



**FACULDADE ESTÁCIO DE SÁ**  
**POLO PRQ DAS NAÇÕES CURSO:**  
**DESENVOLVIMENTO FULL STACK 3º SEMESTRE**  
**MATRÍCULA 202303005181**  
**ALEXANDRE ABREU FERREIRA**

**Por que não paralelizar**

**RELATÓRIO DA MISSÃO PRÁTICA**

**São Paulo 2024**

## **Objetivos da prática**

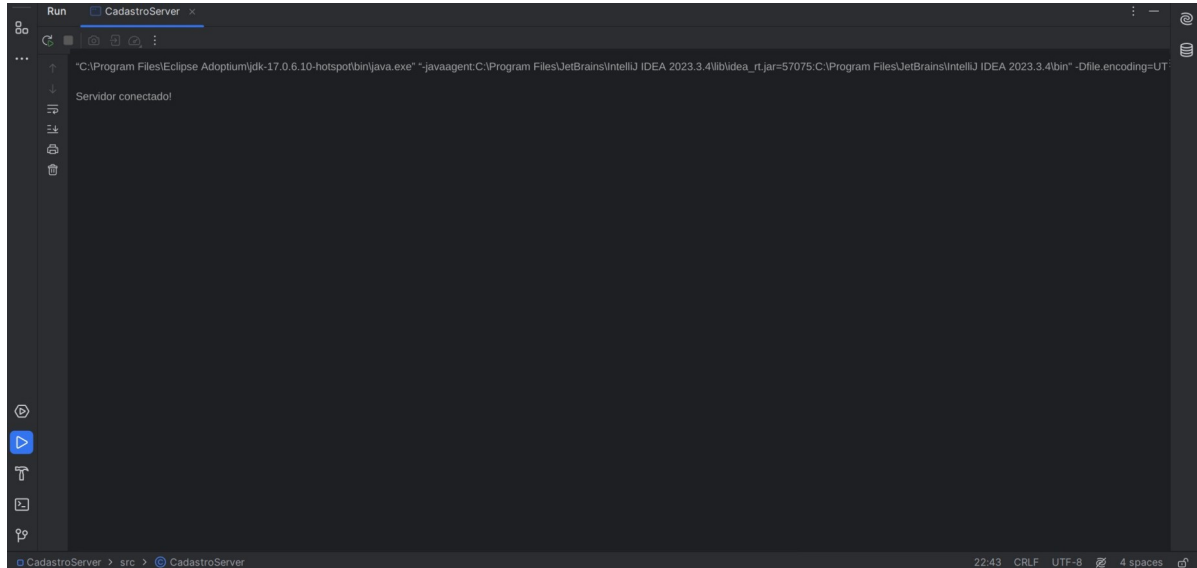
A proposta desta atividade consiste em aprimorar as competências na elaboração de sistemas de comunicação distribuída em linguagem Java, fazendo uso de sockets para realizar a comunicação entre servidores e clientes, além de explorar a utilização de Threads para viabilizar a realização de operações simultâneas tanto no servidor quanto no cliente.

Os metas particulares envolvem desenvolver um servidor em Java apto a aceitar conexões de usuários, autenticar informações de login e senha, atender solicitações dos usuários e acessar um banco de dados SQL Server por meio do JPA. Ademais, busca-se criar um aplicativo cliente para testar a comunicação com o servidor de modo sincronizado, enviando comandos e recebendo informações, e expandir o servidor para lidar com operações de entrada e saída de mercadorias, viabilizando a interação assíncrona com o usuário.

Também está previsto o desenvolvimento de um cliente assíncrono que possibilitará interações dinâmicas através de uma interface de linha de comando e uma janela de mensagens, evidenciando a utilização de Threads para lidar de forma assíncrona com as respostas do servidor. Além disso, serão analisadas as funcionalidades do NetBeans para melhorar a eficiência no desenvolvimento do sistema, bem como será conduzida uma reflexão sobre a aplicação de Threads e sockets em ambientes distribuídos, analisando os resultados alcançados e concluindo sobre a efetividade do modelo adotado.

