Sockets em Java - TCP

O exemplo abaixo mostra uma aplicação cliente servidor.
O cliente envia os dados recebidos do teclado e os envia ao servidor. Em seguida recebe uma mensagem do servidor e mostra no vídeo.
O servidor recebe mensagem do cliente, transforma as letras para maiúsculas e reenvia a mesma mensagem ao cliente.

O exemplo está no livro:
Redes de Computadores e Internet – uma nova abordagem
Kurose&Ross
Addison-Wesley

Fiz algumas adaptações:

TCPClient.java

```
// Abre conexão com o servidor
// Lê uma linha do teclado
// Envia a linha ao servidor
// Lê uma linha do servidor e mostra no vídeo
// Fecha conexão com o servidor

import java.io.*;  // classes para input e output streams
import java.net.*; // classes para socket, serversocket e clientsocket

class TCPClient {
  public static void main(String argv[]) throws Exception
 {
   String sentence;
   String modifiedSentence;

   // cria o stream do teclado
   BufferedReader inFromUser = new BufferedReader( new
        InputStreamReader(System.in));

   // cria o socket de acesso ao server hostname na porta 6789
   Socket clientSocket = new Socket("127.0.0.1", 6789);

   // cria os streams (encadeamentos) de entrada e saida com o servidor
   DataOutputStream outToServer =
         new DataOutputStream(clientSocket.getOutputStream());
   BufferedReader inFromServer =
         new BufferedReader(new inputStreamReader(
                     clientSocket.getInputStream()));

   // le uma linha do teclado e coloca em sentence
   sentence = inFromUser.readLine();

   // envia a linha para o server
   outToServer.writeBytes(sentence + '\n');
```

```
   // lê uma linha do server
   modifiedSentence = inFromServer.readLine();

   // apresenta a linha do server no vídeo
   System.out.println("FROM SERVER " + modifiedSentence);

   // fecha o cliente
   clientSocket.close();
  }
}
```

TCPServer.java

```
// Aguarda pedido de conexão de algum cliente
// Lê uma linha do cliente
// Transforma as minúsculas em maiúsculas
// Envia ao cliente
// Volta para o início

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

class TCPServer {
   public static void main(String argv[]) throws Exception
      {
         String clientSentence;
         String capitalizedSentence;

         // cria socket de comunicação com os clientes na porta 6789
         ServerSocket welcomeSocket = new ServerSocket(6789);

         // espera msg de algum cliente e trata
         while(true) {

            // espera conexão de algum cliente
            Socket connectionSocket = welcomeSocket.accept();

            // cria streams de entrada e saida com o cliente que chegou
            BufferedReader inFromClient =
               new BufferedReader(new InputStreamReader(
                  connectionSocket.getInputStream()));
            DataOutputStream outToClient =
               new DataOutputStream(
                  connectionSocket.getOutputStream());

            // lê uma linha do cliente
            clientSentence = inFromClient.readLine();

            // transforma a linha em maiúsculas
            capitalizedSentence =
                  clientSentence.toUpperCase() + '\n';

            // envia a linha maiúscula para o cliente
            outToClient.writeBytes(capitalizedSentence);
         }
      }
}
```

Mais exemplos

Para exemplificar esse mecanismo, vejamos um servidor e um cliente para uma aplicação de eco, isto é:

- servidor – recebe uma linha do cliente e devolve essa mesma linha para o cliente.
- cliente – espera o usuário digitar uma linha, envia essa linha para o servidor, recebe essa linha de volta do servidor e mostra no vídeo.

