

Campus: Santa Cruz da Serra - Duque de Caxias/RJ

Curso: Desenvolvimento full stack

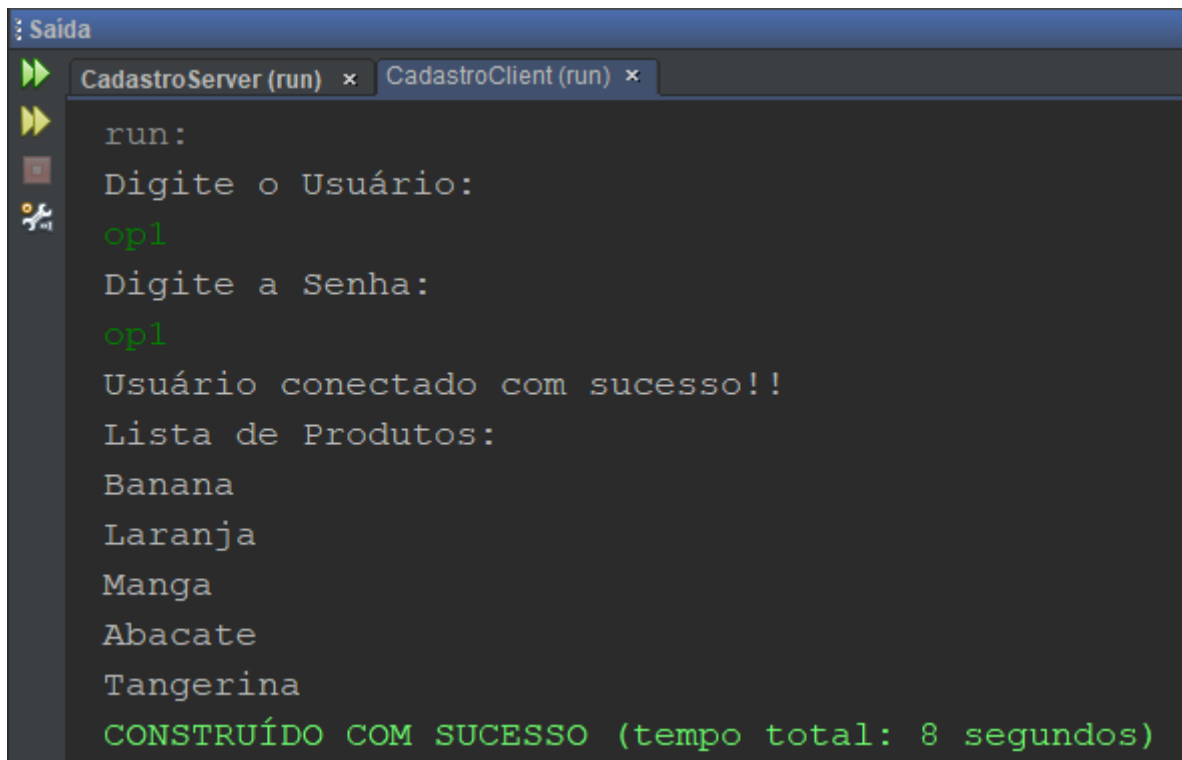
Disciplina: RPG0018 - Por que não paralelizar

Número da Turma: 9001

Semestre letivo: Mundo 3

Integrantes: Felipe Freaza Fidalgo

Por que não paralelizar (CadastroServer)



```
run:
Digite o Usuário:
opl
Digite a Senha:
opl
Usuário conectado com sucesso!!
Lista de Produtos:
Banana
Laranja
Manga
Abacate
Tangerina
CONSTRUÍDO COM SUCESSO (tempo total: 8 segundos)
```

Análise e Conclusão:

1º Procedimento | Criando o Servidor e Cliente de Teste

1.Como funcionam as classes Socket e ServerSocket?

Socket é usado para comunicação de rede em Java, permitindo conexões entre cliente e servidor.

Por outro lado, ServerSocket é empregado apenas pelo lado do servidor para aceitar conexões de clientes, aguardando em uma porta específica. Juntas, essas classes são cruciais na criação de aplicativos de rede em Java.

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

As portas são vitais para a comunicação com servidores, pois permitem identificar serviços específicos. Elas habilitam a multiplexação de diversos serviços em um único servidor, economizando recursos. Além disso, as portas contribuem para a segurança, possibilitando o controle do acesso por meio de firewalls. Também auxiliam no roteamento, direcionando o tráfego para servidores específicos em redes locais. Em resumo, as portas são essenciais para a organização, segurança e eficiência das conexões com servidores.

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

As classes `ObjectInputStream` e `ObjectOutputStream` em Java permitem serializar objetos em bytes para transmissão ou armazenamento. Os objetos transmitidos devem implementar a interface `Serializable` para indicar como devem ser serializados. Isso facilita a comunicação entre aplicativos e sistemas, mas requer cuidado de segurança, pois a desserialização pode ser vulnerável a ataques se os objetos não forem confiáveis. Essas classes são essenciais para transferir dados de objetos completos através de streams de entrada e saída em Java.

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

O uso de classes de entidades JPA no cliente permite o isolamento do acesso ao banco de dados devido ao mapeamento objeto-relacional e ao controle de transações.

O JPA traduz operações do cliente em instruções SQL apropriadas, enquanto mantém um cache de primeiro nível para melhorar o desempenho. O gerenciamento de entidades pelo JPA e a separação de camadas promovem a segurança e integridade do banco de dados, tornando-o mais seguro e eficiente no acesso aos dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

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

As Threads podem ser usadas para tratar respostas de forma assíncrona e eficiente do servidor. Isso permite que o cliente processe várias respostas em paralelo, melhorando a responsividade do aplicativo. É possível criar Threads individualmente ou usar um pool de Threads para controlar a execução. O uso de callbacks e a gestão adequada de recursos compartilhados são essenciais para garantir um tratamento seguro e eficaz das respostas assíncronas. É fundamental ter cuidado com a concorrência e a sincronização para evitar problemas de execução concorrente.

