



UNIVERSIDADE ESTÁCIO DE SÁ

DESENVOLVIMENTO FULLSTACK

Mundo 03 - Nível 05

RPG0018 - Por que não paralelizar

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

Ruan Hernandes Finamor Correia

Matrícula 202208175252

Gravataí – RS

2023

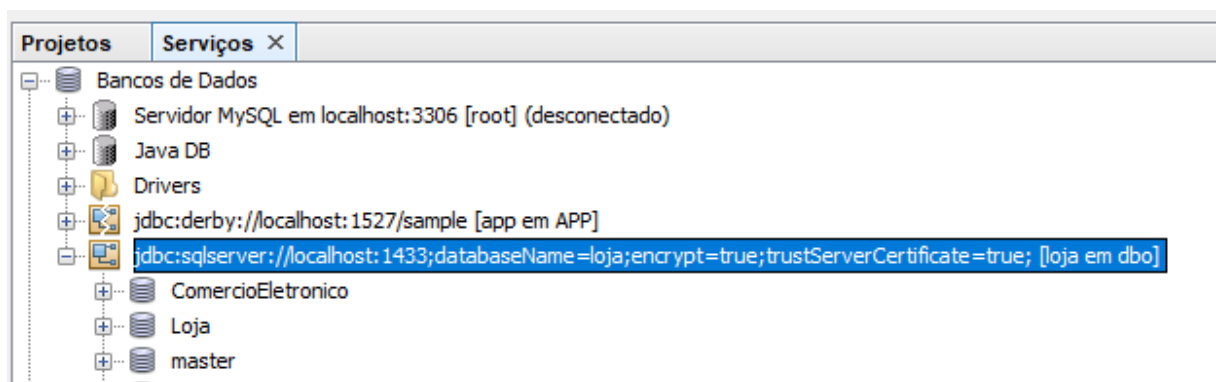
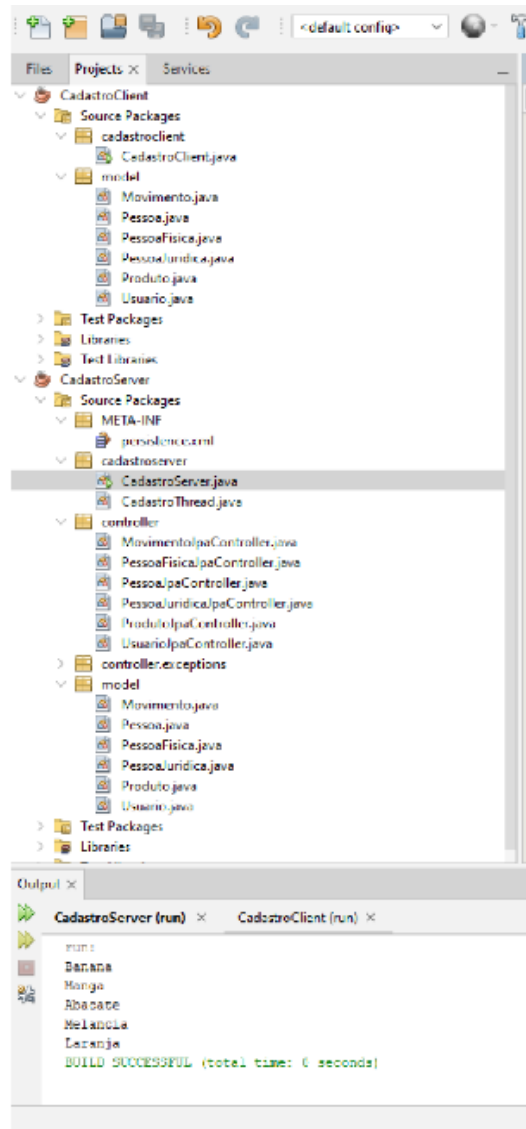
Objetivo da Prática

