Relatório do EP1

Alexandre Kenji Okamoto	11208371
Fernanda Cavalcante Nascimento	11390827
Gabriel Afonso Carnaiba Silva	11270886
Karina Duran	11295911
Mirela Mei	11208392

1. Descrição

O presente relatório tem como objetivo apresentar e analisar o EP de desenvolvimento que foi entregue em conjunto.

Este projeto tem como tema a camada de aplicação, e seu objetivo é servir como um sistema de pizzaria. Ele é composto por dois programas: um para uso interno da pizzaria e outro para uso externo pelos clientes. Ambos são sistemas desktop (terminal), e sua comunicação acontece via servidor.

O programa de uso da pizzaria é responsável por gerenciar informações como a disponibilidade das mesas (alocar e desalocar clientes), disponibilidade de itens do cardápio, além de enviar todas essas informações para o servidor, a fim de serem recebidas e utilizadas pelo programa dos clientes, assim como gerenciar as informações recebidas deles pelo servidor.

Já o programa utilizado pelos clientes, por sua vez, permite que os mesmos solicitem mesas, realizem pedidos, solicitem o fechamento de suas comandas para realizarem o pagamento e liberem as mesas, além de enviar todas essas informações para o servidor para que sejam administradas pelo programa da pizzaria.

2. Detalhes

Dado que o tema do projeto é a camada de aplicação, o protocolo de comunicação utilizado foi o TCP. No código, cada cliente se comunica com o servidor usando um socket, e existe uma thread para cada cliente.

O protocolo TCP, que é um protocolo confiável da camada de transporte, tem como objetivo garantir a transmissão integral e sequencial dos dados para os hosts de destino.

Esse protocolo segmenta as informações recebidas da camada de aplicação em pequenos blocos (datagramas) e insere um cabeçalho de identificação contendo um conjunto de bits (checksum) que fazem possível uma validação dos dados e do próprio cabeçalho. Além da validação, o checksum também permite a recuperação da informação em casos de erros na transmissão.

O host de destino envia, após receber os pacotes, um ACK para o host de origem, a fim de que este tenha a garantia de que eles chegaram sem erros.

3. Como compilar

O projeto foi desenvolvido utilizando a linguagem de programação Java, no ambiente de desenvolvimento Visual Studio Code. A classe main do servidor está no arquivo “PizzariaServer.java”, enquanto que a classe main do cliente está no arquivo “PizzariaClient.java”. Para compilar o código, basta abrir um terminal na pasta "src" e executar o comando "javac *.java". Entretanto, a pasta “bin” já possui todos os arquivos compilados para agilizar o processo, caso necessário.

4. Como executar

Para executar o programa, é necessário executar o arquivo *pizzariaServer.jar* para iniciar o lado do servidor, onde será mostrada a mensagem “Servidor aberto na porta 4000”.

Após isso, para iniciar o lado do cliente é necessário executar o arquivo *pizzariaClient.jar*, fazendo com que o menu de comandos seja exibido.

(Obs.: Para executar no Windows, os arquivos são: *servidor.jar* e *cliente.jar*, dentro da pasta /bin)

Deve-se então, no lado do cliente, digitar o número 1, para realizar a reserva da mesa, indicando a quantidade de pessoas. É possível visualizar os comandos recebidos pelo cliente no terminal do servidor.

Após a reserva da mesa, digitando o número 2 será enviado o cardápio para realizar o pedido. É preciso digitar o número das pizzas desejadas, uma a uma. Depois digitar 0 para finalizar o pedido, voltando para o menu inicial.

Após isso, há a possibilidade de fazer outros pedidos (que serão somados na conta final) ou fechar a conta, para realizar o pagamento e liberar a mesa (opção de número 3).

Então, depois de efetuar o pagamento, deve-se avaliar o atendimento (número 4), e desconectar o cliente (número 5).

5. Como ler o código

- **PizzariaServer:** onde está o método *main* do Server, que irá instanciar o PizzariaServer e abrirPizzaria. A porta utilizada será de número 4000 e há um LinkedHashSet<ClientSocket> para guardar os clientes que irão se conectar.
  O método *start* irá instanciar o ServerSocket e chamará o método *clientConnectionLoop* para gerar um loop infinito, instanciar o ClientSocket e criar threads conforme os clientes iniciam a conexão.
  O método *clientMessageLoop* recebe um ClientSocket como parâmetro e verifica a ocorrência de novos comandos do lado do cliente, para realizar a lógica necessária conforme o número enviado.
  O método *enviarMsgCliente* irá enviar os comandos recebidos pelo cliente, e o *cardapio* irá imprimir as pizzas.

- **PizzariaClient:** onde está o método *main* do Cliente, que irá instanciar o pizzariaClient e chamará o método start. O IP será 127.0.0.1.
  O método *start* irá instanciar o ClientSocket com o IP mencionado e a porta declarada no Server.
  O método *run* aguarda os comandos do cliente e imprime a mensagem.
  O método *messageLoop* irá processar os comandos recebidos e enviar a mensagem para o servidor realizar a lógica necessária. Os métodos *menu*, *ajuda* e *solicitarCardapio* realizam o básico para a impressão do menu inicial.

- **Restaurante:** classe para instanciar o restaurante, possui uma variável List para armazenar mesas, cardapio, pedidos, clientesConectados e avaliacoesPizzaria. O método *GetRestauranteSingleton* irá garantir apenas uma instância de restaurante sendo utilizada.
  O método *abrirPizzaria* limpará as mesas, cardapio, clientes e pedidos, para adicioná-los novamente diante da nova conexão.
  O método *reservarMesa* irá realizar a lógica de reserva, garantindo que ainda existam mesas disponíveis e que o cliente esteja conectado.
  O método *adicionarAoPedido* irá instanciar as pizzas pedidas pelo cliente e adicionar a lista. O método *clienteEntrou* adiciona um novo cliente ao restaurante.
  O método *adicionarPedidoCliente* irá adicionar o pedido ao ID do cliente, para somar ao final e o método *fecharPedido* irá fechar a conta e o cliente, liberando a mesa. O método *adicionarAvaliacao* salva a avaliação do cliente, e o método *acharCliente* busca o cliente pelo ID.

