Estácio	**UNIVERSIDADE ESTACIO DE SÁ** POLO INDAIATUBA – INDAIATUBA/SP DESENVOLVIMENTO FULL STACK - 22.3 Relatório da Missão Prática | Nível 5 | Mundo 3
Aluno:	SAULO HENRIQUE DOS SANTOS
Professor:	Robson Lorbieski
Repositório:	https://github.com/SauloHenriqueSantos/Mundo03Nivel05

Título da Prática: 1º Procedimento | Criando o Servidor e Cliente de Teste

Objetivos da Prática:

- Cria servidores Java com base em Sockets
- Criar clientes síncronos para servidores com base em Sockets
- Criar clientes assíncronos para servidores com base em Sockets
- Utilizar Threads para implementação de processos paralelos
- 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:

Todos os códigos estão no repositório abaixo:

https://github.com/SauloHenriqueSantos/Mundo03Nivel05

Análise e Conclusão PT1:

1. Como funcionam as classes Socket e ServerSocket?

Socket é usado para a comunicação cliente-servidor, permitindo a transferência de dados entre eles.

ServerSocket é usado pelo servidor para aguardar e aceitar conexões de clientes, criando um novo Socket para cada conexão estabelecida. Essas classes são fundamentais paraimplementar aplicativos que se comunicam através de uma rede.

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

As portas são essenciais na comunicação de rede, funcionando como pontos de extremidade em um sistema operacional para diferenciar o tráfego de rede destinado a diferentes serviços ou aplicações.

Elas ajudam a direcionar o tráfego de dados para aplicativos específicos em um servidor. Cada serviço ou aplicativo em um servidor é associado a uma porta única. Isso permite que os pacotes de dados sejam roteados corretamente, garantindo que a informação alcance o aplicativo correto, melhorando a segurança e a eficiência na transmissão de dados.

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 realizar a serialização e deserialização de objetos, respectivamente, permitindo que eles sejam transmitidos através de streams, como sockets ou arquivos. ObjectOutputStream converte um objeto em uma sequência de bytes (serialização) que pode ser enviada por um stream, enquanto ObjectInputStream reconstrói o objeto a partir dessa sequência de bytes (deserialização). Para que um objeto seja elegível para essa transmissão, ele deve ser serializável, o que é indicado pela implementação da interface Serializable. Isso é necessário porque apenas objetos serializáveis podem ser convertidos de maneira confiável em um formato que pode ser enviado e reconstruído em outro local ou momento, mantendo a integridade e o estado dos dados do objeto.

4. **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 pelo uso do padrão JPA (JavaPersistence API) e suas classes de entidades no lado do cliente. A JPA fornece uma camada deabstração que separa as operações no banco de dados do código de negócios.
   As classes de entidades representam objetos persistentes mapeados para tabelas do banco de dados, e as operações sobre esses objetos são realizadas por meio da JPA. Ao utilizar as classes de entidades JPA no cliente, o acesso direto ao banco de dados é evitado, e as operações são realizadas por meio de consultas e atualizações de objetos de entidade. Isso proporciona uma abstração eficaz, permitindo que o código de negócios trabalhe com objetos familiares em vez de lidar diretamente com SQL e detalhes de acesso ao banco de dados. Essa abstração contribui para o isolamento e modularidade do sistema.

Título da Prática: 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 são essenciais para o tratamento assíncrono de respostas em aplicações cliente-servidor. Quando um servidor recebe múltiplas solicitações de clientes, ele pode utilizar Threads para processar cada solicitação simultaneamente, evitando assim o bloqueio enquanto espera por uma operação de longa duração em uma única solicitação. Cada solicitação é processada em uma Thread separada, permitindo que o servidor continue recebendo e respondendo a outras solicitações. Essa abordagem melhora a eficiência e a capacidade de resposta do servidor, pois as Threads operam de forma independente e paralela, garantindo que o processamento de uma solicitação não seja interrompido ou atrasado pelas outras. Em resumo, o uso de Threads no servidor possibilita o tratamento assíncrono das respostas, melhorando o desempenho geral do sistema em ambientes de rede.

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

   O método invokeLater da classe SwingUtilities é usado na programação de interfaces gráficas em Java, especificamente no Swing framework. Ele é essencial para garantir que as alterações na interface do usuário sejam feitas de maneira segura no contexto da Event Dispatch Thread (EDT), que é a thread responsável por gerenciar eventos e atualizações de interface gráfica em Swing. Quando você quer executar um bloco de código que altera a interface do usuário, como atualizar um componente gráfico ou responder a eventos de usuário, invokeLater é utilizado para enfileirar esse bloco de código na EDT. Isso assegura que as mudanças na interface sejam realizadas de maneira sequencial e segura, evitando problemas de concorrência e inconsistências na interface gráfica.

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

   Em Java, os objetos são enviados e recebidos por meio do Socket utilizando as classes ObjectInputStream e ObjectOutputStream. A ObjectOutputStream é usada para serializar um objeto em um fluxo de saída, que pode ser enviado através do Socket. No lado receptor, o ObjectInputStream é utilizado para desserializar o objeto a partir do fluxo de entrada recebido pelo Socket. Isso permite a transmissão eficiente de objetos complexos entre diferentes aplicativos ou máquinas por meio de uma conexão de socket. É importante observar que os objetos que são transmitidos dessa maneira devem ser serializáveis para garantir que possam ser convertidos em bytes e reconstruídos corretamente do outro lado.

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

   No contexto de clientes com Socket Java, o comportamento síncrono implica que as operações de leitura e escrita bloqueiam a execução do programa até que a operação seja concluída, o que pode resultar em espera ociosa do processo. Já o comportamento assíncrono permite que o programa continue a execução enquanto aguarda a conclusão de operações de leitura ou escrita, evitando bloqueios. Isso é alcançado por meio de métodos assíncronos ou threads separadas para manipular eventos de entrada/saída, proporcionando maior eficiência e responsividade ao lidar com operações em segundo plano.