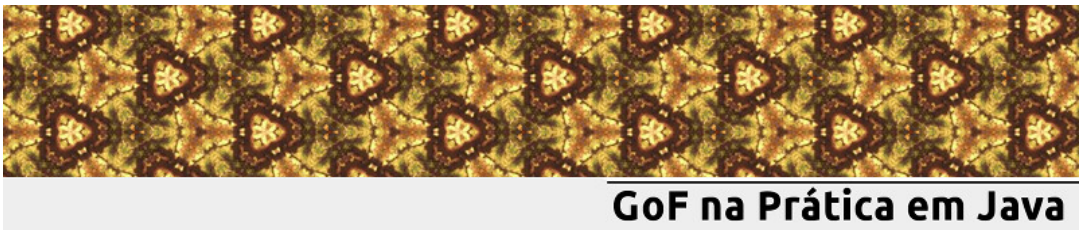
Visitor

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Representar uma operação a ser realizada sobre elementos da estrutura de um objeto. Permitir que se crie uma nova operação sem que se mude a classe dos elementos sobre as quais ela opera.



1 Ficha do Padrão

Tipo : Comportamental, especificamente voltados para a comunicação entre objetos.

Conhecimentos : Interface e Classes Concretas.

Consequências : A adição de novas operações seja relativamente fácil. Reúne as operações relacionadas e separa as não relacionadas.

É usado quando : Uma estrutura de objetos contém várias classes com diferentes interfaces, e deve realizar operações nesses objetos que dependem de suas classes concretas ou para definir novas operações sobre essa estrutura.

2 Problema

O usuário trabalha com montagem de computadores que podem possuir vários periféricos. Ele deseja mostrar a descrição desses computadores de forma mais simples possível.

3 Prévia Estrutura de Classes

Classe para implementação de um periférico:

Listagem 1: *Classe Periferico*

```
1  class Periferico {
2    private String nome;
3
4    public Periferico(String nome) {
5      this.nome = nome;
6    }
7    public String toString() {
8      return nome;
9    }
10 }
```

Classe para implementação de uma CPU:

Listagem 2: *Classe Cpu*

```
class Cpu {
  private ParteComp[] partes;

  public Cpu() {
    partes = new ParteComp[] {
      new Periferico("Mouse"), new Periferico("Teclado"), new Periferico("Monitor")};
  }
}
```

4 Aplicação do Padrão

Interface para distribuição do padrão:

Listagem 3: *Interface ParteVisitor*

```
interface ParteVisitor {
  public void visit(Cpu cpu);
  public void visit(Periferico periferico);
}
```

Classe para a implementação do Padrão:

Listagem 4: *Classe ShowVisitor*

```
class ShowVisitor implements ParteVisitor {
  public void visit(Cpu cpu) {
    System.out.println("Computador");
  }
  public void visit(Periferico periferico) {
    System.out.println("Periferico: " + periferico);
  }
}
```

Interface para organização das Classes:

Listagem 5: *Interface ParteComp*

```
interface ParteComp {
  public void adicionar(ParteVisitor parteVisitor);
}
```

Modificação na classe para implementação de um periférico:

Listagem 6: *Classe Periferico*

```
class Periferico implements ParteComp {
  private String nome;
  public Periferico(String nome) {
    this.nome = nome;
  }
  public String toString() {
    return nome;
  }
  public void adicionar(ParteVisitor parteVisitor) {
    parteVisitor.visit(this);
  }
}
```

Modificação na classe para implementação de uma CPU:

Listagem 7: *Classe Cpu*

```
class Cpu implements ParteComp {
  private ParteComp[] partes;
  public Cpu() {
    partes = new ParteComp[] {
      new Periferico("Mouse"), new Periferico("Teclado"), new Periferico("Monitor")};
  }
  public void adicionar(ParteVisitor parteVisitor) {
    for (int i = 0; i < partes.length; i++) {
      partes[i].adicionar(parteVisitor);
    }
    parteVisitor.visit(this);
  }
}
```

Classe exemplo de uso:

Listagem 8: *Classe Loja*

```
public class Loja {
  public static void main(String[] args) {
    ParteComp computador = new Cpu();
    computador.adicionar(new ShowVisitor());
  }
}
```

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.