Sockets com manipulação de exceptions

Artigo:

Sockets programming in Java: A tutorial

Writing your own client/server applications can be done seamlessly using Java

Qusay H. Mahmoud

http://www.javaworld.com/javaworld/jw-12-1996/jw-12-sockets.html

Sockets – Objetivo de prover a mesma interface de arquivos (open - read/write – close) para a comunicação entre processos

TCP

Como o TCP é orientado a conexão tem uma fase intermediária entre o open e o read/write.

No servidor:

- aguardar um pedido de conexão de algum cliente e
- criação do duto (byte stream) com o cliente

No cliente:

- criação do duto (byte stream) com o cliente

1) Abrir um socket cliente

```
Socket MyClient;
MyClient = new Socket("Machine name", PortNumber);
```

Machine name - o servidor ou a máquina a ser acessada
PortNumber – é o número da porta no servidor que define a aplicação que queremos usar
0 a 1023 – reservado para aplicações especiais (email, FTP, and HTTP, ...)
> 1024 – para ser usada para as aplicações normais

Com exception:

```
Socket MyClient;
try {
  MyClient = new Socket("Machine name", PortNumber);
}

catch (IOException e) {
  System.out.println(e);
}
```

2) Abrir um socket servidor

```
ServerSocket MyService;
try {
  MyService = new ServerSocket(PortNumber);
}
catch (IOException e){
   System.out.println(e);
}
```

3) Para o servidor esperar por uma conexão de algum cliente

```
Socket serviceSocket = null;
try {
  serviceSocket = MyService.accept();
}
catch (IOException e) {
   System.out.println(e);
}
```

4) Para o cliente criar o duto de entrada para receber dados do servidor

```
DataInputStream input;
try {
  input = new DataInputStream(MyClient.getInputStream());
}
catch (IOException e) {
   System.out.println(e);
}
```

5) Para o servidor criar o duto de entrada para receber dados do cliente

```
DataInputStream input;
try {
  input = new DataInputStream(serviceSocket.getInputStream());
}
catch (IOException e) {
  System.out.println(e);
}
```

6) Para o ler os dados (cliente ou servidor)

A classe `DataInputStream` tem algumas funções para ler os dados:

```
input.read()
input.readChar()
input.readInt()
input.readDouble()
input.readLine()
```

7) Para o cliente criar o duto de saida para enviar dados ao servidor

Pode ser usada a classe `PrintStream` or `DataOutputStream`

```
PrintStream output;
try {
  output = new PrintStream(MyClient.getOutputStream());
}
catch (IOException e) {
  System.out.println(e);
}
```

ou então

```
DataOutputStream output;
try {
  output = new DataOutputStream(MyClient.getOutputStream());
}
catch (IOException e) {
  System.out.println(e);
}
```

8) Para o servidor criar o duto de saida para enviar dados ao cliente

Também podem ser usadas `PrintStream` ou `DataOutputStream`.

```
PrintStream output;
try {
  output = new PrintStream(serviceSocket.getOutputStream());
}
catch (IOException e) {
  System.out.println(e);
}
```

ou então

```
DataOutputStream output;
try {
  output = new DataOutputStream(serviceSocket.getOutputStream());
}
catch (IOException e) {
  System.out.println(e);
}
```

9) Para enviar os dados (cliente ou servidor)

A classe `PrintStream` tem algumas funções para enviar os dados:

```
output.Write()
output.println()
```

Idem para a classe `DataOutputStream`.

```
output.writebytes()
```

10) Para fechar o lado cliente

```
try {
  output.close();
  input.close();
  MyClient.close();
}
catch (IOException e) {
  System.out.println(e);
}
```

11) Para fechar o lado servidor

```
try {
  output.close();
  input.close();
  serviceSocket.close();
  MyService.close();
}
catch (IOException e) {
  System.out.println(e);
}
```