PLATAFORMA JAVA

Prof. Daniel Corrêa da Silva

JAVA EM REDE

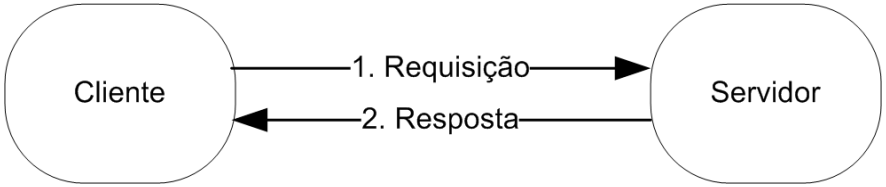
VISÃO GERAL

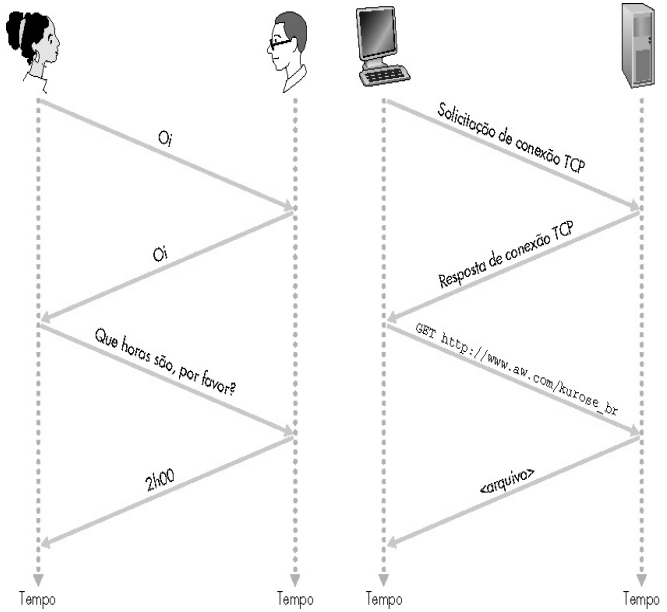
- Introdução ao Modelo Cliente / Servidor

JAVA EM REDE

MODELO CLIENTE / SERVIDOR

Modelo Cliente / Servidor: modelo de arquitetura distribuída onde parte da aplicação é executada no host cliente e parte é executada no host servidor. O host cliente abre a conexão e inicia a comunicação com o servidor, enviando-lhe uma mensagem de solicitação a uma porta que identifica o serviço a ser executado. Após o tratamento da solicitação, o servidor responde com uma mensagem contendo o resultado e após o cliente receber o mesmo, fecha-se a conexão com o servidor.

