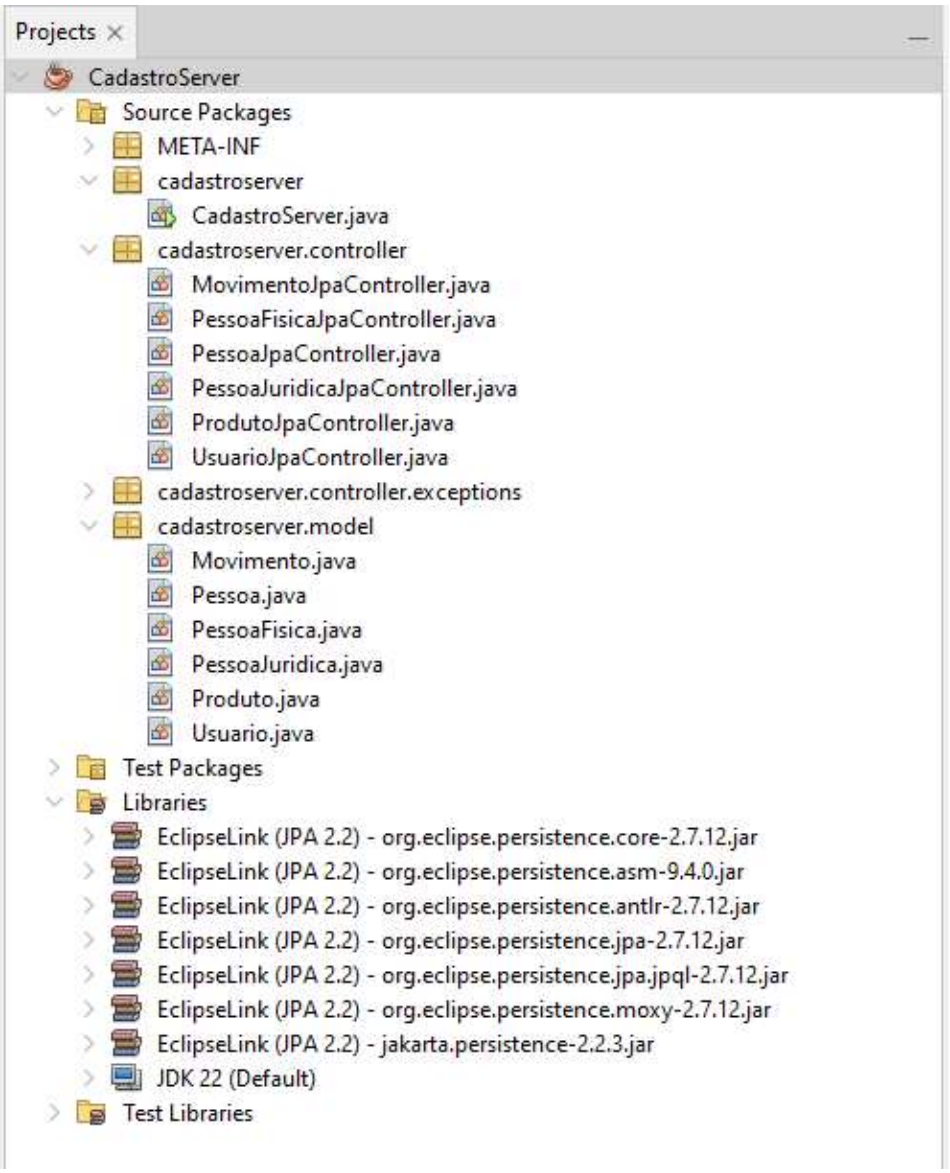
	**UNIVERSIDADE ESTÁCIO DE SÁ** POLO CENTRO – SANTA ROSA – RS **DESENVOLVIMENTO FULL STACK** Relatório da Missão Prática Nível 5 – Mundo 3
Disciplina:	RPG0018 - Por que não paralelizar
Aluno/Matrícula:	Anderson Rech - 202304442215
Turma:	2023.2
Repositório:	https://github.com/4nderech/Mundo3-Missao-Pratica-N5.git

1. 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.

