Java
Sockets

Utilização das classes Java para comunicação TCP/IP e UDP/IP

Autor

- **Autores**
  - **Cláudio Geyer**
  - **Maurício Lima Pilla**
- **Local**
  - **Instituto de Informática**
  - **UFRGS**
  - **disciplina : Programação Distribuída e Paralela**
  - **versão**
    - V14, março de 2010

Súmula

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

Bibliografia

Java x sockets

- GARG, V. J. Concurrent and Distributed Computing in Java. IEEE Press, Wiley, 2004.
- BOGER, M. Java on Distributed Systems. Wiley, 1999.
- HAROLD, E. R. Java Network Programming. O'Reilly, 3a. ed. 2004.
- GRABA, J. An Introduction to Network Programming with Java. Springer, 2006.
- PITT, E. Fundamental Networking in Java. Springer, 2005.
- FARLEY, J. Java - Distributed Computing. Ed. O'Reilly, 1998.

Bibliografia

Java

- FLANAGAN, David. *Java in a Nutshell*. O'Reilly Assoc., 2a. ed., 1997.
- HORSTMANN, Cay S., CORNELL, Gary. *Core Java 1.1 Volume II - Advanced Features*. Prentice Hall PTR, 1998.

Sobre Sockets x redes

- TANEMBAUM, Andrew S. *Computer Networks*. Prentice Hall PTR, 3a. ed., 1996.

Sobre Sockets com Unix/C

- Stevens, R. Advanced Unix Programming.
- Stevens, R. Unix Networking Programming.

Bibliografia

- **Documentação Java: package java.net**
  - Na versão J2SE 6:
    - http://java.sun.com/javase/6/docs/api/
    - -> pacote java.net
- **Orfali, R. and Harkey, D. Client/Server Programming with JAVA and CORBA. John Wiley, 2a edição, 1998.**
  - Capítulo 10
- **Tutorial Sun sobre Java**
  - trail Networking
  - http://java.sun.com/docs/books/tutorial/networking/index.html

Diferenças Entre TCP e UDP

TCP (Transmission Control Protocol)

- **Orientado a conexão**
- **Confiável**
- ***Stream***
- **Controle de fluxo**

UDP (User Datagram Protocol)

- **Orientado a datagrama**
- **Não é confiável**
- **Datagramas (pacotes)**
- **Sem controle de fluxo**

Diferenças Entre TCP e UDP

TCP (Transmission Control Protocol)

- **Mais lento**
- **Garantias de**
  - **Ordem**
  - **Chegada**
  - **Não duplicação**
  - **conteúdo**

UDP (User Datagram Protocol)

- **Sem garantia de**
  - **ordem**
  - **chegada**
  - **não duplicação**
  - **conteúdo**
- **Menor *overhead***
- **Mais apropriado a broadcast**

Conceitos básicos de sockets TCP

- **usa algumas características do modelo cliente/servidor**
- **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

- **Conceitos básicos de 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

- **Conceitos básicos de 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

- **Conceitos básicos de 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

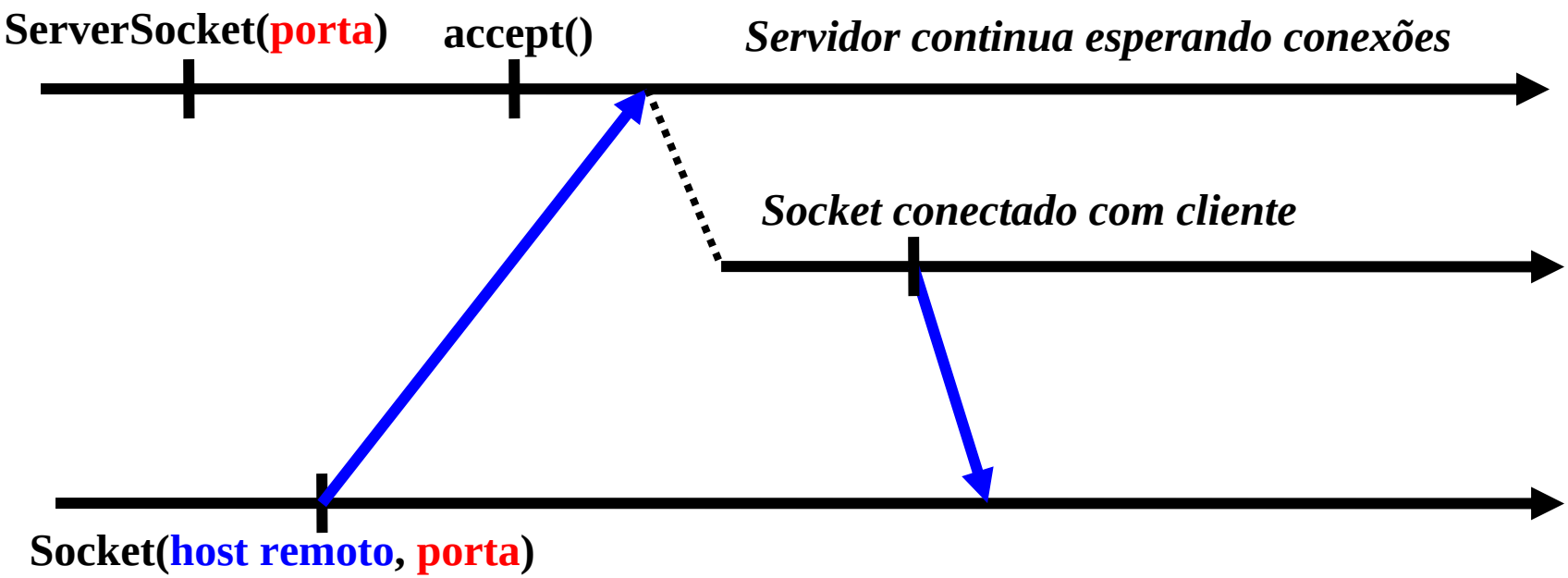
- **Conceitos básicos de sockets TCP**
  - **cliente**
    - conhece hostname (IP) da máquina servidora
    - conhece porta do programa servidor
    - pede conexão
    - se conexão aceita
      - um socket é criado
      - associado a uma porta na máquina cliente