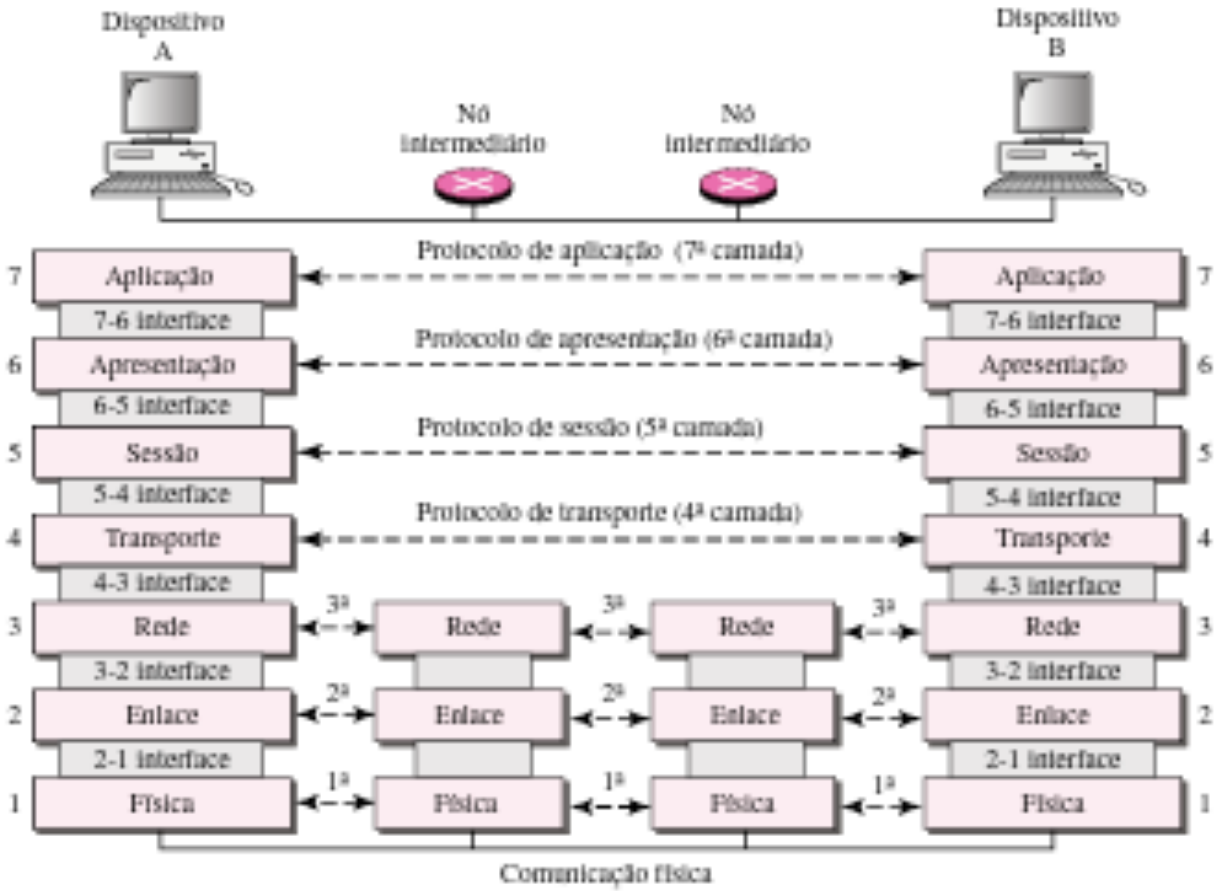




JAVA EM REDE

MODELO CLIENTE / SERVIDOR