O exemplo abaixo e o seguinte estão no livro:
"Aprendendo Java 2"
Mello, Chiara e Villela
Novatec Editora Ltda. – www.novateceditora.com.br

```
// ServidorDeEco.java
import java.io.*;
import java.net.*;
public class ServidorDeEco {
 public static void main(String args[]) {
  try {
   // criando um socket que fica escutando a porta 2000.
   ServerSocket s = new ServerSocket(2000);
   // loop principal.
   while (true) {
    // Aguarda alguém se conectar. A execução do servidor
    // fica bloqueada na chamada do método accept da classe
    // ServerSocket. Quando alguém se conectar ao servidor, o
    // método desbloqueia e retorna com um objeto da classe
    // Socket, que é uma porta da comunicação.
    System.out.print("Esperando alguém se conectar...");
    Socket conexao = s.accept();
    System.out.println(" Conectou!");
    // obtendo os objetos de controle do fluxo de comunicação
    BufferedReader entrada = new BufferedReader(new
         InputStreamReader(conexao.getInputStream()));
    PrintStream saida = new
         PrintStream(conexao.getOutputStream());
    // esperando por alguma string do cliente até que ele
    // envie uma linha em branco.
    // Verificar se linha recebida não é nula.
    // Isso ocorre quando conexão é interrompida pelo cliente
    // Se a linha não for null(o objeto existe), podemos usar
    // métodos de comparação de string(caso contrário,estaria
    // tentando chamar um método de um objeto que não existe)
    String linha = entrada.readLine();
```

```
    while (linha != null && !(linha.trim().equals(""))) {
     // envia a linha de volta.
     saida.println("Eco: " + linha);
     // espera por uma nova linha.
     linha = entrada.readLine();
    }
    // se o cliente enviou linha em branco, fecha-se conexão.
    conexao.close();
    // e volta-se ao loop, esperando mais alguém se conectar
   }
  }
  catch (IOException e) {
   // caso ocorra alguma excessão de E/S, mostre qual foi
   System.out.println("IOException: " + e);
  }
 }
}
```

Vamos agora ao cliente correspondente. Para facilitar o teste do programa, vamos definir o endereço IP do servidor como **`127.0.0.1`**. Este endereço indica a máquina local. Portanto para testarmos abrimos 2 janelas do MSDOS. Em primeiro lugar lançamos o servidor numa das janelas e em seguida o cliente na outra. Experimente lançar o cliente antes do servidor.

```
// ClienteDeEco.java
import java.io.*;
import java.net.*;
public class ClienteDeEco  {
 public static void main(String args[]) {
  try {
   // para se conectar ao servidor, cria-se objeto Socket.
   // O primeiro parâmetro é o IP ou endereço da máquina que
   // se quer conectar e o segundo é a porta da aplicação.
   // Neste caso, usa-se o IP da máquina local (127.0.0.1)
   // e a porta da aplicação ServidorDeEco (2000).
   Socket conexao = new Socket("127.0.0.1", 2000);
   // uma vez estabelecida a comunicação, deve-se obter os
   // objetos que permitem controlar o fluxo de comunicação
   BufferedReader entrada = new BufferedReader(new
       InputStreamReader(conexao.getInputStream()));
   PrintStream saida = new
       PrintStream(conexao.getOutputStream());

   String linha;
   // objetos que permitem a leitura do teclado
```

```
    BufferedReader teclado =
       new BufferedReader(new InputStreamReader(System.in));
    // loop principal
    while (true) {
     // lê a linha do teclado
     System.out.print("> ");
     linha = teclado.readLine();
     // envia para o servidor
     saida.println(linha);
     // pega o que o servidor enviou
     linha = entrada.readLine();
     // Verifica se é linha válida, pois se for null a conexão
     // foi interrompida. Se ocorrer isso, termina a execução.
     if (linha == null) {
      System.out.println("Conexão encerrada!");
      break;
     }
     // se a linha não for nula, deve-se imprimi-la no vídeo
     System.out.println(linha);
    }
   }
   catch (IOException e) {
    // caso ocorra alguma excessão de E/S, mostre qual foi.
    System.out.println("IOException: " + e);
   }
  }
}
```

Qual o problema na solução acima?