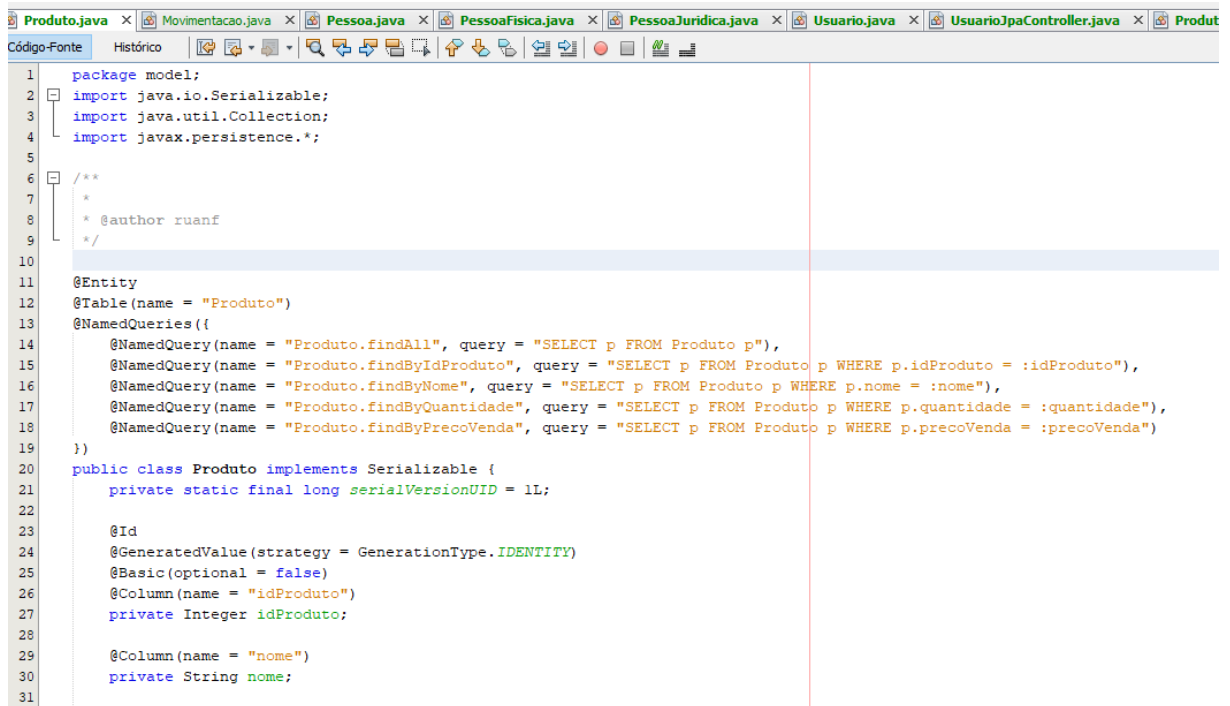
1. Criar servidores Java com base em Sockets.
2. Criar clientes síncronos para servidores com base em Sockets.
3. Criar clientes assíncronos para servidores com base em Sockets.
4. Utilizar Threads para implementação de processos paralelos.
5. No final do exercício, o aluno terá criado um servidor Java 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

Códigos solicitados no roteiro desta aula

```
18 public class CadastroClient {
19
20     /**
21      * @param args the command line arguments
22      * @throws java.io.IOException
23      * @throws java.lang.ClassNotFoundException
24      */
25     public static void main(String[] args) throws IOException, ClassNotFoundException {
26
27         // Instanciar um Socket apontando para localhost, na porta 4321.
28         Socket s1 = new Socket("localhost", 4321);
29
30         // Encapsular os canais de entrada e saída.
31         ObjectOutputStream out = new ObjectOutputStream(s1.getOutputStream());
32         ObjectInputStream in = new ObjectInputStream(s1.getInputStream());
33
34         out.writeObject(s1); // usuário
35         out.writeObject(s2); // senha
36         out.writeObject(s3); // comando
37     }
```





```

1 package model;
2 import java.io.Serializable;
3 import java.util.Collection;
4 import javax.persistence.*;
5
6 /**
7  *
8  * @author ruanf
9  */
10
11 @Entity
12 @Table(name = "Produto")
13 @NamedQueries({
14     @NamedQuery(name = "Produto.findAll", query = "SELECT p FROM Produto p"),
15     @NamedQuery(name = "Produto.findByIdProduto", query = "SELECT p FROM Produto p WHERE p.idProduto = :idProduto"),
16     @NamedQuery(name = "Produto.findByName", query = "SELECT p FROM Produto p WHERE p.nome = :nome"),
17     @NamedQuery(name = "Produto.findByQuantidade", query = "SELECT p FROM Produto p WHERE p.quantidade = :quantidade"),
18     @NamedQuery(name = "Produto.findByPrecoVenda", query = "SELECT p FROM Produto p WHERE p.precoVenda = :precoVenda")
19 })
20 public class Produto implements Serializable {
21     private static final long serialVersionUID = 1L;
22
23     @Id
24     @GeneratedValue(strategy = GenerationType.IDENTITY)
25     @Basic(optional = false)
26     @Column(name = "idProduto")
27     private Integer idProduto;
28
29     @Column(name = "nome")
30     private String nome;
31

```

Análise e conclusão:

P – Como funcionam as classes Socket e ServerSocket?

R – Socket é usado no lado do cliente para estabelecer conexões com um servidor remoto, permitindo o envio e recebimento de dados, enquanto ServerSocket é usado no lado do servidor para aguardar e aceitar conexões de clientes, permitindo a comunicação com múltiplos clientes simultaneamente.

P – Qual a importância das portas para a conexão com servidores?

R – Elas identificam serviços específicos, permitindo o roteamento eficiente do tráfego, além de facilitar a comunicação entre cliente e servidor e também oferecer mais segurança.

