**PADRÃO DE PROJETO DAO**

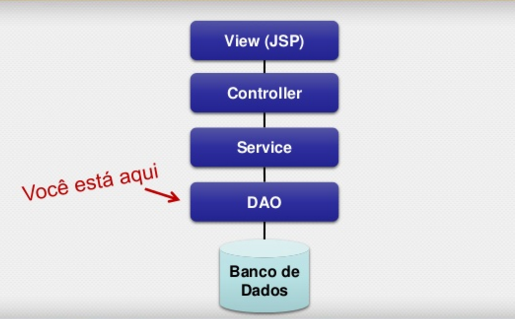
**Definição**

O Data Access Object (DAO) é um padrão de projetos onde um objeto:

* provê uma interface que abstrai o acesso a dados;
* lê e grava a partir da origem de dados (banco de dados, arquivo, memória, etc.);
* encapsula o acesso aos dados, de forma que as demais classes não precisam saber sobre isso.

**Arquitetura**

Numa aplicação web comum seguindo o modelo MVC, os DAOs ficam junto com o Model fazendo um trabalho de suporte, integrando a fonte de dados ao modelo de objetos do sistema.

[](https://i.stack.imgur.com/rEhAT.png)

**Responsabilidades**

Seguindo o princípio de responsabilidade única, um DAO não deve ser responsável por mais do que acesso aos dados.

Definir quem faz o que pode ser um problema quando pensamos na arquitetura de um sistema, mas grande parte disso é porque misturamos as coisas.

Se olhar bem o diagrama acima, fica fácil identificarmos a responsabilidade de cada elemento.

Suponha que o usuário está acessando a página inicial do sistema web. Então uma interação comum poderia ser:

* Um controller recebe a requisição do usuário
* Esse controller chama o método do service adequado para obter as informações para aquela página
* O service chama um ou mais métodos de DAOs para obter as informações necessárias e retorna os dados para o controller
* O controller recebe os dados e redireciona o usuário para uma view que vai renderizar o HTML da página

Adicionalmente, temos que:

* Controllers executam lógica relacionada à navegação do usuário no sistema, isto é, qual URL ou qual ação exibe qual página.
* Services executam a lógica do sistema, que pode incluir gerenciar transações e processar os dados

Portanto, em geral, cada método do DAO deve fazer uma única leitura ou gravação no banco de dados e não deve controlar transações ou realizar operações adicionais, tal como realizar alterações nos dados recebidos do serviço.

**Implementação**

A vantagem de usar uma classe específica para o acesso a dados é evitar espalhar SQLs em todo lugar, tornando a manutenção e evolução de um sistema um pesadelo.

Em geral, agrupa-se os acessos aos dados por similaridade, por exemplo, uma classe por tabela. Porém, não é sempre que isso faz sentido, principalmente quando o sistema não é somente feito de cadastros simples (CRUD).

Um exemplo claro são sistemas que possuem buscas avançadas em que acessam várias tabelas. Nesse caso, cada situação precisa ser analisada caso a caso. No exemplo das buscas, um DAO específico para isso seria interessante.

Usar interfaces é opcional. Veja, Java tem uma péssima reputação por usar muitas interfaces, mas isso tem seus motivos, por exemplo:

Permitir várias implementações para bancos de dados diferentes sem alterar o sistema

Permitir versões diferentes convivendo na mesma versão do sistema (isso pode ser útil em alguns casos, como quando algum campo pode ou não existir e você quer atualizar o sistema sem obrigar a criação do campo)

Facilitar testes unitários criando implementações Fakes dos DAOs, por exemplo, que usam listas em memória, embora frameworks como Mockito consigam gerar mocks dinamicamente sem uma interface.

**Mais Informações**

Saiba mais sobre o padrão de projeto DAO nos seguintes sites:

Explicações:

* <http://www.argonavis.com.br/cursos/java/j550/j550_13.pdf>
* http://www.facom.ufu.br/~bacala/PI/11-WebMVCePatterns.pdf

Explicações com Implementação:

* <http://www.deinf.ufma.br/~geraldo/poo/10.1.DAO.pdf>
* <https://www.devmedia.com.br/implementando-o-data-access-object-no-java-ee/33339>
* http://www.macoratti.net/11/10/pp\_dao1.htm#:~:text=O%20padr%C3%A3o%20DAO%20%C3%A9%20um,utilizam%20banco%20de%20dados%20relacionais

**Referências**

https://pt.stackoverflow.com/questions/113840/como-funciona-o-padr%C3%A3o-dao