Apenas um cliente por vez pode se conectar ao servidor. Imagine agora que em vez de um servidor de eco, tivéssemos um servidor de "chat" (bate papo). Vários clientes tinham que estar conectados ao mesmo tempo no servidor.

Para que isso possa ocorrer, a solução é usar a Thread. A linha de execução inicial dispara outra linha a cada novo cliente e fica esperando por novas conexões.

Um servidor e cliente de chat

Vamos modificar o servidor/cliente de eco acima, para um servidor/cliente de chat.

O servidor de chat deve aceitar conexão de um cliente, disparar uma thread para atender esse cliente e esperar por conexão de um novo cliente.

A thread que atende um cliente específico deve esperar que este envie uma mensagem e replicar esta mensagem para todos os clientes conectados. Quando esse cliente desconectar a thread deve avisar a todos os clientes conectados que isso ocorreu.

Portanto, é necessário que o servidor guarde em um vetor, todos os clientes conectados num dado instante.

```
// ServidorDeChat.java
import java.io.*;
import java.net.*;
import java.util.*;
public class ServidorDeChat extends Thread {
 public static void main(String args[]) {
  // instancia o vetor de clientes conectados
  clientes = new Vector();
  try {
   // criando um socket que fica escutando a porta 2222.
   ServerSocket s = new ServerSocket(2222);
   // Loop principal.
   while (true) {
    // aguarda algum cliente se conectar. A execução do
    // servidor fica bloqueada na chamada do método accept da
    // classe ServerSocket. Quando algum cliente se conectar
    // ao servidor, o método desbloqueia e retorna com um
    // objeto da classe Socket, que é porta da comunicação.
    System.out.print("Esperando alguem se conectar...");
    Socket conexao = s.accept();
    System.out.println(" Conectou!");
    // cria uma nova thread para tratar essa conexão
    Thread t = new ServidorDeChat(conexao);
    t.start();
    // voltando ao loop, esperando mais alguém se conectar.
   }
  }
  catch (IOException e) {
   // caso ocorra alguma excessão de E/S, mostre qual foi.
   System.out.println("IOException: " + e);
  }
 }

 // Parte que controla as conexões por meio de  threads.

 // Note que a instanciação está no main.
 private static Vector clientes;
 // socket deste cliente
 private Socket conexao;
 // nome deste cliente
 private String meuNome;
 // construtor que recebe o socket deste cliente
 public ServidorDeChat(Socket s) {
  conexao = s;
```

```
 }

 // execução da thread
 public void run() {
  try {
   // objetos que permitem controlar fluxo de comunicação
   BufferedReader entrada = new BufferedReader(new
        InputStreamReader(conexao.getInputStream()));
   PrintStream saida = new
        PrintStream(conexao.getOutputStream());
   // primeiramente, espera-se pelo nome do cliente
   meuNome = entrada.readLine();
   // agora, verifica se string recebida é valida, pois
   // sem a conexão foi interrompida, a string é null.
   // Se isso ocorrer, deve-se terminar a execução.
   if (meuNome == null) {return;}
   // Uma vez que se tem um cliente conectado e conhecido,
   // coloca-se fluxo de saída para esse cliente no vetor de
   // clientes conectados.
   clientes.add(saida);
   // clientes é objeto compartilhado por várias threads!
   // De acordo com o manual da API, os métodos são
   // sincronizados. Portanto, não há problemas de acessos
   // simultâneos.

   // Loop principal: esperando por alguma string do cliente.
   // Quando recebe, envia a todos os conectados até que o
   // cliente envie linha em branco.
   // Verificar se linha é null (conexão interrompida)
   // Se não for nula, pode-se compará-la com métodos string
   String linha = entrada.readLine();
   while (linha != null && !(linha.trim().equals(""))) {
    // reenvia a linha para todos os clientes conectados
    sendToAll(saida, " disse: ", linha);
    // espera por uma nova linha.
    linha = entrada.readLine();
   }
   // Uma vez que o cliente enviou linha em branco, retira-se
   // fluxo de saída do vetor de clientes e fecha-se conexão.
   sendToAll(saida, " saiu ", "do chat!");
   clientes.remove(saida);
   conexao.close();
  }
  catch (IOException e) {
   // Caso ocorra alguma excessão de E/S, mostre qual foi.
   System.out.println("IOException: " + e);
  }
```

```
  }

  // enviar uma mensagem para todos, menos para o próprio
  public void sendToAll(PrintStream saida, String acao,
   String linha) throws IOException {
   Enumeration e = clientes.elements();
   while (e.hasMoreElements()) {
    // obtém o fluxo de saída de um dos clientes
    PrintStream chat = (PrintStream) e.nextElement();
    // envia para todos, menos para o próprio usuário
    if (chat != saida) {chat.println(meuNome + acao + linha);}
   }
  }
}
```

