Missão Prática | Nível 5 | Mundo 3

Filipe Alvim Santos - 202208291325

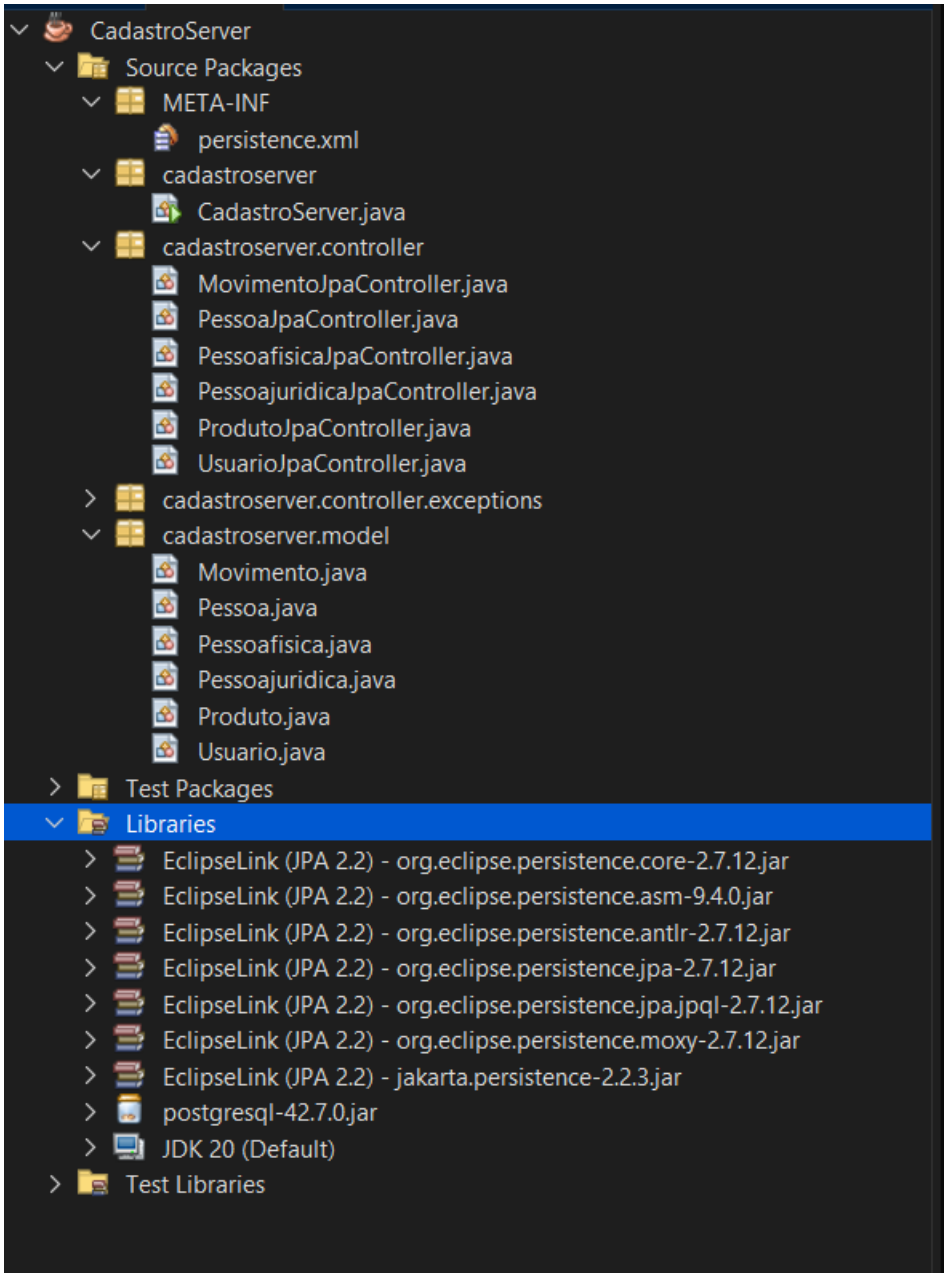
Campus Polo Sulacap – RJ / Desenvolvimento Full Stack
Nível 5: Porque não paralelizar – Turma: 22.3 – 3º Semestre

