



Programação Avançada

8

Padrão de Arquitetura DAO

Bruno Silva, Patrícia Macedo

Sumário


- Padrão **Data Access Object** (DAO)
 - Enquadramento
 - Motivação
 - Solução Proposta (pelo padrão)
 - Exemplo de Aplicação
 - Exercícios
 - Prós e contras

Enquadramento

Um **padrão de arquitetura** consiste numa solução geral e reutilizável para um problema recorrente em *arquitetura de software* num dado contexto.

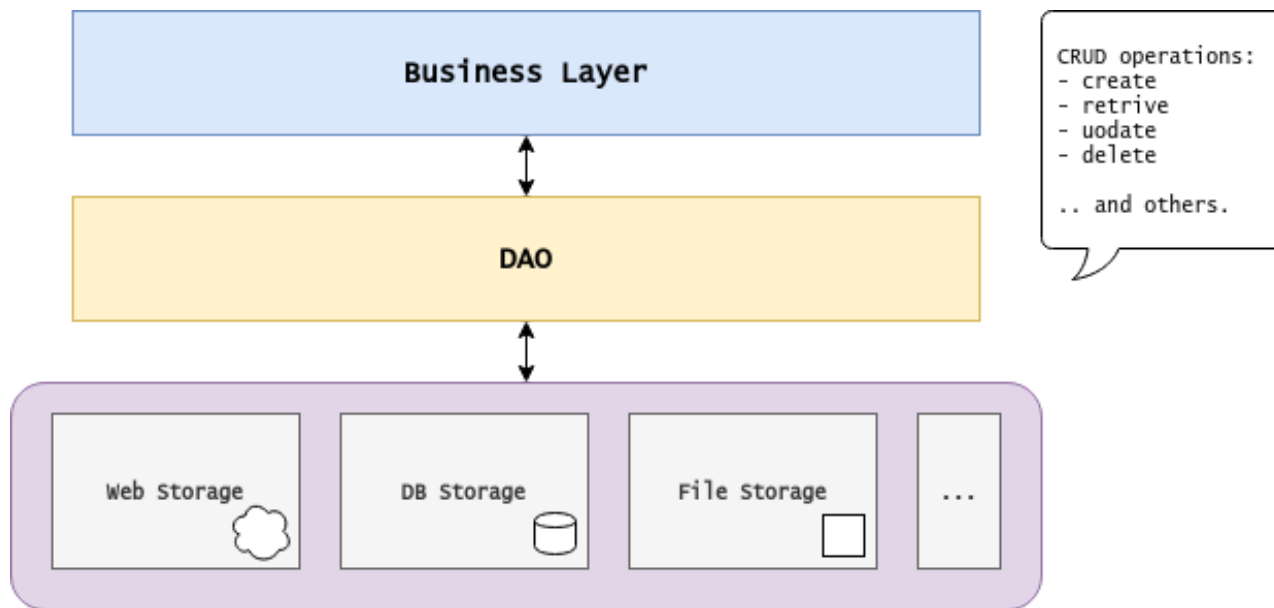
Os padrões de *arquitetura* são semelhantes aos de *desenho*, mas com um âmbito/alcance maior.

? Exemplos de padrões:

- Cliente-servidor;
- *Master-slave*
- *Peer-to-peer*
- *Model-view-controller*
- ***Data Access Object*** (DAO) 

Motivação 🤔

O padrão **DAO** propõe um modelo de camadas que permite separar a lógica de negócio da lógica de acesso aos dados (e persistência).

