Relatório Discente de Acompanhamento

Missão Prática / Nível 5 / Mundo 3

Campus: Rua Visconde de Mauá, Nº 150, Sala 105, - Centro - Canela - RS - CEP.: 95.680-232

Curso: Desenvolvimento Full Stack

Disciplina: Nível 5: Por que não paralelizar

Número da Turma: 9001

Semestre Letivo: 2024.1

Nome: Endrius da Silva dos Santos

Repositório GitHub: https://github.com/endriusssantos/atividade-nivel-5-mundo-3

RELATÓRIO DA MISSÃO PRÁTICA

Objetivos da prática

O objetivo desta prática é desenvolver habilidades na criação de sistemas de comunicação distribuída em Java, utilizando sockets para estabelecer a conexão entre servidores e clientes, e explorar o uso de Threads para permitir operações paralelas tanto no lado do servidor quanto no lado do cliente.

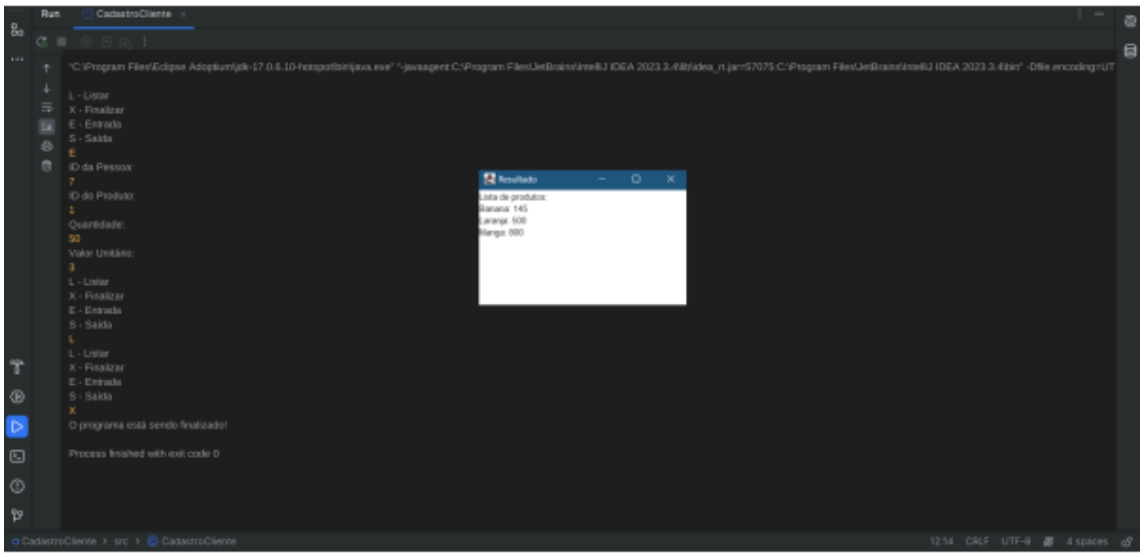
Os objetivos específicos incluem criar um servidor Java capaz de receber conexões de clientes, validar credenciais de login e senha, responder a comandos do cliente e acessar um banco de dados SQL Server via JPA. Além disso, visa-se implementar um cliente de teste para interagir com o servidor de forma síncrona, enviando comandos e recebendo respostas, e ampliar o servidor para lidar com operações de entrada e saída de produtos, permitindo a interação assíncrona com o cliente.

Também será desenvolvido um cliente assíncrono que permita interações dinâmicas por meio de uma interface de linha de comando e uma janela de mensagens, demonstrando o uso de Threads para tratamento assíncrono de respostas do servidor. Adicionalmente, serão exploradas as funcionalidades do NetBeans para aumentar a produtividade no desenvolvimento do sistema e será realizada uma reflexão sobre o uso de Threads e sockets em ambientes distribuídos, analisando os resultados obtidos e tirando conclusões sobre a eficácia do modelo adotado.