O cliente deve aguardar o usuário digitar uma mensagem no teclado e enviar essa mensagem ao servidor.

Mas não é tão simples assim. Há um problema: mensagens podem chegar a qualquer momento do servidor e devem ser mostradas no vídeo. Se você pensou também em thread, acertou. Uma thread é lançada no início e fica esperando qualquer mensagem do servidor para apresentá-la no vídeo. A linha de execução principal do cliente se encarrega de esperar uma mensagem digitada pelo usuário e enviá-la para o servidor.

Da mesma forma que no exemplo anterior, vamos definir o endereço do servidor como **`127.0.0.1`**, isto é, é a máquina local. Neste caso como podem existir vários clientes, podemos abrir uma janela MSDOS para o servidor e várias janelas MSDOS para os clientes. Como antes o servidor deve ser lançado primeiro.

```
// ClienteDeChat.java
import java.io.*;
import java.net.*;
public class ClienteDeChat extends Thread {
 // Flag que indica quando se deve terminar a execução.
 private static boolean done = false;
 public static void main(String args[]) {
  try {
   // Para se conectar a algum servidor, basta se criar um
   // objeto da classe Socket. O primeiro parâmetro é o IP ou
   // o endereço da máquina a qual se quer conectar e o
   // segundo parâmetro é a porta da aplicação. Neste caso,
   // utiliza-se o IP da máquina local (127.0.0.1) e a porta
   // da aplicação ServidorDeChat. Nada impede a mudança
   // desses valores, tentando estabelecer uma conexão com
   // outras portas em outras máquinas.
   Socket conexao = new Socket("127.0.0.1", 2222);
```

```
   // uma vez estabelecida a comunicação, deve-se obter os
   // objetos que permitem controlar o fluxo de comunicação
   PrintStream saida = new
           PrintStream(conexao.getOutputStream());
   // enviar antes de tudo o nome do usuário
   BufferedReader teclado =
      new BufferedReader(new InputStreamReader(System.in));
   System.out.print("Entre com o seu nome: ");
   String meuNome = teclado.readLine();
   saida.println(meuNome);

   // Uma vez que tudo está pronto, antes de iniciar o loop
   // principal, executar a thread de recepção de mensagens.
   Thread t = new ClienteDeChat(conexao);
   t.start();

   // loop principal: obtendo uma linha digitada no teclado e
   // enviando-a para o servidor.
   String linha;
   while (true) {
    // ler a linha digitada no teclado
    System.out.print("> ");
    linha = teclado.readLine();
    // antes de enviar, verifica se a conexão não foi fechada
    if (done) {break;}
    // envia para o servidor
    saida.println(linha);
   }
  }
  catch (IOException e) {
   // Caso ocorra alguma excessão de E/S, mostre qual foi.
   System.out.println("IOException: " + e);
  }
 }

 // parte que controla a recepção de mensagens deste cliente
 private Socket conexao;
 // construtor que recebe o socket deste cliente
 public ClienteDeChat(Socket s) {
  conexao = s;
 }
 // execução da thread
 public void run() {
  try {
   BufferedReader entrada = new BufferedReader
     (new InputStreamReader(conexao.getInputStream()));
```

```
    String linha;
    while (true) {
     // pega o que o servidor enviou
     linha = entrada.readLine();
     // verifica se é uma linha válida. Pode ser que a conexão
     // foi interrompida. Neste caso, a linha é null. Se isso
     // ocorrer, termina-se a execução saindo com break
     if (linha == null) {
      System.out.println("Conexão encerrada!");
      break;
     }
     // caso a linha não seja nula, deve-se imprimi-la
     System.out.println();
     System.out.println(linha);
     System.out.print("...> ");
    }
   }
   catch (IOException e) {
    // caso ocorra alguma exceção de E/S, mostre qual foi.
    System.out.println("IOException: " + e);
   }
   // sinaliza para o main que a conexão encerrou.
   done = true;
  }
}
```

