Universidade Estácio de Sá - Campus Ilha do Governador

202304625751 | DESENVOLVIMENTO FULL STACK

RPG0016 –BACKEND SEM BANCO NÃO TEM

GUSTAVO CALIL

Disciplina RPG0016 -
“Backend sem banco não tem” do curso de
Tecnologia em Desenvolvimento Full Stack da
Universidade Estácio de Sá.
Tutor: Simone Gama

https://github.com/gustavocalil-github/P3_Mission3

1 INTRODUÇÃO

No âmbito do desenvolvimento de aplicativos Java com integração a bancos de dados SQL Server, a habilidade em utilizar o middleware JDBC (Java Database Connectivity) e implementar o padrão DAO (Data Access Object) desempenha um papel fundamental. Este relatório documenta a experiência e os resultados alcançados ao longo do desenvolvimento de um sistema cadastral, focado na persistência de dados em um ambiente de banco de dados relacional. Ao explorar os desafios e conquistas encontrados durante esta missão, busca-se fornecer uma visão abrangente das práticas envolvidas na construção de sistemas Java conectados a bancos de dados SQL Server.

1.1 Objetivos da prática

O objetivo central desta missão prática é capacitar os profissionais a desenvolverem sistemas cadastrais robustos, com uma base sólida de persistência de dados em um ambiente de banco de dados relacional. O foco principal reside em proporcionar aos participantes uma compreensão prática e aprofundada do uso do middleware JDBC e da aplicação eficiente do padrão DAO em aplicativos Java.

Na sequência, são apresentados os objetivos específicos que orientaram a prática:

a) Implementar a persistência de dados utilizando o middleware JDBC para estabelecer conexão com um banco de dados SQL Server.

b) Empregar o padrão DAO de maneira eficiente, separando a lógica de acesso aos dados da lógica de negócios em um aplicativo Java.

c) Desenvolver um sistema cadastral completo, incluindo funcionalidades de inclusão, alteração, exclusão e consulta de registros em um ambiente de banco de dados relacional.

d) Compreender e aplicar conceitos de mapeamento objeto-relacional para facilitar a interação entre objetos Java e as tabelas do banco de dados.

e) Refletir sobre as melhores práticas de desenvolvimento de software em Java, abordando aspectos como organização do código, tratamento de exceções e boas práticas de programação.

2 METODOLOGIA

A metodologia adotada para a realização desta missão prática envolveu o uso da IDE IntelliJ IDEA como ambiente de desenvolvimento principal. A escolha dessa ferramenta se deve à sua ampla gama de recursos e facilidade de uso, que proporcionaram um ambiente produtivo e eficiente para a implementação do projeto. Inicialmente, foi realizada a configuração do ambiente de desenvolvimento, incluindo a instalação do JDK (Java Development Kit) e a configuração do IntelliJ IDEA para suportar projetos Java. Utilizando o IntelliJ IDEA, foi criado um novo projeto Java para o sistema cadastral, seguindo as melhores práticas de organização de código e estrutura de projeto. O próximo passo envolveu a configuração do banco de dados SQL Server, utilizando o SQL Server Management Studio para criar o banco de dados e as tabelas necessárias para armazenar os dados do sistema cadastral. Foi implementada a conexão com o banco de dados SQL Server utilizando o middleware JDBC. Foram criadas classes utilitárias para gerenciar a conexão com o banco de dados e executar consultas SQL. Em seguida, foram criadas classes DAO (Data Access Object) para cada entidade do sistema cadastral. Essas classes foram responsáveis por encapsular a lógica de acesso aos dados, seguindo o padrão DAO e garantindo a separação de responsabilidades. Com a infraestrutura básica configurada, foram implementadas as funcionalidades do sistema cadastral, incluindo a inclusão, alteração, exclusão e consulta de registros. Após a implementação das funcionalidades, foram realizados testes unitários e de integração para validar o funcionamento correto do sistema. Foram simuladas situações de uso para garantir a robustez e confiabilidade do sistema. Por fim, todo o processo de desenvolvimento foi documentado, incluindo a descrição das etapas realizadas, o código-fonte do projeto, os resultados dos testes e uma análise das principais decisões e desafios enfrentados durante o desenvolvimento. Ao adotar essa metodologia, foi possível realizar a implementação do sistema cadastral de forma

estruturada e eficiente, explorando as capacidades do IntelliJ IDEA e seguindo as melhores práticas de desenvolvimento de software em Java.

