Por Que Não Paralelizar?

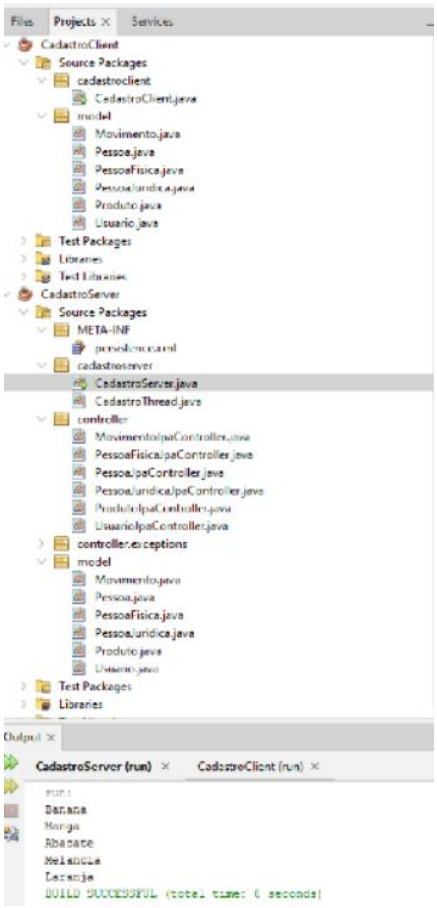
Julio Cesar Marques Junior

Campus: Rua Doutor Bozano, 478 1º Andar, - Centro - Santa Maria - RS - CEP.: 97.015-000.
RPT0018 – Turma: 9001 – Semestre: 2023.4 EAD

Github:
https://github.com/juliomarquesjr/missao_pratica_5_mundo3_estacio_por_que_nao_paralelizar

Objetivo da Prática

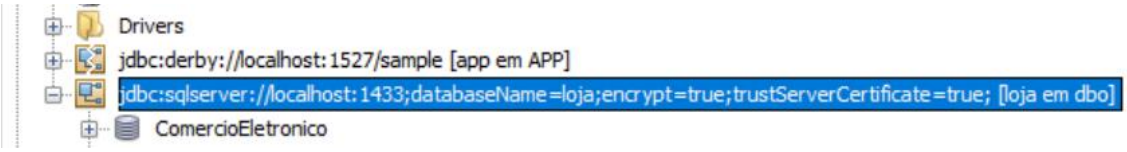
Pessoa Editada:



Estrutura do projeto

```java
package model;
import java.io.Serializable;
import java.util.Collection;
import javax.persistence.*;

/**
 *
 * @author ruanf
 */

@Entity
@Table(name = "Produto")
@NamedQueries({
    @NamedQuery(name = "Produto.findAll", query = "SELECT p FROM Produto p"),
    @NamedQuery(name = "Produto.findByIdProduto", query = "SELECT p FROM Produto p WHERE p.idProduto = :idProduto"),
    @NamedQuery(name = "Produto.findByNome", query = "SELECT p FROM Produto p WHERE p.nome = :nome"),
    @NamedQuery(name = "Produto.findByQuantidade", query = "SELECT p FROM Produto p WHERE p.quantidade = :quantidade"),
    @NamedQuery(name = "Produto.findByPrecoVenda", query = "SELECT p FROM Produto p WHERE p.precoVenda = :precoVenda")
})
public class Produto implements Serializable {
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Basic(optional = false)
    @Column(name = "idProduto")
    private Integer idProduto;

    @Column(name = "nome")
    private String nome;
```

Produto.java



Conexao com o banco

1º Procedimento | Criando o Servidor e Cliente de Teste:

1. Como funcionam as classes Socket e ServerSocket?

   As classes Socket e ServerSocket em Java são usadas para a comunicação em rede entre clientes e servidores. Socket é utilizado por um cliente para se conectar a um servidor, permitindo enviar e receber dados através dessa conexão. Por outro lado, ServerSocket é usado por um servidor para escutar por solicitações de conexão em uma porta específica, aceitando-as para estabelecer uma comunicação. Enquanto Socket facilita a troca de dados entre cliente e servidor, ServerSocket aguarda e aceita conexões, possibilitando a comunicação entre diferentes máquinas em uma rede.

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

   As portas são essenciais na comunicação de rede, funcionando como pontos de entrada específicos em um servidor para diferentes serviços ou aplicações. Elas permitem que múltiplos serviços, como web (HTTP), email (SMTP) e FTP, operem simultaneamente em um único servidor, cada um escutando em sua própria porta designada. Isso facilita a organização e a gestão do tráfego de rede, direcionando as solicitações dos clientes para o serviço correto. Assim, as portas são cruciais para garantir que os dados enviados pela rede cheguem ao aplicativo destinatário correto, permitindo uma comunicação eficiente e organizada entre clientes e servidores.

