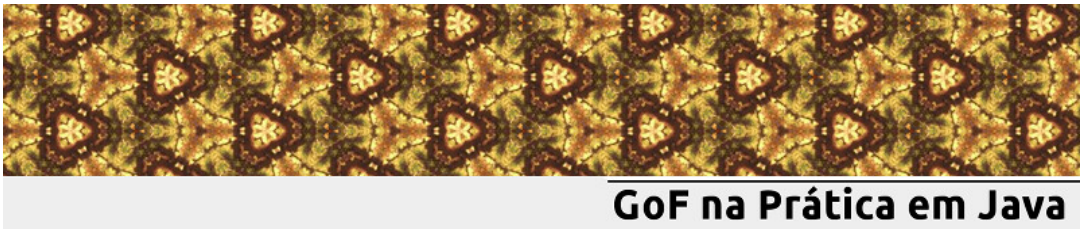
# Composite

***Fernando Anselmo***
*GoF na Prática em Java*

**Função deste Padrão**

**Compor objetos em estruturas de árvore para representar hierarquias do tipo todo-parte. O Composite deixa os clientes tratarem objetos individuais e composições de objetos do mesmo modo.**




## 1 Ficha do Padrão

**Tipo** : Estrutural, diz respeito como classes e objetos podem ser combinados para formar grandes estruturas.

**Conhecimentos** : Classes Abstratas e Classes Concretas.

**Consequências** : Definir uma hierarquia de classes que consiste de objetos primitivos e objetos compostos. O cliente pode tratar estruturas compostas e objetos individuais uniformemente. Os clientes normalmente não sabem (e nem devem se preocupar) se estão tratando de um componente individual ou composto.

**É usado quando** : Para representar hierarquias de objetos do tipo todo-parte. O cliente deve ignorar a diferença entre composições de objetos e objetos individuais.

## 2 Problema

O cliente deseja construir um menu completo para um sistema que está projetando para isso criou uma classe MenuItem, porém não sabe como agregá-la para formar uma estrutura mais complexa.

## 3 Prévia Estrutura de Classes

Classe abstrata que representa um componente do menu (seja um "Item do Menu" ou um "Menu"):

**Listagem 1:** *Classe Abstrata MenuComponente*

```
1 abstract class MenuComponente {
2   private String link;
3   private String descricao;
4
5   public MenuComponente(String descricao, String link) {
6     this.descricao = descricao;
7     this.link = link;
8   }
9   public String toString() {
```

```
    if (link != null) {
      return "\t" + descricao + " - " + link;
    }
    return descricao;
  }
  public abstract void print();
}
```

Classe para conter os itens do menu:

**Listagem 2:** *Classe MenuItem*

```
class MenuItem extends MenuComponente {
  public MenuItem(String descricao, String link) {
    super(descricao, link);
  }
  public void print() {
    System.out.println(super.toString());
  }
}
```

# 4 Aplicação do Padrão

Classe que implementa a composição de um menu:

**Listagem 3:** *Classe Menu*

```
class Menu extends MenuComponente {
  private List<MenuComponente> componentes;

  public Menu(String descricao) {
    super(descricao, null);
    componentes = new ArrayList<MenuComponente>();
  }
  public void add(MenuComponente componente) {
    componentes.add(componente);
  }
  public void print() {
    System.out.println(">> " + super.toString());
    for (MenuComponente menuComponente: componentes) {
      menuComponente.print();
    }
  }
}
```

Classe com um exemplo de uso pelo cliente:

**Listagem 4:** *Classe Cliente*

```
public class Cliente {
  public static void main(String[] args) {
    new Cliente().montarMenu();
  }
  public void montarMenu() {
    Menu parte = new Menu("Parte 1");
    parte.add(new MenuItem("Item 1", "Evento 1"));
    parte.add(new MenuItem("Item 2", "Evento 2"));

    Menu tabela = new Menu("Tabelas");
    tabela.add(new MenuItem("Tabela 1", "Evento 3"));
```

```
12      tabela.add(new MenuItem("Tabela 2", "Evento 4"));
13      tabela.add(new MenuItem("Tabela 3", "Evento 5"));
14
15      Menu menu = new Menu("Menu");
16      menu.add(parte);
17      menu.add(tabela);
18
19      menu.print();
20    }
21  }
```

## Referências


[1] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides *Design Patterns. Elements of Reusable Object-Oriented Software 1 ed.* Estados Unidos, Addison-Wesley, 1995, ISBN 0-201-63361-2.