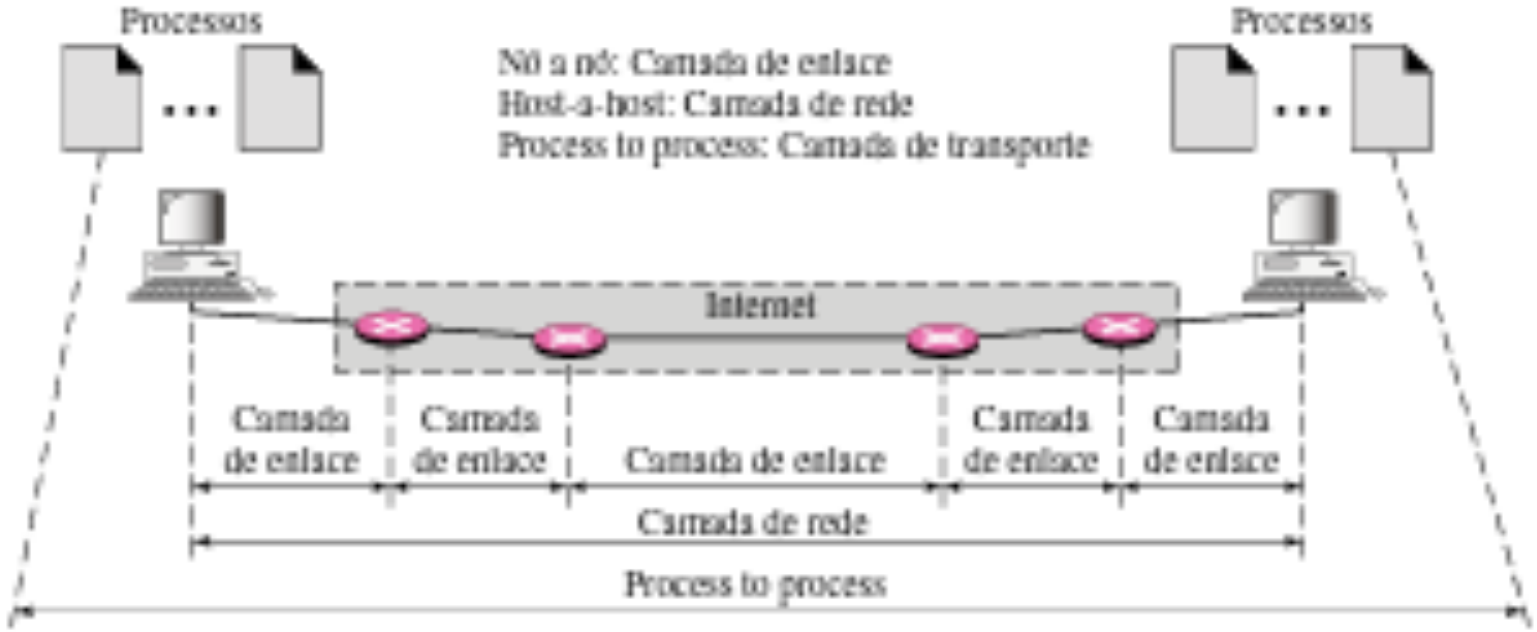
Modelo Cliente / Servidor:



JAVA EM REDE

MODELO CLIENTE / SERVIDOR

Modelo Cliente / Servidor:

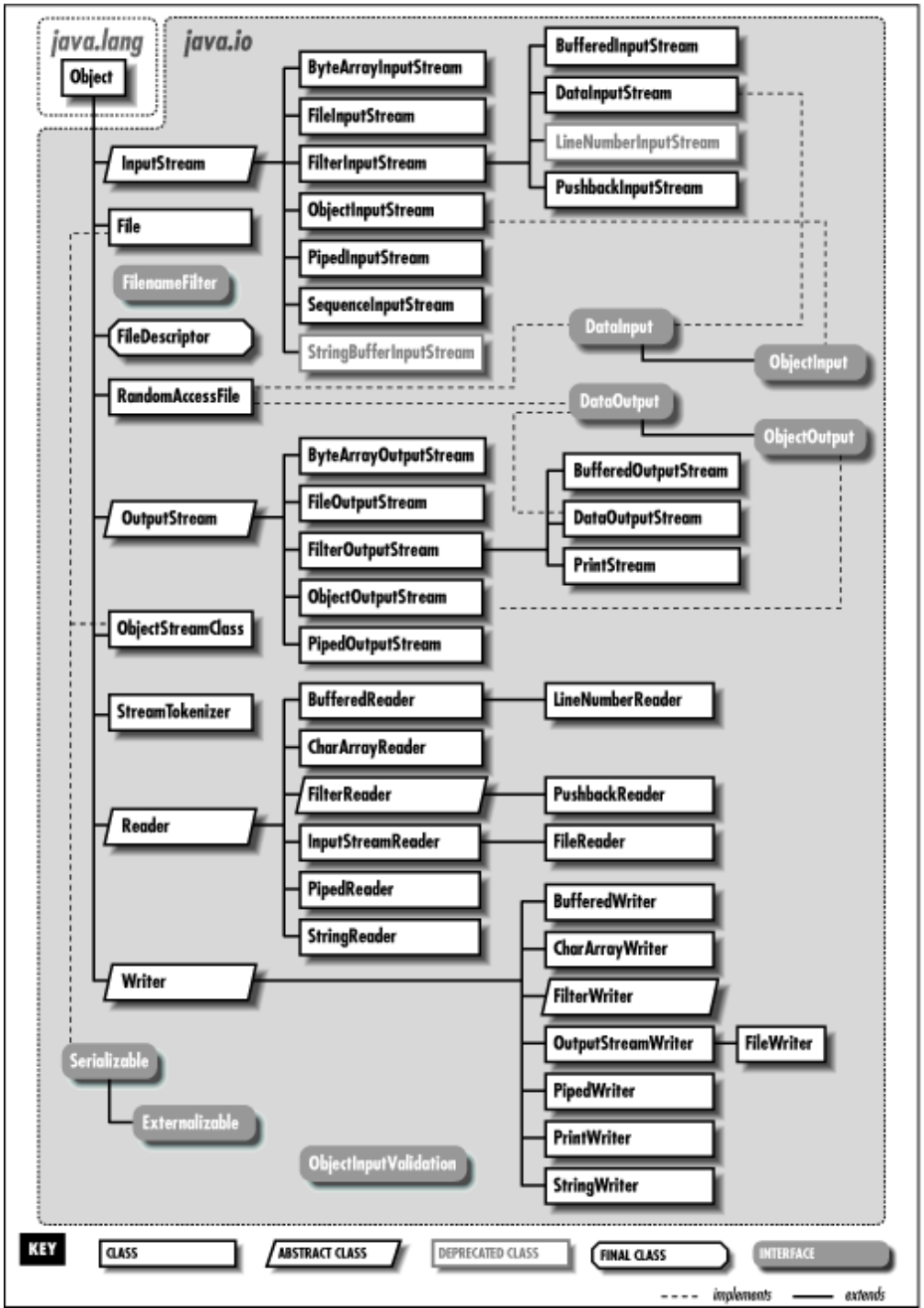


OPERAÇÕES ENTRADA / SAÍDA

PACOTE DE ENTRADA / SAÍDA JAVA

Definição: a plataforma Java inclui pacotes referentes a movimentação de dados. A movimentação de dados ocorre durante a obtenção de dados referentes a entrada e o envio de dados a saída, por meio de entidades capazes de interagirem por meio de operações de entrada e saída: teclado, mouse, placa de rede, disco rígido e programas externos.

- **Pacote java.io:** define operações de E/S em termos de stream. Streams são sequências ordenadas de dados que possuem uma origem (streams de entrada) e um destino (streams de saída).
- **Pacote java.nio:** define operações de E/S em termos de buffers e canais. Buffers são depósitos de dados que podem ser lidos ou escritos. Canais são conexões com entidades capazes de realizar operações de E/S, incluindo buffers, arquivos e sockets.
- **Pacotes java.net:** fornece o suporte específico para operações de E/S em redes, baseado no uso de sockets, com um modelo subjacente em streams ou em canais.


java.lang
java.io
Object
InputStream
File
FilenameFilter
FileDescriptor
RandomAccessFile
OutputStream
ObjectStreamClass
StreamTokenizer
Reader
Writer
Serializable
Externalizable
ObjectInputValidation
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
DataInput
ObjectInput
DataOutput
ObjectOutput
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
ObjectOutputStream
PipedOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
BufferedReader
CharArrayReader
FilterReader
InputStreamReader
PipedReader
StringReader
LineNumberReader
PushbackReader
FileReader
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
PipedWriter
PrintWriter
StringWriter
FileWriter
KEY
CLASS
ABSTRACT CLASS
DEPRECATED CLASS
FINAL CLASS
INTERFACE
implements
extends

OPERAÇÕES ENTRADA / SAÍDA

PACOTE DE ENTRADA / SAÍDA JAVA

As principais subclasses de InputStream e OutputStream, utilizada em programas que se comunicam em rede:

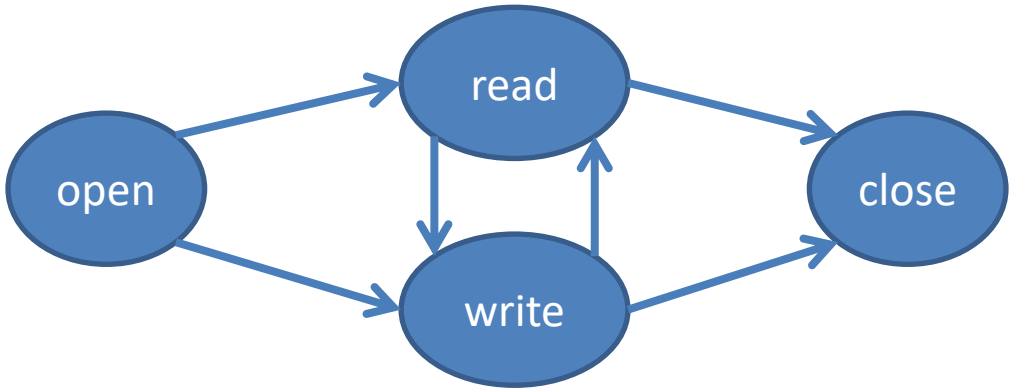
CLASSE	DESCRIÇÃO
ObjectInputStream	Recebe uma sequência de bytes e interpreta como um objetos.
ObjectOutputStream	Envia um objeto “convertendo-o” em uma sequência de bytes.
DataInputStream	Recebe uma sequência de bytes e interpreta como um tipo primitivo.
DataOutputStream	Envia um tipo primitivo “convertendo-o” em uma sequência de bytes.
FileInputStream	Entrada de dados a partir de arquivos.
FileOutputStream	Saída de dados para arquivos.

OPERAÇÕES ENTRADA / SAÍDA

PACOTE DE ENTRADA / SAÍDA JAVA

O pacote java.io subdivide em dois tipos de streams:

- **Stream de Caracteres:** define uma sequência de caracteres, tal como as informações baseadas em texto. As streams de caracteres utilizam as classes ***Reader*** e ***Writer***.
- **Stream de Bytes:** define uma sequência de bytes serializados, tal como as informações baseadas em imagens, vídeo e áudio. As streams de byte utilizam as classes ***InputStream*** e ***OutputStream***.



OPERAÇÕES ENTRADA / SAÍDA

STREAM DE BYTE

InputStream: classe abstrata que fornece a abstração para ler os ***bytes*** de uma ***entidade de origem*** utilizando métodos especificados pela mesma.

METODOS	DESCRIÇÃO
int **read**()	Lê um único byte de dados e retorna o byte que foi lido como um inteiro entre 0 e 255.
int **read**(byte[] buf, int offset, int count)	Lê uma parte de um vetor de bytes. O número máximo de bytes lidos é ***count***. Os bytes são armazenados de ***buf*** [***offset***] até no máximo ***buf*** [***count***].
long **skip**(long count)	Salta ***count*** bytes da entrada ou até que o final da stream seja encontrada.
int **avaible**()	Retorna o número de bytes que podem ser lidos sem bloquear.
void **close**()	Fecha a stream de entrada.

OPERAÇÕES ENTRADA / SAÍDA

STREAM DE BYTE

OutputStream: classe abstrata que fornece a abstração para escrever ***bytes*** em uma ***entidade de destino*** utilizando métodos especificados pela mesma.

METODOS	DESCRIÇÃO
void **write**(int b)	Escreve ***b*** como byte. O byte é passado como um inteiro int.
void **write**(byte[] buf, int offset, int count)	Escreve parte de um vetor de bytes, iniciando em ***buf*** [***offset***] e escrevendo ***count*** bytes. O método bloqueia até que os bytes tenham sido escritos.
void **flush**()	Esvazia todos os buffers referentes a uma stream.
void **close**()	Fecha a stream de saída.

OPERAÇÕES ENTRADA / SAÍDA

STREAM DE CARACTERES

Reader: classe abstrata que fornece a abstração para ler ***caracteres*** em uma ***entidade de origem*** utilizando métodos especificados pela mesma.

METODOS	DESCRIÇÃO
int **read**()	Lê um único caractere e o retorna como um inteiro entre 0 e 65535. Retorna -1 se tiver alcançado o final da stream.
int **read**(char[] buf, int offset, int count)	Lê uma parte de um vetor de caracteres. O número máximo de caracteres lidos é ***count***. Os caracteres são armazenados de ***buf*** [***offset***] até no máximo ***buf*** [***count+cont-1***].
long **skip**(long count)	Salta ***count*** caracteres da entrada ou até que o final da stream seja encontrada.
boolean **ready**()	Retorna true se a stream possui pelo menor um caractere a ser lido.
void **close**()	Fecha a stream de entrada.