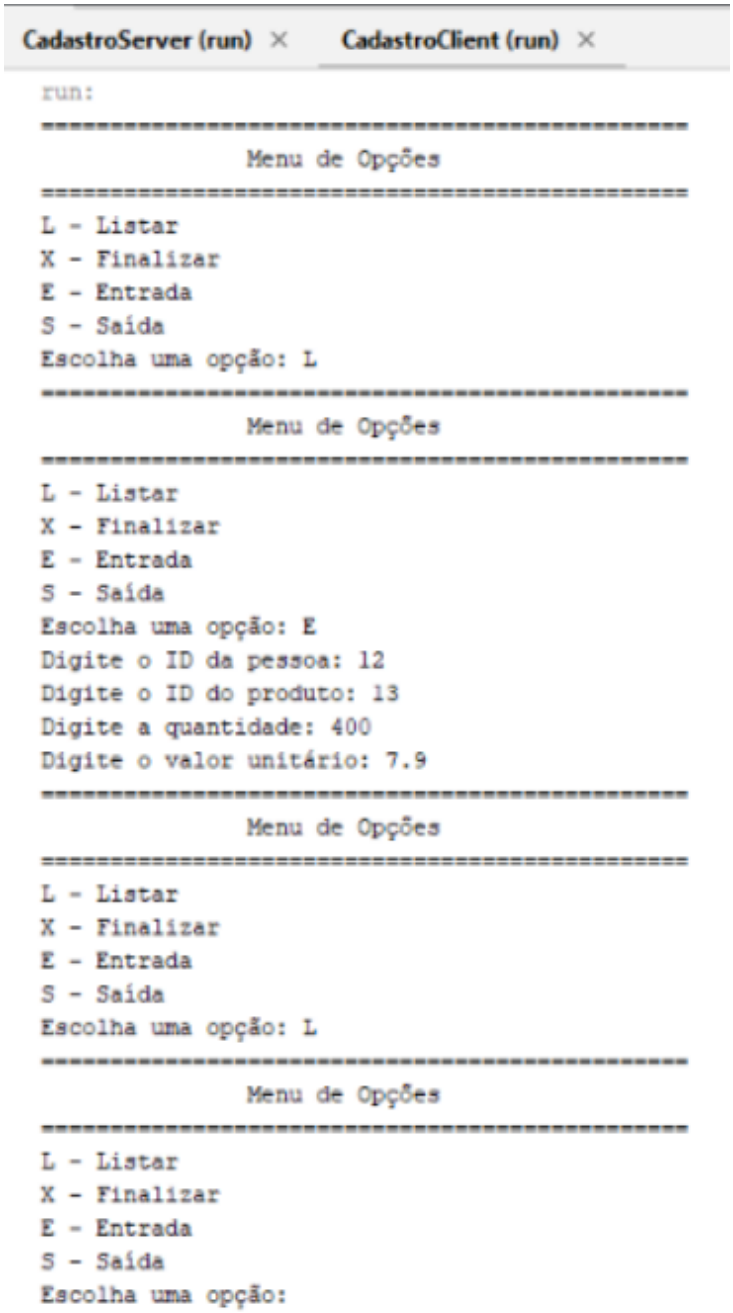
3. Para que servem as classes de entrada e saída ObjectInputStream ObjectOutputStream, e por que os objetos transmitidos devem ser serializáveis?

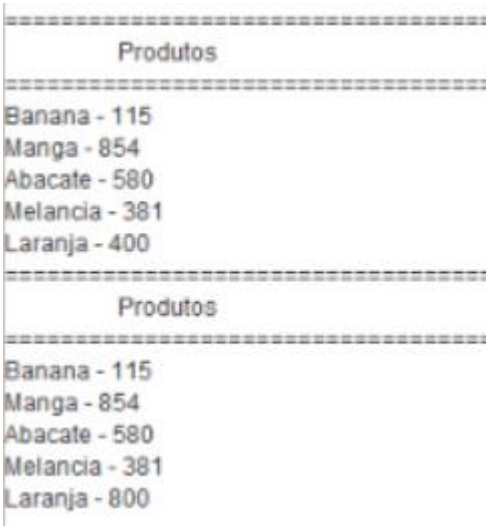
   As classes ObjectInputStream e ObjectOutputStream são utilizadas em Java para realizar a leitura e gravação de objetos em fluxos de dados, como arquivos ou conexões de rede, possibilitando a persistência de estados de objetos ou a transferência de objetos entre diferentes aplicações. Para que um objeto seja apto a ser enviado por esses fluxos, ele precisa ser serializável, isto é, capaz de ser convertido em uma cadeia de bytes e, em seguida, reconstituído a partir dessa cadeia. Essa exigência é crucial porque somente tipos de dados simples podem ser diretamente manipulados por esses fluxos; ao serializar um objeto, permite-se que ele seja desmembrado em um formato transportável por um fluxo e reconstruído posteriormente, viabilizando a comunicação de estruturas de dados complexas entre diferentes contextos de execução.

