	Universidade Estácio Campus Barra – RJ Curso de Desenvolvimento Full Stack Relatório da Missão Prática 5 – Mundo 3
Disciplina:	RPG0018 - Por que não paralelizar
Nome:	Lucia Maria de Lima Martins – 202309761581
Turma:	9001 – 3º Semestre

Servidores e clientes baseados em Socket, com uso de Threads tanto no lado cliente quanto no lado servidor, acessando o banco de dados via JPA.

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.

1º Procedimento | Criando o Servidor e Cliente de Teste

Análise e Conclusão:

a) Como funcionam as classes Socket e ServerSocket?

Um socket é um mecanismo de comunicação (dois sentidos) entre dois programas a funcionar (normalmente) numa rede.

O servidor espera que um cliente faça um pedido de ligação através desse socket Servidor (escuta numa porta específica) Cliente Pedido de ligação.

O servidor ao aceitar a ligação cria um novo socket para uma porta diferente (e assim permite novas ligações), no lado do cliente um socket é criado e é usado para comunicar com o servidor (numa porta disponível na máquina cliente). Servidor (escuta numa porta específica) Cliente.

A classe ServerSocket é usada para fornecer implementação independente de sistema do lado do servidor de uma conexão de soquete cliente/servidor. Esta classe possui um construtor onde passamos a porta que desejamos usar para escutar as conexões, e lança uma exceção se não puder escutar na porta especificada (por exemplo, a porta já está sendo usada).

```
ServerSocket server = new ServerSocket(4321);
```

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

As portas são locais virtuais dentro de um sistema operacional onde as conexões de rede começam e terminam. Elas ajudam os computadores a classificar o tráfego de rede que recebem.

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

ObjectOutputStream grava tipos de dados primitivos e gráficos de objetos Java em um OutputStream. Os objetos podem ser lidos (reconstituídos) usando um ObjectInputStream. O armazenamento persistente de objetos pode ser realizado usando um arquivo para o fluxo.

ObjectInputStream desserializa os dados e objetos primitivos previamente escritos por ObjectOutputStream .

Em um projeto de software, em muitos casos, há necessidade de transferir os dados e isso pode ser tratado usando ObjectOutputStream e ObjectInputStream de pacotes Java IO. Normalmente, os dados são gravados em formato binário e, portanto, não podemos visualizar o conteúdo. A serialização é o processo de gravar um objeto

em um fluxo de saída. Podemos escrever um objeto de classe em si, que contém uma combinação de tipos de dados primitivos e gráficos de objetos Java. A desserialização é o processo de reconstruir um objeto a partir de um fluxo de entrada.

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 pois a JPA, como parte da especificação Java EE, é projetada para separar claramente as operações de lógica de negócios da camada de persistência de dados.

As classes de entidade (@Entity) JPA representam a estrutura dos dados e são usadas tanto no cliente quanto no servidor, mas a gestão real do acesso ao banco de dados é realizada exclusivamente no lado do servidor. Isso significa que as operações de banco de dados, tais como inserções, consultas, atualizações, exclusões (CRUD create, read, update, delete), são executadas por meio de sessões e transações gerenciadas pelo servidor, geralmente através de um provedor de persistência.

Portanto, mesmo que o cliente use as mesmas classes de entidade, ele não tem acesso direto ao banco de dados, o que garante o isolamento e a segurança dos dados.

2º Procedimento | Servidor Completo e Cliente Assíncrono

Análise e Conclusão:

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

Quando um servidor recebe múltiplas solicitações de clientes, pode utilizar Threads parra processar cada solicitação simultaneamente, de modo a evitar assim um bloqueio, enquanto aguarda, por exemplo, por uma operação de longa duração em uma única solicitação.

Cada solicitação é processada em uma Thread separada, o que permite que o servidor continue a receber e responder outras solicitações. Essa abordagem melhora a eficiência e a capacidade de resposta do servidor, pois as Threads operam de forma independente e paralela, isto garante que o processamento de uma solicitação não seja interrompido ou atrasado pelas outras.

O uso de Threads no servidor possibilita o tratamento assincrono das respostas, o que melhora o desempenho geral do sistema em ambientes de rede.

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

Um método invokeLater() é um método estático da classe SwingUtilities e pode ser usado para executar uma tarefa de forma assíncrona no thread do AWT Event dispatcher . O método SwingUtilities.invokeLater() funciona como SwingUtilities.invokeAndWait(), exceto que ele coloca a solicitação na fila de eventos e retorna imediatamente . Um método invokeLater() não espera que o bloco de código dentro do Runnable referenciado por um alvo seja executado.

c) Como os objetos são enviados e recebidos pelo Socket Java?

Em Java, os objetos são enviados e recebidos através de sockets com uso de fluxos (streams), especificamente as classes ObjectOutputStream e ObjectInputStream.

Para enviar um objeto, primeiro serializa-se o objeto, ou seja, o objetivo é convertido em uma sequência de bytes, com uso de ObjectOutputStream; em seguida, é enviado através de um Socket.

Do lado receptor, ObjectInputStream é usado para ler a sequência de bytes do Socket e desserializá-lo (reconstruir o objeto original). É crucial que o objeto a ser enviado implemente a interface Serializable, o que permite essa serialização e desserialização.

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

Na programação de rede com Sockets em Java, o comportamento síncrono e assíncrono possui implicações significativas no bloqueio do processamento.

As operações síncronas bloqueiam a execução do programa até que a operação de rede seja concluída; por exemplo, um método read() em um Socket sincrono pausará a execução até que os dados sejam recebidos. Isso pode simplificar a lógica de programação, mas pode levar à ineficiência, pois o thread que executa a operação fica inativo enquanto espera.

Por outro lado, um comportamento assíncrono, frequentemente implementado através de callbacks ou promises, permite que o programa continue a executar outras tarefas enquanto aguarda a conclusão da operação de rede. Isso aumenta a eficiência e a capacidade de resposta do aplicativo, pois os recursos de processamento podem ser utilizados para outras tarefas durante as operações de espera, mas isto torna a lógica do programa mais complexa, devido ao gerenciamento das operações não sequenciais.