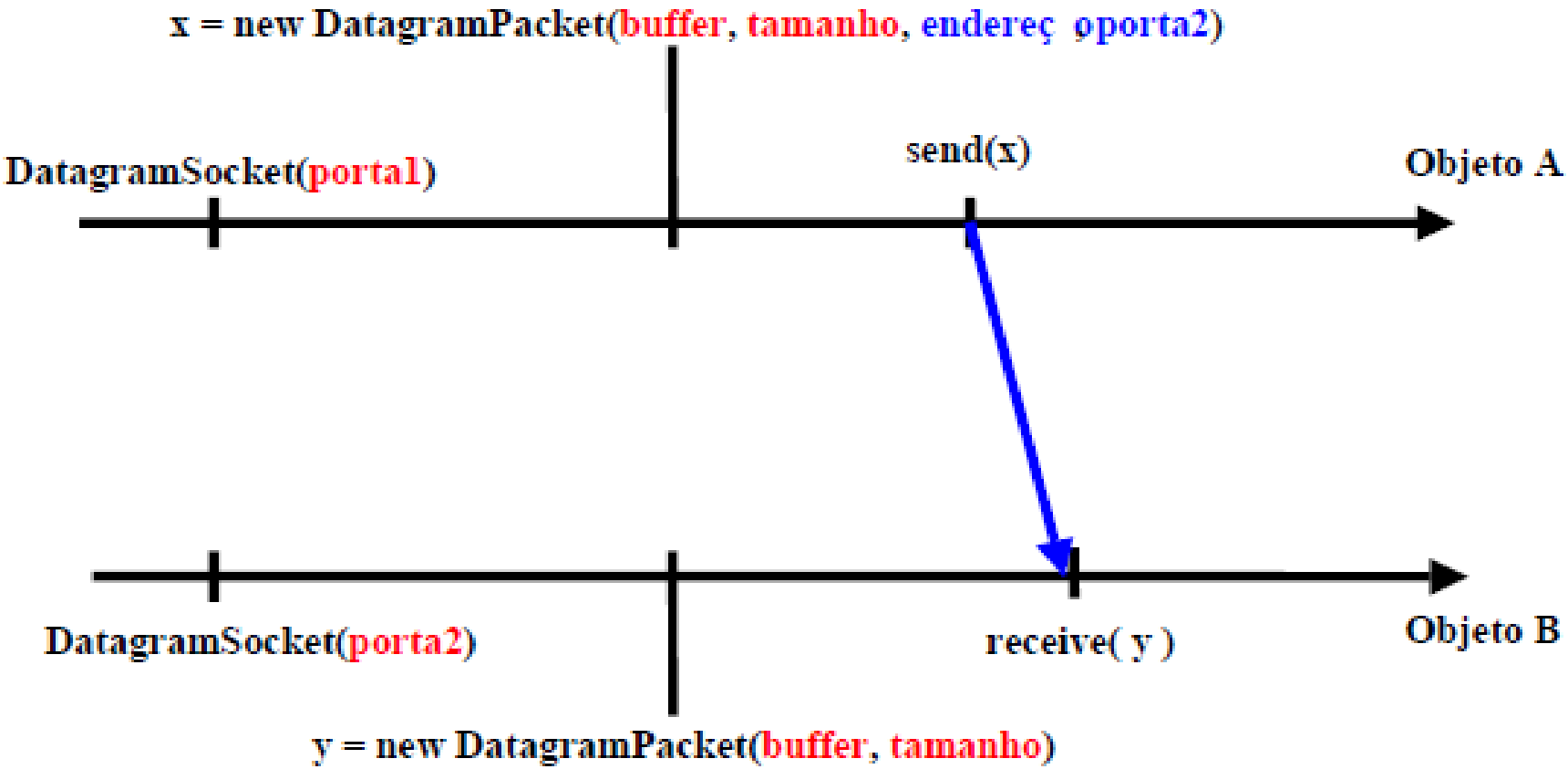
- **Exemplo Código Echo**

```
        // fecha os streams e a conexã o
        out.close();
        in.close();
        stdIn.close();
        echoSocket.close();
        }
}
```

Socket UDP

- Sumário
  - Modelo de comunicação com classes UDP
  - Esqueleto de programa

Socket UDP e Classes Java



Socket UDP e Classes Java

- **Ambos os lados da conexão criam um novo *DatagramSocket***
  - receptor deve informar sua porta de recepção
  - pode ser usado para receber e enviar
- **Ambos os objetos criam *DatagramPacket***
  - mas o objeto que vai enviar o pacote tem que informar o endereço e porta do destinatário
- **Para cada mensagem a ser enviada**
  - criar um novo *DatagramPacket*
  - *informar mensagem (buffer e tamanho) e destino (endereço e porta)*

Socket UDP e Classes Java

- **Utilizar os métodos send e receive**
  - programador precisa empacotar/desempacotar dados em um buffer
- **Receive**
  - Pode receber pacote com diferentes tamanhos do enviado
  - Maior
    - alinha à esquerda
    - resto: não preenchido
  - Menor
    - alinha à esquerda
    - resto: truncado
- **Para terminar a conexão, utilizar close**

