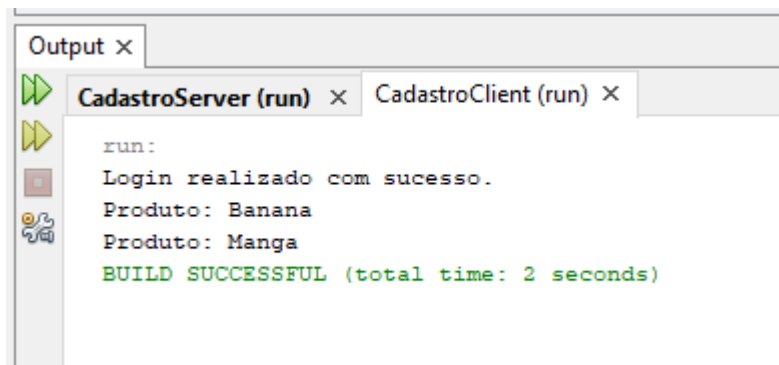
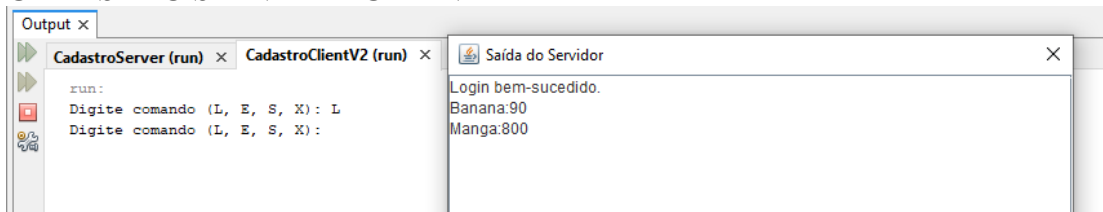


CURSO: DESENVOLVIMENTO FULL STACK**DISCIPLINA: POR QUE NÃO PARALELIZAR?****TURMA: 2024.1****SEMESTRE: 3º****ALUNO: ENILDO ARAÚJO DE OLIVEIRA JÚNIOR MAT.: 202402159101****GIT: <https://github.com/Enildo/JavaSocket>****Missão Prática - Nível 5****Objetivos da prática**

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.

CÓDIGOS JAVANo GIT <https://github.com/Enildo/JavaSocket>**Resultados****CADASTRO SERVER E CLIENTE 1**

```
Output x
CadastroServer (run) x CadastroClient (run) x
run:
Login realizado com sucesso.
Produto: Banana
Produto: Manga
BUILD SUCCESSFUL (total time: 2 seconds)
```

CADASTRO SERVER E CLIENTE 2

```
Output x
CadastroServer (run) x CadastroClientV2 (run) x Saída do Servidor x
run:
Digite comando (L, E, S, X): L
Digite comando (L, E, S, X):
Login bem-sucedido.
Banana:90
Manga:800
```

ANÁLISE E CONCLUSÃO

1. Como funcionam as classes Socket e ServerSocket?

A classe ServerSocket é usada no servidor para escutar conexões em uma porta específica.

A classe Socket é usada tanto no cliente quanto no servidor para estabelecer a conexão e permitir a comunicação entre as duas máquinas por meio de streams de entrada e saída.

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

As portas servem para identificar diferentes serviços dentro de uma mesma máquina.

Por exemplo, o servidor web pode estar na porta 80, o servidor de banco na 1433, e o seu sistema de cadastro na 4321.

Sem as portas, o sistema operacional não saberia para qual serviço direcionar uma conexão recebida.

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

ObjectOutputStream envia objetos pela rede.

ObjectInputStream recebe objetos do outro lado.

Os objetos devem ser serializáveis (implementando Serializable) porque eles precisam ser convertidos para bytes antes de serem enviados.

A serialização garante que o objeto mantenha sua estrutura ao ser transmitido.

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

Porque o cliente não acessa o banco diretamente.

Ele envia comandos para o servidor, que é o único que possui acesso ao banco via JPA.

No cliente, as classes JPA são usadas apenas como modelos de dados para facilitar a comunicação, mas não há conexão com o banco.

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

As Threads permitem que o cliente continue executando outras tarefas enquanto espera pelas respostas do servidor. Com uma thread separada, a leitura dos dados recebidos pode acontecer em segundo plano, sem bloquear a execução do restante da aplicação (como a interface gráfica ou o menu de comandos).

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

O método invokeLater é usado para garantir que atualizações na interface gráfica (Swing) sejam feitas na Thread de interface do usuário (EDT – Event Dispatch Thread). Isso evita problemas de concorrência e travamentos ao alterar componentes visuais fora da thread principal da interface.

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

Os objetos são enviados através de `ObjectOutputStream.writeObject()` e recebidos com `ObjectInputStream.readObject()`. Para isso funcionar, os objetos devem implementar a interface `Serializable`, permitindo que sejam convertidos em bytes para transmissão via rede e reconstruídos no destino.

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

No comportamento síncrono, o cliente espera (bloqueia) até receber a resposta do servidor antes de continuar. Isso pode travar a aplicação enquanto espera.

No comportamento assíncrono, uma thread separada lida com a resposta, permitindo que o cliente continue operando normalmente (ex: aceitando comandos ou mantendo a interface ativa).

O assíncrono é ideal quando o cliente precisa ser responsivo, como em interfaces gráficas.