

Bacharelado em Ciências da Computação

BC1501 – Programação Orientada a Objetos



Aula 07

Prof. André G. R. Balan

andre.balan@ufabc.edu.br

Nesta Aula

- Conceitos Básicos sobre redes de computadores
- Sockets
- Trabalhando com sockets em JAVA

Redes de computadores

- ▶ Uma **rede de computadores** consiste de 2 ou mais computadores e outros dispositivos conectados entre si de modo a poderem compartilhar informações e serviços .
- ▶ Uma rede compreende basicamente:
 - **estruturas físicas** (equipamentos, fios, modems, hubs, ..)
 - **estruturas lógicas** (programas, protocolos de comunicação)

Redes de computadores

- ▶ Um **protocolo de comunicação** compreende um conjunto de regras lógicas e padrões para troca de informações em um meio físico (fios, fibras)

Redes de computadores

- ▶ Exemplos de protocolos bem conhecidos
 - IP (Internet Protocol)
 - DHCP (Dynamic Host Configuration Protocol)
 - TCP (Transmission Control Protocol)
 - HTTP (Hypertext Transfer Protocol)
 - FTP (File Transfer Protocol)
 - Telnet (Telnet Remote Protocol)
 - POP3 (Post Office Protocol 3)
 - SMTP (Simple Mail Transfer Protocol)
 - IMAP (Internet Message Access Protocol)

Redes de computadores

- ▶ Entretanto, com um único protocolo não é possível gerar uma comunicação entre dois sistemas
- ▶ Para isso, vários protocolos são combinados entre si, sendo que cada um fica responsável por uma etapa da comunicação:
 - Etapas de baixo nível: comunicação entre o hardware
 - Etapas de alto nível: comunicação entre aplicativos
 - Etapas de nível médio:
 - Compreende técnicas de envio, recebimento, confirmação, recuperação de erros, etc...

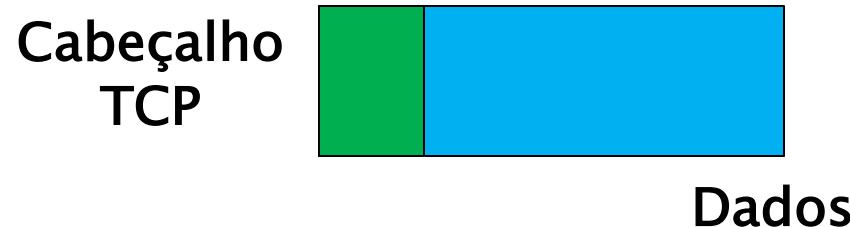
Redes de computadores

- ▶ O “TCP/IP” , é a coleção de protocolos desenvolvida para e utilizada para comunicação via Internet, que vem se consolidando como o padrão de comunicação digital no mundo todo
- ▶ “TCP/IP” é independente de plataforma, e foi implementado em plataformas e sistemas operacionais

Redes de computadores

► Exemplo:

- Os protocolo TCP e responsável por **quebrar os dados da comunicação em pequenos pedaços**, chamados **pacotes**. Juntamente com cada pacote, é adicionado um **cabeçalho TCP (ou cabeçalho de transporte)**



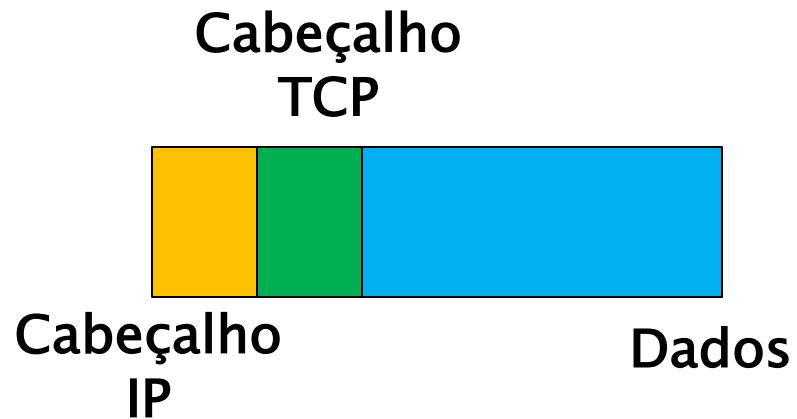
Redes de computadores

- ▶ Informações importantes que constituem o cabeçalho TCP
 - Número do pacote
 - Informações para verificação de integridade (checksum)
 - Porta de origem
 - Porta de destino
- ▶ A porta é uma identificação numérica utilizada para direcionar os dados para um programa desejado