3 RESULTADOS

O sistema passa por quatro etapas principais ao lidar com os dados: criação, leitura, atualização e exclusão, conhecidas como CRUD. Durante a etapa de criação, novos registros são adicionados ao banco de dados por meio de classes específicas, como PessoaFisicaDAO e PessoaJuridicaDAO. Os dados inseridos incluem informações como nome, endereço, telefone, CPF/CNPJ, entre outros

Na etapa de leitura, o sistema recupera e exibe os registros existentes com base em critérios específicos, como o ID da pessoa ou outros atributos de pesquisa. Isso é realizado por meio dos métodos de leitura das classes PessoaFisicaDAO e PessoaJuridicaDAO, apresentando os dados ao usuário para visualização.

Durante a etapa de atualização, os registros no banco de dados são modificados com novas informações utilizando os métodos de atualização das classes mencionadas anteriormente. É possível realizar a alteração de dados como nome, endereço, telefone, CPF/CNPJ, entre outros.

Na etapa de exclusão, registros são permanentemente removidos do banco de dados por meio dos métodos de exclusão das classes relevantes. O usuário seleciona o registro a ser removido, com base em critérios como o ID da pessoa.

Além disso, também testada a opção de finalização do programa, obtendo os resultados esperados de interrupção do *loop* e encerramento da execução do programa.

4 DISCUSSÕES

Nesta seção de discussão, analisaremos aspectos cruciais relacionados à implementação do sistema cadastral em Java, integrado ao banco de dados SQL Server, à luz de conceitos essenciais como o uso do JDBC e do padrão DAO. Além disso, exploraremos como a herança é tratada em um ambiente estritamente relacional de banco de dados. Para isso, abordaremos uma série de perguntas fundamentais que surgiram durante a prática e as relacionaremos diretamente com as experiências vivenciadas e os resultados obtidos. Examinaremos como a persistência em diferentes meios, seja em arquivos ou em banco de dados, impacta o desenvolvimento e a manutenção do sistema. Também discutiremos como o uso do operador lambda tem simplificado certas operações em Java, destacando sua aplicabilidade em nosso projeto. Por fim, compreenderemos por que métodos acionados diretamente pelo método *main* precisam ser marcados como *static*, considerando o contexto da nossa implementação. Ao fim dessa análise, visamos oferecer uma compreensão mais profunda e prática desses conceitos, enriquecendo nossa experiência no desenvolvimento de aplicativos Java conectados a bancos de dados relacionais.

4.1 Qual a importância dos componentes de *middleware*, como o JDBC?

Os componentes de *middleware*, como o JDBC (Java Database Connectivity), desempenham um papel fundamental na integração de aplicativos Java com bancos de dados relacionais. O JDBC fornece uma interface padronizada para acessar e manipular dados no banco de dados, independentemente do sistema de gerenciamento de banco de dados subjacente. Isso permite que os desenvolvedores escrevam código Java

portável que possa ser executado em diferentes ambientes de banco de dados, sem a necessidade de modificar a lógica de acesso aos dados. Além disso, o JDBC oferece recursos avançados, como suporte a transações, consultas parametrizadas e manipulação eficiente de conjuntos de resultados, contribuindo para o desenvolvimento de aplicativos robustos e escaláveis.

4.2 Qual a diferença no uso de *Statement* ou *PreparedStatement* para a manipulação de dados?

A diferença principal entre *Statement* e *PreparedStatement* reside na forma como os comandos SQL são tratados e executados. Um objeto *Statement* é usado para executar comandos SQL estáticos, onde o texto do comando é definido diretamente no código Java. Isso pode levar a vulnerabilidades de segurança, como ataques de injeção de SQL, se os dados de entrada não forem adequadamente validados e sanitizados. Por outro lado, um objeto *PreparedStatement* é usado para executar comandos SQL parametrizados, onde os parâmetros do comando são definidos como espaços reservados e preenchidos dinamicamente com valores durante a execução. Isso torna o *PreparedStatement* mais seguro e eficiente, pois os parâmetros são tratados separadamente do comando SQL, evitando ataques de injeção de SQL e permitindo o reuso eficiente de consultas preparadas.

4.3 Como o padrão DAO melhora a manutenibilidade do software?

O padrão DAO (*Data Access Object*) melhora a manutenibilidade do software ao promover a separação de responsabilidades entre a lógica de negócios e a lógica de acesso a dados. Ele encapsula a lógica de acesso ao banco de dados em classes específicas, reduzindo o acoplamento entre os componentes do sistema e facilitando a modificação e extensão do código. Com o padrão DAO, é possível alterar a fonte de dados subjacente (por exemplo, de um banco de dados relacional para um serviço *web*) sem afetar outras partes do sistema. Além disso, o padrão DAO facilita a realização de testes unitários e a aplicação de práticas de desenvolvimento ágil, como a refatoração de código e a melhoria contínua.