- **Cliente:** classe para instanciar o cliente. Possui um ID único, um Pedido, mesaReservada e uma tag possuiReserva para indicar se há uma mesa instanciada em seu ID.

- **ClientSocket:** classe do Socket do cliente. Realiza as operações de *getMessage*, *sendMessage*, *getRemoteSocketAddress*, *getLocalPort* e *close*, para finalizar o cliente.

- **MesasPizzaria:** classe para instanciar uma mesa. Possui um int para quantidadeLugares, um String com o cliente que reservou, e uma tag para indicar se está disponível.

- **Pedido:** classe para instanciar um pedido novo. Possui um identificador do pedido, um ID do cliente que o fez, e um ArrayList para guardar os itens. O método *totalPedido* irá somar os itens e seus respectivos valores.

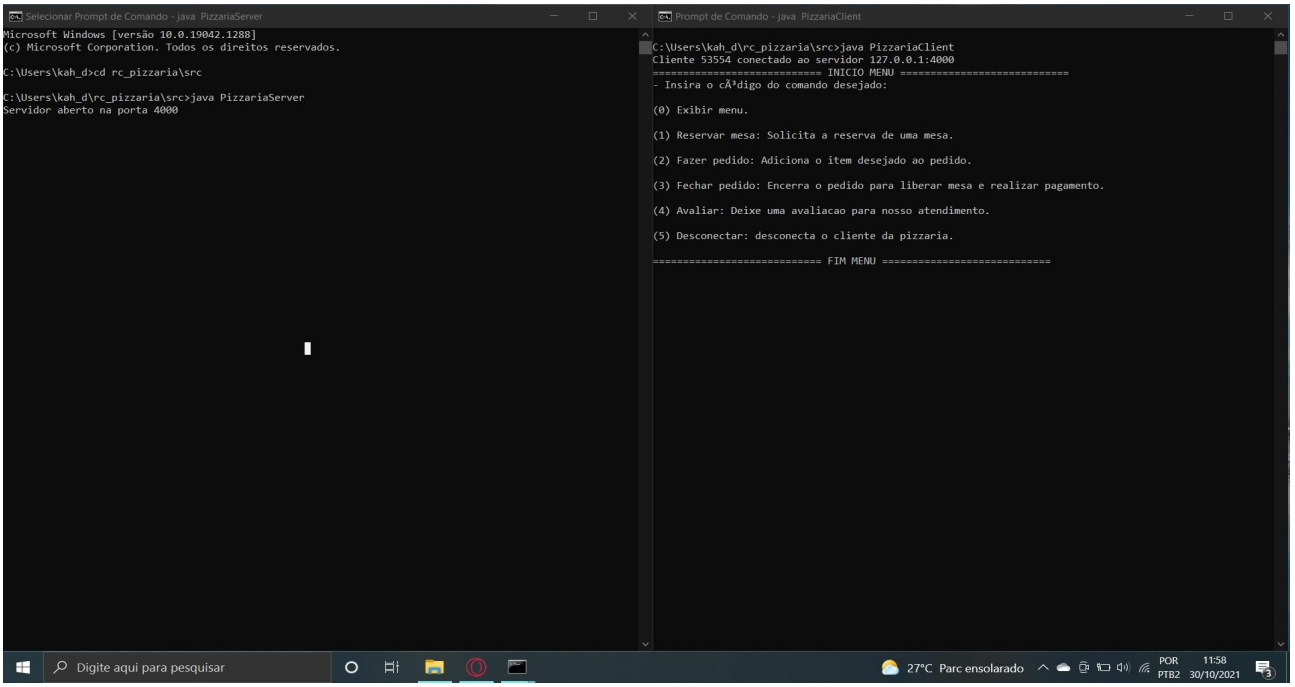
- **ItemCardapio:** classe para um item do cardápio, métodos *getSabor* e *getValor*.

- **Avaliacao:** classe para avaliações recebidas. Possui um int para nota e uma String para o comentário.

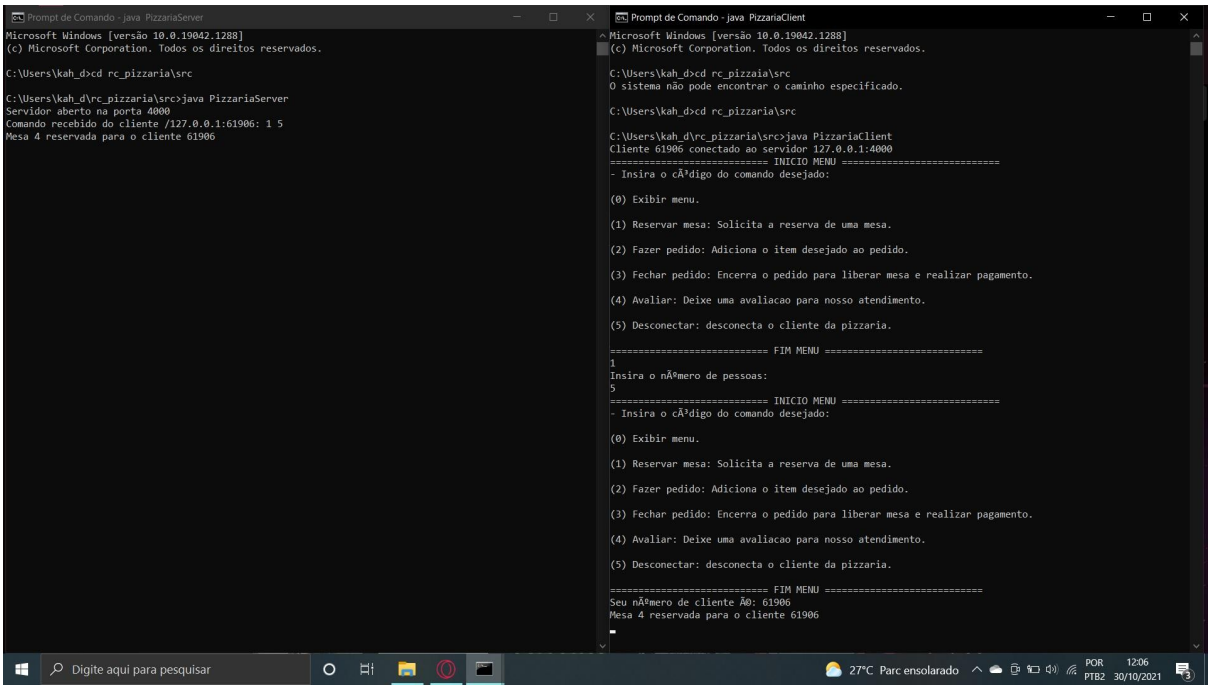
- **Pizza*:** classes com as pizzas disponíveis e seus respectivos valores.

