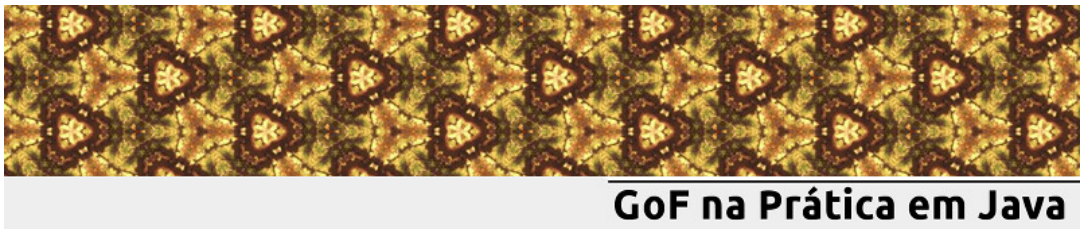
Bridge

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Desacopla uma abstração de sua implementação de tal modo que ambos possam variar independentemente.



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 : Desacoplar a interface da implementação para melhorar sua extensibilidade e ocultar os detalhes de implementação do cliente.

É usado quando : Se deseja evitar uma ligação permanente entre uma abstração e sua implementação. O que ocorre, por exemplo, quando a implementação deve ser selecionada ou trocada em tempo de execução. Tanto a abstração quanto sua implementação devem ser extensíveis por especialização. Neste caso, permite combinar estas diferentes abstrações e implementações e estendê-las independentemente.

2 Problema

Desacoplar a construção de publicações Livros pois o cliente não precisa tomar Conhecimento de métodos do tipo get/set.

3 Prévia Estrutura de Classes

Classe abstrata para manter os padrões das famílias de objeto:

Listagem 1: *Classe Abstrata Publicacao*

```
1 abstract class Publicacao {
2 }
```

Classe que implementa a Publicação de Livros e deve ser isolada do cliente:

Listagem 2: *Classe LivroImpl*

```
1 class LivroImpl extends Publicacao {
2
3   private String titulo;
4   private String autor;
```

```
5
6   public String getTitulo() {
7     return titulo;
8   }
9   public void setTitulo(String titulo) {
10    this.titulo = titulo;
11  }
12  public String getAutor() {
13    return autor;
14  }
15  public void setAutor(String autor) {
16    this.autor = autor;
17  }
18 }
```

4 Aplicação do Padrão

Adição de uma nova classe ponte para implementação da Publicação:

Listagem 3: *Classe Abstrata BridgePublicacao*

```
1  abstract class BridgePublicacao {
2    private Publicacao publicacao;
3
4    public BridgePublicacao(Publicacao publicacao) {
5      this.publicacao = publicacao;
6    }
7    public Publicacao getPublicacao() {
8      return publicacao;
9    }
10 }
```

Modificação na classe abstrata que controla a família das publicações para interligação da ponte:

Listagem 4: *Classe Abstrata Publicacao*

```
1  abstract class Publicacao {
2    private BridgePublicacao bridgePublicacao;
3
4    public BridgePublicacao getBridgePublicacao() {
5      return bridgePublicacao;
6    }
7    public void setBridgePublicacao(BridgePublicacao bridgePublicacao) {
8      this.bridgePublicacao = bridgePublicacao;
9    }
10 }
```

Classe auxiliar ponte para Livro:

Listagem 5: *Classe Livro*

```
1  class Livro extends BridgePublicacao {
2    public Livro(String titulo, String autor) {
3      super(new LivroImpl());
4      ((LivroImpl)getPublicacao()).setTitulo(titulo);
5      ((LivroImpl)getPublicacao()).setAutor(autor);
6    }
7    public String toString() {
8      return ((LivroImpl)getPublicacao()).getTitulo() + " de " +
9        ((LivroImpl)getPublicacao()).getAutor();
```

```
  }
}
```

Classe com um exemplo de uso pelo cliente:

Listagem 6: *Classe Cliente*

```
public class Cliente {
  public static void main(String[] args) {
    new Cliente().publicar();
  }
  public void publicar() {
    Livro livro = new Livro("Design Patterns", "GoF");
    System.out.println(livro);
  }
}
```

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.