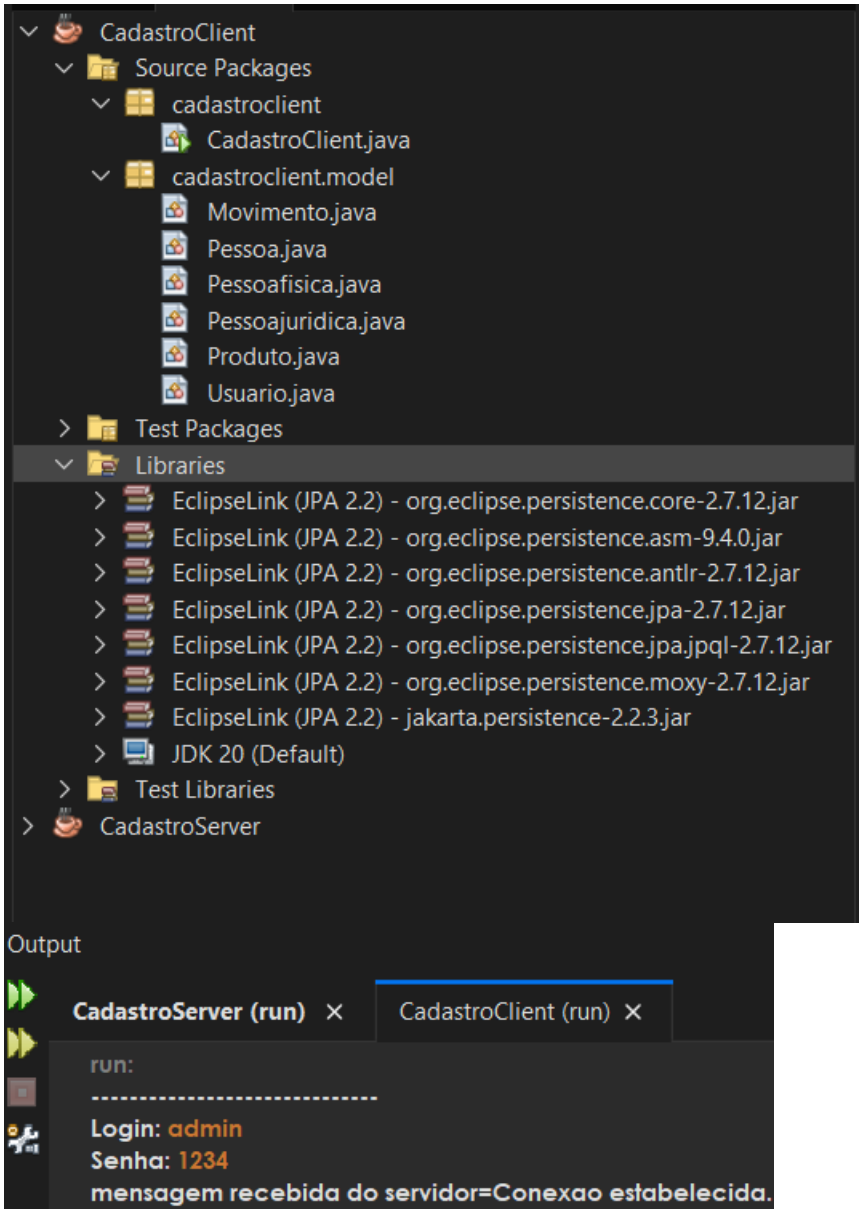
Objetivo da Prática

Os objetivos da prática são: 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 e implementar a resposta assíncrona.

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 | Criando o Servidor e Cliente de Teste


CadastroServer
Source Packages
META-INF
persistence.xml
cadastroserver
CadastroServer.java
cadastroserver.controller
MovimentoJpaController.java
PessoaJpaController.java
PessoafisicaJpaController.java
PessoajuridicaJpaController.java
ProdutoJpaController.java
UsuarioJpaController.java
cadastroserver.controller.exceptions
cadastroserver.model
Movimento.java
Pessoa.java
Pessoafisica.java
Pessoajuridica.java
Produto.java
Usuario.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
postgresql-42.7.0.jar
JDK 20 (Default)
Test Libraries



a) Como funcionam as classes Socket e ServerSocket?

Resposta: As classes Socket e ServerSocket são usadas para estabelecer uma comunicação de rede em Java. A classe Socket é usada para conectar-se a um servidor a partir de um sistema cliente, permitindo o envio e recebimento de dados. Já a classe ServerSocket é usada no lado do servidor para aguardar solicitações de clientes e estabelecer conexões.

A classe ServerSocket possui um construtor onde passamos a porta que desejamos usar para escutar as conexões. O método accept() escuta uma conexão e aceita se alguma for encontrada, bloqueando todo o restante até que uma conexão seja feita. Quando alguma conexão é aceita, ele retorna um objeto Socket.

Por outro lado, a classe Socket representa um endpoint para enviar e receber dados em uma rede. Ela é usada para estabelecer uma conexão com um host remoto e facilita a comunicação bidirecional.

b) **Qual a importância das portas para a conexão com servidores?**

Resposta: As portas são essenciais para a conexão com servidores porque elas permitem que os computadores diferenciem facilmente entre diferentes tipos de tráfego. Cada porta está associada a um processo ou serviço específico e é identificada por um número de 16 bits, conhecido como número de porta. Isso permite que aplicações e processos em um único computador compartilhem uma única conexão física com uma rede, como a internet. Além disso, as portas ajudam os computadores a entender o que fazer com os dados que recebem. Por exemplo, diferentes tipos de dados fluem de e para um computador na mesma conexão de rede. O uso de portas ajuda os computadores a direcionar esses dados para os serviços ou aplicativos corretos.