OPERAÇÕES ENTRADA / SAÍDA

STREAM DE CARACTERES

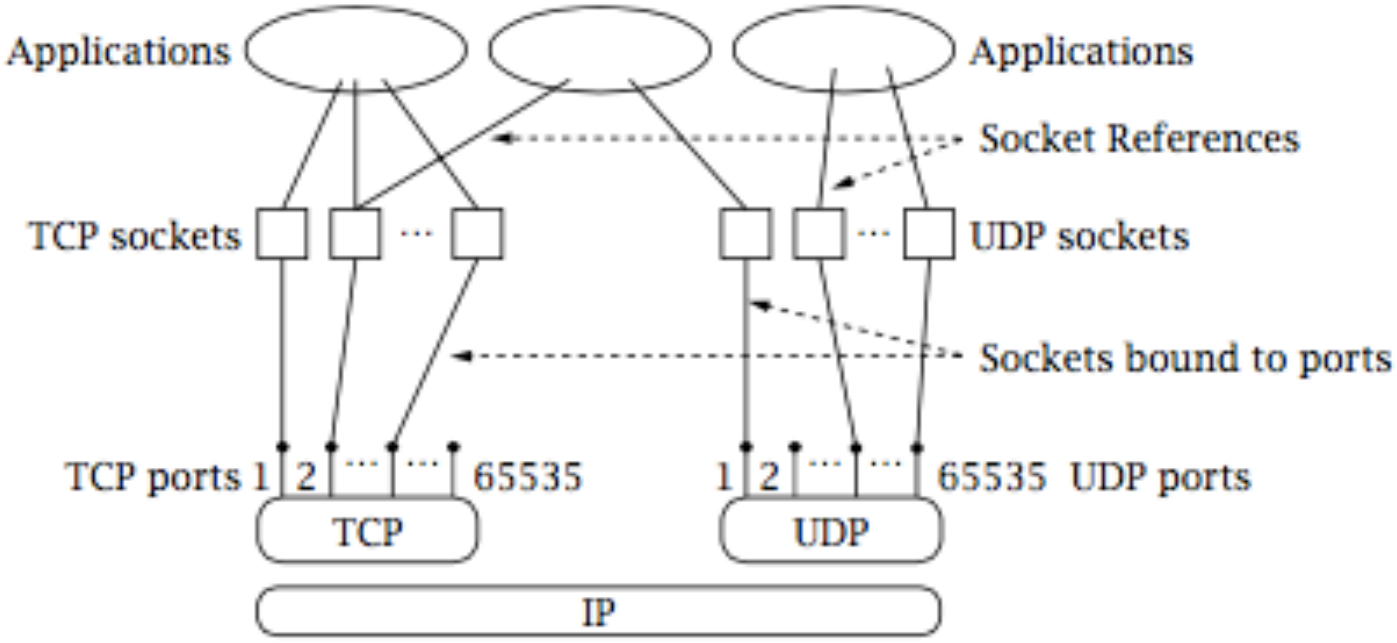
Writer: classe abstrata que fornece a abstração para escrever ***caracteres*** em uma ***entidade de destino*** utilizando métodos especificados pela mesma.

METODOS	DESCRIÇÃO
void **write**(int ch)	Escreve ***ch*** como caractere. O caractere é passado como um inteiro int.
void **write**(char[] buf, int offset, int count)	Escreve parte de um vetor de caracteres, iniciando em ***buf*** [***offset***] e escrevendo ***count*** caracteres. O método bloqueia até que os caracteres tenham sido escritos.
void **flush**()	Esvazia todos os buffers referentes a uma stream.
void **close**()	Fecha a stream de saída.

