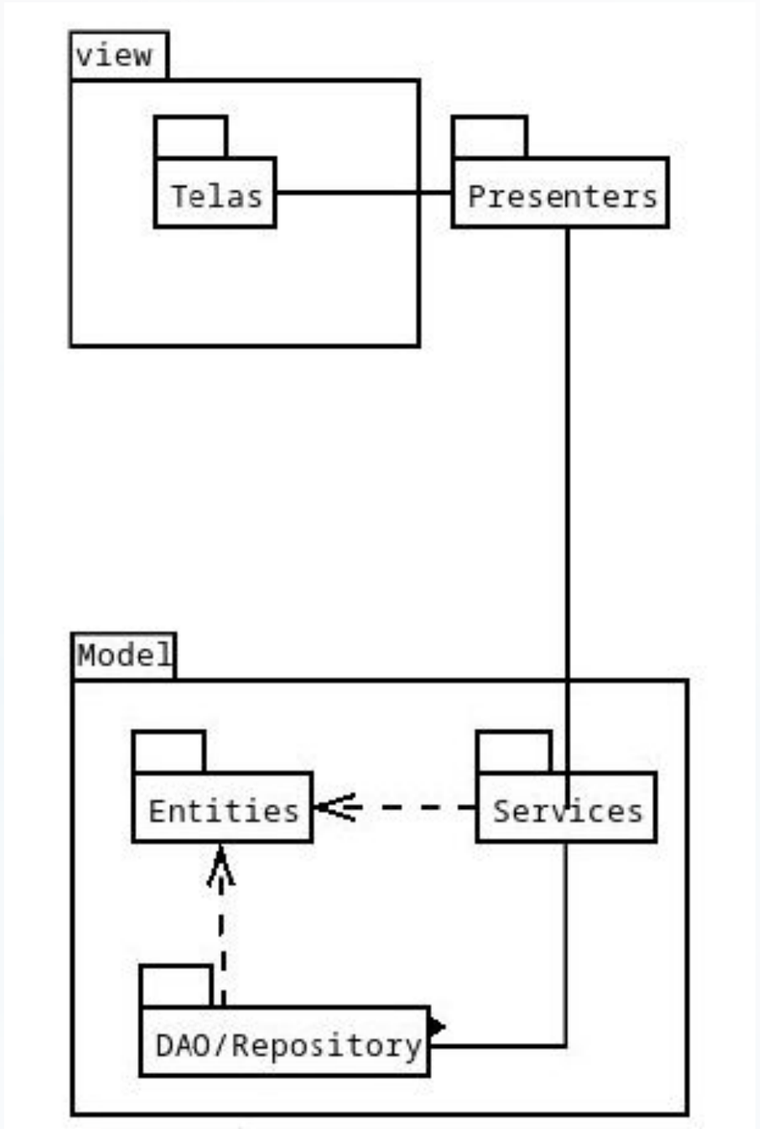
Programação Orientada a Objetos

Camada de Service e DAO

Rayanne Giló da Silva
rayanne.silva@alunos.ufersa.edu.br

Estrutura do Projeto



Diferentes responsabilidades

- Camada de ***View***:
  - interface com o usuário;
  - exibe dados e capta eventos (cliques, inputs);
  - delega ações ao Presenter.
- Camada de ***Presenters***:
  - orquestra fluxo entre View e Model;
  - adapta dados do Service/DAO para a View;
  - trata eventos de UI.
- Camada de ***Service;***
- Camada de ***DAO;***
- Camada de ***Entity*** ;

Diferentes responsabilidades

- Camada de ***View;***
- Camada de ***Presenters***;
- Camada de ***Service***:
  - contém regras de negócio de alto nível;
  - valida dados;
  - orquestra operações transacionais via DAO.
- Camada de ***DAO***:
  - encapsula acesso a dados;
  - realiza CRUD;
  - gerencia transações;
  - mapeamento objeto-relacional.
- Camada de ***Entity***:
  - representa o modelo de domínio;
  - mapeia atributos para colunas de banco;
  - contém validações básicas.

Problema

Visão | Service | Modelo | ??????? | Persistência

S1

S2

M1

M2

Problema

Visão | Controle | Modelo | **DAO** | Persistência

C1

C2

M1

M2

DAO

DAO

DAO

- Data Access Object

- É responsável por encapsular toda a lógica de acesso a dados;.

- Geralmente, o DAO é implementado como uma **interface com métodos** específicos para operações de CRUD

- Responsabilidades:
  - Operações CRUD.
  - Mapeamento objeto-relacional.
  - Gestão de transações (begin, commit, rollback).
  - Isolar detalhes do provedor de persistência.

