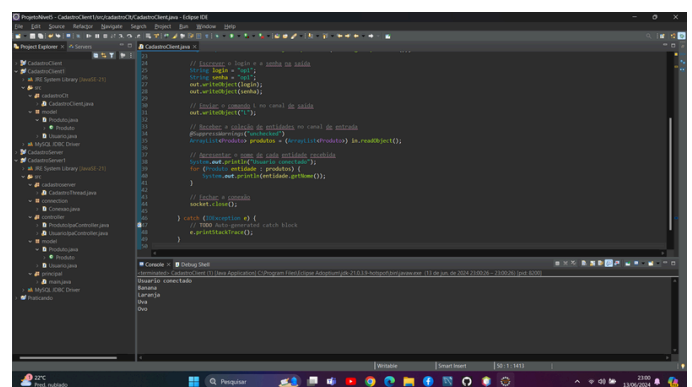
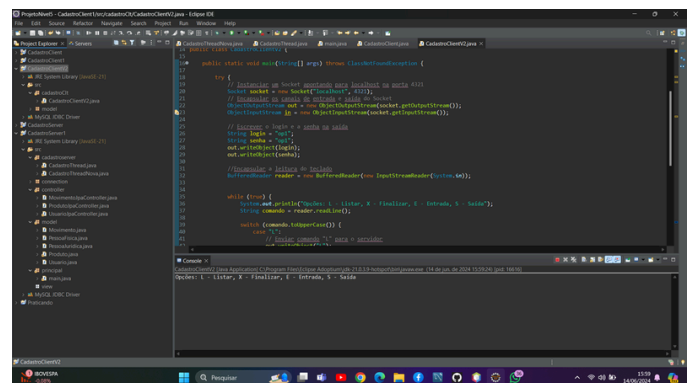


Por que não paralelizar

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

```
1 package cadastroserver;  
2  
3 import java.io.ObjectInputStream;  
4  
5 public class CadastroThread extends Thread {  
6     private ProdutoJpaController ctrl;  
7     private UsuarioJpaController ctrlUsu;  
8     private Socket s1;  
9  
10    public CadastroThread(ProdutoJpaController ctrl, UsuarioJpaController ctrlUsu, Socket socket) {  
11        this.ctrl = ctrl;  
12        this.ctrlUsu = ctrlUsu;  
13        this.s1 = socket;  
14    }  
15  
16    @Override  
17    public void run() {  
18        try {  
19            // Encapsula os canais de entrada e saída do Socket  
20            ObjectInputStream entrada = new ObjectInputStream(s1.getInputStream());  
21            ObjectOutputStream saida = new ObjectOutputStream(s1.getOutputStream());  
22  
23            // Obtém o login e a senha a partir da entrada  
24            String login = (String) entrada.readObject();  
25            String senha = (String) entrada.readObject();  
26  
27            // Utilize ctrlUsu para verificar o login  
28            boolean usuarioValido = ctrlUsu.findUsuario(login, senha);  
29  
30            // Loop de resposta  
31            while (usuarioValido) {  
32                // Obtém o comando a partir da entrada  
33                String comando = (String) entrada.readObject();  
34  
35                if (comando.equals("1")) {  
36                    // Utilize ctrl para retornar o conjunto de produtos através da saída  
37                    ctrl = new ProdutoJpaController();  
38                    ArrayList<Produto> listaProduto = ctrl.getListaProduto();  
39                    saida.writeObject(listaProduto);  
40                }  
41            }  
42        } catch (Exception e) {  
43            e.printStackTrace();  
44        }  
45    }  
46 }
```



Análise e Conclusão:

A classe Socket representa um ponto de comunicação entre duas máquinas em uma rede. Ela permite enviar e receber dados de forma bidirecional.

A classe ServerSocket é usada para criar um ponto de comunicação do lado do servidor que escuta as solicitações de conexão dos clientes¹.

O método accept() do ServerSocket aguarda e aceita conexões de clientes, bloqueando até que uma conexão seja estabelecida.

A função bind() vincula um endereço ao socket, mas é importante evitar erros de “já vinculado” ao usar essa função.

Importância das portas:

As portas são locais virtuais dentro de um sistema operacional onde as conexões de rede começam e terminam.

Elas permitem que os computadores diferenciem facilmente entre diferentes tipos de tráfego, como e-mails (porta diferente da das páginas web) mesmo usando a mesma conexão com a internet.

Classes ObjectOutputStream e ObjectInputStream:

Essas classes em Java são usadas para serializar e desserializar objetos.

ObjectOutputStream grava objetos em um fluxo de saída, enquanto ObjectInputStream lê objetos de um fluxo de entrada.

A serialização é o processo de gravar um objeto em formato binário, permitindo que ele seja transmitido ou armazenado.

Objetos transmitidos devem ser serializáveis (implementar a interface Serializable) para que possam ser corretamente gravados e lidos.

Isolamento do acesso ao banco de dados com JPA:

Mesmo usando as classes de entidades JPA no cliente, o isolamento do acesso ao banco de dados é possível devido ao ciclo de vida das entidades.

O JPA gerencia o estado das entidades, garantindo que as operações de persistência (como persist(), merge(), find()) sejam executadas de forma transacional e consistente⁴.

Além disso, o uso de anotações como @Transient permite ignorar campos específicos durante a persistência⁵.

Threads para tratamento assíncrono:

As threads em Java permitem que partes do seu programa executem de forma paralela. Isso é útil para tratar respostas assíncronas enviadas pelo servidor.

Por exemplo, se o servidor está processando uma tarefa demorada, você pode criar uma thread separada para lidar com a resposta enquanto a thread principal continua sua execução.

Isso evita que o processamento bloqueie a interface do usuário ou outras operações importantes.

Método `invokeLater` da classe `SwingUtilities`:

O método `invokeLater` é usado em aplicativos Swing para executar código na thread de eventos (EDT).

A EDT é responsável por atualizar a interface gráfica. Quando você precisa modificar componentes Swing (como atualizar um rótulo ou botão), use `invokeLater` para garantir que isso ocorra na EDT.

Isso evita problemas de concorrência e mantém a responsividade da interface.

Envio e recebimento de objetos via `Socket` Java:

Para enviar objetos via socket, você pode usar as classes `ObjectInputStream` e `ObjectOutputStream`.

No lado do servidor, leia o objeto usando `ObjectInputStream`. Se o objeto for uma instância da classe esperada (por exemplo, `User`), processe os dados.

Para repassar esses dados para o próprio usuário, você pode armazená-los em uma estrutura de dados (como um `ArrayList`) e enviar essa estrutura para o cliente.

Comparação entre comportamento assíncrono e síncrono com `Socket` Java:

Comportamento assíncrono:

Permite que várias operações ocorram simultaneamente.

Evita bloqueio: outras tarefas podem continuar enquanto uma operação assíncrona é executada.

Ideal para processamento demorado, como comunicação de rede.

Comportamento síncrono:

Executa operações sequencialmente, uma após a outra.

Pode bloquear o processamento se uma operação demorar.

Adequado para tarefas rápidas e interações em tempo real.

Link GitHub : [Yuri6406/3-Semestre-ProjetoNivel5: 3-Semestre-ProjetoNivel5 \(github.com\)](https://github.com/Yuri6406/3-Semestre-ProjetoNivel5)