Programação Orientada a Objetos

Prof.ª Ma. Jessica Oliveira

Católica
Universidade Católica de Brasília

Aula 11 – 26/05/2025

Integração com Banco de Dados (Parte I).

Católica
Universidade Católica de Brasília

Persistência de dados na POO.

Conceito.

- A persistência de dados pode ser definida como a capacidade de um sistema armazenar informações de maneira que estas permaneçam disponíveis e acessíveis mesmo após o término da execução do programa (SILBERSCHATZ; KORTH; SUDARSHAN, 2011).
- Ou seja, diferentemente dos dados armazenados temporariamente na memória RAM, que são voláteis e se perdem ao encerrar o programa, os dados persistentes são gravados em mídias não voláteis, como bancos de dados, arquivos ou sistemas de armazenamento em nuvem.

Conceito.

- De acordo com Elmasri e Navathe (2011), a persistência é um requisito fundamental para qualquer aplicação corporativa, pois permite a manutenção de informações críticas, como dados de clientes, produtos, pedidos, transações financeiras, registros acadêmicos, entre outros.

Importância.

- Em sistemas desenvolvidos segundo o paradigma da POO, os dados estão encapsulados em objetos, que representam entidades do mundo real.
- No entanto, durante a execução do programa, esses objetos residem na memória principal (RAM) e deixam de existir quando o programa é encerrado.
- Portanto, para garantir que as informações associadas aos objetos sejam preservadas de forma durável, é necessário implementar mecanismos de persistência.

Importância.

- Conforme Horstmann e Cornell (2019), a persistência permite que os objetos “sobrevivam” além do ciclo de vida da aplicação, possibilitando que seus atributos sejam armazenados e posteriormente recuperados de uma fonte permanente, como um banco de dados relacional.
- Essa característica é fundamental para o desenvolvimento de aplicações empresariais, websites dinâmicos, sistemas de gestão, plataformas educacionais, entre outros, nos quais a manutenção dos dados é uma necessidade operacional e estratégica.

Abordagens.

- **Persistência em arquivos:** simples, porém limitada. Utiliza leitura e escrita de arquivos texto, binários ou XML/JSON.
- **Persistência em bancos de dados relacionais:** a mais comum em sistemas corporativos, utilizando SQL para operações de armazenamento, recuperação e manipulação de dados.
- **Mapeamento Objeto-Relacional (ORM):** ferramentas como Hibernate e JPA facilitam o mapeamento entre objetos Java e tabelas de bancos relacionais, abstraindo parte da complexidade da persistência.

Modelo Relacional e Persistência.

Católica
Universidade Católica de Brasília

Visão geral do modelo relacional.

- O modelo relacional foi proposto por Edgar F. Codd em 1970, e organiza os dados em tabelas bidimensionais chamadas de relações (SILBERSCHATZ; KORTH; SUDARSHAN, 2011).
- Cada tabela possui linhas (tuplas) e colunas (atributos), representando registros e suas características, respectivamente.
- As principais características do modelo relacional são:
  - Dados estruturados em tabelas.
  - Relacionamentos entre tabelas estabelecidos por chaves primárias e estrangeiras.
  - Independência física e lógica dos dados.
  - Uso da linguagem SQL para definição, manipulação e consulta dos dados.

Desafio do Mapeamento Objeto-Relacional.

- A integração entre o paradigma orientado a objetos e o modelo relacional não é trivial.
- Esse desafio é conhecido como Impedância de Mapeamento Objeto-Relacional (ORM *Impedance Mismatch*) (FOWLER, 2003).Isso ocorre porque:
  - Na POO, os dados são representados por objetos com atributos e comportamentos (métodos), enquanto no modelo relacional, os dados são representados em tabelas, que não possuem comportamento.
  - As estruturas de herança, composição e polimorfismo da POO não possuem equivalentes diretos no modelo relacional.
  - O ciclo de vida de objetos não corresponde exatamente ao ciclo de vida dos registros no banco.
- Portanto, o uso de JDBC ou frameworks ORM é essencial para estabelecer essa ponte entre objetos e dados relacionais.