Exemplo de Código para Datagramas

SEND

```
(...) DatagramSocket s;
try {
    s= new DatagramSocket( );
}catch(SocketException e) { (....)
   }
byte[] b = {0,1,2,3,4,5,6,7};
DatagramPacket p = new
   DatagramPacket ( b, 8,
   iaddr,2000);
try{
   s.send( p );
}catch (IOException e) { (...) }
```

RECEIVE

```
(...) DatagramSocket s;
try {
    s= new DatagramSocket(
   2000 );
}catch(SocketException e) { (....)
   }
DatagramPacket p = new
   DatagramPacket (new byte[8],
   8);
try{
   s.receive( p );
}catch (IOException e) { (...) }
```

iaddr é o endereço InetAddr do host para onde será mandada a mensagem

Exemplo sockets Java UDP

- **Exemplo completo de sockets Java UDP**
  - **fonte: tutorial da Sun**

  - **especificação**
    - cliente
      - solicita uma sentença do dia
    - servidor
      - responde com uma sentença
      - sentença é lida de um arquivo

Exemplo sockets Java UDP

- **código cliente**

```
import java.io.*;
import java.net.*;
import java.util.*;
public class QuoteClient {
    public static void main(String[] args) throws
IOException {

// verifica 1o argumento: nome do servidor
    if (args.length != 1) {
        System.out.println("Usage: java
QuoteClient <hostname>");
        return;
    }
```

Exemplo sockets Java UDP

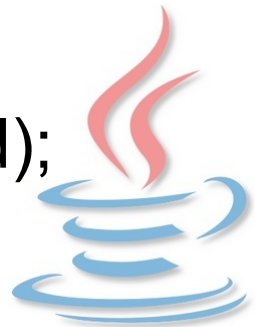
- **Exemplo completo de sockets Java UDP**
  - **código cliente**

```
// cria um datagram socket
    DatagramSocket socket = newDatagramSocket();
// envia pedido;
// nome do server é 1o argumento do programa
// porta é constante: 4445
    byte[] buf = new byte[256];
    InetAddress address =
InetAddress.getByName(args[0]);
    DatagramPacket packet = new
DatagramPacket(buf,
buf.length, address, 4445);
    socket.send(packet);
```

Exemplo sockets Java UDP

- **Exemplo completo de sockets Java UDP**
- **Código cliente**

```
// recebe resposta
// porta do cliente é passada implicitamente ao servidor
        packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);
// mostra resposta
        String received = new String(packet.getData());
        System.out.println("Quote of the Moment: " +
                                                received);
        socket.close();
    }
}
```



Exemplo sockets Java UDP

Exemplo completo de sockets Java UDP

código servidor

```
import java.io.*;
import java.net.*;
import java.util.*;

public class QuoteServerThread extends Thread {

protected DatagramSocket socket = null;
protected BufferedReader in = null;
protected boolean moreQuotes = true;

// construtor básico para exceç ã o
public QuoteServerThread() throws IOException {
this("QuoteServerThread");
}
```

Exemplo sockets Java UDP

Exemplo completo de sockets Java UDP

código servidor

```
// construtor normal: argumento nome da thread
public QuoteServerThread(String name) throws
IOException {
    super(name);
    socket = new DatagramSocket(4445);
    try {
        in = new BufferedReader(new
                FileReader("oneliners.txt"));
    } catch (FileNotFoundException e) {
        System.err.println("Could not open quote file.
Serving time instead.");
    }
        }
```

Exemplo sockets Java UDP

Exemplo completo de sockets Java UDP

código servidor

```
// método principal da thread (servidor)
public void run() {
// loop enquanto houver sentenç as
   while (moreQuotes) {
         try {
              byte[] buf = new byte[256];

// recebe pedido do cliente
              DatagramPacket packet =
                   new DatagramPacket(buf, buf.length);
              socket.receive(packet);
```

Exemplo sockets Java UDP

- **Exemplo completo de sockets Java UDP**
  - **código servidor**

```
// cria resposta
        String dString = null;
        // se nã o háa rquivo de Quotes
        if (in == null)
            dString = new Date().toString();
        else
                // lê próxima quote
            dString = getNextQuote();
        buf = dString.getBytes();
```

Exemplo sockets Java UDP

- **Exemplo completo de sockets Java UDP**
  - **código servidor**

```
// envia resposta ao client em "address" e "port"
//    "address" e "port" obtidos na mensagem recebida
            InetAddress address = packet.getAddress();
            int port = packet.getPort();
            packet = new DatagramPacket(buf, buf.length,
address, port);
            socket.send(packet);
        } catch (IOException e) {
          e.printStackTrace();
                moreQuotes = false;
        }
}
socket.close();
```

Exemplo sockets Java UDP

- **Exemplo completo de sockets Java UDP**
  - **código servidor**

```
protected String getNextQuote() {
    String returnValue = null;
    try {
        if ((returnValue = in.readLine()) == null) {
            in.close();
            moreQuotes = false;
            returnValue = "No more quotes. Goodbye.";
        }
    } catch (IOException e) {
        returnValue = "IOException occurred in server.";
    }
    return returnValue;
}
}
```