Um outro exemplo mais simples

Um servidor que recebe msgs de um cliente (news) e mostra a msg recebida.
Um cliente que a cada minuto envia a última msg digitada ao servidor. Assim que é digitada uma nova mensagem elea é enviada ao servidor.

```
// RecebeDoEnviaCliente.java

// Fica esperando um cliente enviar mensagens

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

class RecebeDoEnviaCliente {
   public static void main(String argv[]) throws Exception  {

         String LinhaCliente;

         // cria socket de comunicação com o cliente na porta 1234

         ServerSocket w = new ServerSocket(1234);

         // espera msg de algum cliente e trata

         // espera conexão de algum cliente
         System.out.println("Esperando algum cliente se conectar\n");
```

```
         Socket c = w.accept();

         System.out.println("Um cliente se conectou\n");

         // cria streams de entrada e saida com o cliente que chegou

         BufferedReader in = new BufferedReader(new
InputStreamReader(c.getInputStream()));

         DataOutputStream out = new
DataOutputStream(c.getOutputStream());

         while (true) {

            // lê uma linha do cliente
            LinhaCliente = in.readLine();

            //mostra a msg que chegou do cliente
            System.out.println("Chegou do cliente:" + LinhaCliente +
'\n');
         }

   }
}

// EnviaCliente.java
// main():  msg = "\n";
//          while(true) {espera(1 minuto); envia a msg;}

// Thread:  while(true) {espera msg ser digitada; envia a msg;}

import java.lang.Thread; //classes para Thread
import java.io.*;        // classes para input e output streams
import java.net.*;       // classes para socket, serversocket

public class EnviaCliente extends Thread {

   static String msg = "nada digitado ainda";

   public static void main(String argv[]) throws Exception {

     // cria o socket de acesso ao servidor na porta 1234
     Socket cliente = new Socket("127.0.0.1", 1234);

     // cria o duto de saida com o servidor
     DataOutputStream saida = new
DataOutputStream(cliente.getOutputStream());

     // Cria o thread
     Thread t = new EnviaCliente(saida);
     // Lança o thread criado
     t.start();

     while (true) {
        // espera 1 minuto
        Thread.sleep (60000);
```

```
        // envia msg ao servidor
        saida.writeBytes(msg + '\n');
      }
    }

    private DataOutputStream ss;

    public EnviaCliente (DataOutputStream s) {
      ss = s;
    }

    public void run() {
     try {
       // cria o stream do teclado
       BufferedReader teclado =
          new BufferedReader(new InputStreamReader(System.in));

       while (true) {
         // aguarda msg ser digitada
         System.out.println("\nDigite a msg: ");
         msg = teclado.readLine();
         // envia msg ao servidor
         ss.writeBytes(msg + '\n');
       }
     }
     catch (IOException e) {
       // Caso ocorra alguma excessão de E/S, mostre qual foi.
       System.out.println("IOException: " + e);
     }
    }
}
```