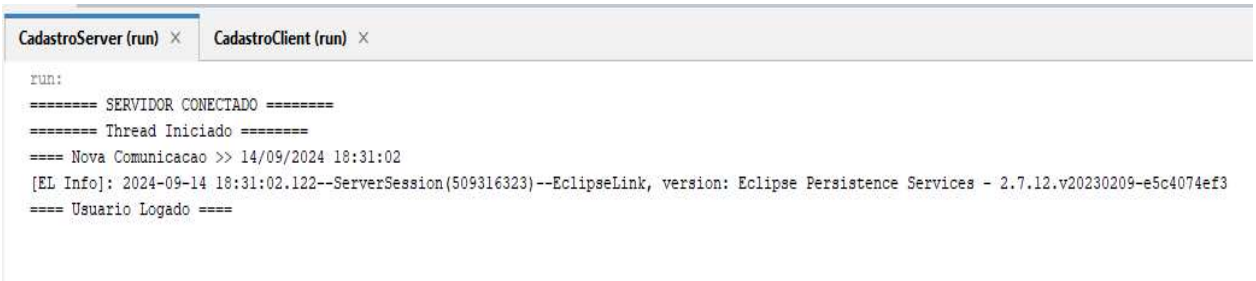
2. Objetivos da prática

1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.
5. No final do exercício, o aluno terá criado um servidor Java baseado em Socket, com acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para implementar a resposta assíncrona.

3. 1º Procedimento | Criando o Servidor e Cliente de Teste



Configuração do primeiro procedimento.



Servidor conectado e thread iniciado com sucesso.

4. Análise e Conclusão:

a. Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket em Java permitem comunicação em rede usando TCP/IP. O Socket é usado pelo cliente para se conectar ao servidor em uma porta específica e trocar dados através de fluxos de entrada e saída. Já o ServerSocket é usado pelo servidor para escutar conexões de clientes em uma porta. Quando um cliente se conecta, o servidor aceita a conexão e cria um Socket para se comunicar. Ambos podem trocar dados até que a conexão seja encerrada.

b. Qual a importância das portas para a conexão com servidores?

As portas são fundamentais para a conexão com servidores, pois permitem que um servidor diferencie e gerencie várias conexões simultâneas. Cada porta representa um ponto de entrada para um serviço específico em uma máquina, permitindo que diferentes serviços, como web (porta 80) ou e-mail (porta 25), sejam acessados independentemente. Ao se conectar a um servidor, o cliente utiliza o número da porta para identificar o serviço desejado, garantindo que os dados sejam direcionados corretamente ao processo responsável por atender aquela requisição.