SOCKET EM JAVA

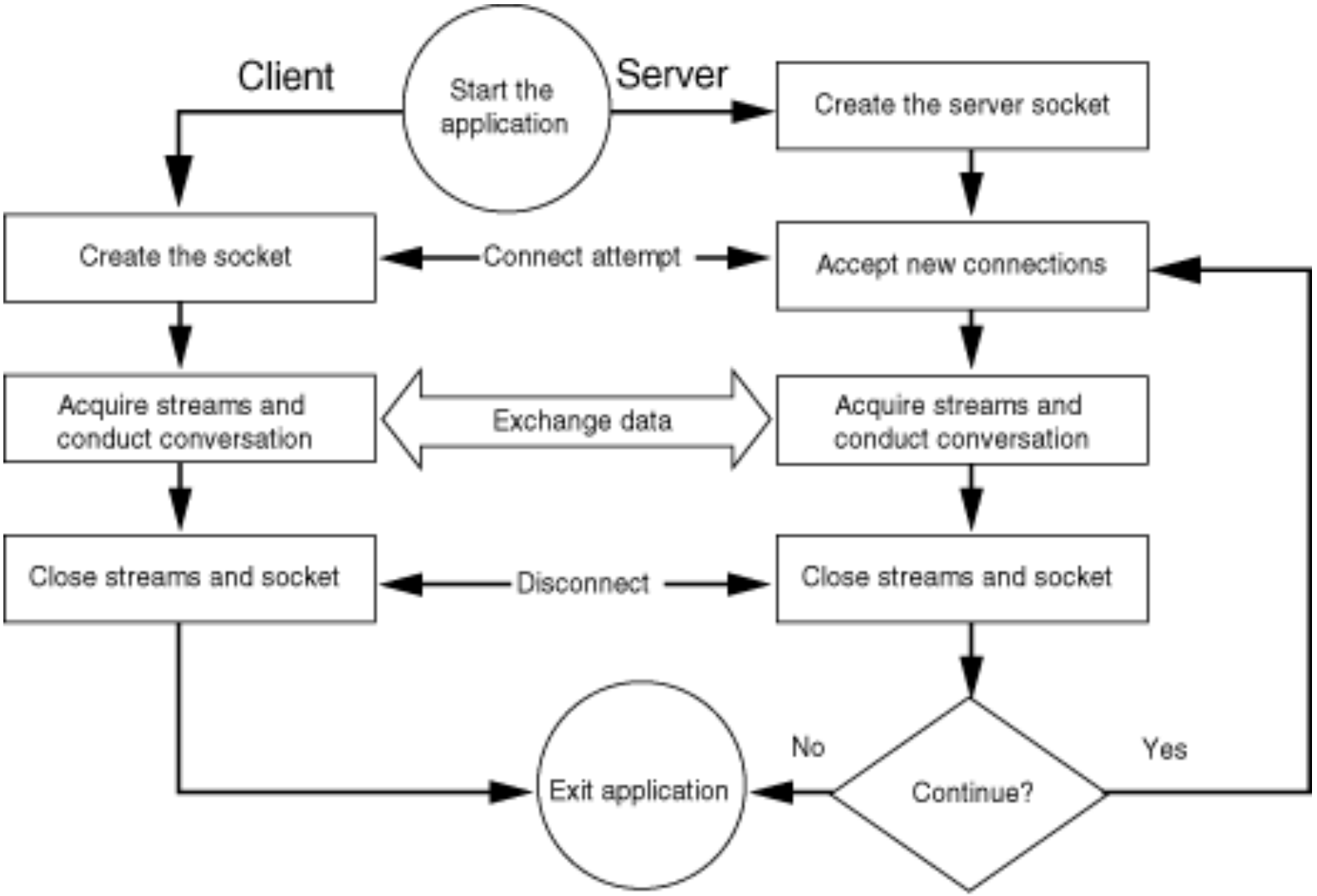
SOCKET EM JAVA:

Socket: permite uma aplicação conectar a uma rede e comunicar-se com outra aplicação também conectada a mesma rede. Informações escrita sobre o socket de uma aplicação de uma máquina, pode ser lida por outra aplicação de outra máquina diferente ou vice-versa.



