Padrão de Projeto
VISITOR

Isac Mendes

05/08/2018

Resumo

Padrões de projetos são soluções para problemas que alguém um dia teve e resolveu aplicando um modelo que foi documentado e que você pode adaptar integralmente ou de acordo com necessidade de sua solução.

Palavras-chaves: padrões de projeto, Java, comportamentais visitor.

Introdução

Este é o terceiro trabalho da disciplina de Programação Orientada a Objeto. Desta vez cada estudante trabaha usando Padrões de Projetos escolhidos pela responsável da disciplina.

O conhecimento de padrões de projeto permite a criação de sistemas melhores, com manutenção menos custosa e reduzindo o tempu de desenvolvimento. Os padrões facilitam o entendimento dos dos autores envolvidos no projeto e focam em técnicas testadas e aprovadas para um determinado problema.
Neste artigo aborda especificamente sobre o Padrão de Projeto **Visitor**. No artigo mostra os diagramas classes do próprio padrão e também do exemplo feito pelos alunos com os códigos feitos usando linguagem Java.

1 Desemvolvimento

1.1 Padrão de Projeto

Foi proposto pelo pelo arquiteto e urbanista austríaco chamado Christopher Wolfgang Alexander. O interesse dele em traduzir tais padrões em maneiras de fácil entendimento para a aplicação em projetos de construção futuros ou melhoria dos pré-existentes resultou na publicação, em 1977, do seu

livro mais famoso: “A Pattern Language: Towns, Buildings, Construction”.

O padrão de projeto nada mais é do que um modelo de uma experiência amplamente testada e aprovada que pode ser usado como um guia para a resolução de problemas específicos de arquiteturas orientadas a objetos. É uma alternativa reutilizável, baseada na abstração das escolhas bem sucedidas para atender uma dificuldade conhecida que é composta de quatro elementos essenciais: o nome, o problema que pretende resolver, a solução para o problema e as consequências da aplicação dele.

1.2 Características de Padrão de Projeto

Os Padrões de Projetos tem as seguintes características:

1. Devem possuir um nome, que descreva o problema, as soluções e conseqüências. Um nome permiti definir o vocabulário a ser utilizado pelos projetistas e desenvolvedores em um nível mais alto de abstração.

2. Todo padrão deve relatar de maneira clara a qual (is) problema(s) ele deve ser aplicado, ou seja, quais são os problemas que quando inserido em um determinado contexto o padrão conseguirá resolve-lo.Alguns podendo exigir pré-condições.

3. Solução descreve os elementos que compõem o projeto, seus relacionamentos, responsabilidades e colaborações. Um padrão deve ser uma solução concreta, ele deve ser exprimido em forma de gabarito (algoritmo) que, no entanto pode ser aplicado de maneiras diferentes.

4. Todo padrão deve relatar quais são as suas conseqüências para que possa ser analisada a solução alternativa de projetos e para a compreensão dos benefícios da aplicação do projeto.

1.3 Importância de Padrão de Projeto

O mais importante sobre os padrões é que eles são soluções aprovadas. Cada catálogo inclui apenas padrões que foram considerados úteis por diversos desenvolvedores em vários projetos. Os padrões catalogados também são bem definidos; os autores descrevem cada padrão com muito cuidado e em seu próprio contexto, portanto será fácil aplicar o padrão em suas próprias circunstâncias. Eles também formam um vocabulário comum entre os desenvolvedores.

1.4 Tipos de Padrões de Projetos

Existem três categoria de Padrões de Projetos, tais como:
1. Padrão Estrutural
2. Padrão de Criação
3. Padrão Comportamental

1.4.1 Padrão Estrutural

Os padrões estruturais se preocupam com a forma como classes e objetos são compostos para formar estruturas maiores. Os de classes utilizam a herança para compor interfaces ou implementações, e os de objeto ao invés de compor interfaces ou implementações, eles descrevem maneiras de compor objetos para obter novas funcionalidades. A flexibilidade obtida pela composição de objetos provém da capacidade de mudar a composição em tempo de execução o que não é possível com a composição estática (herança de classes).

Exemplo: Adapter, etc.

1.4.2 Padrão de Criação

Os padrões de criação são aqueles que abstraem e ou adiam o processo criação dos objetos. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Um padrão de criação de classe usa a herança para variar a classe que é instanciada, enquanto que um padrão de criação de objeto delegará a instanciação para outro objeto.