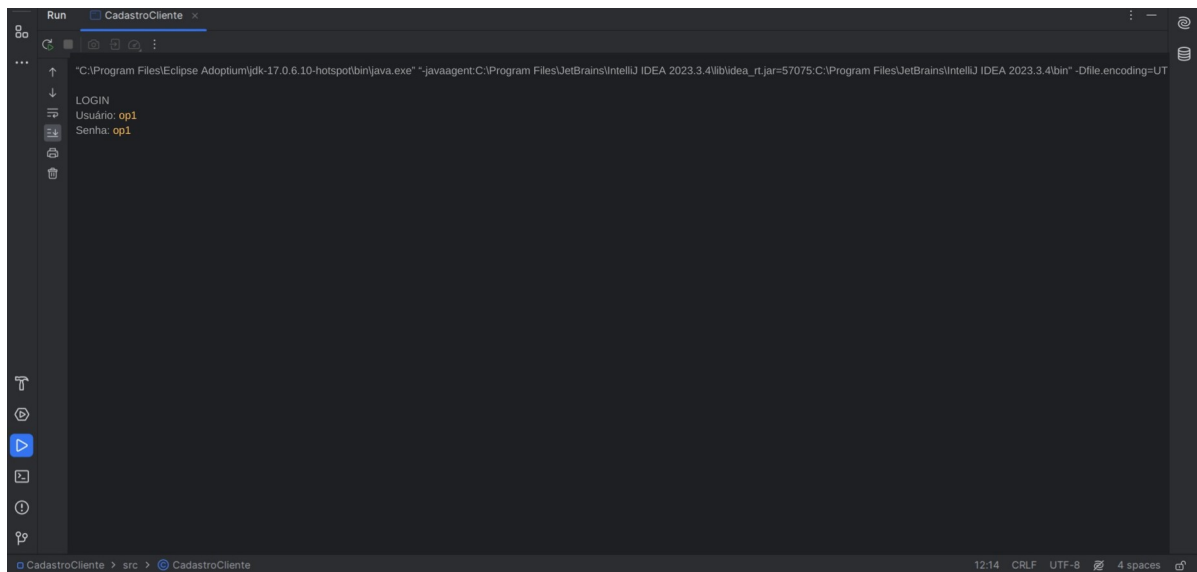
## Resultados

### 1º Procedimento | Criando o servidor e cliente de teste



The screenshot shows the 'Run' console for the 'CadastroServer' application. The console output displays the full Java command used to run the application, followed by the message 'Servidor conectado!'. The status bar at the bottom indicates the file encoding is UTF-8 and the line separator is CRLF.

