Certified Tech Developer
The Ultimate Degree

Padrão DAO

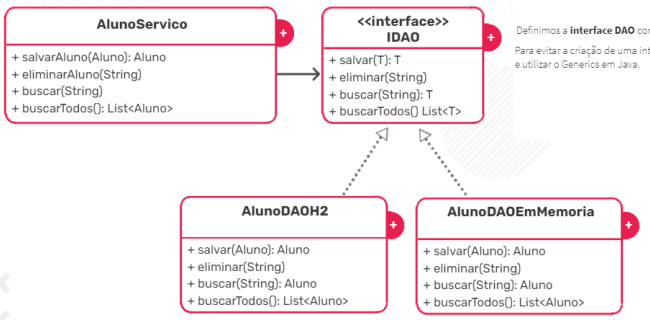
Como sabemos, com JDBC temos a possibilidade de utilizar diferentes mecanismos de banco de dados, sem afetar nosso código. Se respeitarmos os contratos, ou seja, sempre programamos contra as interfaces e nunca contra as implementações, ficaremos imunes às mudanças de driver. Com apenas dois toques podemos abandonar o MySQL e migrar para o PostgreSQL, já que ambos os motores usam o padrão SQL.

Mas, infelizmente, nem todos os provedores respeitam o padrão. Isso significa que as consultas que estávamos usando no MySQL provavelmente falhariam no PostgreSQL, pois há detalhes que mudam. Vamos ver como o padrão DAO nos ajudará a mudar o mecanismo de banco de dados sem afetar a lógica principal do nosso sistema.

Implementação

Agora vamos ver como podemos representar o padrão DAO. Vamos supor que estamos modelando um sistema para uma academia e precisamos gerenciar o cadastro de alunos com a possibilidade de mudar a forma de persistir (armazenar) os alunos sem afetar o funcionamento da camada de serviço.

Na **camada de negócio**, criamos um serviço que utiliza essas implementações. Como podemos ver, o serviço AlunoServico usa a interface IDAO independente de qual implementação foi realizada, pois se comunicará da mesma maneira.

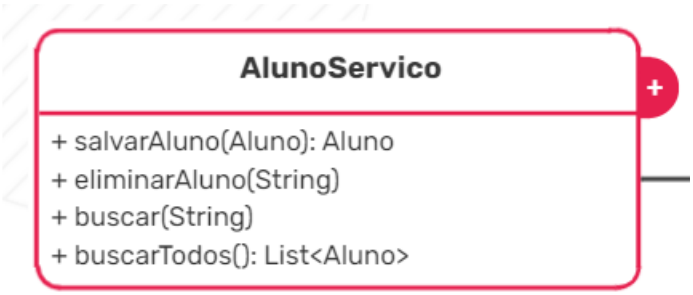


Definimos a **interface DAO** com as operações mais comuns, que teremos que implementar posteriormente.

Para evitar a criação de uma interface por entidade, por exemplo: IAlunoDAO e IMaterialDAO, podemos definir uma única interface e utilizar o Generics em Java.

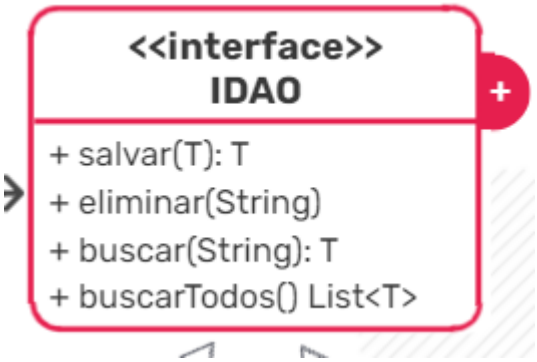
Em seguida, devemos criar as implementações dessa interface, no nosso caso iremos criar duas, uma para se conectar a um banco de dados H2 e outra para persistir em uma lista.

