EAD - Polo Santa Luiza – Vitória – ES

DESENVOLVIMENTO FULL STACK

RELATÓRIO DE PRÁTICA
MUNDO 3 – NÍVEL 5 – MISSÃO PRÁTICA
RPG0018 - Por que não paralelizar

CARLOS ALTOMARE CATÃO

Matrícula: 202403460912

Semestre Letivo: 2025/1 - 3º Período

Data: 2025/06

🎯 Objetivos da Prática

A Missão Prática do Mundo 3 Nível 5 se compõe por 2 procedimentos que permitirão navegar por várias tecnologias, conforme detalhado nos itens:

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.

📌 Requisitos

- SQL Server, com o banco de dados gerado em prática anterior (loja).
- JDK e IDE NetBeans.
- Navegador para Internet, como o Chrome.
- Banco de dados SQL Server com o Management Studio.
- Computador com acesso à Internet.

▶️ Procedimento 1 - Criando o Servidor e Cliente de Teste

✅ Objetivos

O objetivo principal deste projeto foi desenvolver uma aplicação cliente/servidor em Java, com foco em autenticação de usuários e gerenciamento de produtos, utilizando comunicação via sockets e persistência de dados com JPA. O sistema foi pensado para ser modular, escalável e aplicável a contextos reais.

🛠️ Tecnologias Utilizadas

- **Linguagem:** Java (JDK 17+)
- **IDE:** NetBeans
- **Comunicação:** Sockets TCP/IP (Object Streams)
- **Persistência:** JPA (Java Persistence API)
- **Banco de dados:** Microsoft SQL Server. Base configurada via persistence.xml
- **Modelo de dados:** Entidades JPA. Usuario e Produto

🧱 Estrutura do Projeto

O projeto foi dividido em dois módulos principais:

1. **Módulo Servidor (cadastroserver):**
   - CadastroServer: ponto de entrada principal, escuta conexões.
   - CadastroThread: thread dedicada a cada cliente conectado.
   - UsuarioJpaController e ProdutoJpaController: classes para acesso ao banco de dados via JPA.
2. **Módulo Cliente (cadastroclient):**
   - CadastroClient: aplicação que conecta ao servidor, envia comandos, recebe e exibe dados.
   - Utiliza ObjectInputStream e ObjectOutputStream para comunicação binária.

⚙️ Funcionalidades

- Autenticação de usuários via login e senha.
- Validação de credenciais contra o banco de dados.
- Retorno de mensagens de autenticação ao cliente.
- Comando L (listar): retorna todos os produtos cadastrados.
- Comando S: encerra a sessão do usuário.
- Comunicação segura entre cliente e servidor via objetos serializados.

📝 Atividades Realizadas

- Modelagem das entidades Usuario e Produto.
- Implementação dos controladores JPA para acesso ao banco.
- Criação da estrutura de sockets para comunicação bidirecional.
- Controle de múltiplas conexões simultâneas no servidor com threads.
- Tratamento de exceções e encerramento seguro de conexões.

🚧 Desafios Enfrentados

- Gerenciamento correto de fluxos de entrada e saída com ObjectInputStream e ObjectOutputStream.
- Evitar exceções como EOFException, causadas por falhas de sincronização entre cliente e servidor.
- Manutenção de conexões persistentes sem travar o servidor.
- Garantia de integridade dos dados durante a serialização/deserialização.

📄 Códigos

Os códigos foram desenvolvidos com a IDE NetBeans e se encontram em repositório no GitHub onde podem ser acessados pelos links:

Servidor:

https://github.com/CarlosCatao/Mundo_3_Nivel_5_Missao_Pratica/tree/main/Procedimento-1/CadastroServer

Cliente:

https://github.com/CarlosCatao/Mundo_3_Nivel_5_Missao_Pratica/tree/main/Procedimento-1/CadastroClient

🧪 Testes

- Testes manuais com múltiplos usuários e comandos.
- Simulação de falhas de autenticação.
- Verificação de resposta do servidor em tempo real (mensagens e lista de produtos).
- Teste de robustez com entrada de comandos inválidos.

📜 Resultados

Os resultados da execução dos códigos se encontram ilustrados no arquivo *Resultados.pdf* que se encontra em repositório no GitHub onde podem ser acessados pelo link: https://github.com/CarlosCatao/Mundo_3_Nivel_5_Missao_Pratica/blob/main/Procedimento-1/RESULTADOS%20PROCEDIMENTO%201.pdf

📌 Conclusão

O projeto alcançou todos os seus objetivos: foi possível implementar uma comunicação cliente-servidor robusta com autenticação, listagem de dados e controle de sessões. A integração de Java com JPA e sockets mostrou-se eficiente e adequada ao contexto proposto. O aprendizado com os desafios enfrentados, especialmente relacionados à comunicação de rede, reforçou conceitos fundamentais de programação distribuída e persistência de dados.

