



Mapeamento Objeto-Relacional e DAO

Fabio Lazaro de Almeida

Polo Centro – São Bernardo do Campo
Digital – Virtual – 2024.1

Link para código fonte: <https://github.com/scooutzz/Paralelizando.git>

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.

1º Procedimento | Criando o Servidor e Cliente de Teste

- a) Como funcionam as classes Socket e ServerSocket?

A classe 'Socket' é usada para representar o lado do cliente de uma conexão de rede, enquanto o 'ServerSocket' é usado para representar o lado do servidor.

- b) Qual a importância das portas para a conexão com servidores?

As portas são usadas para identificar processos específicos em um servidor. Cada porta corresponde a um serviço específico, permitindo que múltiplos serviços sejam executados simultaneamente.

- c) Para que servem as classes de entrada e saída ObjectInputStream e ObjectOutputStream, e por que objetos transmitidos devem ser serializáveis?

As classes 'ObjectInputStream', e "ObjectOutputStream" são usadas para ler e escrever objetos em um fluxo de entrada ou saída. E são

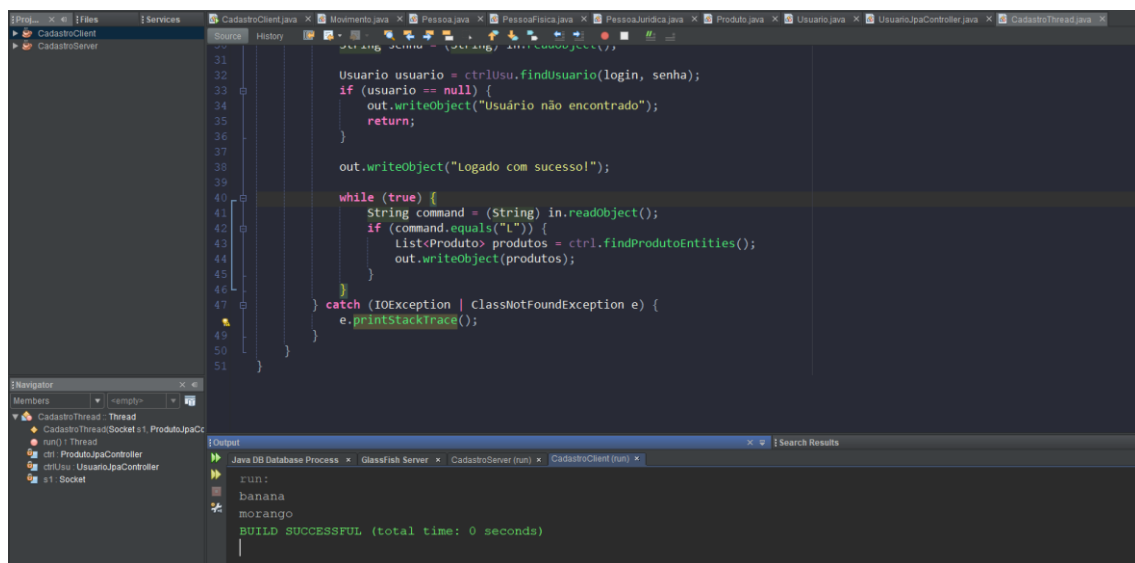
serializáveis para que possam ser convertidos em um formato que pode ser enviado, e depois reconstruído.

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

O isolamento do acesso ao banco de dados é garantido porque o gerenciamento das transações e persistência dos dados ocorrem no lado do servidor. O cliente apenas envia comandos, e não acessa o banco diretamente.

Códigos solicitados:

Pesquisa Cadastaro Server



```
31
32
33 Usuario usuario = ctrlUsu.findUsuario(login, senha);
34 if (usuario == null) {
35     out.writeObject("Usuário não encontrado");
36     return;
37 }
38
39 out.writeObject("Logado com sucesso!");
40
41 while (true) {
42     String command = (String) in.readObject();
43     if (command.equals("L")) {
44         List<Produto> produtos = ctrl.findProdutoEntities();
45         out.writeObject(produtos);
46     }
47 } catch (IOException | ClassNotFoundException e) {
48     e.printStackTrace();
49 }
50
51 }
```

Output:

```
run:
banana
morango
BUILD SUCCESSFUL (total time: 0 seconds)
```

2º Procedimento | Servidor Completo e Cliente Assíncrono

Criar uma segunda versão da Thread de comunicação no projeto do servidor, com a funcionalidade de receber comandos para entrada (E) ou saída (S) de produtos, configurar um objeto Movimento, persistir o movimento e atualizar a quantidade de produtos.

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

As Threads permitem que o servidor processe múltiplas requisições simultaneamente, garantindo que o sistema não fique bloqueado enquanto aguarda as respostas. Isso é crucial para manter a responsividade e a eficiência do sistema, especialmente em ambientes com alto volume de transações.

b) Para que serve o método `invokeLater`, da classe `SwingUtilities`?