Os padrão de criação tornam-se importantes à medida que os sistemas evoluem no sentido de dependerem mais da composição de objetos do que a herança de classes. O desenvolvimento baseado na composição de objetos possibilita que os objetos sejam compostos sem a necessidade de expor o seu interior como acontece na herança de classe, o que possibilita a definição do comportamento dinamicamente e a ênfase desloca-se da codificação de maneira rígida de um conjunto fixo de comportamentos, para a definição de um conjunto menor de comportamentos que podem ser compostos em qualquer número para definir comportamentos mais complexos.

Exemplo: Factory Method, Singleton, etc.

1.4.3 Padrão Comportamental

Os padrão de comportamento se concentram nos algoritmos e atribuições de responsabilidades entre os objetos. Eles não descrevem apenas padrões de objetos ou de classes, mas também os padrões de comunicação entre os ob-

jetos.

Os padrões comportamentais de classes utilizam a herança para distribuir o comportamento entre classes, e os padrões de comportamento de objeto utilizam a composição de objetos em contrapartida a herança. Alguns descrevem como grupos de objetos cooperam para a execução de uma tarefa que não poderia ser executada por um objeto sozinho.

Exemplo: Template Method, Visitor, etc.

1.5 Padrão de Projetos Visitor

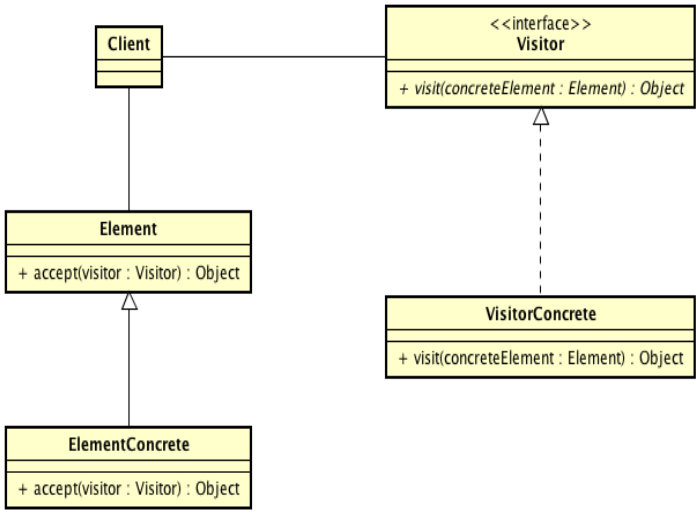
Como já disse na introdução que este artigo aborda especificamente no Padrão de Projeto **Visitor**

Visitor - Representa uma operação a ser realizada sobre elementos da estrutura de um objeto, sendo permitida a criação de uma nova operação sem que mude a classe dos elementos sobre as quais é operado. O Visitor faz parte do Padrões GoF comportamentais. Ele encapsula as extensões de funcionalidade em objetos separados.

Para ajudar o entendimento sobre o Padrão do Projeto Visitor, mostramos a baixo os diagramas de classes do proprio padrão e do exemplo proposto pelo autor.

Diagrama de classe do Padrão Visitor

figura 1. *fonte: wikipedia*



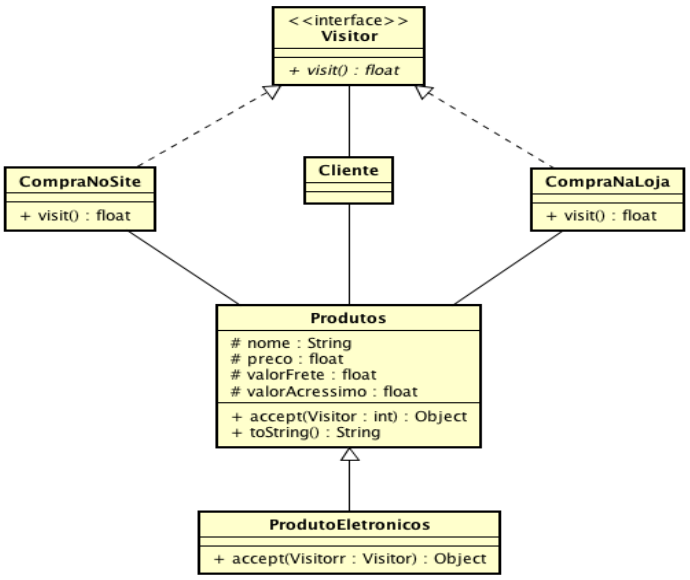
A classe cliente é classe sistema ou é propria classe que aceita o visitor e possui o elemento que contém o método **accept**. Nesta caso o elemento é uma classe base que é dada pelo elementos concretos do sistema ou seja os elemetos que realmente utilizamos na aplicação.O método **accept** ele recebe um parâmetro **visitor**. O **visitor** nada mais é o interface que reutilizará os visitors concreto. A interface **visitor** contém apenas um método **visit** que recebe um parâmetro o elemento concreto.