Redes de computadores

- ▶ O protocolo IP é responsável pelo endereçamento de cada pacote para a máquina destino.
- ▶ Este endereçamento é feito por um identificador único atribuído à interface de rede de cada máquina: o endereço IP

- ▶ Um novo cabeçalho é adicionado ao pacote



Redes de computadores

- ▶ O protocolo IP, em sua versão 4 (IPv4) utiliza endereços numéricos de 32 bits que para maior comodidade são escritos na forma de 4 octetos, como por exemplo 200.231.191.10.
- ▶ Um endereço IPv4 é muito frequentemente chamado de também de *número IP* ou simplesmente de *IP*. Cada octeto corresponde a 8 bits do endereço, e pode portanto variar de 0 a 255.

Redes de computadores

- ▶ Recentemente, com a grande demanda da rede mundial, tem havido escassez de números IPs para identificar tantas máquinas
- ▶ Por este motivo já há uma versão 6 do protocolo IP (IPv6), onde são utilizados endereços de 128 bits

Sockets

Comunicação entre sockets

Sockets

- Em inglês, o termo socket é utilizado para representar uma tomada elétrica, que nada mais é que uma **extremidade de conexão** elétrica



Sockets

- ▶ Na computação os sockets são uma construção de software (**uma API**) que representa as extremidades de um fluxo de comunicação bidirecional
- ▶ Esta API é responsável por uma etapa de comunicação de nível intermediário, sendo que para isso, utiliza os protocolos TCP, UDP e IP.
- ▶ Assim, existem dois tipos de sockets:
 - Sockets de fluxo (TCP/IP)
 - Sockets de datagrama (UDP/IP)

Sockets

▶ Sockets de fluxo (TCP/IP)

- São utilizados para estabelecer uma **conexão fixa** entre dois processos (geralmente um processo servidor e um processo cliente)
- Enquanto a conexão estiver ativa, os dados fluem entre os processos em fluxos contínuos
- Oferece:
 - ✓ *Detecção de e recuperação de erros de transmissão*
 - ✓ *Controle de fluxo para* compatibilização de velocidade entre o transmissor e o receptor
 - ✓ *Garantia de ordem* no recebimento dos blocos de dados (pacotes)

Sockets

- ▶ Os sockets de fluxo são utilizados em aplicações que precisam de uma entrega segura e confiável dos dados
 - Transferência de arquivos
 - Instant messengers
 - Email
- ▶ Desvantagem: é mais lento que o socket de datagrama

Sockets

► Sockets de datagrama (UDP/IP)

- São utilizados para na transmissão de pacotes (blocos de dados) independentes.
- Não é estabelecida uma conexão entre processos
- Não garante nem a transmissão correta dos pacotes e nem que os pacotes cheguem na ordem em que foram enviados
- São utilizados quando o volume de informação é grande e qualidade da troca de dados não é um fator crítico
 - Exemplo: Transmissão de fluxos de vídeo e som pela Internet – **Youtube** (se um ou outro pacote for perdido o usuário vai perceber apenas pequenos ruídos no vídeo ou no som)

Trabalhando com Sockets em JAVA

Sockets de fluxo

Sockets de datagrama

Sockets de fluxo

- ▶ Ao utilizar sockets de fluxo precisamos estabelecer dois tipos de processo:
 - Um processo **servidor**
 - Um processo **cliente**
- ▶ O servidor é um processo que fica a espera de **solicitações de conexão**
 - Ao receber uma solicitação, ele mesmo cria e estabelece esta conexão
- ▶ O cliente é um processo que solicita a criação de uma conexão