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

Servidor: classe java.net.ServerSocket



Cliente: classe java.net.Socket

- **Sockets TCP e Classes Java**
  - **Primeiro**
    - servidor cria *ServerSocket*
    - espera por pedidos de conexão 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 conexões na mesma porta

Código para Criar Conexão

SERVIDOR

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

while (true){

 Socket conexao =
   s.accept();

 /* Disparar uma thread que
   faça algo, passando
   conexão como parâmetro
   */

 (...)

}
```

CLIENTE

```
(...)Socket s;

try{

s =new
   Socket("poncho",8189);

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

   System.exit(0);

}

/* Socket conectado */
```

Como Enviar e Receber Mensagens

- **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

Como Enviar e Receber Mensagens

- **Como Enviar e Receber Mensagens**
  - **Várias classes e métodos para leitura e escrita em *streams***
    - podendo transmitir desde bytes até certos objetos
  - **Para fechar uma conexão, utilizar *close( )***
  - **Sincronização**
    - Receives são síncronos (espera send)
    - Sends: assíncrono bloqueante
      - Não espera “receive”
      - Espera somente sincronização interna do TCP

Exemplos de Uso de Streams

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
    - Provavelmente o limite

Exemplo Echo

- **Exemplo Echo**
  - **fonte: tutorial Java da Sun**
    - trail networking, lição Sockets, 1o exemplo
  - **descrição**
    - le string da standard input
    - envia o string ao servidor Echo
    - recebe resposta do servidor Echo
    - imprime resposta

Exemplo Echo

- **Exemplo Echo**
  - **código**

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

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

        Socket echoSocket = null;  // declara socket
        PrintWriter out = null;    // declara vars stream
        BufferedReader in = null;
```

Exemplo Echo

código

```
try {
        // cria socket local e conecta ao servidor
        echoSocket = new Socket("taranis", 7);
        // geral: cria streams de in (receive) e out (send)
        // PrintWriter:  1o arg: OutputStream
        //                2o arg: println com ação 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 Echo**
  - **código**