JUnit

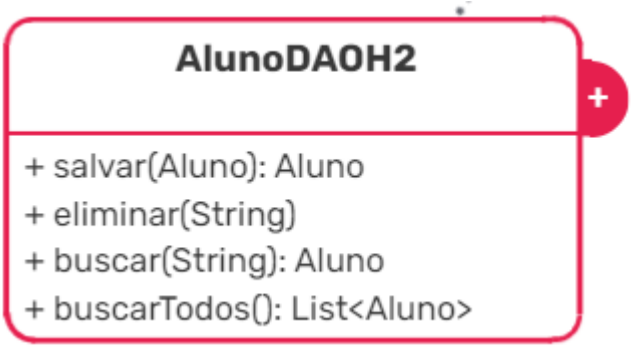



```
public class AlunoServico {

private IDAO<Aluno> alunoDAO;

public AlunoServico() {

}

public AlunoServico( IDAO<Aluno> alunoDAO) {
this.alunoDAO = alunoDAO;
}

public void setAlunoDao( IDAO<Aluno> alunoDAO) {
this.alunoDAO = alunoDAO;
}

public Aluno salvarAluno(Aluno aluno){
alunoDAO.salvar(aluno);
return aluno;
}
public List<Aluno> buscarTodos(){
```

```
return alunoDAO.buscarTodos();
}

public Aluno buscar(String id){
return alunoDAO.buscar(id);
}

public void  eliminarAluno(String id){
alunoDAO.eliminar(id);
}

}
```

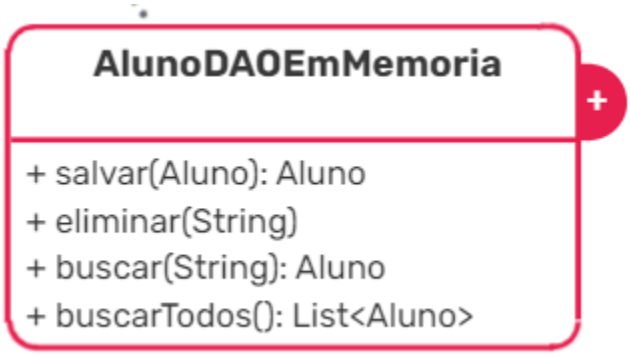



```
public interface IDAO<T> {

public T salvar(T t);
public void eliminar(String id);
public T buscar(String id);
public List<T> buscarTodos();
}
```



```
public class AlunoDAOH2 implements IDAO<Aluno> {

private ConfigurationJDBC configurationJDBC;

public AlunoDAOH2(ConfigurationJDBC configurationJDBC) {
this.configurationJDBC = configurationJDBC;
}

@Override
public Aluno salvar(Aluno aluno) {
Connection connection = configurationJDBC.conectarBancoDeDados();
Statement stmt = null;
String query = String.format("INSERT INTO Aluno
VALUES('%s','%s','%s')", aluno.getId(), aluno.getNome(), aluno.getApelido());
try {
stmt = connection.createStatement();
stmt.executeUpdate(query);
stmt.close();
} catch (SQLException throwables) {
throwables.printStackTrace();
}
return aluno;
}

@Override
public void eliminar(String id) {
Connection connection = configurationJDBC.conectarBancoDeDados();
Statement stmt = null;
String query = String.format("DELETE FROM Aluno where id = %s", id);
try {
stmt = connection.createStatement();
stmt.executeUpdate(query);
stmt.close();
} catch (SQLException throwables) {
throwables.printStackTrace();
}

}

@Override
public Aluno buscar(String id) {
Connection connection = configurationJDBC.conectarBancoDeDados();
Statement stmt = null;
String query = String.format("SELECT id, nome, apelido  FROM Aluno where id =
'%s'", id);
Aluno aluno = null;
try {
stmt = connection.createStatement();
ResultSet result = stmt.executeQuery(query);
while (result.next()) {
String idAluno = result.getString("id");
```

```
String nome = result.getString("nome");
String apelido = result.getString("apelido");
aluno = new Aluno(idAluno, nome, apelido);
}


stmt.close();
} catch (SQLException throwables) {
throwables.printStackTrace();
}


return aluno;
}


@Override
public List<Aluno> buscarTodos() {
Connection connection = configurationJDBC.conectarBancoDeDados();
Statement stmt = null;
String query = "SELECT *  FROM Aluno";
List<Aluno> aluno = new ArrayList<>();
try {
stmt = connection.createStatement();
ResultSet result = stmt.executeQuery(query);
while (result.next()) {
String id = result.getString("id");
String nome = result.getString("nome");
String apelido = result.getString("apelido");
aluno.add(new Aluno(id, nome, apelido));


}


stmt.close();
} catch (SQLException throwables) {
throwables.printStackTrace();
}


return aluno;
}
}
```




```java
public class AlunoDAOEmMemoria implements IDAO<Aluno> {
private List<Aluno> alunoRepositorio;

public AlunoDAOEmMemoria(List<Aluno> alunoRepositorio) {
this.alunoRepositorio = alunoRepositorio;
}

@Override
public Aluno salvar(Aluno aluno) {
alunoRepositorio.add(aluno);
return aluno;
}

@Override
public void eliminar(String id) {
alunoRepositorio.removeIf(aluno -> aluno.getId().equals(id));
}

@Override
public Aluno buscar(String id) {
return alunoRepositorio.stream().filter(aluno ->
aluno.getId().equals(id)).findFirst().orElseGet(null);
}

@Override
public List<Aluno> buscarTodos() {
return alunoRepositorio;
}
}
```