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

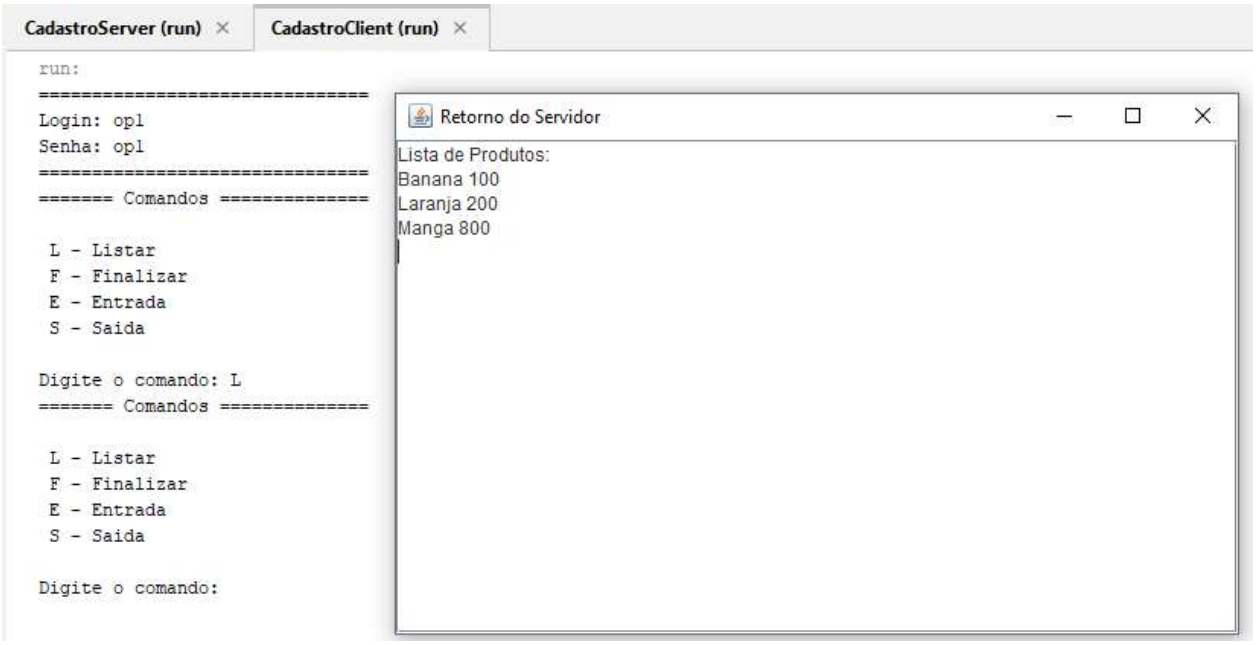
As classes ObjectInputStream e ObjectOutputStream são usadas em Java para ler e escrever objetos complexos, como instâncias de classes, em fluxos de entrada e saída, permitindo a transmissão de objetos entre sistemas. O ObjectInputStream lê objetos de um fluxo de entrada, convertendo dados binários em objetos Java, já o ObjectOutputStream escreve objetos em um fluxo de saída, convertendo objetos Java em dados binários para serem transmitidos. Os objetos transmitidos devem ser serializáveis porque a serialização é o processo de converter o estado de um objeto em um formato que pode ser armazenado ou transmitido. Para isso, a classe do objeto precisa implementar a interface Serializable, garantindo que o objeto possa ser convertido para um formato binário e reconstruído posteriormente. Se o objeto não for serializável, ele não pode ser convertido e transmitido corretamente.

d. 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, o isolamento do acesso ao banco de dados foi garantido porque o cliente não acessa diretamente o banco de dados. No modelo JPA, as entidades representam apenas o estado dos dados, mas a lógica de

acesso e manipulação desses dados (como transações, consultas e persistência) é controlada no lado do servidor, através de uma camada intermediária, como um serviço ou repositório. O servidor controla o acesso ao banco e aplica regras de segurança, transações e validação, evitando que o cliente tenha acesso direto ao banco de dados.

5. 2º Procedimento | Servidor Completo e Cliente Assíncrono



Listagem dos produtos.

CadastroServer (run) × CadastroClient (run) ×

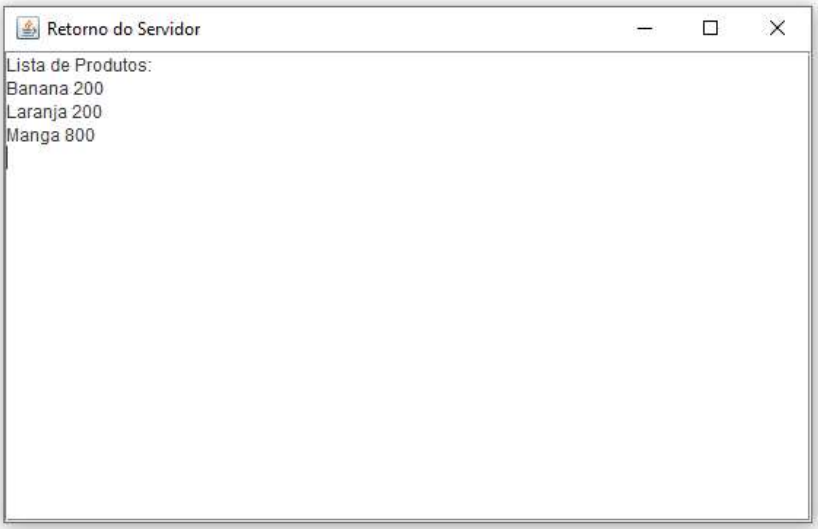
```
run:
================================
Login: op1
Senha: op1
================================
======= Comandos ==============

 L - Listar
 F - Finalizar
 E - Entrada
 S - Saida

Digite o comando: L
======= Comandos ==============

 L - Listar
 F - Finalizar
 E - Entrada
 S - Saida

Digite o comando: E
======== Entrada ========
ID Pessoa: 1
ID Produto: 1
ID Usuario: 1
Quantidade: 150
Valor Unitario: 10
```



Entrada de produtos.

