Estácio

Campus: Dorival Caymmi

Nome: Filippe Markouizos Duarte - 202308599615

Curso: Desenvolvimento Full-Stack

Disciplina: Por que não paralelizar

Turma: 2023.1 **3º Semestre**

Título:

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.

Objetivo:

O objetivo deste trabalho é desenvolver competências na criação e implementação de sistemas baseados em comunicação por Sockets utilizando a linguagem Java. Para isso, o aluno deverá:

1. Criar servidores Java com base em Sockets, capazes de gerenciar conexões e processar múltiplas solicitações simultaneamente por meio de Threads.
2. Desenvolver clientes síncronos e assíncronos para interagir com esses servidores, aplicando conceitos de paralelismo e programação concorrente.
3. Implementar o acesso a banco de dados utilizando JPA (Java Persistence API), integrando a persistência de dados ao sistema.
4. Dominar o uso de Threads para garantir a execução paralela, tanto no servidor, para atender múltiplos clientes, quanto no cliente, para lidar com respostas assíncronas.

Ao final do projeto, o aluno terá desenvolvido um sistema completo, compreendendo a arquitetura cliente-servidor, comunicação eficiente, e manipulação de dados persistentes, consolidando o uso de recursos nativos do Java.

Análise e Conclusão

1. Como as Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor?

Threads permitem que a aplicação cliente continue executando outras tarefas enquanto aguarda ou processa a resposta do servidor. Elas evitam que a interface ou o fluxo principal fiquem bloqueados, realizando operações de I/O em paralelo.

2. Para que serve o método `invokeLater`, da classe `SwingUtilities`?

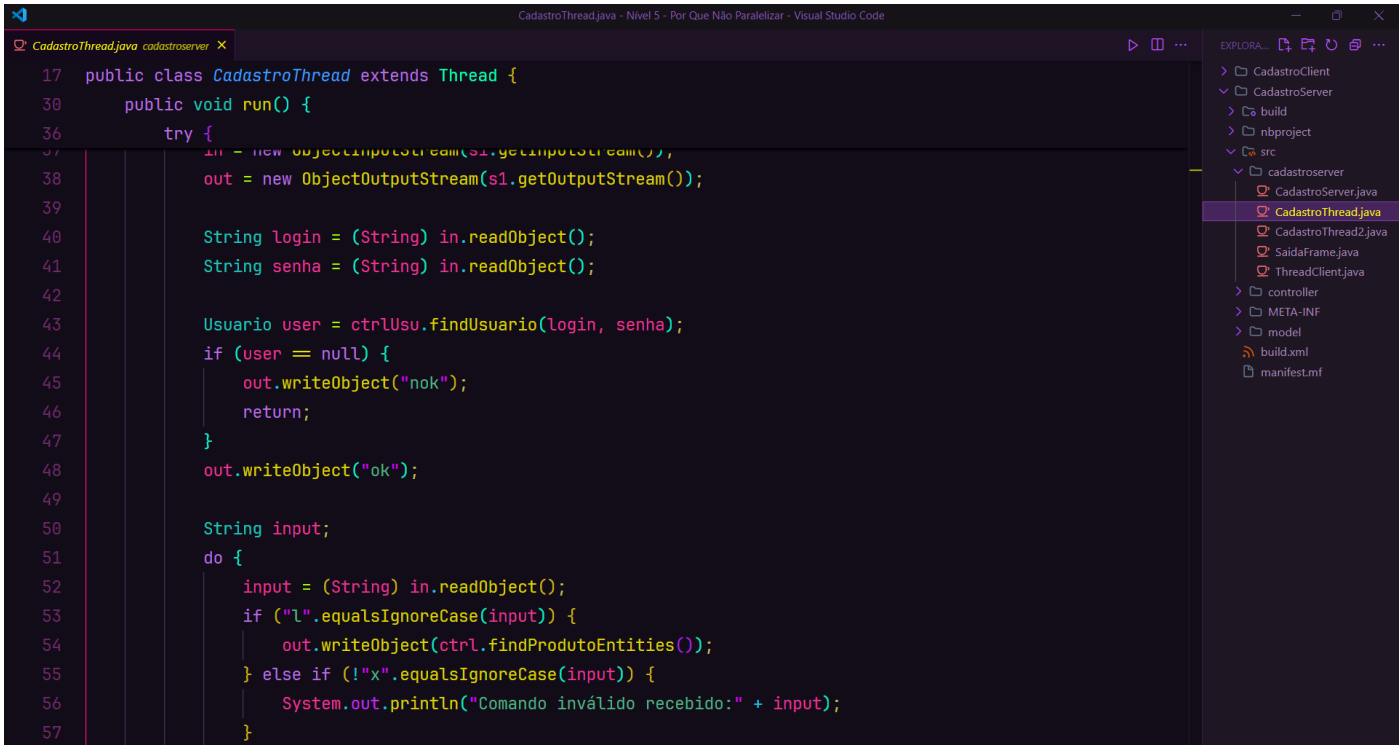
O método `invokeLater` é usado para garantir que uma atualização de interface gráfica (GUI) seja executada na *Event Dispatch Thread* (EDT), o que é essencial para manter a thread de interface segura e responsiva.

3. Como os objetos são enviados e recebidos pelo Socket Java?

Objetos podem ser enviados e recebidos utilizando as classes `ObjectOutputStream` e `ObjectInputStream`. O objeto deve ser serializável (implementar a interface `Serializable`), permitindo sua conversão em um fluxo de bytes.

4. Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento.

- **Síncrono:** O cliente aguarda a resposta do servidor antes de continuar a execução, o que pode bloquear o fluxo principal se a resposta demorar.
- **Assíncrono:** Permite que o cliente continue executando outras tarefas enquanto aguarda a resposta, utilizando Threads ou *callbacks*. Reduz bloqueios, mas requer maior complexidade no código.


CadastroThread.java - Nível 5 - Por Que Não Paralelizar - Visual Studio Code
CadastroThread.java cadastroserver
17 public class CadastroThread extends Thread {
30 public void run() {
36 try {
38 out = new ObjectOutputStream(s1.getOutputStream());
40 String login = (String) in.readObject();
41 String senha = (String) in.readObject();
43 Usuario user = ctrlUsu.findUsuario(login, senha);
44 if (user == null) {
45 out.writeObject("nok");
46 return;
47 }
48 out.writeObject("ok");
50 String input;
51 do {
52 input = (String) in.readObject();
53 if ("l".equalsIgnoreCase(input)) {
54 out.writeObject(ctrl.findProdutoEntities());
55 } else if (!"x".equalsIgnoreCase(input)) {
56 System.out.println("Comando inválido recebido:" + input);
57 }
EXPLORA...
CadastroClient
CadastroServer
build
nbproject
src
cadastroserver
CadastroServer.java
CadastroThread.java
CadastroThread2.java
SaidaFrame.java
ThreadClient.java
controller
META-INF
model
build.xml
manifest.mf