Nome: **Ariel Freitas dos Santos**
Matrícula: **202208291694**
Curso: **Desenvolvimento Full Stack**
Turma : **2023.3**
Campus: **POLO ST MORADA DO SOL - GOIÂNIA - GO**
Github: **https://github.com/ArielFSantos/Java_Estacio_Nivel_5**

Missão Pratica | Mundo 3 | Nível 5

RPG0018 - Por que não paralelizar

Objetivo:

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
6. acesso ao banco de dados via JPA, além de utilizar os recursos nativos do Java para
7. implementação de clientes síncronos e assíncronos. As Threads serão usadas tanto
8. no servidor, para viabilizar múltiplos clientes paralelos, quanto no cliente, para
9. implementar a resposta assíncrona.

Análise e Conclusão:

Como funcionam as classes Socket e ServerSocket?

Socket é usado para a comunicação cliente-servidor, permitindo a transferência de dados entre eles. ServerSocket é usado pelo servidor para aguardar e aceitar conexões de clientes, criando um novo Socket para cada conexão estabelecida. Essas classes são fundamentais para implementar aplicativos que se comunicam através de uma rede.

Qual a importância das portas para a conexão com servidores?

As portas são essenciais para a comunicação entre sistemas em uma rede. Elas ajudam a direcionar o tráfego de dados para aplicativos específicos em um servidor. Cada serviço ou aplicativo em um servidor é associado a uma porta única. Isso permite que os pacotes de dados sejam roteados corretamente, garantindo que a informação alcance o aplicativo correto, melhorando a segurança e a eficiência na transmissão de dados.

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 em Java são usadas para entrada e saída de objetos, respectivamente. Elas permitem a serialização de objetos, que é o processo de converter objetos em uma sequência de bytes. Essa serialização é crucial para transmitir objetos por meio de fluxos de dados, como em comunicação de rede. Os objetos devem ser serializáveis para que o Java possa converter o estado do objeto em bytes e reconstruí-lo em outro ambiente. Isso é fundamental para a comunicação entre diferentes máquinas ou para salvar o estado de um objeto em um arquivo. A serialização garante que a estrutura do objeto seja preservada durante a transmissão ou armazenamento, permitindo uma reconstrução fiel do objeto original.

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

O isolamento do acesso ao banco de dados é garantido pelo uso do padrão JPA (Java Persistence API) e suas classes de entidades no lado do cliente. A JPA fornece uma camada de abstração que separa as operações no banco de dados do código de negócios. As classes de entidades representam objetos persistentes mapeados para tabelas do banco de dados, e as operações sobre esses objetos são realizadas por meio da JPA. Ao utilizar as classes de entidades JPA no cliente, o acesso direto ao banco de dados é evitado, e as operações são realizadas por meio de consultas e atualizações de objetos de entidade. Isso proporciona uma abstração eficaz, permitindo que o código de negócios trabalhe com objetos familiares em vez de lidar diretamente com SQL e detalhes de acesso ao banco de dados. Essa abstração contribui para o isolamento e modularidade do sistema.

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

As Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo servidor ao permitir que determinadas operações ocorram de forma concorrente e independente no código. Isso é especialmente útil em operações de entrada/saída (I/O) e em situações em que é necessário aguardar a resposta do servidor sem bloquear o fluxo principal do programa. Ao criar Threads dedicadas para manipular as respostas, é possível continuar a execução do programa enquanto aguarda simultaneamente as informações do servidor, melhorando a eficiência e responsividade do sistema.

Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities é utilizado para executar uma tarefa de forma assíncrona no despachador de eventos (event dispatch thread) do Swing. Ele é comumente usado para agendar a execução de operações que interagem com a interface gráfica do usuário (GUI) de maneira segura, garantindo que essas operações sejam realizadas no contexto apropriado da thread de eventos, evitando problemas de concorrência e tornando a interface mais responsiva. Em resumo, o invokeLater ajuda a garantir a consistência e a segurança nas operações GUI em aplicativos Swing.

Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, os objetos são enviados e recebidos por meio do Socket utilizando as classes ObjectInputStream e ObjectOutputStream. A ObjectOutputStream é usada para serializar um objeto em um fluxo de saída, que pode ser enviado através do Socket. No lado receptor, o ObjectInputStream é utilizado para desserializar o objeto a partir do fluxo de entrada recebido pelo Socket. Isso permite a transmissão eficiente de objetos complexos entre diferentes aplicativos ou máquinas por meio de uma conexão de socket. É importante observar que os objetos que são transmitidos dessa maneira devem ser serializáveis para garantir que possam ser convertidos em bytes e reconstruídos corretamente do outro lado.

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

No contexto de clientes com Socket Java, o comportamento síncrono implica que as operações de leitura e escrita bloqueiam a execução do programa até que a operação seja concluída, o que pode resultar em espera ociosa do processo. Já o comportamento assíncrono permite que o programa continue a execução enquanto aguarda a conclusão de operações de leitura ou escrita, evitando bloqueios. Isso é alcançado por meio de métodos assíncronos ou threads separadas para manipular eventos de entrada/saída, proporcionando maior eficiência e responsividade ao lidar com operações em segundo plano.