 <b>Estácio</b>	<p align="center"> <b>Universidade Estácio de Sá</b>  <b>Desenvolvimento Full Stack</b>  <b>Nível 5 – Por que não paralelizar</b>  <b>Turma 2022.3 – 3º Semestre</b> </p>
Nome:	Luan Augusto Vieira Bandeira
Repositório:	<a href="https://github.com/luanguto/nivel-5-mundo-3">https://github.com/luanguto/nivel-5-mundo-3</a>

### **Objetivos da prática**

Criar servidores Java com base em Sockets.










































Criar clientes síncronos para servidores com base em Sockets.

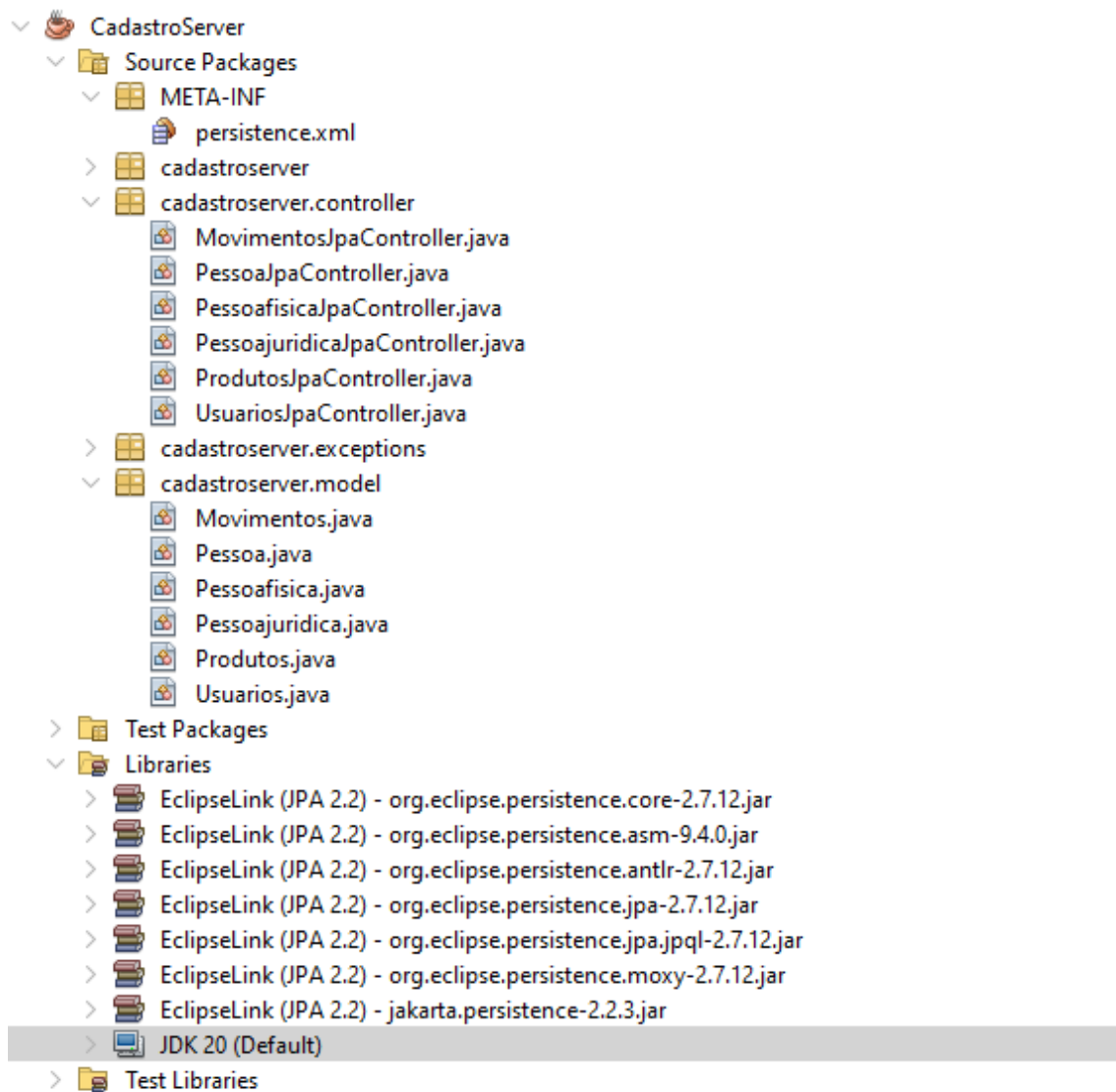
Criar clientes assíncronos para servidores com base em Sockets.