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

   Utilizando classes de entidades da Java Persistence API (JPA) no cliente ainda permite manter o acesso ao banco de dados isolado porque a JPA atua como uma camada de mediação, administrando o acesso aos dados sem revelar os detalhes das operações do banco de dados ao cliente. Isso é feito mapeando objetos Java a registros do banco de dados, controlando como esses objetos são persistidos e recuperados. Assim, o gerenciamento do acesso aos dados é centralizado na aplicação, permitindo a implementação de políticas de segurança robustas, gerenciamento de transações e controle sobre as operações de dados, sem expor o banco de dados a acessos diretos ou indevidos pelo cliente. Portanto, mesmo empregando entidades JPA do lado do cliente, a integridade e o isolamento do banco de dados são preservados pela gestão cuidadosa na camada de servidor ou persistência, assegurando que somente operações autorizadas sejam executadas.

2º Procedimento | Servidor Completo e Cliente Assíncrono:





Conclusão:

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

   Threads podem ser empregadas para gerenciar respostas assíncronas de servidores, facilitando o processamento de múltiplas operações em paralelo sem interromper a execução do fluxo principal da aplicação. Ao enviar uma requisição a um servidor, a aplicação pode criar uma Thread separada dedicada a aguardar e tratar essa resposta, o que permite que a thread principal continue operando outras atividades ou realizando novas solicitações. Essa abordagem evita que a aplicação fique parada à espera de respostas do servidor, otimizando a eficácia e a interatividade do usuário, particularmente em softwares que necessitam de resposta rápida ou que gerenciam várias requisições e respostas simultaneamente. Por meio do uso de Threads para a manipulação assíncrona, é possível desenvolver aplicações mais dinâmicas e com performance aprimorada, mesmo sob demandas de processamento elevadas.

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

   O método `invokeLater` da `SwingUtilities` é usado em aplicações com interface gráfica Java Swing para assegurar que certas ações sejam executadas na Event Dispatch Thread (EDT), que é a thread designada para gerir a interface de usuário. Esse procedimento é fundamental para modificar a GUI de forma segura, prevenindo problemas de sincronização e falhas visuais, pois alterações na interface precisam ocorrer nesta thread específica. Utilizando `invokeLater`, o trecho de código desejado é postergado para ser processado pela EDT, facilitando a realização de ajustes na GUI de maneira confiável, mesmo quando essas ações provêm de threads distintas. Essa abordagem é particularmente valiosa para atualizar elementos da interface do usuário em reação a eventos gerados fora da EDT, mantendo a interface responsiva e corretamente funcional.

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

   Em Java, objetos são enviados e recebidos através de sockets utilizando a serialização, que permite converter um objeto em uma sequência de bytes para transmissão, e posteriormente reconstituí-lo no lado receptor. Para enviar um objeto, o desenvolvedor deve primeiro criar um ObjectOutputStream ligado ao fluxo de saída do socket, e então utilizar o método writeObject para serializar e enviar o objeto. No lado receptor, é necessário criar um ObjectInputStream a partir do fluxo de entrada do socket, que pode então ser usado para ler (deserializar) o objeto transmitido com o método readObject. Essa abordagem requer que os objetos enviados implementem a interface Serializable, garantindo que possam ser convertidos para e a partir de uma sequência de bytes de forma eficaz. Esse processo facilita a comunicação de dados complexos entre clientes e servidores em uma rede, permitindo que objetos inteiros sejam transmitidos com facilidade, mantendo a integridade dos dados.

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

   Na comunicação via sockets em Java, a escolha entre operações assíncronas e síncronas impacta diretamente o bloqueio e o fluxo do processamento. Em uma abordagem síncrona, o cliente aguarda ativamente por uma resposta após enviar uma solicitação, resultando em um bloqueio até que a resposta seja recebida, o que pode levar a uma ineficiência em aplicações que requerem alta responsividade ou lidam com múltiplas tarefas simultaneamente. Por outro lado, o uso de comportamento assíncrono permite que o cliente continue executando outras operações enquanto espera pela resposta, eliminando o bloqueio desnecessário e melhorando a eficiência geral. Essa diferença essencial entre esperar ativamente (comportamento síncrono) e continuar o processamento sem interrupções (comportamento assíncrono) destaca a importância da escolha do modelo de comunicação baseado nas necessidades específicas da aplicação, balanceando a complexidade do código com a performance desejada.