CES/JF Centro de Ensino Superior de Juiz de Fora

MANTENEDORA

SVD - ESDEVA

Desenvolvimento Orientado à Objetos

Persistência - JDBC

Christien Lana Rachid

2019/1

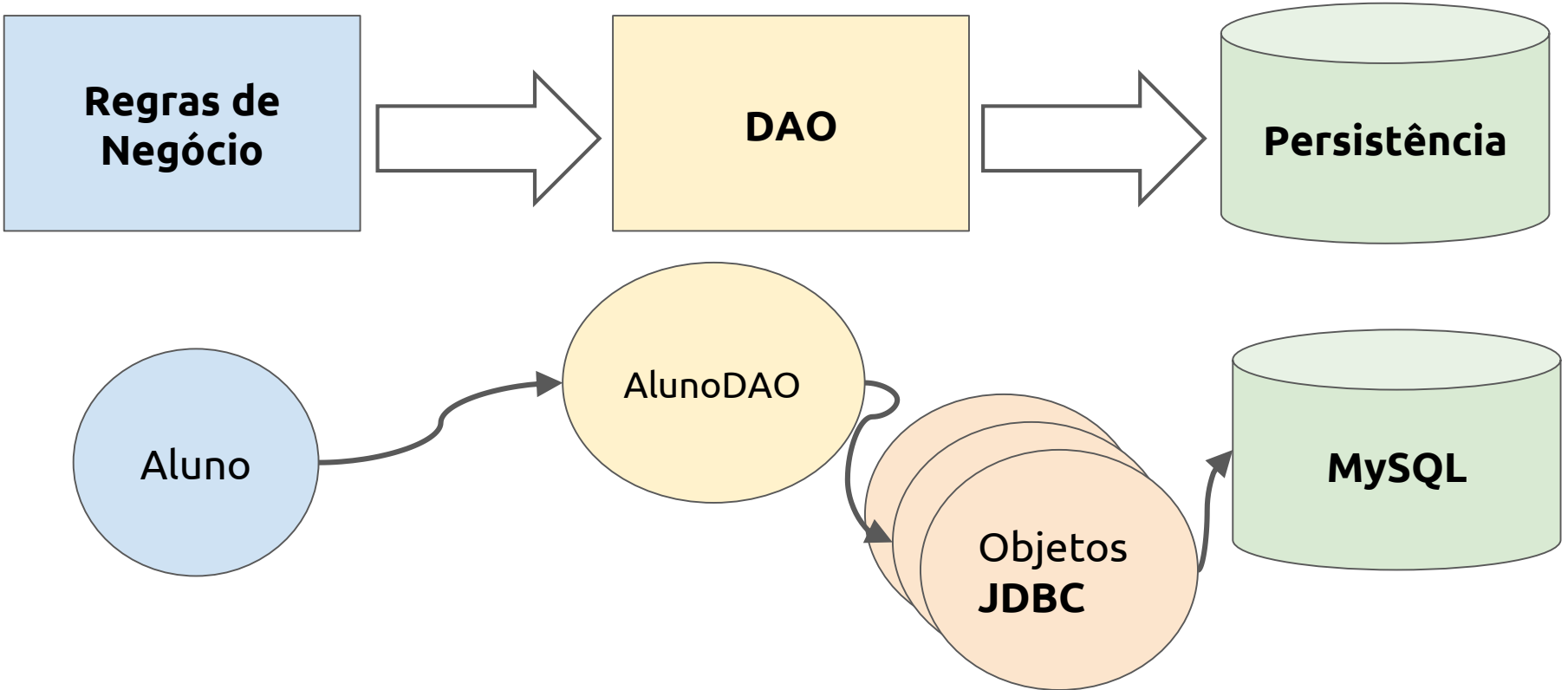
Camada de Persistência | Como organizar o código?

- O acesso à base de dados é frequente em Sistemas de Informação.
  - Persistir, recuperar, atualizar e remover informações.
- Fragmentação do acesso à base de dados
  - Presença de **Connection**s, **Statement**s e **SQLException** por todo código.
  - Presença de SQL em strings difíceis de ler e manter.
  - Replicação de código entre diversas partes da aplicação.

Camada de Persistência | Data Access Objects

- Uma solução para o esse problema é o uso do padrão de projeto **Data Access Object (DAO)**
  - Objeto de Acesso a Dados
- Introduzidos como um padrão do J2SE e J2EE.
- Objetos responsáveis por abstrair o acesso a um repositório de dados.
  - Arquivo em disco, Planilha, Banco de Dados
- Dessa forma o código de regra de negócios não se comunica diretamente com um repositório de dados, e sim com um **Camada DAO (DAO *Layer*)**

Camada de Persistência | Data Access Objects



Data Access Objects | Vantagens

- Isolando as regras de negócio do acesso à camada de dados, aplica o **Princípio da Responsabilidade Simples**.
- Permite que a persistência seja substituída sem impacto nas regras de negócio.
  - Substituição do MySQL pelo SQL Server
- Por ser um padrão de projeto, pode ser utilizado em diversas linguagens, frameworks e bancos de dados.

Data Access Objects | Implementação

- Tradicionalmente, sempre criamos uma interface que define as operações suportadas pela nossa camada de persistência.
  - Exemplo: AlunoDAO
- Criamos implementações específicas das operações suportadas.
  - Exemplo: AlunoDAOJdbc, AlunoDAOHibernate
- No restante do código, deve-se referenciar apenas à interface e às implementações padrão, nunca às implementações específicas.