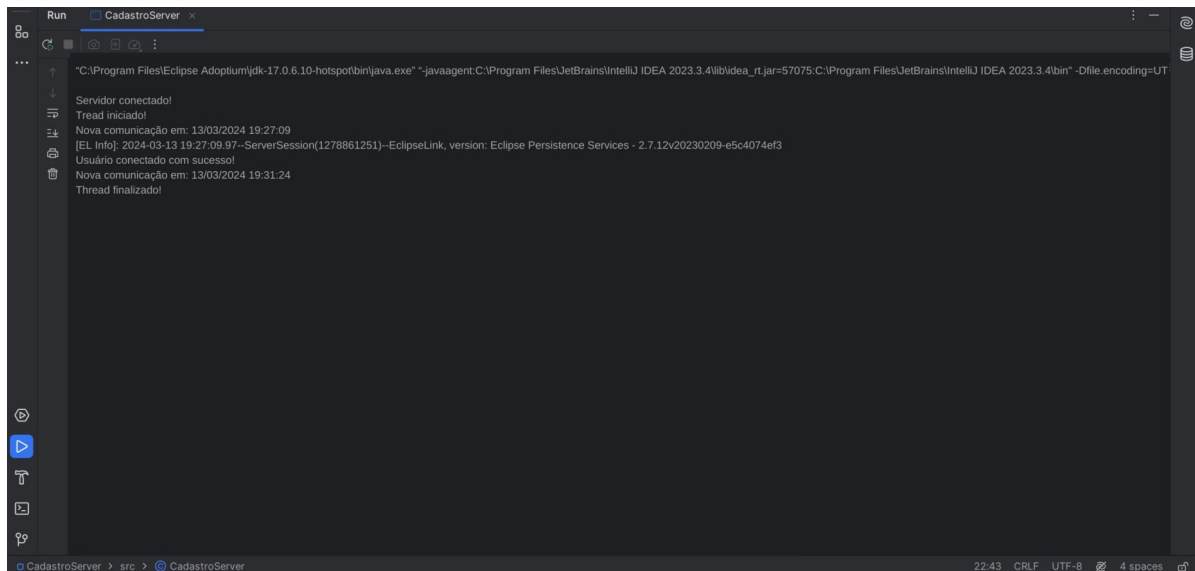
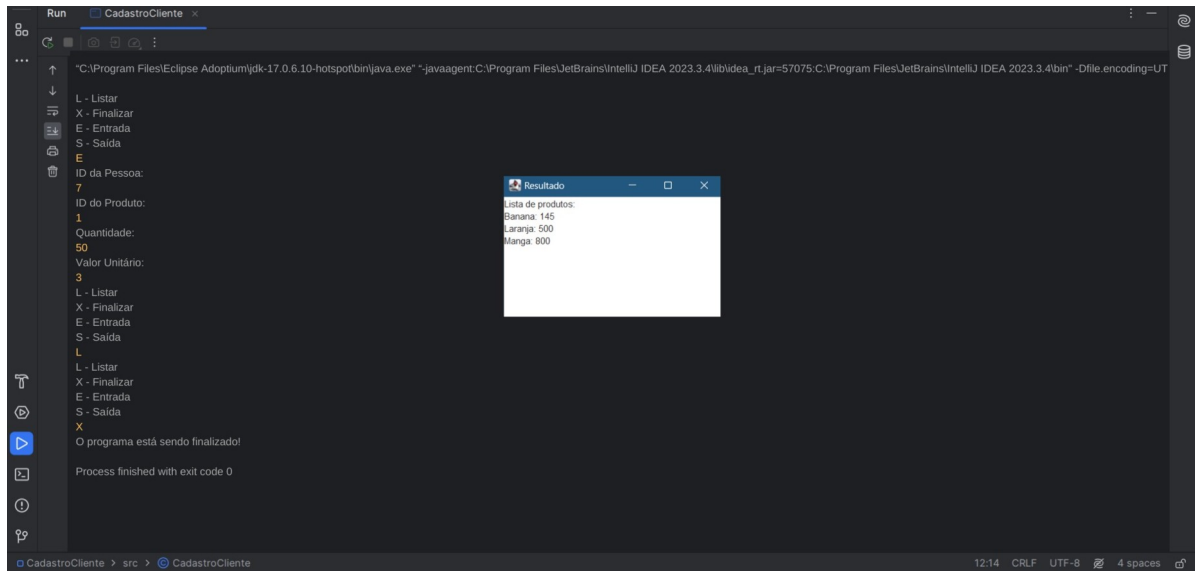
```
Run CadastroServer x
"C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.4\lib\idea_rt.jar=57075:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.4\bin" -Dfile.encoding=UTF-8
Servidor conectado!
```



The screenshot shows the 'Run' console for the 'CadastroCliente' application. The console output displays the command to run the application, followed by the 'LOGIN' section and the input values 'Usuário: op1' and 'Senha: op1'. The status bar at the bottom indicates the file encoding is UTF-8 and the line separator is CRLF.

```
Run CadastroCliente x
"C:\Program Files\Eclipse Adoptium\jdk-17.0.6.10-hotspot\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.4\lib\idea_rt.jar=57075:C:\Program Files\JetBrains\IntelliJ IDEA 2023.3.4\bin" -Dfile.encoding=UTF-8
LOGIN
Usuário: op1
Senha: op1
```

## 2º Procedimento | Servidor completo e cliente assíncrono



## Análise e Conclusão

### 1. Como funcionam as classes **Socket** e **ServerSocket**?

As classes **Socket** e **ServerSocket** em Java são utilizadas para estabelecer conexões de rede entre um cliente e um servidor. O **ServerSocket** é empregado pelo servidor para aguardar e aceitar solicitações de conexão dos clientes. Após a conexão ser estabelecida, um objeto **Socket** é criado no servidor para representar essa conexão. O **Socket** permite ao servidor se comunicar com o cliente, possibilitando o envio e recebimento de dados.