Utilizar Threads para implementação de processos paralelos.

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.

### **1º Procedimento**

- ▼  CadastroServer
  - ▼  Source Packages
    - ▼  META-INF
      -  persistence.xml
    - ▼  cadastroclient
      -  CadastroClient.java
    - ▼  cadastroserver
      -  CadastroServer.java
      -  CadastroThread.java
      -  MainServer.java
      -  build.xml
      -  persistence.xml
    - ▼  cadastroserver.controller
      -  MovimentosJpaController.java
      -  PessoaJpaController.java
      -  PessoafisicaJpaController.java
      -  PessoajuridicaJpaController.java
      -  ProdutosJpaController.java
      -  UsuariosJpaController.java
    - ▼  cadastroserver.exceptions
      -  IllegalOrphanException.java
      -  NonexistentEntityException.java
      -  PreexistingEntityException.java
    - ▼  cadastroserver.model
      -  Movimentos.java
      -  Pessoa.java
      -  Pessoafisica.java
      -  Pessoajuridica.java
      -  Produtos.java
      -  Usuarios.java
  - >  Test Packages
  - ▼  Libraries
    - >  EclipseLink (JPA 2.2) - org.eclipse.persistence.core-2.7.12.jar
    - >  EclipseLink (JPA 2.2) - org.eclipse.persistence.asm-9.4.0.jar
    - >  EclipseLink (JPA 2.2) - org.eclipse.persistence.antlr-2.7.12.jar
    - >  EclipseLink (JPA 2.2) - org.eclipse.persistence.jpa-2.7.12.jar
    - >  EclipseLink (JPA 2.2) - org.eclipse.persistence.jpa.jpql-2.7.12.jar
    - >  EclipseLink (JPA 2.2) - org.eclipse.persistence.moxy-2.7.12.jar
    - >  EclipseLink (JPA 2.2) - jakarta.persistence-2.2.3.jar
    - >  JDK 20 (Default)
  - >  Test Libraries



## Análise e Conclusão:

### Como funcionam as classes Socket e ServerSocket?

A classe Socket é usada para estabelecer uma conexão entre um cliente e um servidor. Um objeto Socket do lado do cliente tenta se conectar a um ServerSocket em um servidor específico. Uma vez que a conexão é estabelecida, a comunicação pode ocorrer utilizando fluxos de entrada e saída.

A classe ServerSocket é usada pelo servidor para escutar as conexões de entrada em uma porta específica. Quando um ServerSocket aceita uma conexão, ele cria e retorna um novo objeto Socket que é conectado ao cliente que fez a solicitação.

### Qual a importância das portas para a conexão com servidores?

As portas são essenciais para a comunicação em redes TCP/IP porque servem como pontos de extremidade em um host ou IP específico. Cada serviço em um servidor que utiliza TCP ou UDP especifica uma porta para que o tráfego destinado a esse serviço seja recebido corretamente.

Portas diferentes permitem que múltiplos serviços escutem e respondam a requisições simultaneamente em um único host.

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

`ObjectInputStream` e `ObjectOutputStream` são fluxos de entrada e saída que permitem ler e escrever objetos Java de e para fluxos subjacentes (como aqueles de um `Socket`), respectivamente. Isso permite a transmissão de objetos através de uma conexão de rede.

Os objetos que são transmitidos por esses fluxos precisam ser serializáveis porque a serialização é o processo de converter um objeto em um formato de byte que pode ser facilmente restaurado em um objeto posteriormente. A serialização é necessária para transportar o estado do objeto através de uma rede ou para salvá-lo em disco.

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

Mesmo que as classes de entidade JPA sejam usadas no cliente, o acesso ao banco de dados é isolado porque a manipulação real dos dados (como persistência, atualização, consulta e remoção) só ocorre no lado do servidor. O cliente apenas envia e recebe objetos ou comandos serializados, sem ter acesso direto ao banco de dados.

No servidor, é o `EntityManager` (do JPA) que realiza operações no banco de dados com base nos comandos recebidos. O cliente não pode executar operações de banco de dados diretamente, ele só pode solicitar ações que o servidor irá interpretar e executar se apropriado.

Esse isolamento é importante para segurança e integridade dos dados, pois restringe o acesso direto ao banco de dados apenas ao código do servidor que pode validar e autenticar as solicitações antes de executá-las.