Por que não paralelizar

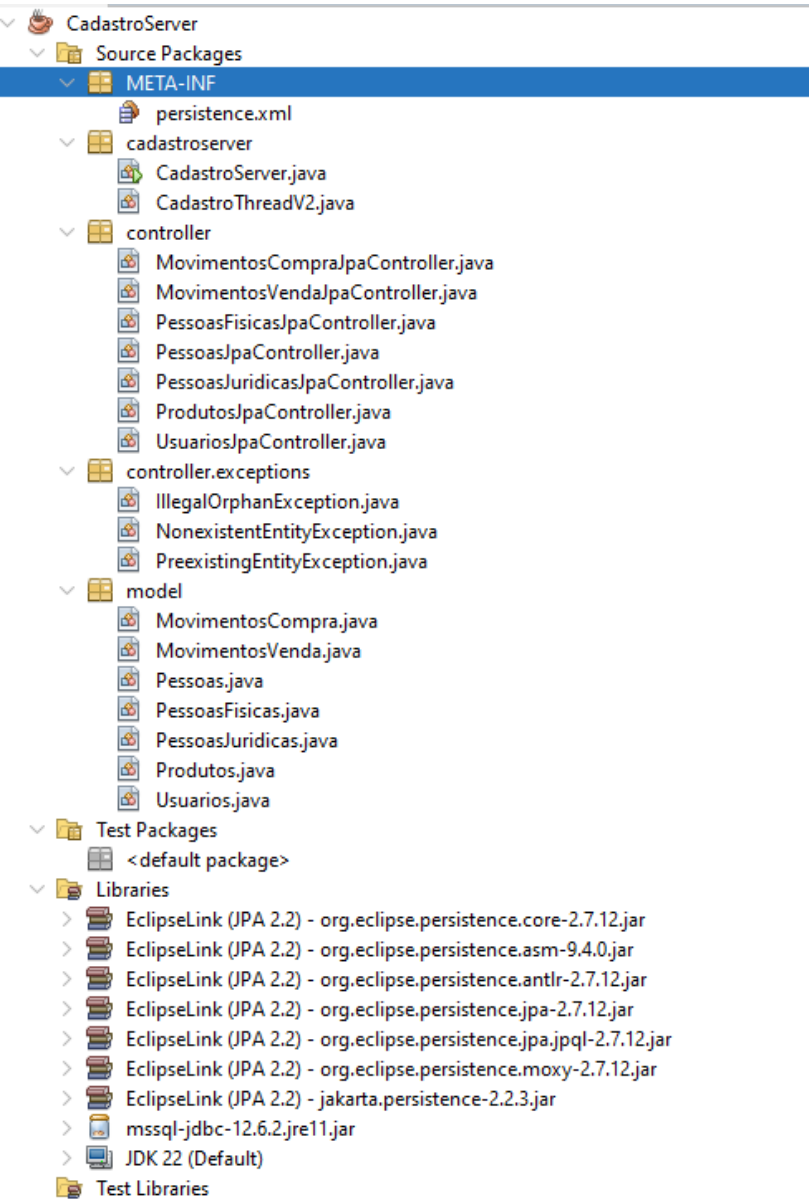
Ramon Barros Correa, 202208939635

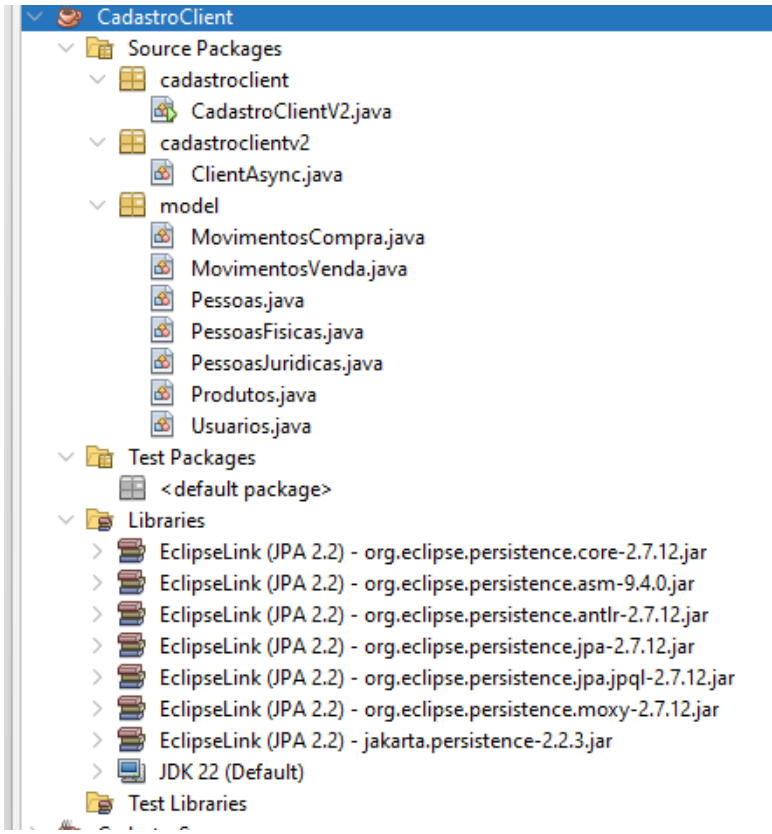
Inserir aqui o Campus: Polo Sete lagoas (MG)
Nível 4 – Por que não paralelizar, turma 2022.2, semestre 2024.1

Objetivo 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, comacesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java paraimplementação de clientes síncronos e assíncronos. As Threads serão usadas tantono servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, paraimplementar a resposta assíncrona.

1º e 2º Procedimento | Criando o Servidor Completo





Como funcionam as classes Socket e ServerSocket?

A classe ServerSocket é usada pelo lado do servidor para aguardar e aceitar conexões de clientes.
A classe Socket é usada pelo lado do cliente para se conectar a um servidor
As classes Socket e ServerSocket são fundamentais para comunicação de rede em Java, permitindo que aplicativos cliente-servidor se comuniquem através do protocolo TCP/IP. Elas fornecem uma interface simples e eficaz para estabelecer conexões, enviar e receber dados e gerenciar a comunicação de rede de forma geral.

Qual a importância das portas para a conexão com servidores?

Roteamento de Tráfego
As portas permitem que um único servidor ofereça vários serviços diferentes. Cada serviço é identificado por uma porta única.
Os roteadores e firewalls utilizam as informações de porta para rotear o tráfego de rede para o servidor e os serviços corretos.
Segurança
As portas ajudam a garantir a segurança, permitindo que os administradores de rede controlem quais serviços estão disponíveis para acesso externo.
Os firewalls podem ser configurados para bloquear ou permitir o tráfego em determinadas portas com base nas políticas de segurança da rede.
Concorrência e Escalabilidade
O uso de portas permite que múltiplos clientes se conectem simultaneamente ao mesmo servidor, cada um utilizando uma porta diferente.
Isso permite que os servidores lidem com grandes volumes de tráfego de maneira eficiente e escalável.
Flexibilidade
As portas são atribuídas dinamicamente a serviços específicos, permitindo a flexibilidade na configuração de servidores.
Os administradores de sistemas podem configurar novos serviços em portas não utilizadas sem interferir nos serviços existentes.