6. Testes e resultados obtidos

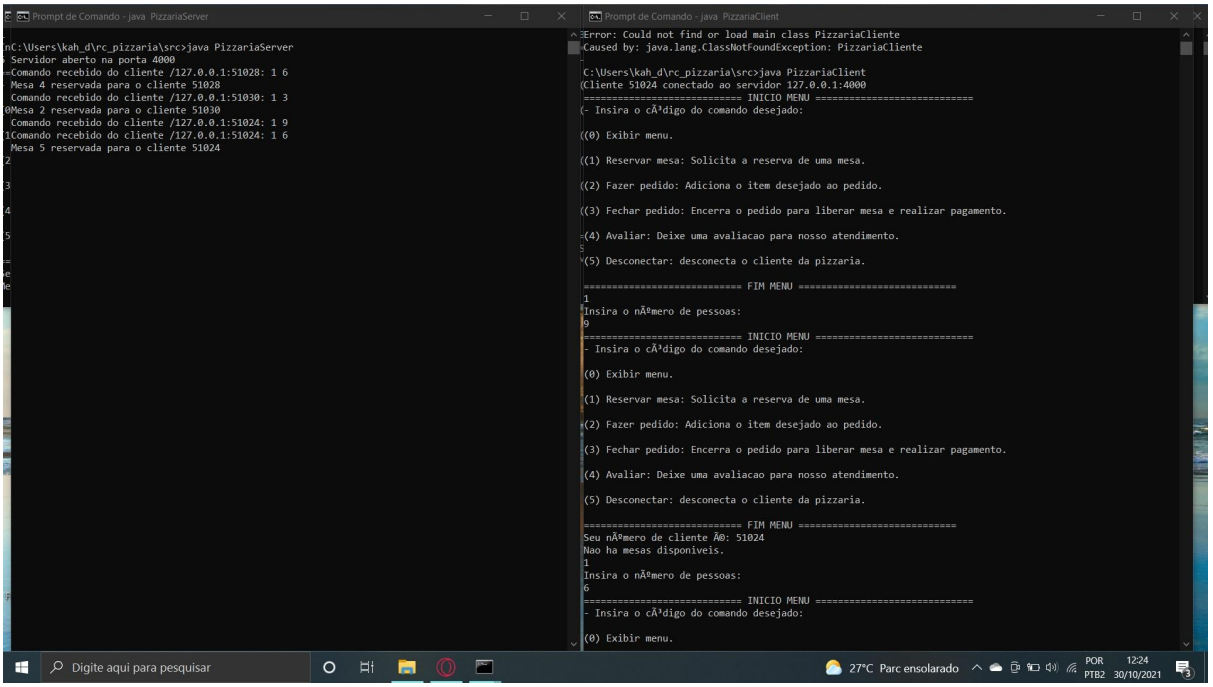
O teste inicial: Abertura do socket.



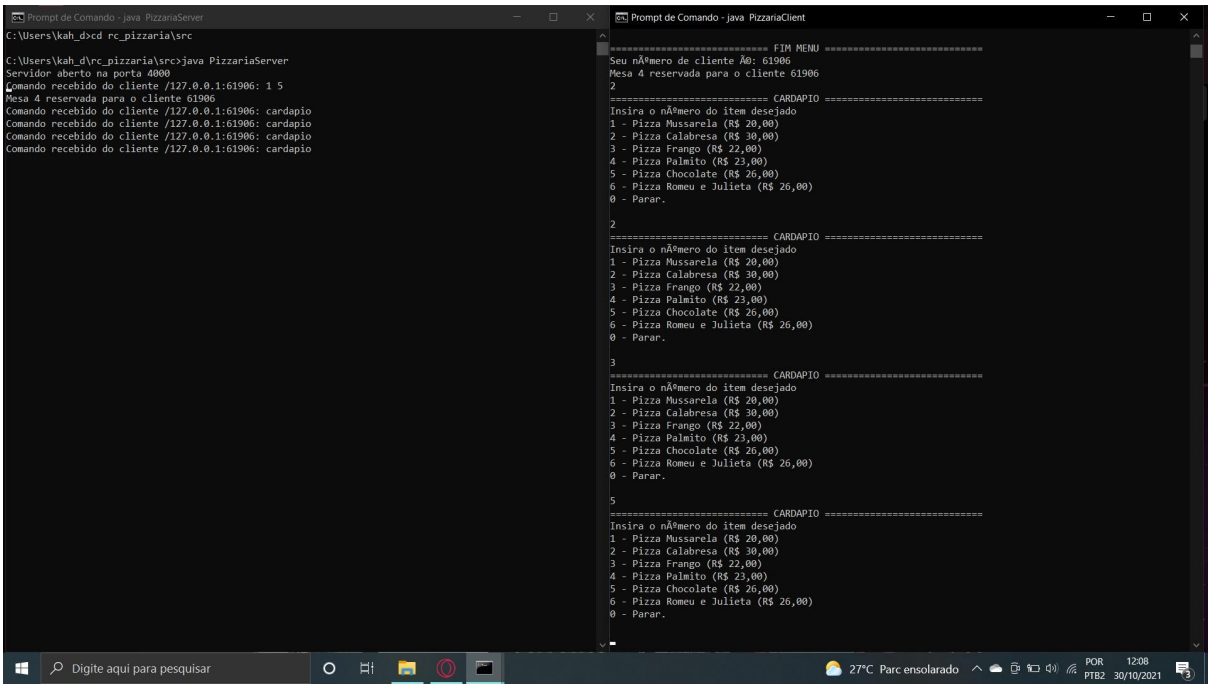
Teste para reserva de mesa para 5 pessoas