Sockets de fluxo

- ▶ Este tipo de aplicação é denominada aplicação cliente-servidor:
 - Exemplos de servidores
 - Servidor de páginas HTTP
 - Servidor de hora
 - Servidor de mensagens instantâneas
 - Exemplos de clientes
 - Browser HTTP
 - Cliente MSN, ICQ
 - Software que corrige a hora do computador

Sockets de fluxo

- ▶ Passos para criação do servidor:

1 - Importar as classes do pacote java.net

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

2 - Criar um objeto da classe ServerSocket

```
ServerSocket serverSocket =  
    new ServerSocket (númeroDaPorta);
```

Sockets de fluxo

- ▶ O *númeroDaPorta* é utilizado para possibilitar que um cliente identifique o processo servidor no computador servidor
- ▶ Os números de porta podem estar entre 0 e 65.535
- ▶ Vários números de portas são reservados para processos servidores específicos. EX
 - porta 21 – servidor FTP
 - porta 80 – servidor HTTP (página Web)
- Os endereços de porta 0 a 1023 **são reservados para aplicações especiais.**
- O novo servidor deve escolher um número de porta maior que 1024

Sockets de fluxo

- ▶ Passos para criação do servidor:
 - 3 – Colocar o servidor para ficar aguardando solicitações de conexão:

```
Socket novaConexao = serverSocket.accept();
```

Neste ponto, a execução do aplicativo servidor fica parada, até receber uma solicitação para nova conexão.

Sockets de fluxo

- ▶ Passos para criação do servidor:
 - 4 – Após receber uma solicitação, criar streams de entrada e/ou saída de dados.

```
DataInputStream stream_entrada =  
    new DataInputStream(novaConexao.getInputStream());
```

```
DataOutputStream stream_saida =  
    new DataOutputStream(novaConexao.getOutputStream());
```

Sockets de fluxo

- ▶ Passos para criação do servidor:

5 – Enviar e receber dados pela nova Conexão

```
int k = stream_entrada.readInt();
String s = stream_entrada.readUTF() ;

stream_saida.writeInt( 3 );
stream_saida.writeUTF( "Hello from side A" );
```

Sockets de fluxo

- ▶ Passos para criação do servidor:

6 – fechar as streams de entrada e saída

```
stream_entrada.close();  
stream_saida.close();
```