```
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 Echo**
  - **código**
    - // objeto para I/O do teclado: a enviar ao servidor

```
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

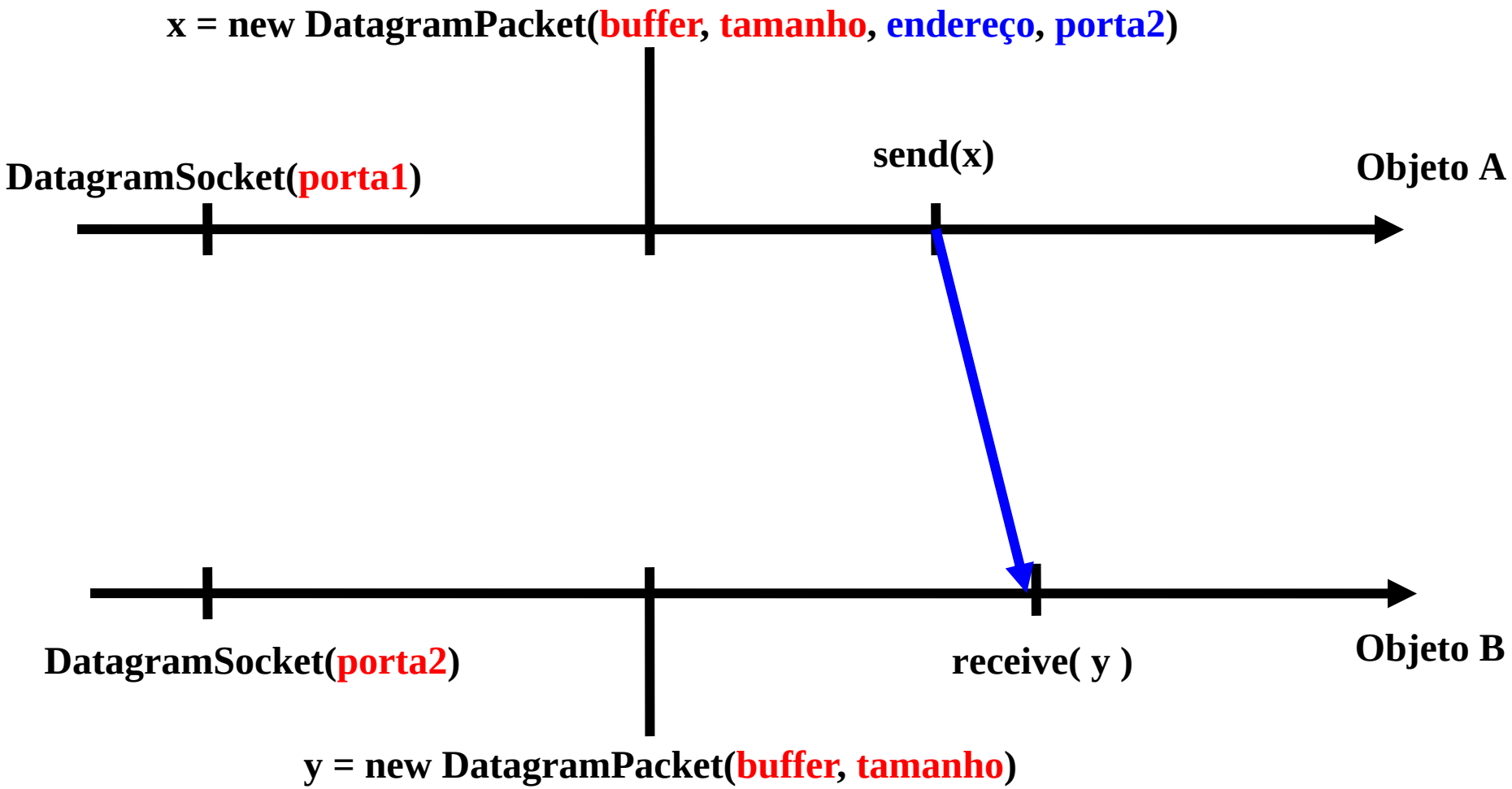
- **Exemplo Echo**
  - **código**
    - // fecha os streams e a conexão
      out.close();
      in.close();
      stdIn.close();
      echoSocket.close();
      }
      }

Exercícios Sockets

- **Exercícios**
  - **A) identifique e explique os comandos send e receive no exemplo Echo**
  - **B) faça um esqueleto (pseudo-código) do programa Echo servidor**
  - **C) modele uma solução multithreaded no servidor**
  - **D) modele o algoritmo de difusão usando Java TCP sockets**
    - quem pede a conexão
    - como

- **Sockets UDP**
  - **sumário**
    - modelo de comunicação com classes UDP
    - esqueleto de programa

Sockets UDP e Classes Java



Sockets 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*
  - *infomar mensagem (buffer e tamanho) e destino (endereço e porta)*

Sockets 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) { (...) }
```

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 completo de 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 = new DatagramSocket();
// 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 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 exceção
 public QuoteServerThread() throws IOException {
        this("QuoteServerThread");
    }
```

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("one-
liners.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 sentenças
      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 não há arquivo de Quotes
if (in == null)
    dString = new Date().toString();
else
        // lê próxima quote
    dString = getNextQuote();
buf = dString.getBytes();
```

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 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;
    }
}
```

Exercícios: sockets Java UDP

- **Exercícios**
  - **A) modele o algoritmo de difusão com Java sockets UDP**
  - **B) discuta possíveis efeitos caso a difusão seja programada com Java sockets UDP**

- **Revisão**
  - **caracterize a TM via sockets conforme conceitos PDP**
    - nomeação: estática/dinâmica, explícita/implícita
    - criação de canais: idem
    - uni ou bidirecional
    - síncrona, assíncrona bloqueante/não-bloqueante
    - buferizada ou não
  - **quais os tipos de sockets em Java?**

- **Revisão**
  - **sockets TCP**
    - propriedades
  - **sockets TCP em Java**
    - nomeação
    - criação de conexão (canal)
    - criação e tipos de links
    - primitivas para send/receive
    - tipos de dados

- **Revisão**
  - **sockets UDP**
    - propriedades
  - **sockets UDP em Java**
    - nomeação
    - canal
    - mensagem
    - send/receive
    - tipos de dados