Campus polo Rua Teresa - Desenvolvimento Full Stack

RPG0016 - BackEnd sem banco não tem
Turma 9001 - 3o semestre
Aluno: Marcos Valerio R S de Carvalho
Github: https://github.com/Voidrrrr/Mundo-3-nivel-3

RPG 0016 - Vamos manter as informações?

Objetivos da prática:

- **Implementar persistência com base no middleware JDBC.**
- **Utilizar o padrão DAO (Data Access Object) no manuseio de dados.**
- **Implementar o mapeamento objeto-relacional em sistemas Java.**
- **Criar sistemas cadastrais com persistência em banco relacional.**
- **No final do exercício, o aluno terá criado um aplicativo cadastral com uso do SQL Server na persistência de dados.**

Análise e Conclusão:

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

  O middleware JDBC é essencial para conectar aplicações Java a bancos de dados de forma independente e eficiente. Ele oferece uma camada de abstração que facilita o acesso a diferentes bancos, simplificando o código e permitindo que a aplicação seja portável entre sistemas. Além disso, o JDBC melhora o desempenho e a escalabilidade, gerenciando conexões de forma eficiente e reduzindo o custo de comunicação com o banco de dados.

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

O `Statement e o PreparedStatement` são ambos usados para executar comandos SQL no JDBC, mas possuem diferenças importantes:

- `Statement`: Executa comandos SQL diretamente, o que o torna mais suscetível a ataques de *SQL Injection*. É mais adequado para consultas simples sem parâmetros variáveis.
- `PreparedStatement`: É mais seguro, pois permite a parametrização das consultas, protegendo contra *SQL Injection*. Além disso, ele é pré-compilado pelo banco, o que pode melhorar o desempenho, especialmente para consultas repetidas.

- **Como o padrão DAO melhora a manutenibilidade do software?**

  O padrão DAO (Data Access Object) melhora a manutenibilidade do software ao separar a lógica de acesso a dados da lógica de negócio. Com essa

separação, mudanças no banco de dados (como na estrutura ou nos comandos SQL) afetam apenas a camada DAO, sem impactar outras partes do sistema. Isso facilita atualizações, testes e reutilização do código, promovendo um design mais modular e organizado.

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

Quando refletimos herança em um banco de dados relacional, temos três abordagens principais:

Tabela Única por Hierarquia: Armazena toda a hierarquia de classes em uma única tabela, usando uma coluna para identificar o tipo. Simples, mas pode gerar muitas colunas nulas para subclasses que não usam certos campos.

Uma Tabela por Subclasse: Cria uma tabela para cada subclasse com suas colunas específicas e uma chave estrangeira para a tabela da superclasse. Reduz colunas nulas, mas aumenta a complexidade das consultas com *joins*.

Tabela por Classe Concreta: Cada subclasse tem sua própria tabela com todas as colunas necessárias (incluindo as da superclasse). Evita *joins*, mas pode duplicar dados e dificultar a manutenção.

Cada abordagem tem vantagens e desvantagens, e a escolha depende de requisitos específicos de desempenho, simplicidade e manutenção.

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

Persistência em Arquivo

- Estrutura Simples: Dados armazenados em arquivos (texto, CSV, JSON, etc.) com uma estrutura mais simples.
- Facilidade de Uso: Simples de implementar e usar para dados pequenos ou não estruturados.
- Desempenho: Pode ser mais rápido para leituras e gravações de pequenas quantidades de dados, mas não escala bem.
- Concorrência: Difícil de gerenciar acesso simultâneo por múltiplos usuários, aumentando o risco de corrupção de dados.
- Recursos Limitados: Não oferece funcionalidades avançadas como consultas complexas, integridade referencial ou transações.

Persistência em Banco de Dados

- Estrutura Complexa: Utiliza tabelas e relacionamentos, permitindo uma estrutura de dados mais complexa e organizada.
- Escalabilidade: Projetado para gerenciar grandes volumes de dados e múltiplos usuários simultaneamente.
- Consultas Avançadas: Suporta linguagens de consulta (como SQL) para operações complexas e análises de dados.
- Integridade dos Dados: Oferece mecanismos para garantir integridade referencial, transações e segurança.
- Gerenciamento de Conexões: Facilita o controle de acesso e a concorrência entre múltiplos usuários.

- **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 nas versões mais recentes do Java simplificou a impressão de valores em entidades de várias maneiras:

Sintaxe Concisa: Expressões lambda permitem implementar interfaces funcionais de forma mais sucinta, eliminando a necessidade de classes anônimas.

Integração com Streams: A API de Streams facilita operações funcionais em coleções, permitindo transformar, filtrar e imprimir dados de maneira fluida.

Melhor Legibilidade: O código se torna mais legível e expressivo, facilitando a compreensão sem a verbosidade típica.

Funcionalidade Adicional: Operações como `map`, `filter` e `reduce` podem ser utilizadas para manipular dados antes da impressão.

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

Métodos acionados diretamente pelo método `main` em Java precisam ser marcados como `static` porque:

Contexto Estático: O método `main` é estático e é chamado pela JVM sem criar uma instância da classe. Portanto, não há um objeto associado.

Acesso a Membros Estáticos: Apenas membros `static` podem ser acessados diretamente pelo `main`. Métodos não estáticos requerem uma instância da classe, o que não é possível no contexto do `main`.

Em resumo, métodos chamados diretamente no `main` devem ser `static` para que possam ser acessados sem a necessidade de criar um objeto da classe.