Interface DAO

```java
public interface UserRepository {
    User findById(Long id);
    List<User> findAll();
    void save(User user);
    User update(User user);
    void delete(User user);
    User findByEmail(String email);
}
```

Implementação do DAO

```
public class UserRepositoryImpl implements UserRepository {

    // Conexão com o banco de dados
    private final EntityManager em = JPAUtil.getEntityManagerFactory();




    (métodos...)
}
```

Busca por ID e busca de todos os usuário

```java
// Buscar pelo ID
@Override
    public User findById(Long id) {
        return em.find(User.class, id);
    }

// Buscar todos os usuários
    @Override
    public List<User> findAll() {
        return em.createQuery("FROM User", User.class).getResultList();
    }
```

Busca por ID e busca de todos os usuário

```java
// Buscar pelo ID
@Override
    public User findById(Long id) {
        return em.find(User.class, id);
    }

// Buscar todos os usuários
    @Override
    public List<User> findAll() {
        return em.createQuery("FROM User", User.class).getResultList();
    }
```

Salvando e atualizando um usuário

```java
// Salvando um usuário
public void save(User user) {
        EntityTransaction ts = em.getTransaction();
        ts.begin();
        em.persist(user);
        ts.commit();
    }

// Atualizando um usuário
    @Override
    public User update(User user) {
        EntityTransaction ts = em.getTransaction();
        ts.begin();
        User merged = em.merge(user);
        ts.commit();
        return merged;
    }
```

Deletando e buscando por e-mail

```java
// Deletando um usuário
@Override
    public void delete(User user) {
        EntityTransaction ts = em.getTransaction();
        ts.begin();
        em.remove(em.contains(user) ? user : em.merge(user));
        ts.commit();
    }

// Buscando por e-mail
    @Override
    public User findByEmail(String email) {
        TypedQuery<User> q = em.createQuery(
                "SELECT u FROM User u WHERE u.email = :e", User.class);
        q.setParameter("e", email);
        return q.getResultStream().findFirst().orElse(null);
    }
```

Camada de Serviço

- Contém regras de negócio de alto nível e orquestração de operações.

- Geralmente, também possui um interface.

- Responsabilidades principais:
  - Validar dados e aplicar regras de domínio.
  - Delegar operações de persistência ao DAO.
  - Tratar/expor exceções de negócio (BusinessException).

Interface da Camada de Serviço

```java
public interface UserService {
    User getById(Long id);
    List<User> getAll();
    void register(User user);
    void changeEmail(Long id, String newEmail);
    void remove(Long id);
}
```

Implementação

```
public class UserServiceImpl implements UserService {

    // Chamada da implementação do respositório
    private final UserRepository repo = new UserRepositoryImpl();



  (...outros métodos...)

}
```

Busca por ID e busca de todos os usuário

```java
@Override
public User getById(Long id) {
  return repo.findById(id);
}

public List<User> getAll() {
  return repo.findAll();
}
```

Busca por ID e busca de todos os usuário

```java
@Override
public User getById(Long id) {
  return repo.findById(id);
}

public List<User> getAll() {
  return repo.findAll();
}
```

Exemplo de Registro

```
@Override
public void register(User user) {
  if(repo.findByEmail(user.getEmail()) != null) {
      throw new IllegalArgumentException("Email já cadastrado");
    }

   repo.save(user);
}
```

Exemplo de Deleção e de Alteração de e-mail

```java
@Override
public void changeEmail(Long id, String newEmail) {
    User u = repo.findById(id);
    if (u == null) throw new IllegalArgumentException("Usuário não encontrado");
    u.setEmail(newEmail);
    repo.update(u);
}

@Override
public void remove(Long id) {
    User u = repo.findById(id);
    if (u != null) repo.delete(u);
}
```

Exemplo de Uso na Main

```
public class Main {
    public static void main(String[] args) {
        UserService service = new UserServiceImpl();

        User u1 = new User("Pedro de Sousa", "maria.joana.silva@example.com");
        User u2 = new User("Maria Clara Esteves", "joao.oliveira@example.com");

        service.register(u1);
        System.out.println("Salvo: " + u1);

        service.register(u2);
        System.out.println("Salvo: " + u2);

        service.getAll().forEach(System.out::println);

        JPAUtil.shutdown();
    }
}
```

Boas práticas

- DAO:
  - usar try/catch para rollback;
  - lançar erros de acesso ao banco.
- Service:
  - capturar erros de negócio e lançar exceções .
- Validar parâmetros nos setters da entidade (usar *IllegalArgumentException*).
- Fechamento de conexões:
  - sempre chamar *JPAUtil.shutdown()* ao final da aplicação para fechar o *EntityManagerFactory* e liberar recursos (conexões de banco, caches internos, threads), evitando vazamentos de memória e garantindo encerrar corretamente os pools de conexão.

Dúvida?

Atividade

- Agora tente implementar utilizando como base o template disponível no link:
  - https://github.com/rayannegsilva/template_poo/tree/servicesedao