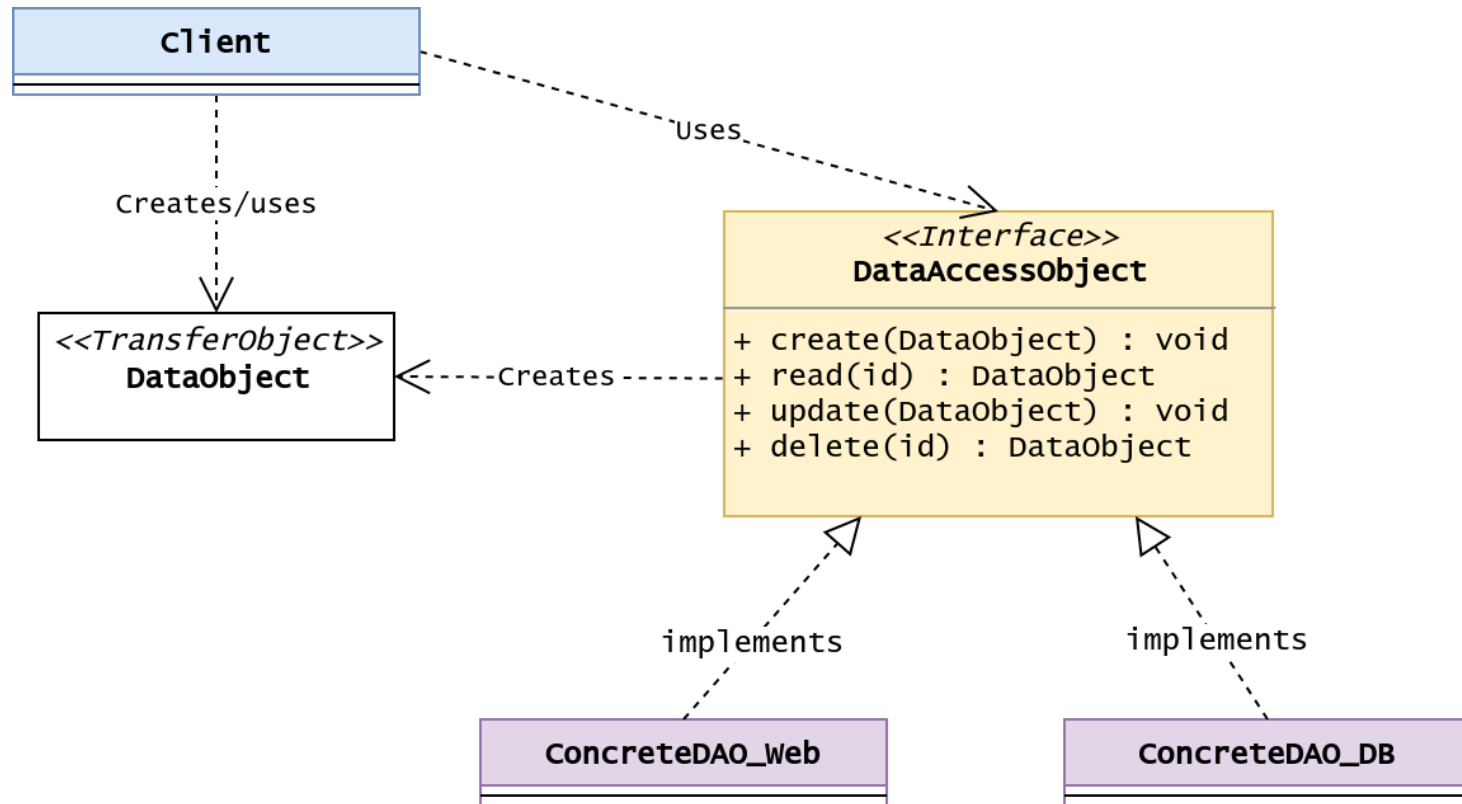


⚠️ permite alterar o mecanismo de "armazenamento" sem quebrar o código da lógica de negócio.

Solução Proposta 😊

- Como na maioria dos padrões, promove o desacoplamento de classes, através do uso de *interface(s)* que a camada de negócio utiliza e cuja implementação concreta pode variar.
 - Conceptualmente pode ser encarado como uma variante do padrão *Strategy* para acesso a um repositório de dados de um tipo.

Solução Proposta 😊



💡 as cores utilizadas mapeiam as camadas apresentadas.