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

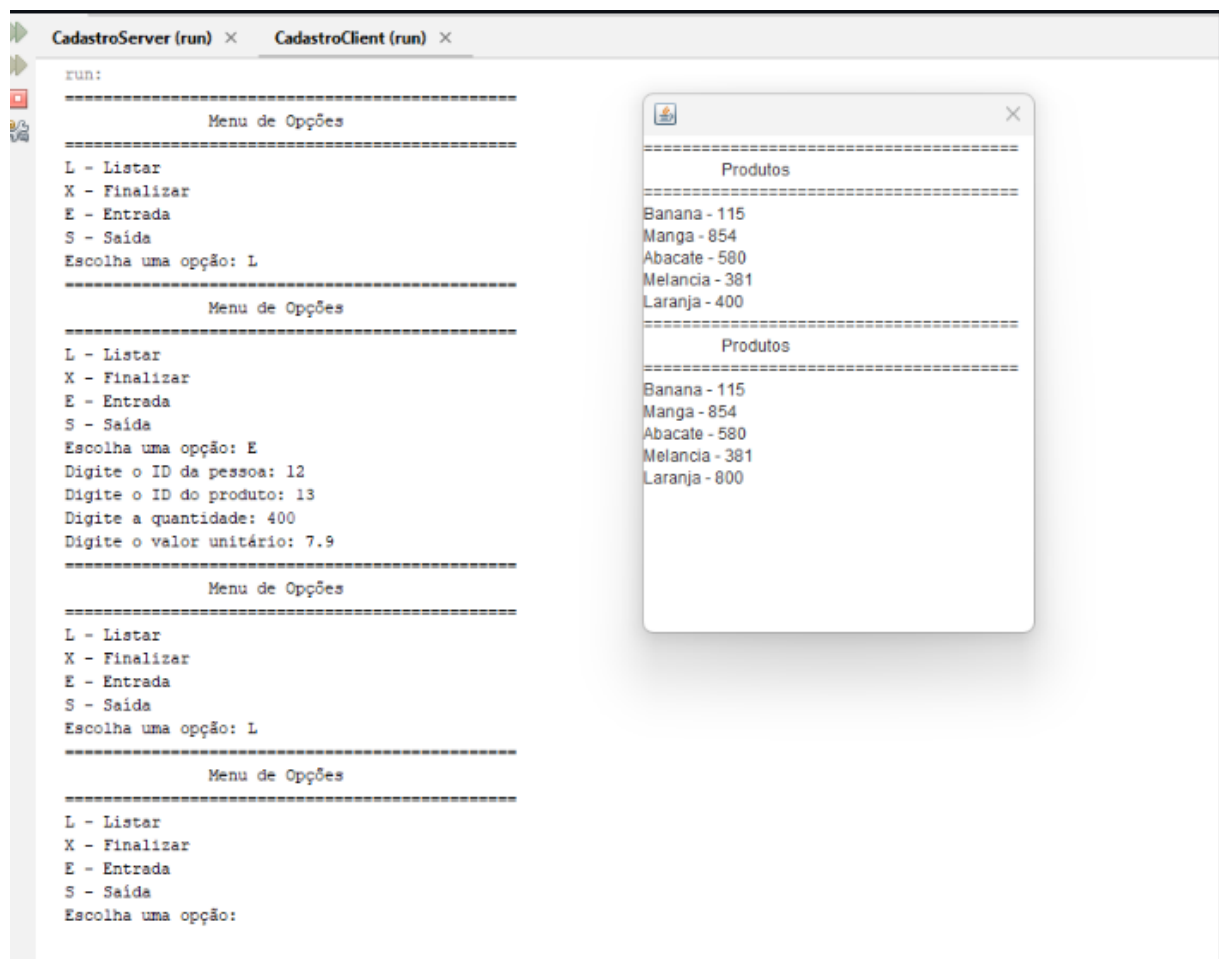
R – Elas servem para serializar e desserializar objetos, permitindo a transferência de objetos entre sistemas. Os objetos precisam ser serializáveis para que possam ser convertidos em bytes e depois reconstruídos.

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

R – Por conta da arquitetura das camadas, onde o acesso ao banco de dados ocorre apenas no servidor.

2º Procedimento

Resultado da execução dos códigos



Análise e conclusão:

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

R – Criando threads separadas que executam tarefas sem bloqueia a thread principal, o que permite que a aplicação continue executando outras tarefas enquanto aguarda a resposta.

P – Para que serve o método `invokeLater`, da classe `SwingUtilities`?

R – Para garantir que o código seja executado na thread de UI, o que evita problemas de concorrência e ajuda a manter a interface responsiva.

P – Como os objetos são enviados e recebidos pelo Socket Java?

R – Usando as classes `ObjectOutputStream` e `ObjectInputStream`. Os objetos precisam ser serializáveis para serem transmitidos assim.

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

R – O comportamento síncrono bloqueia a thread principal durante operações, tornando a aplicação menos responsiva, porém mais simples. Já o comportamento assíncrono não bloqueia a thread, o que torna a aplicação mais responsiva, porém requer uma lógica mais complexa.