

**Campus:** Polo Ingleses

**Curso:** Desenvolvimento Full Stack

**Disciplina:** Nível 5 – Por que não paralelizar

**Turma:** 9001

**Semestre:** 23.3

**Aluna:** Maria Carolina Knudsen Boabaid

## **1º Procedimento:**

**Objetivo da prática:** Criar sistema servidor cliente utilizando comunicação por meio de sockets, implementando threads tanto no lado do cliente quanto no servidor, e acessando o banco de dados por meio da API de Persistência Java (JPA).

## **Resultados da execução dos códigos:**



```
18 |  
|  
PROBLEMAS 1 SAÍDA CONSOLE DE DEBURAÇÃO TERMINAL PORTAS Run: CadastroClientApplication + - □ □ ... X  
/dev/fd/12:18: command not found: compdef  
(base) mariacarolinaboabaid@MacBook-Air-de-Maria CadastroClient % /usr/bin/env /Library/Java/JavaVirtualMachines/jdk-21.jdk/Contents/Home/bin/java @/var/folders/4y/ggj8mv0d4h100c0wp600_klc0000gn/T/cp_eqy059x69qkie1fw3tj0b1x0r.argfile com.CadastroClient.CadastroClientApplication  
  
Spring  
:: Spring Boot :: (v3.2.0)  
2023-11-25T18:37:57.709-03:00 INFO 9892 --- [main] c.C.CadastroClientApplication : Starting CadastroClientApplication  
using Java 21.0.1 with PID 9892 (/Users/mariacarolinaboabaid/Downloads/CadastroClient/target/classes started by mariacarolinaboabaid in /Users/mariacarolinaboabaid/Downloads/CadastroClient)  
2023-11-25T18:37:57.711-03:00 INFO 9892 --- [main] c.C.CadastroClientApplication : No active profile set, falling back to 1 default profile: "default"  
2023-11-25T18:37:57.793-03:00 INFO 9892 --- [main] c.C.CadastroClientApplication : Started CadastroClientApplication in 0.387 seconds (process running for 0.683)  
Usuário conectado com sucesso  
Banana  
Laranja  
Manga
```

## **Análise e Conclusão:**

### **1. Como funcionam as classes Socket e ServerSocket?**

As classes Socket e ServerSocket são classes do Java que permitem a comunicação entre dois computadores em uma rede. A classe Socket representa uma conexão entre um cliente e um servidor, enquanto a classe ServerSocket representa um servidor que está escutando por conexões de clientes.

O funcionamento básico das classes Socket e ServerSocket é o seguinte:

- **Socket:** O cliente cria um objeto Socket e especifica o endereço e a porta do servidor com o qual deseja se conectar. Em seguida, o cliente chama o método `connect()` para iniciar a conexão.
- **ServerSocket:** O servidor cria um objeto ServerSocket e especifica a porta na qual deseja ouvir por conexões. Em seguida, o servidor chama o método `accept()` para aceitar uma conexão de um cliente.

Uma vez que a conexão entre o cliente e o servidor é estabelecida, os dois lados podem trocar dados usando os métodos de entrada e saída da classe Socket.

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

As portas são números que identificam diferentes serviços em um servidor. Cada serviço em um servidor tem uma porta associada. Por exemplo, o serviço HTTP, que é usado para servir páginas da web, usa a porta 80.

Quando um cliente deseja se conectar a um servidor, ele especifica a porta do serviço que deseja usar. Por exemplo, para se conectar a um servidor web, o cliente especifica a porta 80.

As portas são importantes para garantir que os diferentes serviços em um servidor não interfiram uns com os outros.

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

As classes `ObjectInputStream` e `ObjectOutputStream` são classes do Java que permitem a transmissão de objetos entre dois computadores em uma rede.

A classe `ObjectInputStream` é usada para ler objetos de um fluxo de entrada. A classe `ObjectOutputStream` é usada para escrever objetos em um fluxo de saída.

Para que um objeto possa ser transmitido usando as classes `ObjectInputStream` e `ObjectOutputStream`, ele deve ser serializável. Um objeto serializável é um objeto que pode ser convertido em uma sequência de bytes e convertido de volta em um objeto.

A serialização é um processo que converte um objeto em uma sequência de bytes. A desserialização é um processo que converte uma sequência de bytes em um objeto.

Os objetos serializáveis são usados em uma variedade de aplicações de rede, incluindo:

- Aplicações cliente-servidor
- Aplicações distribuídas
- Aplicações de comunicação

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

A JPA é uma especificação que permite a abstração do acesso ao banco de dados em aplicações Java. A JPA fornece uma API que permite aos desenvolvedores interagir com o banco de dados de forma independente do provedor de banco de dados.

A JPA usa o conceito de transações para garantir o isolamento do acesso ao banco de dados. Uma transação é um conjunto de operações que são executadas como uma unidade.