O funcionamento ocorre da seguinte maneira: o cliente é o elemento concreto aceita o visitor, desta forma quando chama o método **accept**, ele chama do visitor a sua função visit que pode alterar o estado do elemento concreto e implementar novas funcionalidades ou mesmo alterar seus atributos.

Com base do diagrama de classe mostrando acima vim a propor um exemplo simples sobre um sistema para calcular os valores dos preços dos produtos electrônicos. O sistema calcula os valores dos produtos comprados direta na loja e os produtos comprados pelo sites.

A figura a seguir mostra o diagrama de classe do exemplo prosposto pelo autor do artigo.

figura 2. *fonte: Isac Mendes*



O diagrama de classe do exemplo proposto tem uma interface e cinco classes. A classe CompraNoSite e CompraNaLoja implementam a interface Visitor e ao mesmo tempo associam-se com classe produtos que é uma classe abstrata. Para implementar classe produto temos que implementar uma classe concreta. A classe ProdutosElectronicos é uma classe concreta que foi especiliazada pela classe Produtos. Cliente é uma classe que utilizar a interface

Visiotor e a classe Produto. Na classe Produtos tem quastro atributos, todos atributos são protegidos e dois métodos públicos.O método **accept** acita como parâmetro o Visitor que retornaria um objeto. O método **accept** tem a sua função de aceitar qualquer visita feito por classes que implementam o método **visit**, neste caso o método **visit é usada através da interface que é implementada através das duas classes que são CompraNaLoja e CompraNoSite**

1.6 Código da implementação do exemplo proposto usando Netbean

Interaface **Visitor**

```
1  /**
2  * comentario
3  */
4  package visitor;
5
6  public interface Visitor {
7
8     public float visit (Produto produto);
9
10
11 }
```

Classe **Produto**

```
1  /**
2  * comentario
3  */
4  package visitor;
5
6  //classe Produto
7  public abstract class Produto {
8      //Declaracao de variavel
9      protected String nome;
10     protected float preco;
11     protected float valorFrete;
12     protected float acressimo;
13
14     //Construtor
15     public Produto(String nome, float preco, float
           valorFrete, float acressimo) {
16         this.nome = nome;
17         this.preco = preco;
18         this.valorFrete = valorFrete;
19         this.acressimo = acressimo;
20     }
21
22     public abstract Object accept (Visitor visitor);
23
```

```
24      public float getPreco (){
25          return this.preco;
26      }
27
28      public float getValorFrete() {
29          return valorFrete;
30      }
31
32      public float getAcressimo() {
33          return acressimo;
34      }
35
36      @Override
37      public String toString() {
38          return "Nome do Produto: "+ nome +  " \nPreco: R$
                " + preco +
39                   " \nValor Frete: R$ " + valorFrete + "\
                      nValor acressimo: R$ " + acressimo +"\n
                      ";
40      }
41  }
```

Classe **ProdutoElectro**

```
1
2   package visitor;
3
4
5   public class ProdutoElectro extends Produto {
6
7       public ProdutoElectro(String nome, float preco, float
            valorFrete, float acressimo) {
8           super(nome, preco, valorFrete, acressimo);
9       }
10
11      @Override
12      public Object accept (Visitor visitor){
13          return visitor.visit (this);
14      }
15
16
17  }
```

Classe **CompraNaLoja**

```
1
2   package visitor;
3
4   public class CompraNaLoja implements Visitor {
5
6       @Override
7       public float visit(Produto produto) {
8           return produto.getPreco();
```

```
9         }
10
11    }
```