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

ObjectOutputStream
Propósito: A classe ObjectOutputStream é utilizada para serializar objetos em Java, ou seja, converter objetos em uma sequência de bytes que podem ser transmitidos pela rede, armazenados em arquivos ou transferidos entre diferentes processos.
Funcionamento: Ela escreve os bytes que representam o estado dos objetos em um fluxo de saída (como um arquivo ou uma conexão de rede).
ObjectInputStream
Propósito: A classe ObjectInputStream é utilizada para desserializar objetos, ou seja, reconstruir objetos a partir de uma sequência de bytes previamente serializada.
Funcionamento: Ela lê os bytes do fluxo de entrada e reconstrói os objetos correspondentes.
Por que os objetos transmitidos devem ser serializáveis?
Transmissão de Objetos: Quando objetos são transmitidos pela rede ou armazenados em arquivos, eles precisam ser convertidos em uma forma que possa ser transmitida e recriada no destino. A serialização é o processo de fazer isso.
Objetos Serializáveis: Para que um objeto possa ser serializado em Java, ele deve implementar a interface Serializable. Isso permite que o Java saiba que os objetos dessa classe podem ser transformados em uma sequência de bytes e depois restaurados de volta para objetos em tempo de execução.
Segurança: A serialização também é importante para garantir que apenas os dados relevantes do objeto sejam transmitidos e que a integridade dos dados seja mantida durante a transmissão.
Compatibilidade: A serialização também permite a comunicação entre diferentes versões de aplicativos Java, desde que a estrutura de dados do objeto serializado não seja alterada significativamente entre as versões.

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

Embora as classes de entidades JPA possam ser usadas no lado do cliente para representar os dados, o acesso ao banco de dados é sempre intermediado pelo servidor. Isso permite que o servidor mantenha o controle total sobre o acesso ao banco de dados, garantindo o isolamento e a segurança dos dados. O cliente apenas solicita operações e recebe os resultados, sem acesso direto aos dados subjacentes.

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

Thread Separada para Cada Requisição; Thread Pool; Modelagem Assíncrona; Callbacks; CompletableFuture (Java); Thread.sleep() com Polling; Tratamento de Eventos.

Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities serve para executar uma tarefa de forma assíncrona na thread de despacho de eventos do Swing (também conhecida como Event Dispatch Thread - EDT).
O método invokeLater da classe SwingUtilities é uma ferramenta essencial para garantir a atualização segura e assíncrona da interface gráfica do usuário em aplicativos Swing. Ele permite que tarefas sejam executadas de forma assíncrona na EDT, evitando problemas de concorrência e garantindo a consistência e a responsividade da GUI.

Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, os objetos são enviados e recebidos através de sockets utilizando as classes ObjectOutputStream e ObjectInputStream, respectivamente.

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

Comportamento Síncrono:

Bloqueio de Thread Única:

No modelo síncrono, cada operação de leitura ou escrita em um socket bloqueia a thread que está realizando a operação até que ela seja concluída.
Isso significa que a thread cliente fica inativa durante a execução da operação de I/O, o que pode resultar em uma experiência de usuário não responsiva, especialmente em operações de rede lentas.

Operações em Série:

Como as operações de leitura e escrita são executadas de forma sequencial em uma única thread, múltiplas operações podem ser realizadas em série.
Isso pode levar a atrasos na execução de operações subsequentes, especialmente se uma operação de I/O demorar mais do que o esperado.

Simplicidade no Controle de Fluxo:

O modelo síncrono é mais simples de entender e implementar, pois as operações de I/O são executadas de forma previsível e em ordem.

Comportamento Assíncrono:

Não Bloqueio de Thread:

No modelo assíncrono, as operações de I/O são executadas em threads separadas, permitindo que a thread principal do cliente continue executando outras tarefas enquanto as operações de I/O estão em andamento.
Isso evita o bloqueio da thread principal e mantém a aplicação responsiva, mesmo durante operações de I/O demoradas.

Paralelismo e Concorrência:

Como as operações de I/O são executadas em threads separadas, múltiplas operações podem ser realizadas em paralelo, aumentando a eficiência e o desempenho da aplicação.
Isso é especialmente útil em situações onde várias operações de rede precisam ser realizadas simultaneamente.

Complexidade Adicional:

O modelo assíncrono introduz uma complexidade adicional devido à necessidade de lidar com concorrência e sincronização entre threads.
O código assíncrono pode ser mais difícil de entender e depurar, especialmente para desenvolvedores menos experientes.