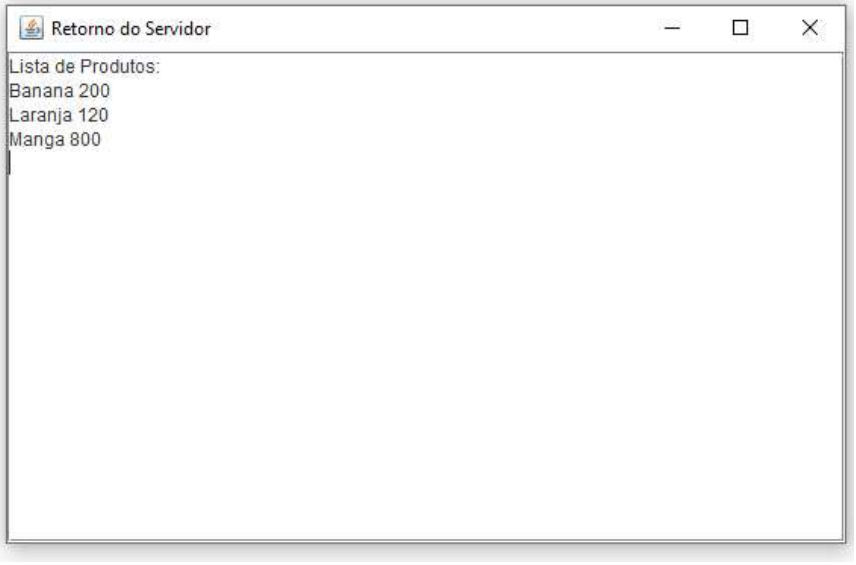
```
======= Comandos ==============

 L - Listar
 F - Finalizar
 E - Entrada
 S - Saida

Digite o comando: S
======== Saida ========
ID Pessoa: 1
ID Produto: 2
ID Usuario: 1
Quantidade: 80
Valor Unitario: 10
======= Comandos ==============

 L - Listar
 F - Finalizar
 E - Entrada
 S - Saida

Digite o comando:
```



Saída de produtos.

6. Análise e Conclusão:

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

As Threads permitem o tratamento assíncrono de respostas do servidor ao processar a comunicação em segundo plano. Enquanto uma Thread aguarda a resposta do servidor, o programa principal continua executando normalmente, sem ser bloqueado. Isso melhora a responsividade da aplicação, já que a espera pela resposta não interrompe outras operações. Quando a resposta é recebida, a Thread pode processá-la e acionar as ações necessárias, mantendo o fluxo de execução do cliente eficiente e fluido.

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

O método invokeLater, da classe SwingUtilities, é utilizado para garantir que uma operação seja executada na Thread de Despacho de Eventos (EDT - Event Dispatch Thread) do Swing. Isso é essencial para atualizar a interface gráfica corretamente e evitar problemas de concorrência, pois todas as alterações na interface devem ser feitas na EDT para garantir a integridade e o comportamento esperado da aplicação

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

Em Java, objetos são enviados e recebidos pelo Socket usando as classes ObjectOutputStream e ObjectInputStream. Para enviar um objeto, cria-se um ObjectOutputStream a partir do OutputStream do socket e utiliza-se o método writeObject(). Para receber um objeto, cria-se um ObjectInputStream a partir do InputStream do socket e utiliza-se o método readObject(). O objeto deve implementar a interface Serializable para garantir que possa ser corretamente serializado e desserializado durante a comunicação.

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

A comunicação com Socket em Java pode ser assíncrona o cliente envia uma solicitação e continua realizando outras tarefas sem esperar pela resposta. Utiliza Threads ou frameworks de concorrência para gerenciar a comunicação, melhorando a responsividade e a eficiência do aplicativo, mas é mais complexa de implementar, ou síncrona, onde o cliente envia uma solicitação e bloqueia a execução até receber a resposta. Isso pode resultar em uma interface não responsiva e atraso em outras tarefas. A escolha entre assíncrono e síncrono depende da necessidade de responsividade e da complexidade aceitável na implementação.