Data Access Objects | Implementação - Exemplo

- Primeiro criamos a interface com as operações suportadas.

```
public interface AlunoDAO {

   Aluno inserir(Aluno aluno);

   void atualizar(Aluno aluno);

   void remover(Aluno aluno);

   List<Aluno> listarTodos();
}
```

Definimos na interface todas a operações que precisamos para:

- Inserir um novo aluno.
- Atualizar um aluno.
- Excluir um aluno.
- Consultar todos os alunos.

Data Access Objects | Implementação - Exemplo

```
public class AlunoDAOJdbc implements AlunoDAO {
        // Implementação
}
public class AlunoDAOHibernate implements AlunoDAO {
        // Implementação
}

// Utilização:
AlunoDAO dao = new AlunoDAOJdbc();
dao.inserir(new Aluno(10001, “john Doe”));
```

Alguns autores utilizam o sufixo **Impl** para a implementação “padrão”:

- **AlunoDAOImpl**

Data Access Objects | Implementação - Exemplo

```
public class DAOBase {


   protected Connection novaConexao() throws DAOException {
      try {
         Class.forName("com.microsoft.sqlserver.jdbc.SQLServerDriver");
         String url = "jdbc:sqlserver://localhost:1433;databaseName=academia";
         return DriverManager.getConnection(url,"sa","p@ssw0rd");
      } catch (ClassNotFoundException | SQLException e) {
         throw new DAOException("Erro ao estabelecer conexão.", e);
      }
   }
}
```

Como todos os DAOs podem ter muitos comportamentos e características diferentes, podemos colocar esses comportamentos em uma superclasse comum.

Para separar os erros que a camada de DAO pode apresentar dos erros específicos de bancos de dados (Lembre que o repositório de dados pode ser um Excel), criamos uma exceção específica.

Data Access Objects | Implementação - Exemplo

```
public class DAOException extends Exception {

   public DAOException(String message, Throwable e){
          super(message, e);
   }

   public DAOException(String message){
      super(message);
   }
}
```

DAOException é uma exceção.

Vamos criar os construtores para nossa exceção, que por sua vez utilizam a palavra chave **super** para usar os construtores de Exception.

Data Access Objects | Implementação - Exemplo

```
public class AlunoDAOJdbc extends DAOBase implements AlunoDAO {

}
```

O **AlunoDAOJdbc** herda da classe **DAOBase** e **implementa** a interface **AlunoDAO.**

Dessa forma ele herda todas as características de DAOBase e é obrigado a implementar os métodos definidos no contrado da interface AlunoDAO: **inserir**, **atualizar**, **remover**, **listarTodos**.

Podemos dizer que o **AlunoDAOJdbc é um DAOBase**, e ao mesmo tempo também é valido dizer que **AlunoDAOJdbc é um AlunoDAO**.

Data Access Objects | Implementação - Exemplo

```
@Override
public void inserir(Aluno aluno) throws DAOException {
   Connection conn=null;
   PreparedStatement stmt=null;
   try {
         // Inserção
   } catch (SQLException e) {
      throw new DAOException("Erro ao inserir!", e);
   } finally {
      if( stmt != null ){
         try{
            stmt.close();
         }catch( SQLException e ){ e.printStackTrace(); }
      }
      if( conn != null ){
         try{
            conn.close();
         } catch( SQLException e ){ e.printStackTrace(); }
      }
   }
}
```

Inserção

```
st
st
st
st
stmt.setDate(5,  new
java.sql.Date(aluno.getDataNascimento().getTime()));
st
st
```

Como **inserir** é um método da interface, precisamos adicionar a anotação **Override.**

Precisamos transformar os **SQLException** em **DAOException**, assim podemos mudar a persistência alterando somemente a camada DAO.

Devemos fechar todos os recursos inicializados, tratando os erros possíveis: **Connection** e **Statement** devem ser fechados.

Data Access Objects | Implementação - Exemplo

Podemos utilizar o método **novaConexao** da classe **DAOBase** para conectar com o banco.

```
try {
    // Inserção
} catch (SQLException e) {
```

Inserção

```
conn = novaConexao();

String sql = "INSERT INTO aluno (nome, data_matricula,
endereco, telefone, data_nascimento, altura, peso) " +
      "VALUES (?, ?, ?, ?, ?, ?, ?)";

stmt = conn.prepareStatement(sql);
stmt.setString(1, aluno.getNome());
stmt.setDate(2,  new java.sql.Date(aluno.getDataMatricula().getTime()));
stmt.setString(3,  aluno.getEndereco());
stmt.setString(4,  aluno.getTelefone());
stmt.setDate(5,  new java.sql.Date(aluno.getDataNascimento().getTime()));
stmt.setFloat(6,  aluno.getAltura());
stmt.setFloat(7,  aluno.getPeso());

stmt.executeUpdate();
```

Montamos o SQL necessário e criamos um **PreparedStatement**.

Executamos o Statement construído no banco de dados.

```
    }
    if( conn != null ){
}
```

Data Access Objects | Implementação - Exemplo

- O código para os métodos **atualizar**, **remover** e **listarTodos** é análogo.
- Dessa forma, o DAO pode ser facilmente utilizado para processamento de regras de negócio:

```
Aluno john = new Aluno(null, "John Doe", new Date(),"Rua 7", "32323232", new Date(),1.80f, 90f);
AlunoDAO dao = new AlunoDAOJdbc();

dao.inserir(john);
john.setTelefone("3299999-9999");
dao.atualizar(john);

for(Aluno aluno : dao.listarTodos()) {
  if (aluno.getNome().equals("John Doe")) {
     System.out.println("John Está matriculado");
  }
}
```

Data Access Objects | Atividade 1

- Utilizando o CRUD de Aluno desenvolvido, refatore-o para a utilização do Padrão **DAO**.