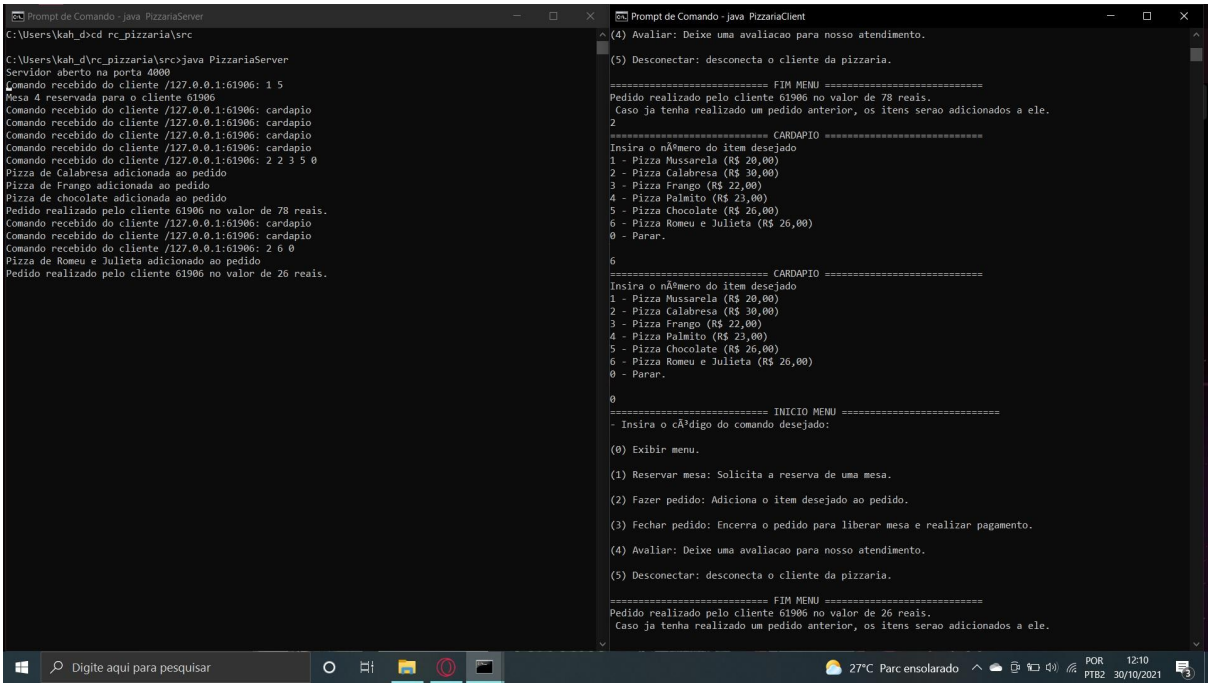
Teste com número excedente de pessoas(9, sendo o máximo 6):