Solução Proposta 🤗

Participantes do padrão:

- **Data access object (DAO):** Interface que declara os métodos das operações passíveis de serem efetuadas sobre o repositório de informação.
- **Data object:** Tipo de objeto utilizado na transferência de dados. O repositório contém "instâncias" deste tipo (explicitamente ou implicitamente; e.g., uma BD regra geral não guarda "objetos", mas registos).
- **Concrete DAOs:** Implementação da *interface DAO* sobre um repositório específico de dados (Ficheiros locais, BD, *web service*, etc.).
- **Cliente:** Camada/módulo que necessita de acesso aos dados. Fá-lo através da *interface DAO*.

Exemplo de aplicação

Repositório de apoio à aula:

- https://github.com/brunomnsilva/JavaPatterns_DAO

Apresenta um programa (interação com o utilizador fornecida através de uma *prompt* de comandos) que permite gerir uma coleção de livros:

```
Available commands: GET, ADD, DEL, LIST, SEARCH, RANGE, COUNT, QUIT  
Type command >
```

💡 O mecanismo de acesso aos dados / persistência é feito através do padrão de arquitetura **Data Access Object**; o *transfer object* é o tipo `Book`.

Data Access Object


A interface `DAO<T, K>` descreve, genericamente, um DAO cujo **data object** é do tipo `T` e cujo **identificador único** de cada *data object* é do tipo `K`.

```
public interface Dao<T, K> {  
    T get(K key);  
    Collection<T> getAll();  
    void save(T instance) throws DaoException;  
    void update(T instance) throws DaoException;  
    T delete(K key);  
    int count();  
}
```


⚠ genericamente, descreve o conjunto de operações CRUD sobre um repositório de "instâncias" do tipo `T`.

Exemplo de aplicação

A interface anterior pode ser "extendida" para um *data object* concreto, e.g., um livro (ver classe `domain.Book`)

- `Book` é o **data object**;
- `String` é o tipo do identificador único (ISBN) 

```
public interface BookDao extends Dao<Book, String> {  
    /* Additional operations besides CRUD (inherited): */  
    Collection<Book> getAllFromAuthorSearch(String queryString);  
    Collection<Book> getAllFromYearRange(int yearStart, int yearEnd);  
}
```

 as operações "adicionais" são resultado de análise das funcionalidades necessárias no acesso aos dados.

Exemplo de aplicação

O *cliente* apenas interage com a *interface* BookDAO, e.g.:

```
BookDao dao = ... /* Instância de concrete dao */

//...

Collection<Book> all = dao.getAll();
for(Book b : all) {
    System.out.println(b);
}

//...
Book deleted = dao.delete("A6G-8H2-E9P");
System.out.println("Book was deleted: " + deleted);

//...
```

Exemplo de aplicação

! Implementações fornecidas de `BookDao` :

- O exemplo mais básico de um repositório de armazenamento é a memória RAM (⚠ persistência *volátil*).
 - `BookDaoVolatileList` : utiliza como coleção subjacente uma lista, i.e., `List<Book>`
- Repositórios de armazenamento com persistência *não-volátil* podem envolver a utilização de ficheiros locais:
 - `BookDaoSerialization` : estende o mecanismos anterior de forma a persistir a coleção de livros para disco sempre que houver alterações; através de *Java serialization*.
 - Ficheiro persistido: `storage/books.dat`
 - ...

Exemplo de aplicação

! Implementações fornecidas de `BookDao` :

- Repositórios de armazenamento com persistência *não-volátil* podem envolver a utilização de ficheiros locais:
 - ...
 - `BookDaoTextFiles` : guarda a informação dos livros do repositório individualmente em ficheiros de texto (legíveis); o nome do ficheiro é o ISBN do livro (identificador único).
 - Ficheiros individuais em: `storage/*.book`

Exercícios

1. (A) Teste o programa fornecido, variando o repositório de armazenamento.

