



MISSÃO PRÁTICA / NÍVEL 5 / MUNDO 3

MATHEUS DE CARVALHO MARQUES – 202302172881

CAMPUS: TAQUARA, R9

DISCIPLINA: RPG0018 POR QUE NÃO PARALELIZAR – 2024.1

<https://github.com/mathcarvalhom/Missao-Pratica-5-Mundo-3>

Objetivo 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.

1º PROCEDIMENTO | Análise e conclusão:

<https://github.com/mathcarvalhom/Missao-Pratica-5-Mundo-3>

Como funcionam as classes Socket e ServerSocket? A classe ServerSocket é utilizada no servidor para aguardar e aceitar conexões na rede, enquanto a classe Socket é empregada no cliente para estabelecer uma conexão com o servidor. Ambas as classes viabilizam a comunicação bidirecional entre aplicativos por meio de fluxos de dados. Após a conexão ser estabelecida com sucesso, essas duas entidades podem se comunicar utilizando o InputStream do objeto Socket para receber dados e enviando informações de volta através do OutputStream.

- Qual é a importância das portas para a conexão com servidores? As portas são essenciais para as conexões com servidores, pois associam serviços a números específicos, permitindo a operação simultânea de vários serviços em um único servidor. Essa abordagem facilita o direcionamento preciso do tráfego, otimizando a comunicação entre sistemas. Além disso, as portas são vitais para a segurança, possibilitando a aplicação de medidas de controle de acesso baseadas nas portas designadas para cada serviço, reforçando assim a integridade do sistema.

- 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 ferramentas importantes no universo Java, proporcionando a capacidade de serialização e desserialização de objetos. Essa funcionalidade é crucial para a transmissão de objetos entre sistemas distribuídos, persistência de dados em arquivos e, em última instância, para garantir a integridade e eficácia da comunicação entre objetos em diferentes ambientes.

- Por que, mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados? O uso das classes de entidades JPA no cliente garante o isolamento do acesso ao banco de dados devido ao mapeamento objeto-relacional, gerenciamento de transações, uso de caches e estratégia de lazy loading. Essas características, quando combinadas, criam uma camada de abstração que permite ao cliente interagir com o banco de dados de maneira eficiente, enquanto o JPA gerencia os detalhes da comunicação com o banco de dados.

2º PROCEDIMENTO | Análise e conclusão:

<https://github.com/mathcarvalhom/Missao-Pratica-5-Mundo-3>

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

do servidor é uma estratégia eficaz na gestão de operações de rede em programas clientes. Essa abordagem permite que o cliente continue executando diferentes tarefas enquanto aguarda uma resposta do servidor. A chave para esse cenário é a execução das operações de rede em Threads independentes, evitando o bloqueio da Thread principal do cliente. Com essa prática, a resposta do servidor pode ser processada ou comunicada ao usuário pela Thread principal assim que estiver disponível. Enquanto isso, outras Threads podem continuar a realizar suas tarefas, resultando em uma melhoria significativa na eficiência e na responsividade geral do programa. Essa abordagem paralela e dinâmica proporciona uma experiência mais fluida ao usuário, pois minimiza o impacto das operações de rede no fluxo principal de execução, tornando o programa mais ágil e capaz de executar múltiplas atividades simultaneamente.

- Para que serve o método `invokeLater`, da classe `SwingUtilities`? O método `invokeLater` da classe `SwingUtilities` em Java é utilizado para executar uma tarefa Swing de forma assíncrona na Thread de despacho de eventos Swing (EDT). Isso é especialmente útil quando se deseja atualizar a interface gráfica do usuário (GUI) a partir de uma Thread diferente da EDT, evitando problemas de concorrência e garantindo a consistência na interação com componentes Swing. O `invokeLater` agenda a execução da tarefa na EDT, permitindo uma comunicação segura e eficiente com a GUI em ambientes multithreaded.

- Como os objetos são enviados e recebidos pelo Socket Java? Em Java, a comunicação entre objetos por meio de sockets envolve a serialização e desserialização dos objetos. Para enviar um objeto por um Socket, é necessário convertê-lo em uma sequência de bytes por meio do processo de serialização. Isso é feito geralmente usando a interface `Serializable`. Do lado receptor, o objeto é reconstruído a partir da sequência de bytes recebida usando o processo de desserialização. Isso permite a transmissão eficiente de objetos entre máquinas distintas por meio de sockets, facilitando a comunicação em redes.

- Compare a utilização de comportamento assíncrono ou síncrono nos clientes com Socket Java, ressaltando as características relacionadas ao bloqueio do processamento. A diferença fundamental entre o comportamento assíncrono e síncrono nos clientes com Socket Java está relacionada ao bloqueio do processamento. Em operações síncronas, o cliente espera ativamente a conclusão de uma operação antes de continuar, o que pode resultar em bloqueios, especialmente em redes lentas. Em contraste, no comportamento assíncrono, o cliente pode continuar executando outras tarefas enquanto aguarda a conclusão da operação de Socket. Isso evita bloqueios desnecessários, melhorando a eficiência e a responsividade do programa, especialmente em situações onde a latência de rede pode ser um fator significativo.