Resultados

1º Procedimento | Criando o servidor e cliente de teste

2º Procedimento | Servidor completo e cliente assíncrono



Análise e Conclusão

1. Como funcionam as classes Socket e ServerSocket?

As classes Socket e ServerSocket em Java são usadas para criar conexões de rede entre um cliente e um servidor. O ServerSocket é usado pelo servidor para aguardar e aceitar solicitações de conexão dos clientes. Uma vez que uma conexão é estabelecida, um objeto Socket é criado no servidor para representar essa conexão. O Socket é usado pelo servidor para se comunicar com o cliente, enviando e recebendo dados.

2. Qual a importância das portas para a conexão com servidores?

As portas são fundamentais para a conexão com servidores porque elas permitem que vários serviços de rede sejam executados em um mesmo computador, cada um em sua própria porta. As portas são números de identificação associados a serviços específicos em um sistema. Ao conectar-se a um servidor, o cliente especifica a porta na qual o serviço desejado está sendo executado. Isso permite que o servidor saiba a que serviço o cliente está se conectando e direcione os dados apropriados para esse serviço.

3. 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 usadas para ler e escrever objetos Java a partir de e para fluxos de dados, respectivamente. Elas são especialmente úteis para transmitir objetos complexos via conexões de rede em Java. Os objetos transmitidos devem ser serializáveis para poderem ser convertidos em uma sequência de bytes que podem ser enviados pela rede. Isso é necessário porque os dados transmitidos pela rede devem ser representados como uma sequência de bytes para serem enviados e recebidos corretamente entre o cliente e o servidor.

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

Mesmo utilizando as classes de entidades JPA no cliente, foi possível garantir o isolamento do acesso ao banco de dados porque o acesso ao banco de dados é realizado exclusivamente no servidor. O cliente interage com o servidor através de solicitações e respostas de dados, mas não acessa diretamente o banco de dados. O servidor é responsável por receber as solicitações do cliente, acessar o banco de dados, processar os dados conforme necessário e enviar as respostas de volta ao cliente. Isso ajuda a garantir o isolamento do acesso ao banco de dados e a manter a segurança e a integridade dos dados.

5. 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 através da criação de Threads separadas para lidar com cada solicitação de cliente. Quando uma solicitação é recebida, uma nova Thread pode ser iniciada para processar essa solicitação, permitindo que o servidor continue a aguardar por novas solicitações enquanto processa as solicitações existentes em paralelo. Isso permite que o servidor atenda a múltiplos clientes de forma simultânea e responda de forma assíncrona, garantindo assim uma maior eficiência e escalabilidade.

6. Para que serve o método invokeLater, da classe SwingUtilities?

O método invokeLater da classe SwingUtilities é usado para executar uma determinada tarefa de forma assíncrona na thread de despacho de eventos do Swing. Isso é útil quando se deseja atualizar a interface gráfica do usuário (GUI) em resposta a eventos ou solicitações, garantindo que as operações de atualização da GUI sejam realizadas na thread apropriada para evitar problemas de concorrência e bloqueios.

7. Como os objetos são enviados e recebidos pelo Socket Java?

Os objetos são enviados e recebidos pelo Socket Java por meio de fluxos de entrada e saída. Ao enviar um objeto, ele é serializado em uma sequência de bytes que podem ser transmitidos através do Socket. No lado receptor, a sequência de bytes é lida e deserializada de volta ao objeto original. Isso permite que objetos complexos sejam transmitidos pela rede entre clientes e servidores em Java de forma eficiente e confiável.

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

A utilização de comportamento assíncrono nos clientes com Socket Java permite que várias operações sejam executadas de forma simultânea, sem bloquear o processamento principal. Isso é particularmente útil em cenários onde o cliente precisa continuar respondendo a eventos ou interações do usuário enquanto aguarda por respostas do servidor. Por outro lado, o comportamento síncrono bloqueia o processamento até que uma operação seja concluída, o que pode resultar em tempos de resposta mais longos e uma experiência de usuário menos responsiva. Assim, o comportamento assíncrono é preferível em situações onde a escalabilidade e a responsividade são importantes, enquanto o comportamento síncrono é mais adequado para operações sequenciais ou de curta duração.