❓ Questionamentos

▶️ Como funcionam as classes Socket e ServerSocket?

As classes *Socket* e *ServerSocket* fazem parte da API de redes (networking) e são usadas para implementar comunicação entre computadores (ou processos)

A Classe *ServerSocket* cria um **servidor** TCP que escuta conexões de clientes e a Classe *Socket* cria um **cliente** TCP ou lida com a conexão de um cliente no servidor.

Estas Classes funcionam segundo o seguinte fluxo:

1. O **servidor** cria um ServerSocket e fica esperando conexões.
2. O **cliente** cria um Socket e se conecta ao IP/porta do servidor.
3. O ServerSocket aceita essa conexão e retorna um Socket para se comunicar com o cliente.
4. Ambos os lados trocam dados usando *InputStream* e *OutputStream*.

▶️ Qual a importância das portas para a conexão com servidores?

As **portas** são importantes para a comunicação entre computadores na rede, pois elas funcionam como **"canais"** que permitem que múltiplas aplicações ou serviços se comuniquem simultaneamente em um mesmo dispositivo.

Quando o cliente se conecta ao IP do servidor, ele também precisa saber , senão, não saberá qual serviço usar.

▶️ 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 utilizadas para ler e escrever objetos inteiros (com estado e atributos) em fluxos de entrada/saída entre arquivos, redes ou memória, por exemplo. Elas fazem parte do mecanismo de **serialização de objetos** em Java.

Objetos em Java precisam ser **"serializáveis"** (ou seja, implementar a interface *Serializable*) para que possam ser convertidos em uma sequência de bytes em um processo denominado de **serialização**.

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

Porque as classes de entidade JPA no cliente são apenas representações de dados, não têm capacidade de acessar o banco diretamente. O isolamento é garantido porque a lógica de acesso ao banco (via *EntityManager*, transações, queries etc.) permanece no servidor.

▶️ Procedimento 2 - Servidor Completo e Cliente Assíncrono

✅ Objetivos

O projeto tem como objetivo desenvolver uma aplicação cliente-servidor para cadastro e movimentação de produtos em estoque. Os usuários autenticados podem listar produtos, registrar movimentações de entradas e saídas de produtos, bem como consultar informações diretamente de um banco de dados relacional, utilizando comunicação em rede baseada em sockets Java.

🛠️ Tecnologias Utilizadas

- **Linguagem:** Java (JDK 17+)
- **IDE:** NetBeans
- **Comunicação:** Sockets TCP/IP (Object Streams)
- **Persistência:** JPA (Java Persistence API)
- **Banco de dados:** Microsoft SQL Server. Base configurada via persistence.xml
- **Modelo de dados:** Entidades JPA. Usuario e Produto

📦 Estrutura do Projeto

- **Projeto Servidor (cadastroserver)**
  - Camada de controle (controllers): ProdutoJpaController, UsuarioJpaController, PessoaJpaController, OperacaoJpaController
  - Modelo de dados (entidades JPA): Produto, Pessoa, Usuario, Operacao
  - Servidor principal: CadastroServer
  - Thread de atendimento por cliente: CadastroThreadV2
- **Projeto Cliente (cadastroclient)**
  - Aplicação de console: CadastroClient e CadastroClientV2 (versão assíncrona)
  - Comunicação via Socket com servidor
  - Envio de comandos e leitura de respostas

⚙️ Funcionalidades

- Login de usuários com verificação via banco de dados.
- Listagem de todos os produtos com informações de estoque e preço.
- Registro de entrada (compra) e saída (venda) de produtos, com atualização de estoque.
- Armazenamento de operações no banco, com rastreamento de comprador, vendedor, data, quantidade e valor unitário.
- Interface gráfica para exibir mensagens e operações recebidas do servidor.
- Armazenamento de movimentações com data, quantidade e valor unitário.
- Controle de estoque com atualização automática.
- Atendimento concorrente de múltiplos clientes via Thread.

📝 Atividades Realizadas

- Criação das entidades JPA com anotações e relacionamentos
- Implementação dos controladores (JpaControllers) para cada entidade
- Desenvolvimento do servidor multithreaded (CadastroServer e CadastroThreadV2)
- Implementação do cliente console com menu de comandos
- Testes de conexão, persistência e atualização de dados
- Tratamento de exceções e mensagens de erro

🚧 Desafios Enfrentados

- Tratamento de exceções silenciosas e erros de sincronização nos fluxos de rede.
- Compatibilização entre os tipos de dados enviados pelo cliente e esperados pelo servidor.
- Configuração e testes da unidade de persistência com JPA.
- Gerenciamento correto de ObjectInputStream e ObjectOutputStream.
- Controle de concorrência e sincronização das threads.
- Validação de comandos e consistência de dados no banco.

