UNIVERSIDADE FEDERAL RURAL DO SEMIÁRIDO - UFERSA
Departamento de Computação - DC
Graduação em Ciência da Computação
Disciplina: Sistemas Distribuídos
Prof.: Paulo Henrique Lopes Silva

Prática 1 - Sockets em Java

1. Sockets TCP

- Cliente

```
package sockets.tcp.cliente;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.Socket;
import java.net.UnknownHostException;
import java.util.Scanner;

public class ClienteSimples {

	private static Scanner sc;

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


		// Conectar ao servidor
		Socket socketCliente = new Socket("localhost", 5000);

		// Cria canal para receber dados
		DataInputStream fluxoEntrada = new
DataInputStream(socketCliente.getInputStream());

		// Cria canal para enviar dados
		DataOutputStream fluxoSaida = new
DataOutputStream(socketCliente.getOutputStream());

		sc = new Scanner(System.in);

		System.out.println("Digite uma mensagem: ");

		String msg = sc.nextLine();

		System.out.println("\n\n <-- Mensagem enviada ao servidor: "+ msg);

		fluxoSaida.writeUTF(msg); //Envia mensagem.

		msg = fluxoEntrada.readUTF(); //Aguarda o recebimento de uma
string.

		System.out.println("\n\n --> Mensagem recebida do servidor : "+ msg
+ "\n\n");

		//Fecha os canais de entrada e saída.
		fluxoEntrada.close();

		fluxoSaida.close();
```

```
            //Fecha o socket.
            socketCliente.close();


      }

}
```

- Servidor

```
package sockets.tcp.servidor;

import java.io.DataInputStream;
import java.io.DataOutputStream;
import java.io.IOException;
import java.net.ServerSocket;
import java.net.Socket;

public class ServidorSimples {

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


            //Cria um socket servidor na porta 5000
        ServerSocket serverSocket=new ServerSocket(5000);

        System.out.println("\n\nIniciando servidor TCP...\n");

        System.out.println("Servidor pronto para receber conexões...\n\n");

        // O metodo accept retorna um socket para comunicação com o próximo
        // cliente a conectar.
        // A execução do método bloqueia até que algum cliente conecte no
        // servidor.
        Socket socket = serverSocket.accept();

        // imprime o ip do cliente
        System.out.println("Nova conexão com o cliente " +
socket.getInetAddress().getHostAddress());

        //Cria um canal para receber dados.
        DataInputStream fluxoEntrada = new
DataInputStream(socket.getInputStream());

        //Cria um canal para enviar dados.
        DataOutputStream fluxoSaida = new
DataOutputStream(socket.getOutputStream());

        String msg = fluxoEntrada.readUTF(); //Aguarda o recebimento de uma
string.

        System.out.println("--> Mensagem recebida do cliente: " + msg);

        msg = "Bem vindo e tchau!";

        System.out.println("--> Servidor enviando mensagem: " + msg);

        fluxoSaida.writeUTF(msg); //Envia uma string.
```

```
        //Fecha os canais in e out do socket que estão atendendo ao cliente
        fluxoEntrada.close();

        fluxoSaida.close();

        //Fecha o socket para o cliente.
        socket.close();

        System.out.println("*****Conexão finalizada*****\n");

        //Fechando o servidor.
        serverSocket.close();

      }

}
```

Exercícios:

1. Crie um projeto no Eclipse, codifique, execute e observe o funcionamento dessa aplicação cliente/servidor simples.

2. Crie uma aplicação onde o cliente envia uma mensagem ao servidor e a recebe de volta invertida.

3. Crie uma aplicação onde o cliente envia um valor numérico ao servidor. Se o valor for par, o servidor multiplica-o por 10 e envia o retorno ao cliente. Se o valor for ímpar, o servidor multiplica-o por 11 e envia o retorno ao cliente. Se o valor for zero, o servidor envia o valor zero por extenso ao cliente.