Quando uma transação é iniciada, o banco de dados é bloqueado para outros acessos. As operações da transação são executadas uma após a outra, e todas as alterações são aplicadas ao banco de dados apenas se todas as operações forem bem-sucedidas.

No caso de uma transação ser abortada, todas as alterações feitas durante a transação são revertidas.

No exemplo em questão, o cliente está utilizando as classes de entidades JPA para acessar o banco de dados. A JPA usa transações para garantir o isolamento do acesso ao banco de dados.

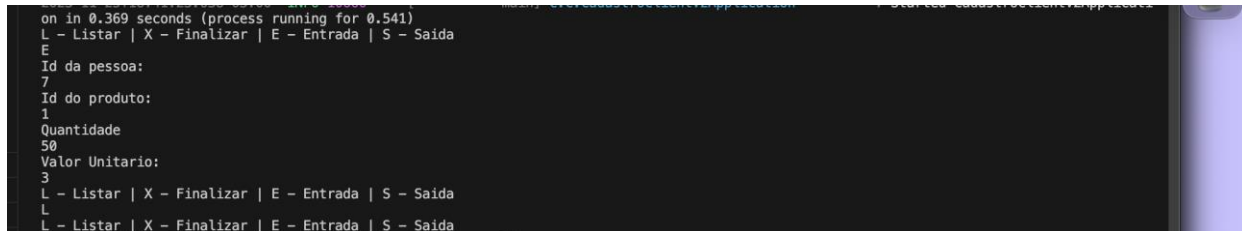
Assim, mesmo que o cliente esteja utilizando as classes de entidades JPA, o isolamento do acesso ao banco de dados é garantido.

Em resumo, as classes Socket e ServerSocket permitem a comunicação entre dois computadores em uma rede. As portas são números que identificam diferentes serviços em um servidor. As classes ObjectInputStream e ObjectOutputStream permitem a transmissão de objetos entre dois computadores em uma rede. Os objetos serializáveis podem ser transmitidos usando as classes ObjectInputStream e ObjectOutputStream. A JPA usa transações para garantir o isolamento do acesso ao banco de dados.

## **2º Procedimento:**

**Objetivo da prática: Criar o lado Cliente do sistema.**

### **Resultados da execução dos códigos:**



```
on in 0.369 seconds (process running for 0.541)
L - Listar | X - Finalizar | E - Entrada | S - Saída
E
Id da pessoa:
7
Id do produto:
1
Quantidade
50
Valor Unitario:
3
L - Listar | X - Finalizar | E - Entrada | S - Saída
L
L - Listar | X - Finalizar | E - Entrada | S - Saída
```

### **Análise e Conclusão:**

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

As Threads podem ser utilizadas para o tratamento assíncrono das respostas enviadas pelo Servidor criando uma Thread separada para cada resposta. Essa Thread é responsável por receber a resposta do Servidor e processá-la.

O processamento da resposta pode ser feito de forma assíncrona, ou seja, a Thread principal do cliente não precisa esperar até que a resposta seja recebida e processada para continuar executando. Isso permite que o cliente continue executando outras tarefas enquanto aguarda a resposta do Servidor.

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

O método `invokeLater()`, da classe `SwingUtilities`, é usado para executar uma tarefa assíncrona na interface gráfica do usuário (GUI).

O método `invokeLater()` recebe uma `Runnable` como argumento. A `Runnable` é executada em uma Thread separada da Thread principal da GUI.

O método `invokeLater()` é útil para executar tarefas que não precisam ser executadas imediatamente na Thread principal da GUI. Por exemplo, podemos usar o método `invokeLater()` para executar uma tarefa que não afeta a interface do usuário, como uma tarefa de acesso ao banco de dados.

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

Os objetos são enviados e recebidos pelo Socket Java usando as classes `ObjectInputStream` e `ObjectOutputStream`.

A classe `ObjectInputStream` é usada para ler objetos de um fluxo de entrada. A classe `ObjectOutputStream` é usada para escrever objetos em um fluxo de saída.

Para enviar um objeto por um Socket Java, precisamos criar um objeto `ObjectOutputStream` associado ao Socket. Em seguida, podemos escrever o objeto no fluxo de saída.

Para receber um objeto por um Socket Java, precisamos criar um objeto `ObjectInputStream` associado ao Socket. Em seguida, podemos ler o objeto do fluxo de entrada.

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

O comportamento assíncrono e o comportamento síncrono têm características diferentes relacionadas ao bloqueio do processamento.

No comportamento assíncrono, o processamento da resposta do Servidor é feito em uma Thread separada da Thread principal do cliente. Isso significa que a Thread principal do cliente não precisa esperar até que a resposta seja recebida e processada para continuar executando.

No comportamento síncrono, o processamento da resposta do Servidor é feito na Thread principal do cliente. Isso significa que a Thread principal do cliente precisa esperar até que a resposta seja recebida e processada para continuar executando.