- Conversão e manipulação de tipos como BigDecimal e Date.
- Criação correta de associações entre entidades como Operacao, Produto e Pessoa.

📄 Códigos

Os códigos foram desenvolvidos com a IDE NetBeans e se encontram em repositório no GitHub onde podem ser acessados pelos links:

Servidor:

https://github.com/CarlosCatao/Mundo_3_Nivel_5_Missao_Pratica/tree/main/Procedimento-2/CadastroServer

Cliente:

https://github.com/CarlosCatao/Mundo_3_Nivel_5_Missao_Pratica/tree/main/Procedimento-2/CadastroClientV2

🧪 Testes

- Testes manuais de todas as operações (L, E, S, X) com dados reais.
- Simulação de múltiplos clientes se conectando simultaneamente ao servidor.
- Verificação de mensagens exibidas na interface gráfica para feedback em tempo real.
- Teste de persistência e consistência de estoque após entrada/saída.
- Simulação de entradas inválidas (ID inexistente, valor negativo, string ao invés de número) para validação de robustez.
- Verificação de atualização do estoque após entradas e saídas.

📜 Resultados

Os resultados da execução dos códigos se encontram ilustrados no arquivo *Resultados.pdf* que se encontra em repositório no GitHub onde podem ser acessados pelo link: https://github.com/CarlosCatao/Mundo_3_Nivel_5_Missao_Pratica/blob/main/Procedimento-2/RESULTADOS%20PROCEDIMENTO-2.pdf

📌 Conclusão

O sistema desenvolvido atinge com sucesso o objetivo de registrar movimentações comerciais em uma estrutura cliente-servidor robusta, segura e modular. A aplicação oferece tanto uma interação via console quanto uma interface gráfica útil para mensagens, integrando os conceitos de rede, banco de dados relacional, programação orientada a objetos e interface com o usuário. Os desafios encontrados ao longo do desenvolvimento foram valiosos para consolidar conhecimentos técnicos e fortalecer práticas de depuração, tratamento de exceções e modelagem de dados.

❓ Questionamentos

▶️ 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 com o objetivo de evitar que a aplicação cliente fique bloqueada enquanto aguarda por uma resposta; isto é especialmente importante em aplicações que precisam continuar responsivas ou lidar com múltiplas tarefas simultaneamente.

▶️ Para que serve o método invokeLater, da classe SwingUtilities?

O método *invokeLater()* da classe *SwingUtilities* é utilizado para agendar a execução de um trecho de código na ***Event Dispatch Thread*** **(EDT)**, a *thread* responsável por atualizar a interface gráfica (GUI) em aplicações que usam **Swing**.

▶ Como os objetos são enviados e recebidos pelo Socket Java?

No Java, os objetos podem ser enviados e recebidos através de *Sockets* usando os fluxos de entrada e saída com suporte à *serialização de objetos*, por meio das classes:

- *ObjectOutputStream* (para enviar objetos),
- *ObjectInputStream* (para receber objetos).

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

A comparação entre comportamento assíncrono e síncrono em clientes que utilizam *Socket* em Java envolve principalmente a forma como o processamento lida com bloqueios durante comunicação de rede.

Característica	Comportamento Síncrono	Comportamento Assíncrono (com Threads ou NIO)
Execução do código	Linha por linha, bloqueia até receber resposta	Continua executando, não bloqueia enquanto espera
Bloqueio (blocking I/O)	Sim, métodos como read() ou readObject() bloqueiam	Não, a comunicação é tratada em segundo plano
Responsividade da aplicação	Baixa — se a rede for lenta, tudo "trava"	Alta — a interface ou lógica principal continua fluindo
Uso de threads	Opcional (mas recomendável em GUIs)	Necessário (Threads, ExecutorService ou NIO)
Complexidade do código	Simples	Mais complexo (gerenciamento de threads ou callbacks)
Controle de concorrência	Baixo	Requer controle (sincronização, locks, filas, etc.)
Escalabilidade	Limitada — cada cliente precisa de uma thread dedicada	Melhor — com NIO ou thread pool, escala bem
Exemplo típico de uso	Programas de linha de comando simples	Chats, servidores multiplayer, GUIs, apps interativas

📌 Conclusão Final da Prática

O projeto atingiu com êxito todos os objetivos propostos, resultando em um sistema cliente-servidor robusto, seguro e modular, capaz de registrar movimentações comerciais com autenticação, controle de sessões e listagem de dados. A integração entre Java, JPA, sockets e banco de dados relacional mostrou-se eficiente, proporcionando uma aplicação funcional tanto por meio de interface gráfica quanto via console.

O desenvolvimento consolidou importantes conceitos de programação orientada a objetos, persistência de dados e comunicação em rede. Os desafios enfrentados durante o processo contribuíram significativamente para o aprofundamento do conhecimento técnico, aprimorando práticas de modelagem, tratamento de exceções e depuração.