

**ServerSocket:** Esta classe é utilizada no lado do servidor para aguardar e aceitar conexões de clientes. O ServerSocket fica esperando por pedidos de

conexão e, quando um pedido é recebido, cria um novo Socket para lidar com a comunicação com o cliente.

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

As portas são números de 16 bits que identificam processos específicos em um dispositivo de rede. Em um servidor, diferentes serviços podem ser associados a diferentes portas. Por exemplo, o serviço da web geralmente usa a porta 80, enquanto o serviço de transferência de arquivos (FTP) usa a porta 21.

As portas são vitais para garantir que os dados cheguem ao processo de destino correto no servidor. Quando você se conecta a um servidor, seu programa cliente precisa especificar a porta do serviço ao qual deseja se conectar. Se não houver uma porta, ou se a porta estiver errada, a comunicação pode falhar.

## **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` em Java são usadas para a serialização de objetos. A serialização é o processo de converter um objeto em uma sequência de bytes, e a desserialização é o processo reverso, convertendo a sequência de bytes de volta para um objeto.

Essas classes são frequentemente usadas em redes para transmitir objetos entre diferentes máquinas ou processos. A serialização é importante para garantir que o objeto possa ser convertido em uma forma que pode ser transmitida pela rede e recriado no destino. Para usar essas classes, os objetos que você deseja transmitir devem implementar a interface `Serializable`. Isso informa ao Java que é seguro transformar esse objeto em uma sequência de bytes.

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

Java Persistence API (JPA) é uma especificação do Java que descreve o gerenciamento de dados relacionais em aplicativos Java. Ela fornece um meio de mapear objetos Java para tabelas em um banco de dados relacional. A utilização de classes de entidades JPA no cliente permite que o acesso ao banco de dados seja isolado por meio de operações CRUD (Create, Read, Update, Delete).

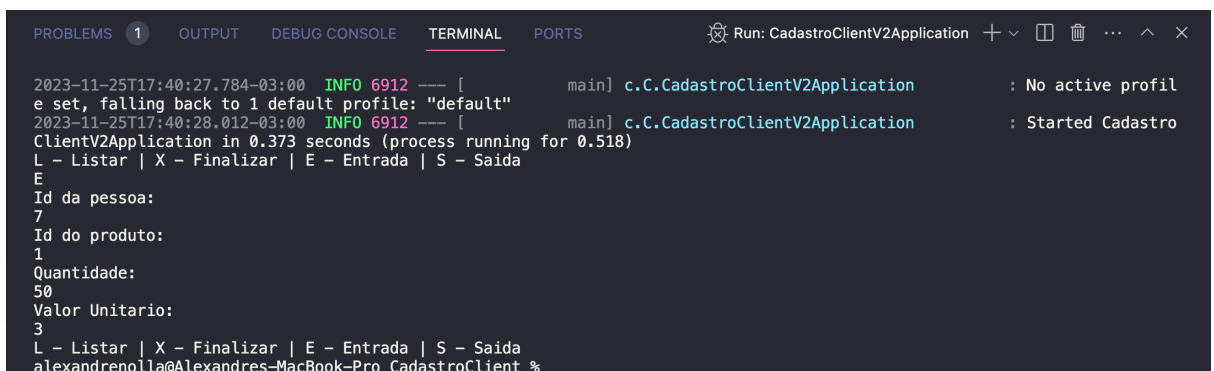
O isolamento do acesso ao banco de dados é alcançado pela camada de persistência. As operações no banco de dados são encapsuladas em métodos de objetos de entidade JPA, e a lógica de negócios do aplicativo interage com esses objetos. A camada de persistência cuida da tradução entre objetos Java e estruturas de banco de dados subjacentes. Isso facilita a manutenção do código, já que as mudanças no banco de dados podem ser refletidas nos objetos de entidade sem afetar diretamente a lógica de negócios.

Além disso, JPA geralmente utiliza transações para garantir a consistência dos dados. As transações permitem que um conjunto de operações seja tratado como uma unidade atômica, garantindo que todas as operações sejam bem-sucedidas ou que nenhuma delas seja realizada. Isso contribui para o isolamento e a integridade dos dados.

## **2º Procedimento:**

**Objetivo da prática: Criar o lado Cliente, integrado com o Servidor, com o uso de Threads e acessando banco de dados via JPA.**

**Resultados da execução dos códigos:**



```
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS Run: CadastroClientV2Application + - [ ] [ ] ... ^ x
2023-11-25T17:40:27.784-03:00 INFO 6912 --- [main] c.C.CadastroClientV2Application : No active profil
e set, falling back to 1 default profile: "default"
2023-11-25T17:40:28.012-03:00 INFO 6912 --- [main] c.C.CadastroClientV2Application : Started Cadastro
ClientV2Application in 0.373 seconds (process running for 0.518)
L - Listar | X - Finalizar | E - Entrada | S - Saida
E
Id da pessoa:
7
Id do produto:
1
Quantidade:
50
Valor Unitario:
3
L - Listar | X - Finalizar | E - Entrada | S - Saida
alexandrenolla@Alexandres-MacBook-Pro CadastroClient %
```

## **Análise e Conclusão:**

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

Threads podem ser utilizadas para tratamento assíncrono das respostas do servidor ao delegar o processamento das respostas para uma thread separada. Isso é particularmente útil em operações de entrada e saída (I/O), onde o bloqueio pode ocorrer enquanto aguarda a chegada de dados.

Ao receber uma resposta do servidor, você pode iniciar uma nova thread para processar essa resposta sem bloquear a execução principal do programa. Isso permite que o programa continue a executar 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 em ambientes de interface gráfica do usuário (GUI) Swing em Java. Ele é usado para executar uma determinada tarefa na thread de despacho de eventos do Swing, também conhecida como a Event Dispatch Thread (EDT).

Como as operações de GUI em Swing não são thread-safe, todas as atualizações na interface gráfica devem ser feitas na EDT. O `invokeLater` permite que você agende a execução de uma tarefa na EDT, garantindo assim a segurança da thread. Isso é especialmente importante quando você está trabalhando com eventos e atualizações de interface do usuário em aplicativos Swing.

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

Em Java, os objetos podem ser enviados e recebidos por meio de sockets usando as classes `ObjectInputStream` e `ObjectOutputStream`. Estas classes permitem a serialização e desserialização de objetos, tornando possível a transmissão desses objetos pela rede.

Quando você envia um objeto através de um `ObjectOutputStream`, o objeto é serializado em uma sequência de bytes que pode ser transmitida pela rede. No lado receptor, o `ObjectInputStream` é usado para desserializar essa sequência de bytes de volta para o objeto original.

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

**Comportamento Síncrono:** No comportamento síncrono, o cliente espera pela resposta do servidor antes de continuar a execução.

O código bloqueia enquanto aguarda a resposta do servidor, o que pode levar a uma experiência de usuário menos responsiva.

Geralmente mais simples de implementar e entender.

**Comportamento Assíncrono:** No comportamento assíncrono, o cliente continua a execução principal enquanto aguarda a resposta do servidor em uma thread separada.

Permite que o programa seja mais responsivo, pois outras tarefas podem ser executadas enquanto a comunicação com o servidor está em andamento.

Requer cuidados extras para garantir a consistência dos dados e evitar condições de corrida.

A escolha entre comportamento assíncrono e síncrono depende dos requisitos específicos do aplicativo. O comportamento assíncrono é muitas vezes preferido em aplicativos que precisam ser responsivos e lidar com operações de rede ou I/O sem bloquear a interface do usuário. No entanto, requer cuidado adicional para lidar com concorrência e sincronização.