## **2. Qual a importância das portas para a conexão com servidores?**

As portas são essenciais para a conexão com servidores, pois permitem que múltiplos serviços de rede operem simultaneamente em um mesmo computador, cada um em sua própria porta. As portas são números identificadores associados a serviços específicos em um sistema. Ao conectar-se a um servidor, o cliente especifica a porta onde o serviço desejado está sendo executado. Isso possibilita ao servidor identificar a qual serviço o cliente está se conectando e direcionar os dados corretamente para esse serviço.

## **3. Para que servem as classes de entrada e saída `ObjectInputStream` e `ObjectOutputStream`, e por que os objetos transmitidos devem ser serializáveis?**

As classes `ObjectInputStream` e `ObjectOutputStream` são utilizadas para ler e escrever objetos Java em fluxos de dados, respectivamente. Elas são particularmente úteis para transmitir objetos complexos através de conexões de rede em Java. Os objetos transmitidos precisam ser serializáveis para que possam ser convertidos em uma sequência de bytes, possibilitando o envio pela rede. Essa serialização é necessária porque os dados transmitidos pela rede devem ser representados como uma sequência de bytes para serem enviados e recebidos corretamente entre o cliente e o servidor.

## **4. Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados?**

Mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados porque o acesso ao banco de dados é realizado exclusivamente no servidor. O cliente interage com o servidor através de solicitações e respostas de dados, sem acessar diretamente o banco de dados. O servidor recebe as solicitações do cliente, acessa o banco de dados, processa os dados conforme necessário e envia as respostas de volta ao cliente. Esse mecanismo garante o isolamento do acesso ao banco de dados, mantendo a segurança e a integridade dos dados.

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

As Threads podem ser usadas para o tratamento assíncrono das respostas enviadas pelo servidor através da criação de Threads separadas para cada solicitação de cliente. Quando uma solicitação é recebida, uma nova Thread pode ser iniciada para processá-la, permitindo que o servidor continue aguardando novas solicitações enquanto processa as solicitações existentes em paralelo. Isso possibilita que o servidor atenda múltiplos clientes simultaneamente e responda de forma assíncrona, garantindo maior eficiência e escalabilidade.

## **6. Para que serve o método `invokeLater`, da classe `SwingUtilities`?**

O método `invokeLater` da classe `SwingUtilities` é utilizado para executar uma tarefa de forma assíncrona na thread de despacho de eventos do Swing. Isso é útil para atualizar a interface gráfica do usuário (GUI) em resposta a eventos ou solicitações, garantindo que as operações de atualização da GUI ocorram na thread apropriada. Utilizar `invokeLater` ajuda a evitar problemas de concorrência e bloqueios, assegurando que as atualizações na GUI sejam realizadas de maneira segura e eficiente.

## **7. Como os objetos são enviados e recebidos pelo Socket Java?**

Os objetos são enviados e recebidos através do Socket Java por meio de fluxos de entrada e saída. Quando um objeto é enviado, ele é serializado em uma sequência de bytes, que pode ser transmitida através do Socket. No lado receptor, essa sequência de bytes é lida e deserializada de volta ao objeto original. Isso permite a transmissão eficiente e confiável de objetos complexos pela rede entre clientes e servidores em Java.

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

A utilização de comportamento assíncrono nos clientes com Socket Java permite que várias operações sejam executadas simultaneamente, sem bloquear o processamento principal. Isso é especialmente útil em cenários onde o cliente precisa continuar respondendo a eventos ou interações do usuário enquanto aguarda respostas do servidor. Em contraste, o comportamento síncrono bloqueia o processamento até que uma operação seja concluída, o que pode resultar em tempos de resposta mais longos e uma experiência de usuário menos responsiva. Portanto, o comportamento assíncrono é preferível em situações onde a escalabilidade e a responsividade são importantes, enquanto o comportamento síncrono é mais adequado para operações sequenciais ou de curta duração.