4.4 Como a herança é refletida no banco de dados, quando lidamos com um modelo estritamente relacional?

Quando lidamos com um modelo estritamente relacional, a herança pode ser refletida no banco de dados de várias maneiras, dependendo da abordagem escolhida para modelar a hierarquia de classes no sistema. Uma abordagem comum é usar uma tabela para cada classe concreta na hierarquia de herança, onde cada tabela contém os atributos específicos da classe, além dos atributos herdados das classes pai. Isso é conhecido como mapeamento de tabela por classe (ou tabela por subclasse). Outra abordagem é usar uma única tabela para todas as classes na hierarquia de herança, onde uma coluna é usada para identificar o tipo de objeto representado por cada linha. Isso é conhecido como mapeamento de tabela por hierarquia (ou tabela única para hierarquia). Ambas as abordagens têm vantagens e desvantagens, e a escolha entre elas depende dos requisitos específicos do sistema e das preferências do desenvolvedor.

4.5 Quais as diferenças entre a persistência em arquivo e a persistência em banco de dados?

A persistência em arquivo e em banco de dados diferem principalmente em termos de armazenamento e recuperação de dados. Na persistência em arquivo, os dados são

armazenados em arquivos no sistema de arquivos do computador. Isso pode ser feito de várias maneiras, como arquivos de texto simples, arquivos binários ou formatos mais complexos, como JSON ou XML. Por outro lado, na persistência em banco de dados, os dados são armazenados em um sistema de gerenciamento de banco de dados (SGBD), que oferece recursos avançados para armazenamento, recuperação, manipulação e consulta de dados. Os bancos de dados relacionais, como o SQL Server mencionado neste contexto, organizam os dados em tabelas com relações definidas entre elas, permitindo consultas complexas e transações seguras.

4.6 Como o uso de operador lambda simplificou a impressão dos valores contidos nas entidades, nas versões mais recentes do Java?

O uso de operadores lambda em Java simplificou significativamente a escrita de código ao lidar com coleções de dados, como listas ou mapas. Os operadores lambda permitem que funções sejam tratadas como objetos, facilitando a passagem de comportamentos como parâmetros de métodos e a execução de operações em coleções de forma mais concisa e legível. Por exemplo, ao imprimir os valores contidos em uma lista de entidades, antes do uso de operadores lambda, era necessário iterar explicitamente sobre a lista e imprimir cada valor individualmente. Com o uso de operadores lambda e a API de fluxo introduzida no Java 8, isso pode ser feito de forma mais elegante e eficiente em termos de código, reduzindo a quantidade de código *boilerplate* necessário.

4.7 Por que métodos acionados diretamente pelo método *main*, sem o uso de um objeto, precisam ser marcados como *static*?

No Java, o método *main* é o ponto de entrada de um programa Java e é chamado pelo sistema Java para iniciar a execução do programa. Para que o método *main* possa ser chamado sem a necessidade de criar uma instância da classe que o contém, ele deve ser declarado como *static*. Isso significa que o método *main* pertence à classe em si, em vez de pertencer a instâncias específicas da classe. Ao marcar o método *main* como *static*, ele pode ser chamado diretamente pela JVM (*Java Virtual Machine*) sem a necessidade de criar um objeto da classe, o que é essencial para iniciar a execução do programa Java.

5 CONSIDERAÇÕES FINAIS

A conclusão desta missão prática proporcionou uma valiosa experiência no desenvolvimento de aplicativos Java integrados a bancos de dados SQL Server, destacando a importância do *middleware* JDBC e do padrão DAO no processo. Ao longo deste relatório, foram documentados detalhes sobre a metodologia adotada, os resultados obtidos e as discussões realizadas durante o desenvolvimento do sistema cadastral. A metodologia empregada, que incluiu a utilização da IDE IntelliJ IDEA e a aplicação das melhores práticas de desenvolvimento Java, mostrou-se eficaz na implementação estruturada e eficiente do projeto. Desde a configuração do ambiente de desenvolvimento até a realização dos testes e a documentação final, cada etapa foi cuidadosamente planejada e executada visando garantir a qualidade e a robustez do sistema. Os resultados alcançados foram significativos, com a implementação bem-sucedida das funcionalidades de criação, leitura, atualização e exclusão de registros. As imagens apresentadas ilustraram claramente o funcionamento do sistema e a interação com o banco de dados, evidenciando a eficácia das soluções desenvolvidas. Na seção de discussões, foram explorados aspectos cruciais relacionados ao uso do JDBC, do padrão DAO e à reflexão sobre a herança em um

ambiente estritamente relacional. Perguntas fundamentais foram respondidas objetivamente, proporcionando uma análise aprofundada dos conceitos abordados. Em resumo, esta missão prática contribuiu significativamente para aprimorar as habilidades técnicas em desenvolvimento de software. As lições aprendidas e as experiências adquiridas neste projeto certamente nos preparam de forma mais abrangente para os desafios futuros no campo do desenvolvimento de aplicativos Java.