SOCKET EM JAVA

SOCKET EM JAVA:



SOCKET EM JAVA

EXMPLO1: TCPServer com DataOutputStream

```
1  package br.principal;
2
3  import java.io.BufferedReader;
8
9  public class TCPServer {
10
11     public static void main(String[] args) throws Exception{
12
13         String clientSentence;
14         String capitalizeSentence;
15         int port = 6789;
16         ServerSocket welcomeSocket = new ServerSocket(port);
17         while(true)
18         {
19             System.out.println("Servidor aguardando o cliente");
20             Socket conectionSocket = welcomeSocket.accept();
21             BufferedReader inFromClient = new BufferedReader(new InputStreamReader(conectionSocket.getInputStream()));
22             DataOutputStream outToClient = new DataOutputStream(conectionSocket.getOutputStream());
23
24             clientSentence = inFromClient.readLine();
25             System.out.println("Texto vindo do cliente: "+ clientSentence);
26             capitalizeSentence = clientSentence.toUpperCase()+'\n';
27             outToClient.writeBytes(capitalizeSentence);
28         }
29     }
30 }
```

SOCKET EM JAVA

EXMPLO1: TCPClient com DataOutputStream

```java
package br.principal;

import java.io.BufferedReader;

public class TCPClient {

    public static void main(String[] args) throws UnknownHostException, IOException{

        String sentence;
        String modifiedSentence;
        int port = 6789;
        Socket clientSocket = new Socket("127.0.0.1", port);
        BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));
        DataOutputStream outServer = new DataOutputStream(clientSocket.getOutputStream());
        BufferedReader inFromServer = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));

        sentence = inFromUser.readLine();
        outServer.writeBytes(sentence + '\n');
        modifiedSentence = inFromServer.readLine();

        System.out.println("Texto enviado do servidor: "+modifiedSentence);
        clientSocket.close();
    }
}
```

SOCKET EM JAVA

EXMPLO2: TCPServer com ObjectOutputStream

```
1  package br.principal;
2
3  import java.io.ObjectInputStream;
4  import java.io.ObjectOutputStream;
5  import java.net.ServerSocket;
6  import java.net.Socket;
7
8  public class TCPServer {
9
10     public static void main(String[] args) throws Exception{
11
12         int port = 6789;
13         ServerSocket welcomeSocket = new ServerSocket(port);
14         while(true)
15         {
16             System.out.println("Servidor aguardando o cliente");
17             Socket conectionSocket = welcomeSocket.accept();
18             ObjectInputStream inFromClient = new ObjectInputStream(conectionSocket.getInputStream());
19             ObjectOutputStream outToClient = new ObjectOutputStream(conectionSocket.getOutputStream());
20
21             Maquina maquina = (Maquina) inFromClient.readObject();
22             maquina.setNome("servidor responde");
23             outToClient.writeObject(maquina);
24         }
25     }
26 }
```

SOCKET EM JAVA

EXMPLO2: TCPClient com ObjectOutputStream

```
1  package br.principal;
2
3  import java.io.IOException;
8
9  public class TCPClient {
10
11     public static void main(String[] args) throws UnknownHostException, IOException, ClassNotFoundException {
12
13         int port = 6789;
14         Socket clientSocket = new Socket("127.0.0.1", port);
15
16         ObjectOutputStream outServer = new ObjectOutputStream(clientSocket.getOutputStream());
17         ObjectInputStream inFromServer = new ObjectInputStream(clientSocket.getInputStream());
18
19         Maquina maquina = new Maquina();
20         maquina.setNome("cliente solicita");
21
22         outServer.writeObject(maquina);
23         maquina = (Maquina) inFromServer.readObject();
24         System.out.println(maquina.getNome());
25
26         clientSocket.close();
27
28     }
29 }
```

SOCKET EM JAVA

EXMPLO2: TCPClient com ObjectOutputStream

```
1  package br.principal;
2
3  import java.io.Serializable;
4
5  public class Maquina implements Serializable{
6
7      private String nome;
8
9      public String getNome() {
10         return nome;
11     }
12
13     public void setNome(String nome) {
14         this.nome = nome;
15     }
16
17
18 }
```