7 – fechar a nova Conexão

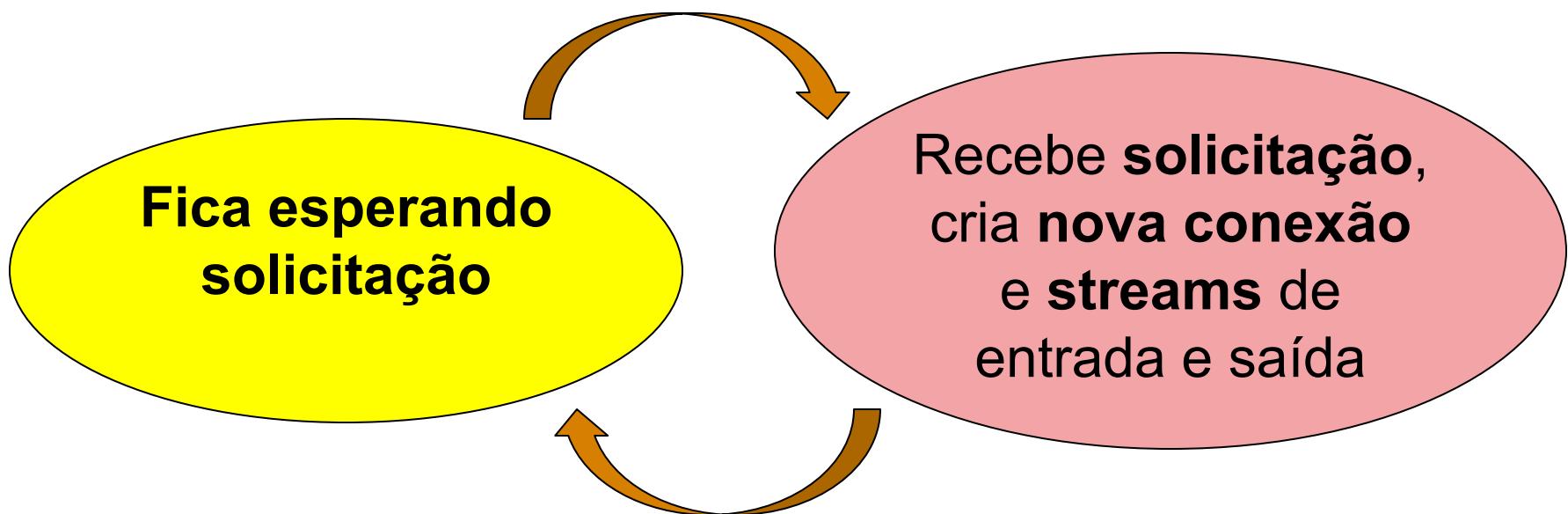
```
novaConexao.close();
```

Sockets de fluxo

- ▶ É interessante notar que:
 - Aquilo que um lado da nova conexão “gravar” na objeto stream de saída, ficará disponível para o outro lado da conexão no objeto stream de entrada

Sockets de fluxo

- Para que o servidor possa receber e tratar várias novas conexões é preciso colocar estas funções dentro de um loop, que ficará rodando continuamente



Sockets de fluxo

- ▶ Passos para criação do **cliente**:

1 – Solicitar conexão ao servidor (IP, porta)

Exemplo

```
Socket novaConexao =  
    new Socket("127.0.0.1", 12345);
```

Sockets de fluxo

- ▶ Passos para criação do **cliente**:

2 – Criar streams de entrada e/ou saída de dados.

```
DataInputStream stream_entrada =  
    new DataInputStream(novaConexao.getInputStream());
```

```
DataOutputStream stream_saida =  
    new  
    DataOutputStream(novaConexao.getOutputStream());
```

Sockets de fluxo

- ▶ Passos para criação do **cliente**:
- 3 – Enviar e receber dados pela nova Conexão

```
stream_saida.writeInt( 3 );
stream_saida.writeUTF( "Hello from side B" );

int k = stream_entrada.readInt();
String s = stream_entrada.readUTF() ;
```

Sockets de fluxo

- ▶ Passos para criação do **cliente**:

4 – fechar as streams de entrada e saída

```
stream_entrada.close();  
stream_saida.close();
```

5 – fechar a nova Conexão

```
novaConexao.close();
```

Sockets de fluxo

► Importante

- A maioria dos métodos da API sockets geram exceções que devem ser tratadas
 - *Circundar todas as chamadas com try...catch*
 - *Passar adiante a exceção : throws*

Sockets de fluxo

► Exercícios:

1- Construir um servidor de conselhos que fornece conselhos para um programa cliente, via sockets de fluxo, toda vez que lhe é solicitado uma nova conexão.

Ao receber o conselho, o aplicativo cliente exibe na tela o conselho, e logo após finaliza.

Conselhos (Servidor)

```
public static void main(String[] args) {  
    ServerSocket serverSocket;  
  
    try {  
        serverSocket = new ServerSocket(12345);  
    } catch (IOException ex) {  
        System.out.println("Não foi possível criar o serversocket");  
        return;  
    }  
  
    String conselhos [] = {"Programe um pouco todo dia",  
                          "Chegue sempre no horário certo",  
                          "Respeite os mais velhos",  
                          "Durma cedo e acorde cedo",  
                          "Coma pouco sal e açúcar"};  
  
    Random r = new Random();  
    int n = 0;  
  
    .  
    .  
    .
```

Conselhos (Servidor)

```
.  
. .  
  
while (true) {  
    try {  
        Socket novaConexao = serverSocket.accept();  
  
        DataOutputStream stream_saida = new  
                                         DataOutputStream(novaConexao.getOutputStream());  
  
        int i = r.nextInt(conselhos.length);  
        stream_saida.writeUTF("Conselho nº " + ++n + ":" + conselhos[i]);  
  
        stream_saida.close();  
        novaConexao.close();  
  
    } catch (IOException ex) {  
        System.out.println("Problemas de conexão com o servidor");  
        return;  
    }  
  
}
```