Java Database Connectivity (JDBC).

Católica
Universidade Católica de Brasília

Definição e propósito.

- O JDBC é uma API que permite que aplicações Java se conectem e interajam com bancos de dados relacionais (HORSTMANN; CORNELL, 2019).
- Ele define um conjunto de classes e interfaces que facilitam a execução de comandos SQL diretamente de aplicações Java.
- O JDBC faz parte da especificação da plataforma Java desde a versão JDK 1.1 e continua sendo amplamente utilizado, tanto de forma direta como base para frameworks ORM mais avançados.

Arquitetura.

- **Driver JDBC:** é a implementação específica que permite a comunicação entre a aplicação Java e o banco de dados. Cada sistema gerenciador de banco de dados (SGBD) possui seu próprio driver.
- **Connection:** representa uma conexão ativa com o banco. Através dela, é possível enviar comandos SQL e obter resultados.
- **Statement:** permite executar comandos SQL estáticos (sem parâmetros).
- **PreparedStatement:** permite executar comandos SQL parametrizados, oferecendo mais segurança e desempenho.
- **ResultSet:** armazena os resultados retornados de uma consulta SQL.

Funcionamento básico.

- **Carregamento do Driver:** o driver JDBC é carregado na aplicação para possibilitar a comunicação com o SGBD.
- **Estabelecimento da Conexão:** por meio da classe **`DriverManager`** e do método **`getConnection()`**, a aplicação abre uma conexão com o banco de dados.
- **Execução de Comandos SQL:** utilizando objetos do tipo **`Statement`** ou **`PreparedStatement`**, a aplicação envia comandos SQL (SELECT, INSERT, UPDATE, DELETE).

Funcionamento básico.

- **Processamento dos Resultados:** as consultas SQL retornam um objeto **`ResultSet`**, que permite percorrer os registros e acessar os dados.
- **Encerramento dos Recursos:** ao final da operação, é necessário fechar o **`ResultSet`**, o **`Statement`** e a **`Connection`** para liberar os recursos do sistema e do banco.

Boas práticas no uso de JDBC.

- Sempre fechar **`ResultSet`**, **`Statement`** e **`Connection`** no bloco **`finally`** ou utilizar **`try-with-resources`** (a partir do Java 7).
- Usar **`PreparedStatement`** para comandos que recebem parâmetros, prevenindo ataques de SQL Injection.
- Manter o gerenciamento de conexões eficiente, utilizando pools de conexões em aplicações maiores (Apache DBCP, HikariCP, etc.).
- Tratar corretamente as exceções, oferecendo mensagens claras e logando os erros para análise posterior.
- Nunca deixar dados sensíveis, como senhas, expostos diretamente no código fonte; utilizar arquivos de configuração externos.

Vamos fazer um teste?

Dúvidas?

jessica.oliveira@p.ucb.br

Referências Bibliográficas.

- ECKEL, B. **Thinking in Java.** 4. ed. Prentice Hall, 2006.
- ELMASRI, R.; NAVATHE, S. B. **Sistemas de Banco de Dados.** 6. ed. São Paulo: Pearson, 2011.
- FOWLER, M. **Patterns of Enterprise Application Architecture.** Addison-Wesley, 2003.
- HORSTMANN, C. S.; CORNELL, G. **Core Java Volume I - Fundamentals.** 11. ed. Prentice Hall, 2019.
- ORACLE. **JDBC Overview.** Documentação Oficial. Disponível em: https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/ Acesso em: 25 maio 2025.
- SILBERSCHATZ, A.; KORTH, H. F.; SUDARSHAN, S. **Sistemas de Banco de Dados.** 6. ed. São Paulo: Pearson, 2011.
- H2 Database. **H2 Database Engine.** Documentação oficial. Disponível em: https://www.h2database.com Acesso em: 25 maio 2025.