UNIVERSIDADE ESTADUAL DO OESTE DO PARANÁ
CENTRO DE CIÊNCIAS EXATAS E TECNOLÓGICAS
COLEGIADO DE INFORMÁTICA

Disciplina: Redes de Computadores **Ano:** 2007
Professor: Luiz Antonio

Trabalho 1º Bimestre – Cliente/Servidor de Bate-Papo

Observações: Verificar alterações no trabalho em vermelho.

Contexto: Camada de Aplicação. **Valor:** 4,0
Realização: em duplas.
Entrega e apresentação: 19/outubro

1. Descrição do Trabalho

O trabalho consiste em implementar dois programas: um cliente e um servidor de bate-papo. O servidor deve aceitar a conexão de vários clientes simultâneos. As mensagens recebidas pelo Servidor devem ser replicadas a todos os clientes conectados, exceto nos casos de mensagens reservadas.

O servidor deve informar a todos os participantes sobre entradas e saídas de participantes. Portanto, é necessário que o servidor guarde uma lista de todos os clientes conectados num dado instante.

Cada cliente deverá ser identificado por um nome. Este nome deverá ser informado quando o cliente entrar no bate-papo, enviar uma mensagem ou quando se desconectar, de acordo com o exemplo:

Fulano conectou.
Fulano disse: <mensagem>
Fulano disse reservadamente: <mensagem>
Fulano desconectou.

A aplicação (cliente e servidor) deve implementar um protocolo padrão, de modo que qualquer cliente possa se conectar a qualquer servidor.

Os trabalhos devem ser desenvolvidos em Java.

2. Definição do Protocolo

Cliente: um cliente é qualquer programa conectado ao servidor, desde que não seja outro servidor. Um cliente é identificado por um nome único (*username*) composto por, no máximo, 10 caracteres. O *username* pode conter apenas letras, números e _ (sublinhado).

Canal: nesta implementação existe um único canal (“sala” do bate-papo). Todos os clientes conectados compartilham o mesmo canal, não havendo interação para entrada ou saída de um canal específico.

Protocolo: O protocolo prevê os seguintes comandos e respectivos parâmetros (baseados na RFC 1459 que define o IRC (*Internet Relay Chat Protocol*) - http://tools.ietf.org/html/rfc1459):

SERVER <ip_servidor> [<porta>]

Comando para solicitar a conexão com o Servidor disponível no endereço IP informado. A porta é opcional, sendo que a porta padrão é 5588. O próximo comando aceito é o USER. Qualquer outro comando implica na desconexão do Cliente no Servidor.

USER <username>

É o comando para identificar o Cliente no Servidor. O nome do Cliente será armazenado no Servidor para futuras referências. Este comando pode ser utilizado a qualquer momento para trocar o nome do usuário no servidor.
Respostas possíveis do Servidor:

- `OK_USERNAME <username>`: nome de usuário informado corretamente. O servidor responde com o nome informado.
- `ERR_INVALIDUSERNAME <username>`: nome de usuário inválido. O servidor responde com o nome informado.
- `ERR_NEEDMOREPARAMS`: o parâmetro <username> está ausente.
- `ERR_ALREADYREGISTRED <username>`: o nome de usuário informado já está registrado no Servidor. O servidor responde com o nome informado.

MSG <mensagem>

Comando usado pelo Cliente para enviar uma mensagem ao Servidor.

MSG_SENDED <nome_emissor> <mensagem>

Comando enviado pelo Servidor e recebido por todos os clientes (inclusive o emissor) contendo o nome do Cliente emissor e a mensagem enviada pelo mesmo. ~~No caso de mensagens reservadas, apenas o Cliente destino deve receber esta mensagem (ver próx. comando)~~.

PRIVMSG <username_destino> <mensagem>

Permite que um cliente converse reservadamente com outro. A confirmação do envio é dada pelo `PRIVMSG_SENDED` (ver comando seguinte).
Respostas possíveis do servidor:

- `ERR_NOSUCHNICK <username>`: `<username_destino>` não encontrado no servidor.
- `NOTEXTTOSEND`: falta parâmetro com a mensagem a ser enviada.

PRIVMSG_SENDED <nome_emissor> <mensagem>

Comando enviado pelo Servidor ao cliente especificado pela mensagem `PRIVMSG` contendo o nome do Cliente emissor e a mensagem enviada pelo mesmo.

NAMES

Comando utilizado pelos clientes para obter a lista de todos os clientes conectados naquele momento.

NAMES <cliente1> <cliente2> ... <clienteN>

Resposta enviada pelo Servidor para o cliente solicitante ao receber a solicitação `NAMES` (comando anterior) com a lista de clientes conectados.

QUIT [<mensagem_saida>]

Comando do cliente para deixar o bate-papo. O servidor deve enviar a informação de saída para todos os demais participantes do bate-papo (inclusive para o próprio cliente que está deixando o programa). Opcionalmente pode ser informada uma mensagem de saída (ver próx. comando).

QUIT <nome_cliente> [<mensagem_saida>]

Comando enviado pelo Servidor aos participantes quando um cliente qualquer deixa o bate-papo (ver comando anterior). O Cliente que está deixando o sistema recebe este comando e termina a conexão com o Servidor.

Anexo I - Código de Apoio

Cliente/Servidor de Eco em Java utilizando Sockets TCP

Servidor.java

```
import java.io.*;
import java.net.*;
public class Servidor {
	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(" cliente "
				+conexao.getInetAddress().getHostAddress()
					+" 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(""))) {
					//imprime a msg na tela
					System.out.println("Recebeu: "+linha);
					// 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);
		}
	}
}
```

Cliente.java

```
import java.io.*;
import java.net.*;
public class Cliente {
	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);
		}
	}
}
```