Agora vamos testar a camada de serviço com o JUnit

Primeiro, criamos uma instância de AlunoServico, salvamos um aluno, alteramos a implementação e salvamos outro aluno novamente.

```
private IDAO<Aluno> AlunoDAOEmMemoria = new AlunoDAOEmMemoria(new ArrayList());
private IDAO<Aluno> AlunoDAOH2 = new AlunoDAOH2(new ConfigurationJDBC());
private AlunoServico AlunoServico = new AlunoServico();

@Before
public void salvarAlunoAlterandoImplementacaoDAO() {
   alunoServico.setAlunoDAO(alunoDAOEmMemoria);
   alunoServico.salvarAluno(new Aluno("1", "Aluno 1", "Apelido 1"));
   alunoServico.setAlunoDAO(alunoDAOH2);
   alunoServico.salvarAluno(new Aluno("2", "Aluno 2", "Apelido 2"));
}
```

```
private IDAO<Aluno>                  = new AlunoDAOEmMemoria(new ArrayList());
private IDAO<Aluno>            = new AlunoDAOH2(new ConfigurationJDBC());
private AlunoServico              = new AlunoServico();



public void salvarAlunoAlterandoImplementacaoDAO() {
   alunoServico.setAlunoDAO(                );
               .salvarAluno(new Aluno("1", "Aluno 1", "Apelido 1"));
               .setAlunoDAO(          );
               .salvarAluno(new Aluno("2", "Aluno 2", "Apelido 2"));
}
```

```java
@Test
public void buscarAlunosAlterandoImplementacaoDAO(){
   alunoServico.setAlunoDAO(alunoDAOEmMemoria);
   Aluno aluno = alunoServico.buscar("1");
   Assert.assertEquals(aluno.getId(),"1");
   Assert.assertEquals(aluno.getNome(),"Aluno 1");
   Assert.assertEquals(aluno.getApelido(),"Apelido 1");

   alunoServico.setAlunoDAO(alunoDAOH2);
   aluno = alunoServico.buscar("2");
   Assert.assertEquals(aluno.getId(),"2");
   Assert.assertEquals(aluno.getNome(),"Aluno 2");
   Assert.assertEquals(aluno.getApelido(),"Apelido 2");

}
```

Como podemos observar, independentemente da implementação que estivermos utilizando, o serviço sempre se comporta da mesma maneira.


```java
@Test
public void buscarAlunosAlterandoImplementacaoDAO(){
   alunoServico.setAlunoDAO(alunoDAOEmMemoria);
   Aluno aluno = alunoServico.buscar("1");
   Assert.assertEquals(aluno.getId(),"1");
   Assert.assertEquals(aluno.getNome(),"Aluno 1");
   Assert.assertEquals(aluno.getApelido(),"Apelido 1");

   alunoServico.setAlunoDAO(alunoDAOH2);
   aluno = alunoServico.buscar("2");
   Assert.assertEquals(aluno.getId(),"2");
   Assert.assertEquals(aluno.getNome(),"Aluno 2");
   Assert.assertEquals(aluno.getApelido(),"Apelido 2");


}
```