Classe **CompraNoSite**

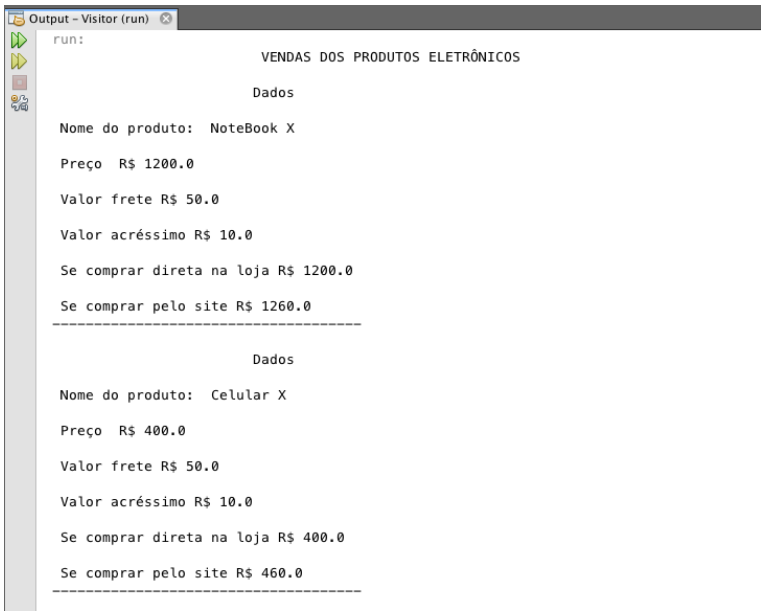
```
1
2  package visitor;
3
4  public class CompraNoSite implements Visitor {
5
6      @Override
7      public float visit(Produto produto) {
8          return produto.getPreco() + produto.getValorFrete
               () + produto.getAcressimo();
9
10     }
11
12 }
```

Classe Cliente/Main

```
1  package visitor;
2
3  public class Cliente{
4
5      /**
6       * @param args the command line arguments
7       */
8      public static void main(String[] args) {
9         Produto produto   =  new  ProdutoElectro ("
              NoteBook X ", 1200f, 50f, 10f);
10        Produto produto1  =  new ProdutoElectro (" Celular
              X ", 400f, 50f, 10f);
11        Produto produto2  =  new ProdutoElectro (" HD
              externo ", 80f, 15f, 10f);
12        Produto produto3  =  new ProdutoElectro (" Pendrive
               X ", 50f, 7f, 2f);
13        Produto produto4  =  new ProdutoElectro (" iPad ",
              800f, 50f, 10f);
14        Visitor valorLoja =  new CompraNaLoja ();
15        Visitor valorSite =  new CompraNoSite();
16
17
18         System.out.println(" \t\t\t VENDAS PRODUTOS
               ELETRONICOS\n ");
19
20
21         System.out.println(produto);
22         System.out.println(" Se comprar direta na loja R$
               "+ produto.accept(valorLoja)+"\n");
23         System.out.println(" Se comprar pelo site R$ "+
               produto.accept(valorSite)+ "");
24         System.out.println("
               ------------------------------------\n");
25
26
```

```
27
28          System.out.println(produto1);
29          System.out.println(" Se comprar direta na loja R$
               "+ produto1.accept(valorLoja)+"\n");
30          System.out.println(" Se comprar pelo site R$ "+
               produto1.accept(valorSite)+ "");
31          System.out.println("
               ---------------------------------------\n");
32
33
34
35
36          System.out.println(produto2);
37          System.out.println(" Se comprar direta na loja R$
               "+ produto2.accept(valorLoja)+"\n");
38          System.out.println(" Se comprar pelo site R$ "+
               produto2.accept(valorSite)+ "");
39          System.out.println("
               ---------------------------------------\n");
40      }
41  }
```

Saída



Considerações finais

Os padrões possibilitam através de uma linguagem clara e concisa, que os projetistas experientes transfiram os seus conhecimentos aos mais novos em um alto nível de abstração e assim facilitam o desenvolvimento e o reaproveitamento de código.

A implentação do exemplo acima usando os conceitos de Programação Orientada a Objetos como herança, polimorfismo e encapsulamento.

O exemplo proposto é embora simples más foi uma resolução pra um problema simples de como calcular os preços dos produtos de uma loja.

Referências

DEVMEDIA, Alessandro. “Conheça os Padrões de Projeto ”.2005.
Link: <https://www.devmedia.com.br/conheca-os-padroes-de-projeto/957>

MACORRATI.NET, José Carlos Macoratti, Padrões de Projeto - Design Patterns.
Link: <http://www.macoratti.net/vb_pd1.htm>

TREINAWEB, Kennedy. “Padrões de projeto: o que são e o que resolvem”. 22 de novembro de 2016.
<https://www.treinaweb.com.br/blog/padroes-de-projeto-o-que-sao-e-o-que-resolvem/>

SHARELATEX ONLINE
Link: <https://www.sharelatex.com/project/5b5d2656c9b2444a59451179>

Wikipedia
<https://pt.wikipedia.org/wiki/Padr%C3%A3o_de_projeto_de_software>