⚠ Note que:

- As alterações efetuadas sobre mecanismos de persistência **não-voláteis** sobrevivem entre execuções do programa.
- As alterações num tipo de repositório não se refletem nos outros; isto deverá ser óbvio.

Exercícios

1. (B) Complete o método `CLI` por forma a implementar o comando "range":

```
///...  
case "range":  
    /* TODO: implement this command */  
    System.out.println("[Not implemented]");  
    break;  
//...
```

- O comando deve utilizar a operação do *Dao*:

```
getAllFromYearRange(int yearStart, int yearEnd)
```

Exercícios

2. Crie uma implementação `BookDaoVolatileMap` que implementa a *interface* `BookDao` utilizando como coleção subjacente uma instância de `Map<String, Book>`.
 - Semelhante a `BookDaoVolatileList`, mas mais fácil de implementar.
 - Altere o método `main` por forma a utilizar este mecanismo e teste.
3. Aplique o padrão **Simple Factory** para criar/instanciar as variantes de `BookDao` disponíveis. Utilize a fábrica no método `main`.

Exercícios

4. Pretende-se implementar um mecanismo de persistência que utilize o formato *JSON*:

JSON (JavaScript Object Notation) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate.

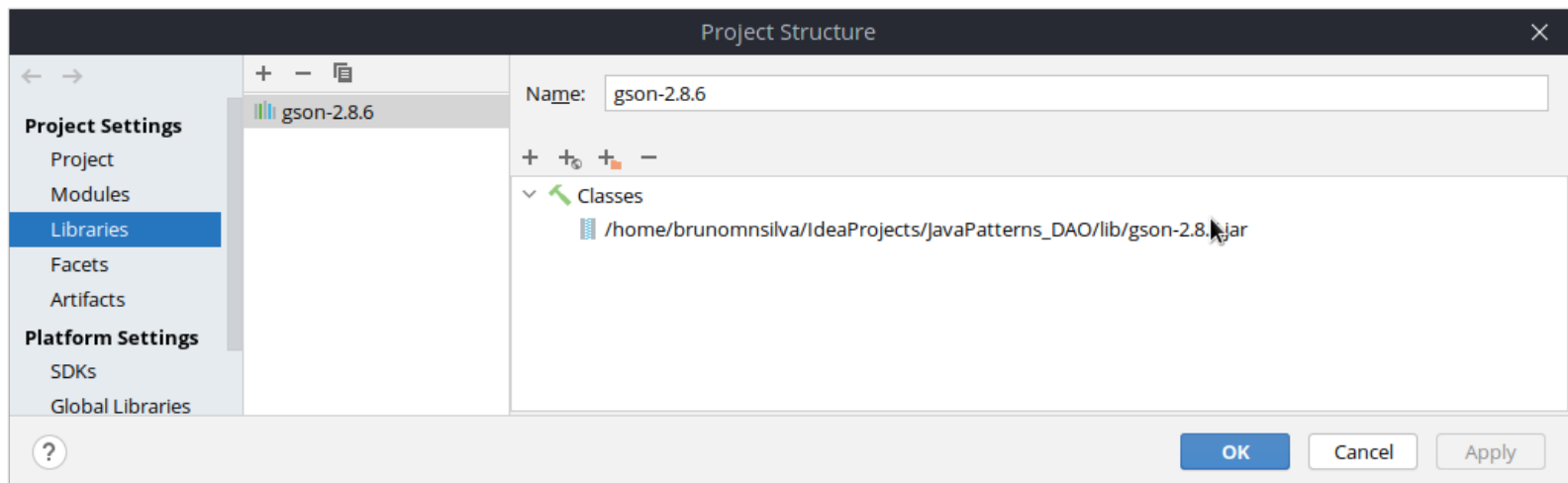
- Crie uma classe `BookDaoJSON` para esse efeito e adicione-a à fábrica anterior.
- ...