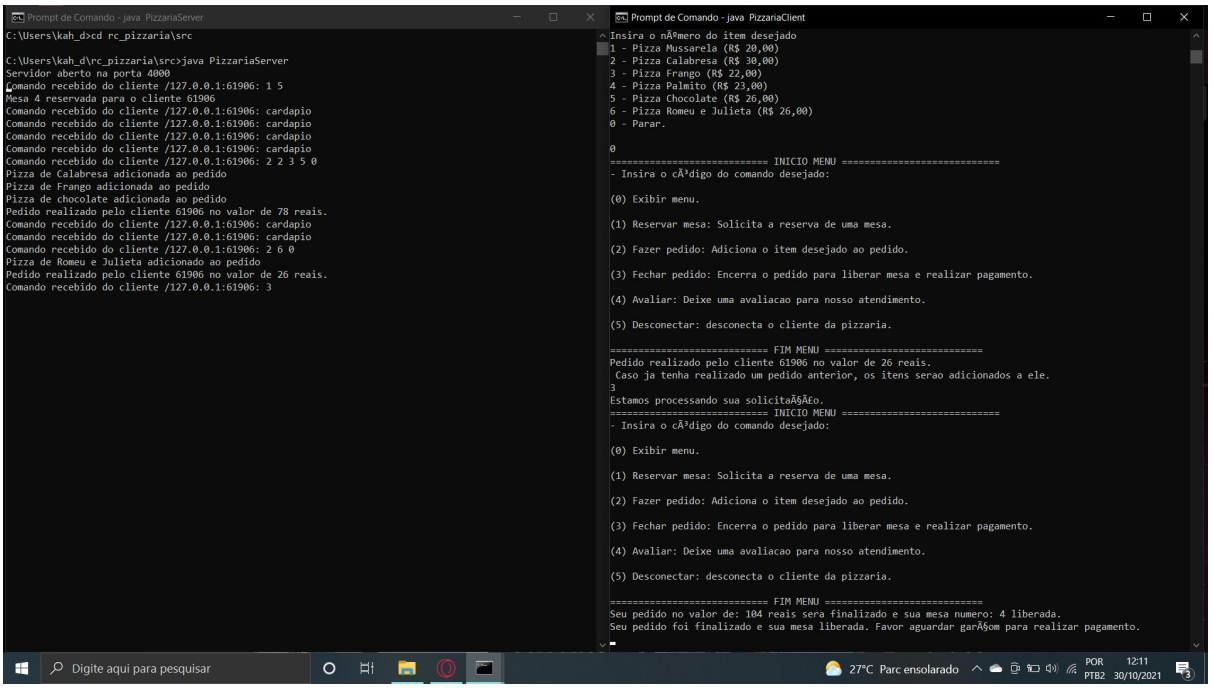
Teste: pedindo as pizzas



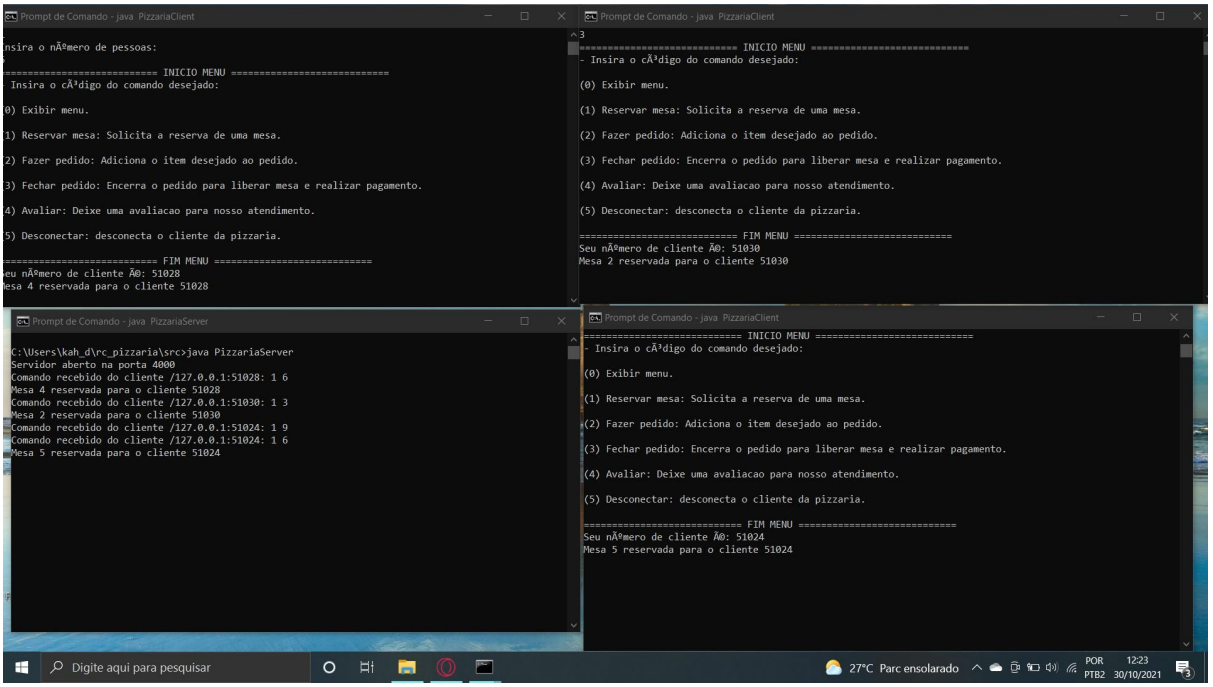
Teste: adicionando novos sabores a um pedido já existente:



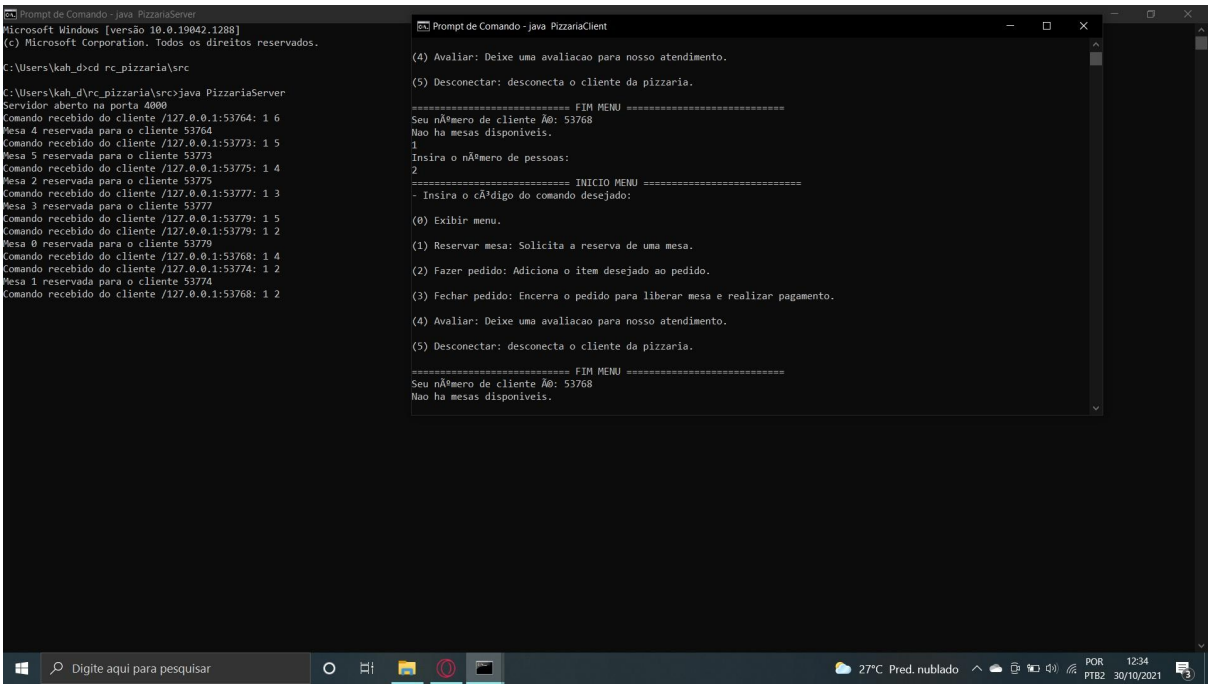
Teste: fechando o pedido:



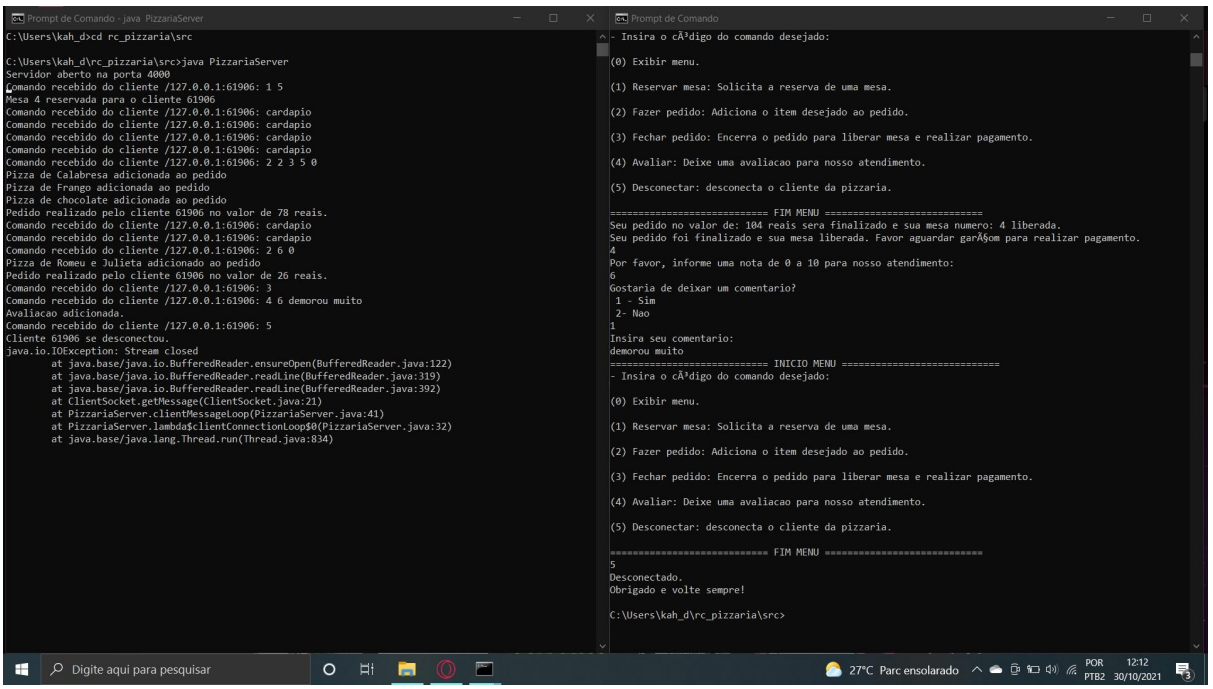
Teste com múltiplos clientes(3):



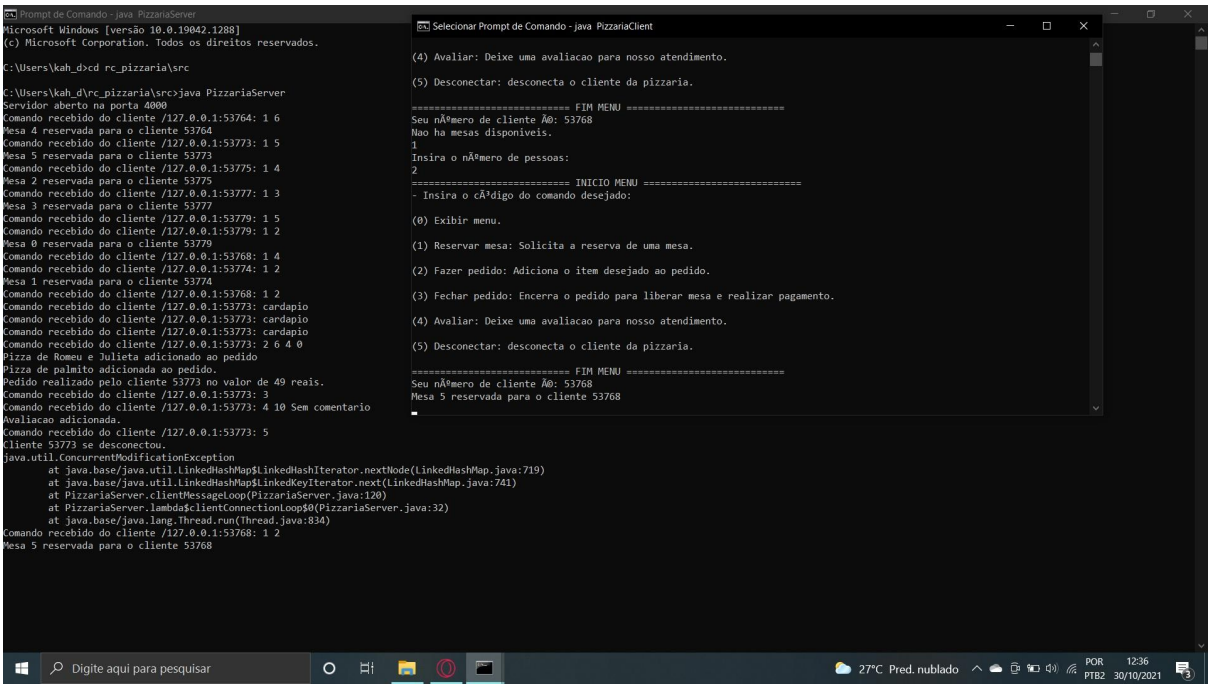
Teste: todas as mesas ocupadas.



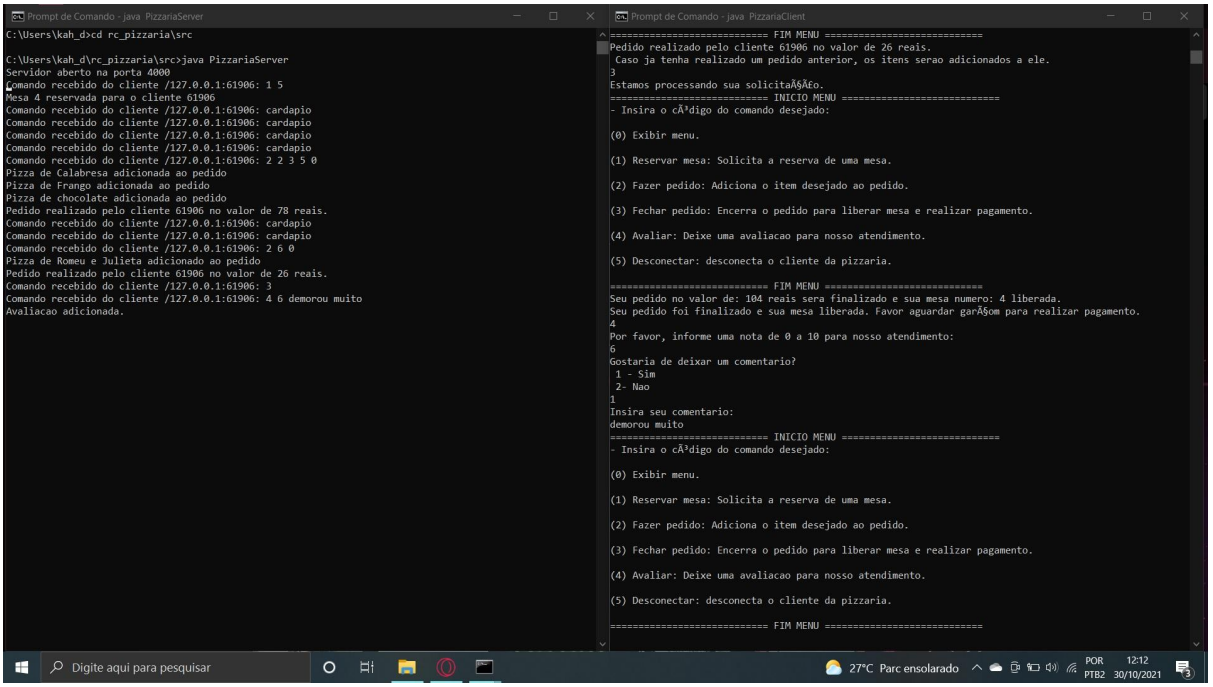
Teste: Cliente desconectado:



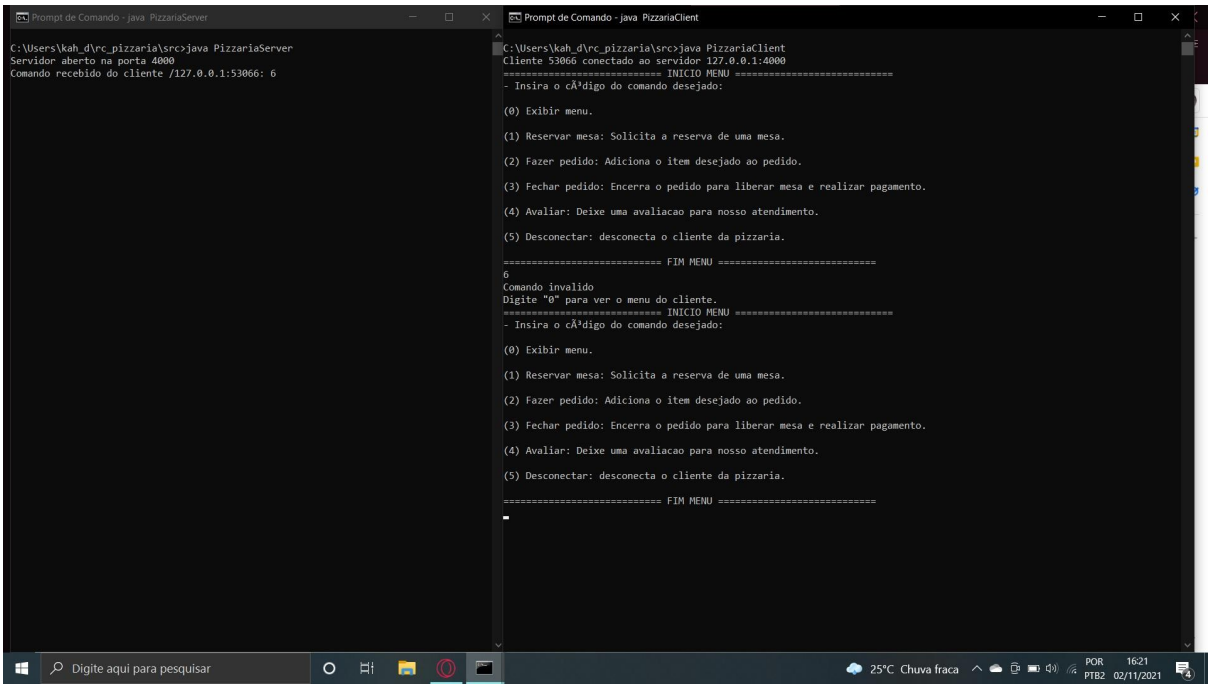
Teste: mesa liberada



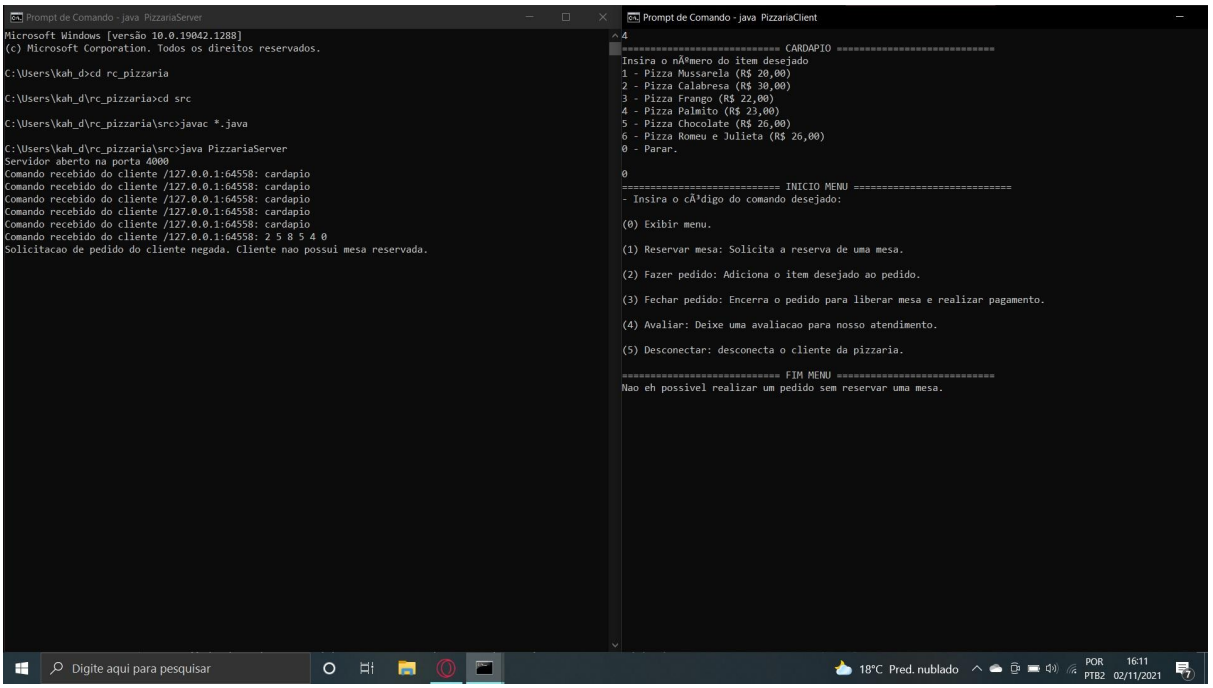
Teste: avaliação feita:



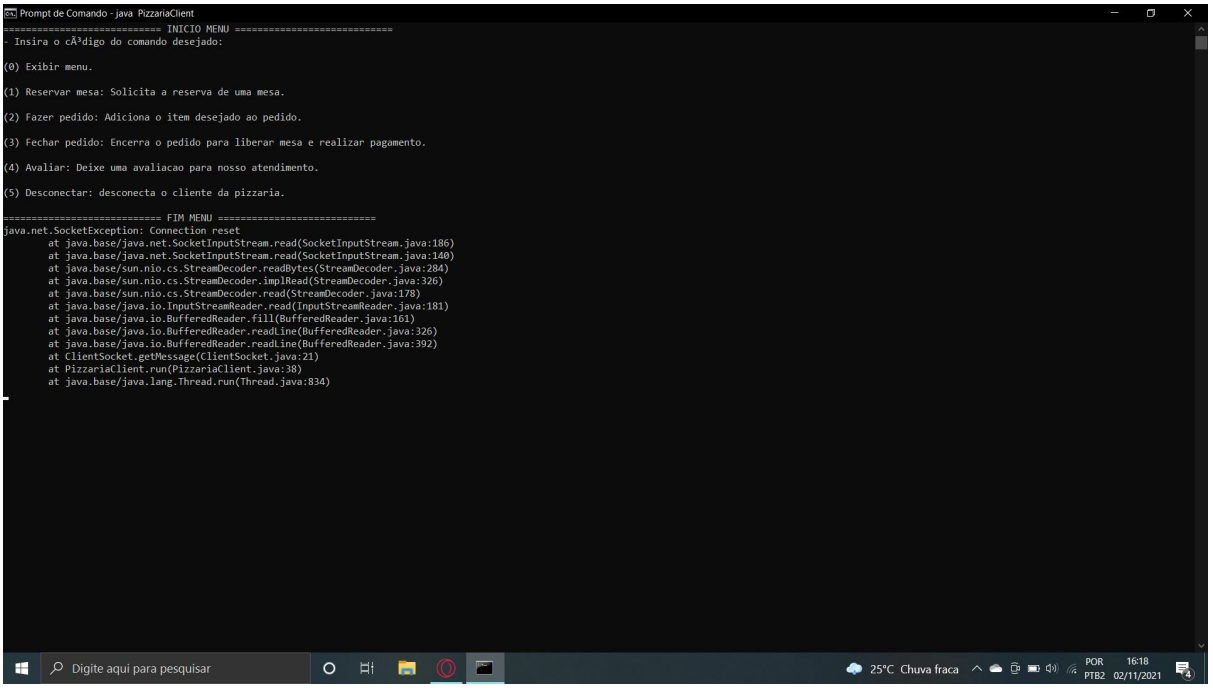
Teste: Comando inválido(6);



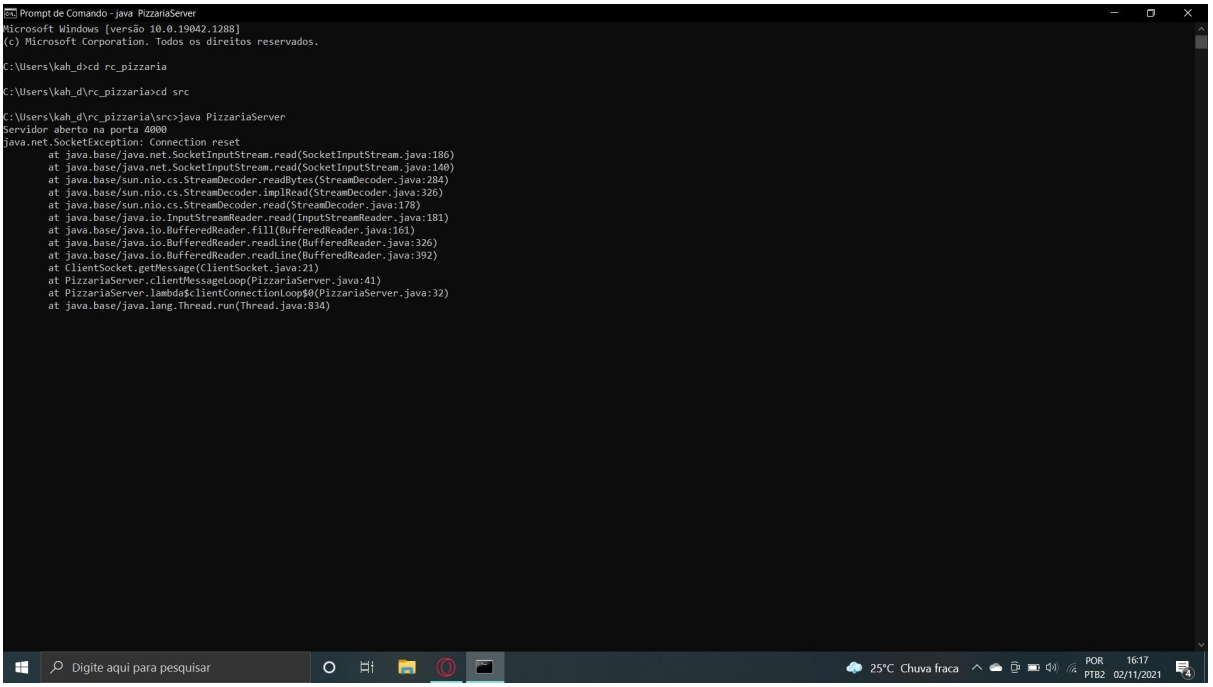
Teste: pedido sem mesa:



Teste: Servidor Fechou inesperadamente



Teste: Se o Client fecha inesperadamente.



7. Fontes

Para uso dos padrões Abstract Factory e Singleton:
https://refactoring.guru/pt-br/design-patterns/java

Para entender o uso das bibliotecas de Sockets e Threads em java:

- #1 Criando aplicação CLIENTE/SERVIDOR de CHAT em JAVA usando SOCK…
- #2 Criando aplicação CLIENTE/SERVIDOR de CHAT em JAVA usando SOCK…