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

O método `invokeLater` da classe `SwingUtilities` em Java serve para agendar a execução de tarefas na "Event Dispatch Thread" (EDT), a thread responsável pela interface gráfica. Isso garante que operações de interface do usuário sejam executadas de maneira segura e evita bloqueios na GUI. É útil para atualizar elementos da interface a partir de threads não-EDT, mantendo a responsividade da aplicação e evitando problemas de concorrência.

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

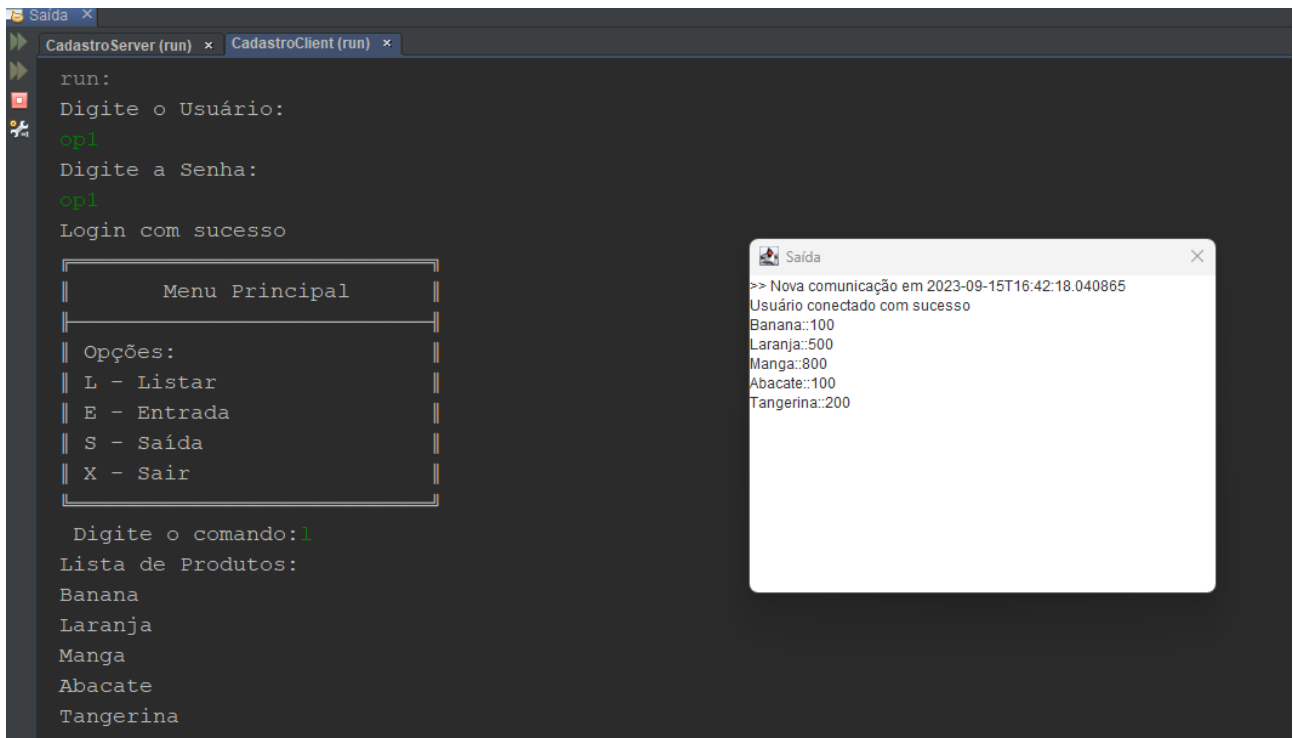
Em Java, objetos são enviados e recebidos por sockets usando `ObjectInputStream` e `ObjectOutputStream`. Para enviar, crie um `Socket` e use `ObjectOutputStream.writeObject()`. Para receber, crie um `Socket`, use `ObjectInputStream.readObject()`. Os objetos devem ser serializáveis e as partes devem concordar com a estrutura dos objetos enviados e recebidos.

4.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 síncrono em clientes com Socket Java bloqueia a thread principal, tornando o aplicativo menos responsivo. Embora seja mais simples de implementar, pode causar problemas de bloqueio. O comportamento assíncrono não bloqueia a thread principal, permitindo maior responsividade, mas é mais complexo de coordenar. A escolha depende das necessidades do aplicativo, considerando fatores como desempenho e complexidade.

Conclusão:

Nesta missão prática de nível 5 em Java, criamos um servidor e cliente com Sockets e Threads para comunicação. Utilizamos JPA para acessar um banco de dados SQL Server. O servidor autentica usuários, responde a comandos e permite entrada/saída de produtos. Implementamos um cliente assíncrono com interface gráfica e threading para interatividade. Essa prática expandiu nossas habilidades em programação distribuída Java, destacando o uso de Sockets, Threads e JPA, promovendo uma experiência interativa do usuário.



```
run:
Digite o Usuário:
cpl
Digite a Senha:
cpl
Login com sucesso

Menu Principal

Opções:
L - Listar
E - Entrada
S - Saída
X - Sair

Digite o comando: l
Lista de Produtos:
Banana
Laranja
Manga
Abacate
Tangerina
```

```
>> Nova comunicação em 2023-09-15T16:42:18.040865
Usuário conectado com sucesso
Banana::100
Laranja::500
Manga::800
Abacate::100
Tangerina::200
```