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

Resposta: As classes ObjectInputStream e ObjectOutputStream são usadas para a serialização e desserialização de objetos em Java. A ObjectOutputStream é usada para converter um objeto em uma série de bytes (processo de serialização), que podem ser enviados por um fluxo de saída. Por outro lado, a ObjectInputStream é usada para ler esses bytes de um fluxo de entrada e reconstruir o objeto (processo de desserialização).

Os objetos transmitidos devem ser serializáveis porque a serialização permite que o estado do objeto seja capturado e transformado em uma cadeia de bytes. Isso é útil quando você deseja transmitir dados de um objeto pela rede ou salvá-lo em um arquivo. Em Java, para tornar um objeto serializável, a classe do objeto deve implementar a interface Serializable.

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

Resposta: As classes de entidades JPA são usadas no cliente para representar os dados que estão sendo manipulados, mas o acesso real ao banco de dados é isolado e gerenciado pelo JPA. Isso permite que o código do cliente se concentre na lógica de negócios, enquanto o JPA cuida das complexidades do gerenciamento de transações e do acesso ao banco de dados. Além disso, o uso de classes de entidades ajuda a evitar a disseminação de SQLs em todo o lugar, tornando a manutenção e evolução de um sistema mais gerenciável. Portanto, mesmo que as classes de entidades

JPA sejam usadas no cliente, o acesso ao banco de dados é efetivamente isolado.

2º Procedimento | Servidor Completo e Cliente Assíncrono

a)

Resposta: As Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor de várias maneiras. Elas permitem que várias tarefas sejam executadas concorrentemente, como o processamento paralelo, a interatividade em programas gráficos e o tratamento de tarefas assíncronas.

Por exemplo, um pool de threads, que é uma coleção de threads de trabalho, pode ser usado para executar com eficiência retornos de chamada assíncronos em nome do aplicativo. Isso é especialmente útil para reduzir o número de threads do aplicativo e fornecer gerenciamento dos threads de trabalho.

Além disso, as threads que trabalham independentemente no tempo são assíncronas, enquanto aquelas que trocam informações em tempo de execução são síncronas. Portanto, ao usar threads para tratar respostas assíncronas de um servidor, é possível melhorar o desempenho e a capacidade de resposta dos aplicativos.

b)

Resposta: O método invokeLater da classe SwingUtilities é usado para enfileirar um evento de interface para ser despachado pela thread de eventos de interface assim que houver oportunidade. Ele recebe um objeto Runnable e executa o método run desse objeto na Event Dispatch Thread (EDT), que é a thread responsável por manipular eventos de interface do usuário em aplicações Java Swing.

Isso é útil para garantir que o código de interface do usuário seja executado de maneira segura, mesmo se chamado de uma thread que não seja a EDT. Além disso, ao utilizar invokeLater, é possível solicitar a execução de código de forma assíncrona na EDT, permitindo que o programa continue sua execução sem aguardar o término desse código específico. Isso ajuda a manter a responsividade da interface gráfica em aplicações Java Swing.

c)

Resposta: Os objetos são enviados e recebidos pelo Socket Java através de um processo chamado serialização e desserialização. Para enviar um objeto, você precisa criar um ObjectOutputStream a partir do fluxo de saída do socket. Em seguida, você pode usar o método writeObject para escrever o objeto no fluxo. O objeto é então serializado em uma sequência de bytes que pode ser enviada através do socket.

Para receber um objeto, você precisa criar um ObjectInputStream a partir do fluxo de entrada do socket. Em seguida, você pode usar o método readObject para ler o objeto do fluxo. O objeto é desserializado a partir da sequência de bytes recebida.

É importante notar que para que um objeto possa ser enviado através de um socket, ele deve ser serializável. Isso significa que a classe do objeto deve implementar a interface Serializable.

d)

Resposta: A utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java tem implicações significativas no bloqueio do processamento.

No processamento síncrono, as operações são executadas uma após a outra, o que significa que uma operação deve ser concluída antes que a próxima possa começar. Isso pode levar ao bloqueio do processamento, pois a thread que está executando o código fica bloqueada (ou seja, não pode fazer mais nada) até que a operação síncrona seja concluída. No contexto dos Sockets Java, isso pode ocorrer quando uma thread está esperando por dados de um socket.

Por outro lado, no processamento assíncrono, várias operações podem ocorrer simultaneamente. Isso significa que a thread que iniciou uma operação assíncrona não fica bloqueada esperando que a operação seja concluída. Em vez disso, ela pode continuar a fazer outras coisas e ser notificada quando a operação assíncrona estiver concluída. No contexto dos Sockets Java, isso pode permitir que uma aplicação continue a fazer outras coisas enquanto espera por dados de um socket.

Portanto, a escolha entre comportamento assíncrono e síncrono nos clientes com Socket Java pode ter um impacto significativo na eficiência e responsividade da aplicação.