Exercícios

4. Pretende-se implementar um mecanismo de persistência que utilize o formato *JSON* (continuação):
 - Importe a biblioteca **GSON** que tratará de gerar/ler o formato *JSON* na linguagem Java.
 - <https://github.com/google/gson>
 - <https://search.maven.org/artifact/com.google.code.gson/gson/2.8.6/jar> (download)
 - Coloque o *jar* numa pasta `/lib` do projeto IntelliJ (criar e configurar - slide seguinte)
 - Implemente de forma semelhante à classe `BookDaoSerialization`, mas fazendo uso desta biblioteca para persistir a coleção. Ver exemplo:
 - <https://futurestud.io/tutorials/gson-mapping-of-arrays-and-lists-of-objects>

Exercícios

Para incluir uma biblioteca *jar* num projeto IntelliJ configure o *Project Settings*:

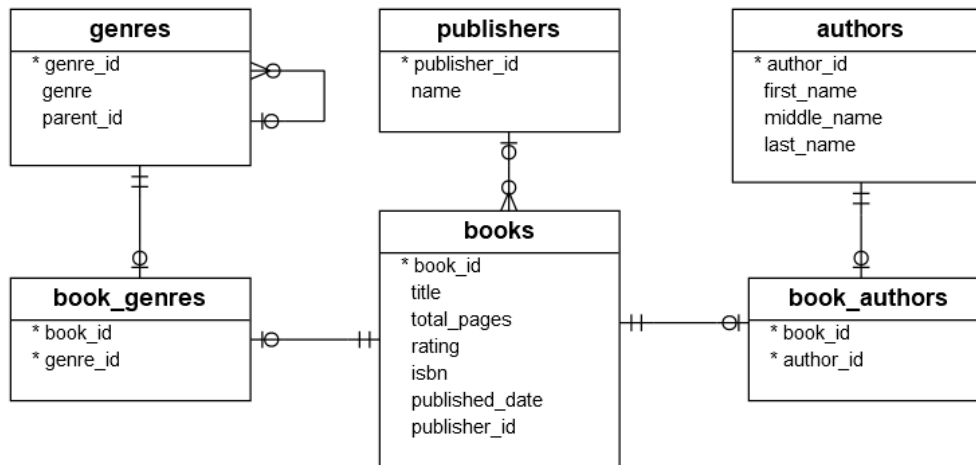


Prós e contras

- ✓ Promove o fraco acoplamento entre o *cliente* (ou camada de negócio) e a lógica de acesso aos dados.
- ✓ *Single Responsibility Principle*. O código de acesso aos dados está centralizado num único sítio do código.
- ✓ *Open/Closed Principle*. É fácil adicionar novos mecanismos de persistência sem "quebrar" o código existente do *cliente*.
- ✗ Se uma aplicação irá utilizar apenas **um** mecanismo de persistência, o padrão introduz uma complexidade desnecessária.
- ✗ O padrão DAO utiliza "mapeamentos" completos dos objetos; mesmo que só seja requerida parte da informação de um *transfer object*, ele é lido na totalidade.

Prós e contras

✗ Utilizar o padrão DAO para dados relacionais é complexo, e.g.,



⚠ Implica um *dao* separado para cada tabela de informação, e.g.,
 BookDao , AuthorDao , BookAuthorDao e a aplicação do padrão **Factory Method**.

ORMs

Na sequência da "desvantagem" anterior, existem *frameworks* de *Object-Relation Mapping* (ORM) disponíveis para Java que simplificam a aplicação deste padrão no mundo real, nomeadamente:

- <https://hibernate.org/orm/>
 - Implementação de *Java Persistence API (JPA)* vocacionada para bases de dados relacionais.
 - Permite alterar o SGBD subjacente sem alterar o *cliente*.

! No problema proposto, outro desafio seria implementar um `BookDaoSQLite` que utiliza uma tabela para guardar a informação dos livros. Não é difícil, mas introduzia entropia desnecessária à compreensão do padrão.

Referência web

- <https://www.journaldev.com/16813/dao-design-pattern>