

**Pacote:** `poo.dao.impl`

**Classe testada:** `ProvaDaoJdbc`

**Classe de teste:** `ProvaDaoJdbcTest`

---

## 1. Contexto e Objetivo

O conjunto de testes definidos em `ProvaDaoJdbcTest` tem como objetivo validar o comportamento da classe `ProvaDaoJdbc`, responsável pelo CRUD-L do model `Prova` no banco de dados.

A camada testada pertence ao DAO, que interage diretamente com o `JdbcTemplate` do Spring.

Os testes utilizam mocking do `JdbcTemplate` para simular consultas e operações SQL, evitando dependência de um banco real. Assim, garantem que os métodos da DAO executem corretamente as instruções SQL e lidem adequadamente com resultados e exceptions.

---

## 2. Unidade Testada

A unidade de teste é a classe `ProvaDaoJdbc`, cuja função é implementar as operações CRUD-L sobre o model `Prova`.

---

## 3. Estratégia de Teste

A estratégia utilizada é testar o comportamento de cada método público da DAO, verificando:

- Se os métodos chamam o `JdbcTemplate` de forma correta.
- Se o retorno faz sentido.

- Se exceções não são lançadas em situações erradas.
  - Se o resultado dos métodos ( `Optional` , `boolean` , ou `Prova` ) reflete corretamente o estado esperado.
- 

## 4. Testes Realizados

### 1. `shouldCreateProva_whenFieldsValid`

- **Objetivo:** verificar se o método `create` envia a instrução `INSERT_SQL` corretamente e retorna o objeto persistido.
- **Cenário:** criação de uma nova prova com todos os campos válidos.
- **Validação:** o retorno é o mesmo objeto mockado; o SQL e parâmetros enviados ao `JdbcTemplate` são corretos.

### 2. `shouldReturnEmptyOptional_whenFindByIdDoesNotMatch`

- **Objetivo:** garantir que `findById` devolva `Optional.empty()` quando o ID não existe.
- **Cenário:** consulta por um ID inexistente.
- **Validação:** o método retorna vazio e a consulta é feita com o SQL correto.

### 3. `shouldUpdateProva_whenStateIsValidAndIdPresent`

- **Objetivo:** confirmar que `update` atualiza registros válidos com ID definido.
- **Cenário:** atualização de uma prova existente.
- **Validação:** o método devolve `Optional` com o objeto atualizado e verifica a chamada do SQL `UPDATE_SQL` .

### 4. `shouldThrowException_whenUpdateCalledWithoutId`

- **Objetivo:** garantir que uma exceção seja lançada ao tentar atualizar uma prova sem ID.
- **Cenário:** Prova criada sem identificador.
- **Validação:** lança `NullPointerException` com mensagem adequada e **sem interação com o banco**.

## 5. `shouldReturnFalse_whenDeleteDoesNotAffectRows`

- **Objetivo:** verificar se `delete` retorna `false` quando nenhuma linha é afetada.
  - **Cenário:** exclusão de um ID inexistente.
  - **Validação:** o método retorna `false` e confirma a execução do comando `DELETE`.
- 

## 5. Conclusão

Os testes realizados garantem que a entidade `prova` tenha confiabilidade e persistência correta, testando casos onde deve falhar e passar para cada operação.

Ao usar mocks, asseguramos que o comportamento da DAO é validado sem acessar o banco real, permitindo mais velocidade, robustez e isolamento (permitindo termos CERTEZA que a falha vem do arquivo e não de outro ponto da arquitetura, como o banco, rede, outro arquivo, etc).

Caso os testes passem, a classe `ProvaDaoJdbc` se prova conforme as boas praticas de implementação e robustez nos cenários de manipulação de dados.