|  |  |
| --- | --- |
|  |  |

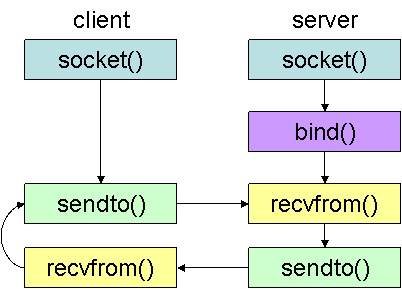
Disciplina de SD 2019.1– Aula Prática

Professor Dr. Windson Viana de Carvalho

Programação em Socket – Prática dos Jóqueis

**Introdução**

Nesta atividade, usaremos a API de sockets para comunicar com um servidor de um jogo de corrida de cavalos. Durante a atividade, usaremos vários tipos de métodos de troca de dados entre os sockets. Para tal, você deve criar um cliente Java, no compilador que você preferir, para se comunicar com o servidor do jogo.



|  |
| --- |
|  |
| Relembrando, para criar um cliente socket você deve adicionar a seguinte linha de código:  Socket clientSocket = new Socket("localhost", 80);  O construtor do objeto socket permite que o primeiro parâmetro seja uma S*tring* que indica o endereço IP e o segundo indica a porta em que o socket vai se conectar no servidor.  Para a comunicação, nós usaremos dois objetos, o de input e o de output. Eles representam formas mais simples de manipulação dos “canais stream” do sockets. O Java fornece uma ampla gama desses tipos de manipuladores de stream. Para começar, usaremos os mais comumente usados: o DataOuputStream e o DataInputStream.  DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());  DataInputStream inFromServer = new DataInputStream(clientSocket.getInputStream());  O parâmetro do construtor de ambos *streams* são sockets clientes criados previamente.  Provavelmente, você já ouviu da forma mais básica de BufferedReader para ler Strings ao usar a sua função readLine(), contudo a comunicação do tipo Data permite que você envie qualquer tipo de informação e não limita ela a apenas uma linha de caracteres.  Com tudo isso explicado, importe pro seu compilador o cliente de exemplo disponibilizado pelo professor (final do documento), procure e leia os comentários da função receberMensagemTCP(). Esta vai ser a função que você usará para receber as mensagens nessa atividade. Atenção a essa classe, pois o servidor vai esperar que você receba e interprete as informações por ele enviadas.  Esse modo de envio é padronizado, você, antes de enviar uma mensagem, deve mandar o tamanho dela. Dessa forma, o servidor sabe exatamente o que deve receber e quanto ele deve ler do seu buffer. Por exemplo:  outToServer.writeInt(sentence.length());  outToServer.writeBytes(sentence); |

**1- Exercício Básico**

Envie ao servidor o nome que você deseja ao seu Jóquei, e receba a mensagem de confirmação do servidor. Mostre um printscreen do que você recebeu e do código do cliente.

**2. Exercício Médio**

Envie ao servidor na sentença o nome que você deseja ao seu Jóquei e um número de 0 a 10 separado por uma vírgula. O servidor vai separar a mensagem e verificar o número, então leia a reposta que ele te mandou. Mostre aqui um printscreen do que você recebeu e do seu código.

**3. Exercício Avançado**

Envie ao servidor o nome que você deseja ao seu Jóquei. Em seguida, o servidor vai responder com uma mensagem que começa com “DESAFIO,” junto com um número aleatório (não tem espaço entre a mensagem e o número). Esse número representa o desafio. Você deve escrever no programa um código que multiplica esse número por 10 e então o dividi por 15 (o número resultado deve ter a precisão de um DOUBLE). Em seguida, o cliente deve enviar ao servidor esse número calculado. Para o envio, utilize o seguinte formato:

*“RESPOSTA,” + \*número\**

Para o servidor, ele irá responder com uma mensagem RIGHT caso tenha acertado ou WRONG caso o contrário tenha ocorrido.

package socket;

import java.io.\*;

import java.net.\*;

class TCPClientBasico {

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

String sentence;

String modifiedSentence;

BufferedReader inFromUser = new BufferedReader(new InputStreamReader(System.in));

System.out.println ("Abrindo conexão na porta 80 ");

/\*Começando por declarar um socket Cliente. Uma String com o IP e a Porta do servidor com o qual ele deve se conectar\*/

Socket clientSocket = new Socket("localhost", 80);

/\*Streams são os canais de troca de dados que um socket possui. Dois tipos principais, os de saida(output) e os de entrada(input). Os valores enviados

\* podem ser de string, de inteiros, de bytes ou objetos java. Mais especificamente os DataInputStream e DataOutputStream

\* servem para troca de dados em byte[]\*/

DataOutputStream outToServer = new DataOutputStream(clientSocket.getOutputStream());

DataInputStream inFromServer = new DataInputStream(clientSocket.getInputStream());

System.out.println ("Digite a mensagem e tecle enter");

sentence = inFromUser.readLine();

/\* Uma técnica comum em conexões sockets é enviar, antes da mensagem em si, o tamanho dela. O que você quer enviar

\* nem sempre pode ser resumido em apenas uma linha a ser lida por um BufferedReader (e.g., usando readLine()). Com o envio do tamanho, o programa

\* não corre o risco de fazer buffers grandes para receber os dados do stream

\*/

outToServer.writeInt(sentence.length()); // enviando o tamanho da cadeia

outToServer.writeBytes(sentence); // enviado a cadeia

modifiedSentence = receberMensagemTCP(inFromServer); //recebendo do servidor com o método escrito abaixo

System.out.println("FROM SERVER: " + modifiedSentence);

clientSocket.close(); //fechando a conexão

}

/\*Esse é um método de recepção de mensagens, imagine que a mensagem foi enviada nos mesmos moldes das mensagens acima

i.e. tamanho da mensagem + mensagem\*/

public static String receberMensagemTCP(DataInputStream stream) throws IOException{

byte[] message = null;//Bytes que irão conter a mensagem, chamamos isso de buffer

int length = stream.read(); // o primeiro int da mensagem e o tamanho dela, útil pra mensagens que são de mais de uma linha

if(length>0) {//Se tiver uma mensagem...

message = new byte[length];//Inicializar o Array de bytes do tamanho da mensagem

stream.readFully(message, 0, message.length); //Ler os bytes do stream até chegar no tamanho indicado pelo primeiro int

}

/\*Como vamos focar principalmente em mensagens de formato String, temos ainda que converter o objeto byte[] para um to tipo

String, por sorte o String tem um construtor especifico para isso\*/

String resultado = new String(message,"ASCII");//Converter de byte[] para um String usando o encoding ASCII

return resultado;//Retorno da função

}

}