**Documentação do padrão de projeto**

**Composite**

**Matheus Carvalho Castro1, Rafaela Ferreira2, Lucas Aurélio3, Samuel Hipólito4**

**Tiago Nunes5 ,Kaique Icaro6**

Centro Universitário Una Contagem  
Caixa Postal – Contagem – MG – Brazil

Curso de análise e desenvolvimento de sistemas-Una Contagem

***Abstract.*** *This meta-article describes the composite design pattern, with its functionalities, characteristics and its objectives within the subject of programming, being very specific in face of its descriptions.*

***Resumo.*** *Este meta-artigo descreve o padrão de projeto composite, com suas funcionalidades, características e seus objetivos dentro do assunto da programação, sendo bem específico diante de suas descrições.*

**1. Resumo da funcionalidade do composite**

O padrão composite utiliza uma composição de vários objetos em árvores, sendo que após essa junção em árvores podem ser trabalhados como objetos de forma singular. Diante disso, resume bem a estrutura de códigos extensos

**2. Construção do código**

**2.1. Início**

Para dar início à criação do código do projeto, o iniciamos respeitando a seguinte ordem de criação:

* Criação Pacote Composite
* Criação Interface Departamento
* Criação Classe DepartamentoComercial
* Criação Classe DepartamentoFinanceiro
* Criação Classe DepartamentoChefe - Composite
* Criação Classe Main

**2.2. Meio**

**2.2.1**Após a criação das classes, inserimos os complementos necessários.

* Interface Departamento

***package composite;***

***public interface Departamento {***

***// Comportamento padr�o aplicado em todas as classes implementadas com essa interface***

***public void imprimeDetalhesDepartamento();***

***}***

* Classe DepartamentoComercial

***package composite;***

***public class DepartamentoComercial implements Departamento {***

***private String nome;***

***private int id;***

***public DepartamentoComercial(int id, String nome) {***

***this.id = id;***

***this.nome = nome;***

***}***

***// Imprime os detalhes dessa classe***

***@Override***

***public void imprimeDetalhesDepartamento() {***

***System.out.println("id: " + id + " | nome: " + nome);***

***}***

***}***

* Classe DepartamentoFinanceiro

***package composite;***

***public class DepartamentoFinanceiro implements Departamento {***

***private String nome;***

***private int id;***

***public DepartamentoFinanceiro(int id, String nome) {***

***this.id = id;***

***this.nome = nome;***

***}***

***// Imprime os detalhes dessa classe***

***@Override***

***public void imprimeDetalhesDepartamento() {***

***System.out.println("id: " + id + " | nome: " + nome);***

***}***

***}***

* DepartamentoChefe – Composite

***package composite;***

***import java.util.ArrayList;***

***import java.util.List;***

***// Molde para as classes Composite (que v�o delegar/organizar)***

***public class DepartamentoChefe implements Departamento {***

***// Lista para agrupar classes Folha (que v�o manipular e manter os dados)***

***private List<Departamento> listaDepartamento = new ArrayList<Departamento>();***

***// Imprime os detalhes das classes filhas***

***@Override***

***public void imprimeDetalhesDepartamento() {***

***for (Departamento dep : listaDepartamento) {***

***dep.imprimeDetalhesDepartamento();***

***}***

***}***

***// Adiciona uma Folha a lista de Folhas acima***

***public void addDepartamento(Departamento dep) {***

***listaDepartamento.add(dep);***

***}***

***// Remove uma Folha da lista de Folhas acima***

***public void removeDepartamento(Departamento dep) {***

***listaDepartamento.remove(dep);***

***}***

***}***

**2.3. Final**

**2.3.1**Estruturação da Classe Main.

***package composite;***

***public class Main {***

***public static void main(String[] args) {***

***// Cria 2 departamentos da classe Folha "DepartamentoComercial"***

***DepartamentoComercial dep1 = new DepartamentoComercial(1, "Departamento de vendas de cosmeticos");***

***DepartamentoComercial dep2 = new DepartamentoComercial(2, "Departamento de vendas de eletronicos");***

***// Cria 1 departamento da classe Composite "DepartamentoChefe" para delegar os dois departamentos comerciais***

***DepartamentoChefe depChefeComercial = new DepartamentoChefe();***

***// Adiciona os 2 departamentos Folha para a lista do objeto "depChefeComercial"***

***depChefeComercial.addDepartamento(dep1);***

***depChefeComercial.addDepartamento(dep2);***

***// Cria 2 departamentos da classe Folha "DepartamentoFinanceiro"***

***DepartamentoFinanceiro dep3 = new DepartamentoFinanceiro(3, "Departamento financeiro de cosmeticos");***

***DepartamentoFinanceiro dep4 = new DepartamentoFinanceiro(4, "Departamento financeiro de eletronicos");***

***// Cria 1 departamento da classe Composite "DepartamentoChefe" para delegar os dois departamentos financeiros***

***DepartamentoChefe depChefeFinanceiro = new DepartamentoChefe();***

***// Adiciona os 2 departamentos Folha para a lista do objeto "depChefeFinanceiro"***

***depChefeFinanceiro.addDepartamento(dep3);***

***depChefeFinanceiro.addDepartamento(dep4);***

***// Cria 1 departamento da classe Composite "DepartamentoChefe" para delegar os dois departamentos chefes (comercial e financeiro)***

***DepartamentoChefe depChefe = new DepartamentoChefe();***

***// Adiciona os 2 departamentos Composite para a lista do objeto "depChefe"***

***depChefe.addDepartamento(depChefeComercial);***

***depChefe.addDepartamento(depChefeFinanceiro);***

***// Imprime os detalhes das classes Folha ultilizando o objeto Composite "depChefe"***

***depChefe.imprimeDetalhesDepartamento();***

***}***

***}***

**3. Vantagens e desvantagens do Composite**

**3.1. Vantagens**

**3.1.1** Pode trabalhar de forma mais simples com árvores mais complexas.

**3.1.2** Pode introduzir novos elementos no código sem problema de “quebrar” o código já existente.

**3.2. Desvantagens**

**3.2.1** Pode ser difícil fazer uma interface simples, uma vez que possue classes com diversas funcionalidades, fazendo com que a interface seja mais genérica, podendo assim sendo de difícil compreensão.

**4. Exemplo de estrutura do Composite**



**5.1** Estrutura do Composite na vida real

**Referências**

<https://refactoring.guru/pt-br/design-patterns/composite>

<https://refactoring.guru/pt-br/design-patterns/composite/java/example>