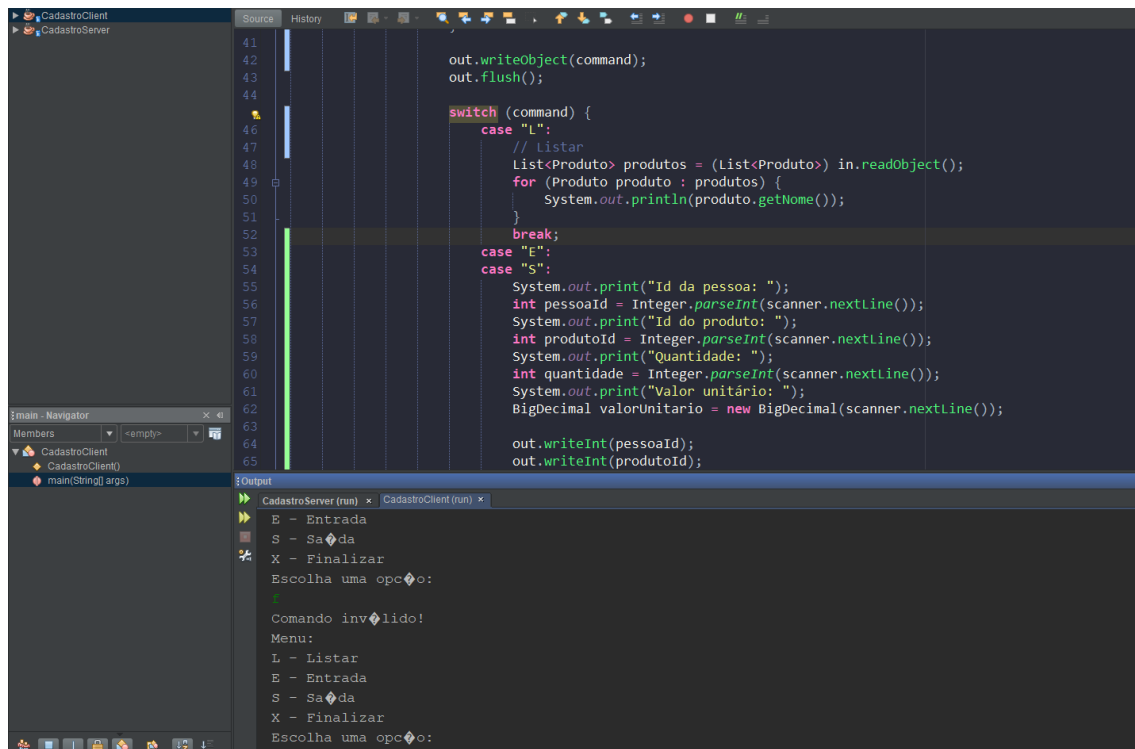
O método `invokeLater` da classe `SwingUtilities` é usado para agendar a execução de um pedaço de código na thread de despacho de eventos do Swing. Isso é essencial para garantir que operações que atualizam a interface gráfica sejam feitas de forma segura, evitando conflitos de concorrência.

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

No comportamento assíncrono, os clientes podem continuar suas operações enquanto aguardam respostas do servidor, o que melhora a responsividade e a eficiência geral do sistema. Por outro lado, no comportamento síncrono, o cliente fica bloqueado até receber a resposta do servidor, o que pode causar lentidão se houver demora na resposta ou se o servidor estiver ocupado com outras requisições.

Códigos solicitados:

Segunda execução do cadastro server



```
41  
42 out.writeObject(command);  
43 out.flush();  
44  
45 switch (command) {  
46     case "L":  
47         // Listar  
48         List<Produto> produtos = (List<Produto>) in.readObject();  
49         for (Produto produto : produtos) {  
50             System.out.println(produto.getNome());  
51         }  
52         break;  
53     case "E":  
54     case "S":  
55         System.out.print("Id da pessoa: ");  
56         int pessoaId = Integer.parseInt(scanner.nextLine());  
57         System.out.print("Id do produto: ");  
58         int produtoId = Integer.parseInt(scanner.nextLine());  
59         System.out.print("Quantidade: ");  
60         int quantidade = Integer.parseInt(scanner.nextLine());  
61         System.out.print("Valor unitário: ");  
62         BigDecimal valorUnitario = new BigDecimal(scanner.nextLine());  
63  
64         out.writeInt(pessoaId);  
65         out.writeInt(produtoId);  
66     }
```

main - Navigator
Members
CadastroClient
main(String[] args)

Output
CadastroServer (run) x CadastroClient (run) x
E - Entrada
S - Salva
X - Finalizar
Escolha uma opção:
Comando inválido!
Menu:
L - Listar
E - Entrada
S - Salva
X - Finalizar
Escolha uma opção:

Conclusão

A prática de comunicação cliente servidor em Java, utilizando sockets e threads, mostra a importância de compreender os conceitos de serialização, fluxos de entrada e saída, e a utilização de portas para gerenciar conexões de rede. As threads permitem um tratamento assíncrono eficiente, melhorando a capacidade de resposta e a escalabilidade do servidor. A escolha entre comunicação síncrona e assíncrona depende das necessidades específicas da aplicação, com a comunicação assíncrona oferecendo uma melhor utilização dos recursos do sistema, embora seja mais complexa de implementar. Em resumo, a combinação dessas tecnologias proporcionam uma base sólida para o desenvolvimento de sistemas distribuídos robustos e eficientes.