Conselhos (Cliente)

```
public static void main(String[] args) {  
    try {  
        Socket novaConexao = new Socket("localhost", 12345);  
  
        DataInputStream stream_entrada = new DataInputStream(novaConexao.getInputStream());  
  
        String mensagem = stream_entrada.readUTF();  
  
        System.out.println(mensagem);  
  
    } catch (IOException ex) {  
        return;  
    }  
}
```

Sockets de fluxo

► Interessante:

- Também é possível transmitir pela conexão **objetos**
- Para isso é preciso trabalhar com as classes :
 - **ObjectOutputStream**
 - **ObjectInputStream**

Sockets de fluxo

- ▶ Exemplo de envio de objetos serializáveis:

```
ObjectOutputStream saida_serial = new  
ObjectOutputStream(novaConexao.getOutputStream());  
  
saida_serial.writeObject(new Date());
```

Sockets de fluxo

- ▶ Exercícios:
 - 2- Construir um servidor de **hora certa** que fornece a hora do computador servidor (um objetos da classe date) para um programa cliente, via sockets de fluxo, toda vez que lhe é solicitado uma nova conexão.

Ao receber a hora certa, o aplicativo cliente exibe na tela o conselho, e logo após finaliza.

HoraCerta (Servidor)

```
public static void main(String[] args) {  
    ServerSocket serverSocket;  
  
    try {  
        serverSocket = new ServerSocket(12345);  
    } catch (IOException ex) {  
        System.out.println("Não foi possível criar o serversocket");  
        return;  
    }  
  
    .  
    .  
    .
```

HoraCerta (Servidor)

```
.  
. .  
.  
  
while (true) {  
    try {  
        Socket novaConexao = serverSocket.accept();  
  
        ObjectOutputStream saida_serial = new  
            ObjectOutputStream(novaConexao.getOutputStream());  
  
        saida_serial.writeObject(new Date());  
  
        saida_serial.close();  
        novaConexao.close();  
  
    } catch (IOException ex) {  
        System.out.println("Problemas de conexão com o servidor");  
        return;  
    }  
}  
}
```

Sockets de fluxo

- ▶ Para receber objetos

```
ObjectInputStream entrada_serial = new  
ObjectInputStream(novaConexao.getOutputStream());
```

```
Date horacerta = (Date) entrada_serial.readObject();
```



Aqui, temos que fazer uma conversão forçada de tipos, chamada “Casting”.

Isto porque, o readobject sempre retorna um objeto genérico (Object)

HoraCerta (Cliente)

```
public static void main(String[] args) {  
    try {  
        Socket novaConexao = new Socket("localhost", 12345);  
  
        ObjectInputStream entrada_serial = new  
                                         ObjectInputStream(novaConexao.getInputStream());  
  
        Date horacerta;  
  
        horacerta = (Date) entrada_serial.readObject();  
  
        System.out.println("A hora certa é: " + horacerta);  
  
    } catch (Exception ex) {  
        System.out.println("Não foi possível obter a hora certa");  
        return;  
    }  
}
```

HoraCerta (Cliente)

```
public static void main(String[] args) {  
    try {  
        Socket novaConexao = new Socket("localhost", 12345);  
  
        ObjectInputStream entrada_serial = new  
                                         ObjectInputStream(novaConexao.getInputStream());  
  
        Date horacerta;  
  
        horacerta = (Date) entrada_serial.readObject();  
  
        System.out.println("A hora certa é: " + horacerta);  
  
    } catch (Exception ex) {  
        System.out.println("Não foi possível obter a hora certa");  
        return;  
    }  
}
```

Sockets de fluxo

- ▶ Exercícios para casa

1- Programar e testar os dois exemplos

Servidor/Cliente: Conselhos e HoraCerta

Sockets de fluxo

- ▶ Exercícios para casa

2- Alterar o Servidor e o Cliente HoraCerta para que vários clientes possam ficar conectados ao Servidor recebendo de 1 em 1 segundo a hora certa

Para isso, o servidor deve iniciar uma nova thread toda vez que um cliente se conecta. Essa thread fica em um loop infinito enviando a hora certa de 1 em 1 segundo para o cliente. Do lado do cliente, depois de se conectar com o servidor, ele deve ficar num loop infinito, recebendo a hora certa e imprimindo na tela.