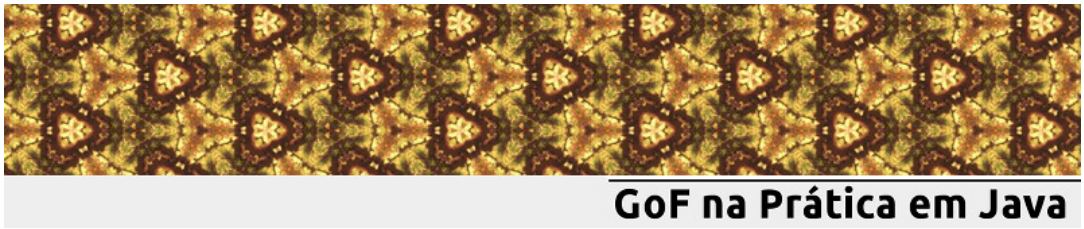
Chain of Responsability

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Evitar o acoplamento de um transmissor com uma requisição aos seus receptores, fazendo com que mais de um projeto, tenha a chance de manipular esta requisição.



1 Ficha do Padrão

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

Conhecimentos : Interface e Classes Concretas.

Consequências : Redução do acoplamento. Maior flexibilidade na associação de responsabilidades aos objetos. A recepção e tratamento da requisição não é garantida.

É usado quando : Mais de um objeto pode tratar uma requisição, e estes não são conhecidos a priori. O Objeto a tratar a requisição deve ser definido automaticamente. Emitir uma requisição para um dos vários objetos envolvidos, sem especificá-lo explicitamente. O conjunto de objetos que pode manipular uma requisição deve ser especificado dinamicamente.

2 Problema

O usuário precisa criar um conjunto circular de Redes que serão hospedadas em um WebServer, por exemplo: Linux e Microsoft. Porém se o WebServer estiver ocupado deve ser chamado outro.

3 Prévia Estrutura de Classes

Interface com a estrutura para as Redes:

Listagem 1: *Interface Rede*

```
1 interface Rede {
2   String tipoRede();
3 }
```

Classe concreta de implementação da rede Linux:

Listagem 2: *Classe Linux*

```
1 class Linux implements Rede {
2   public String tipoRede() {
3     return "Rede Linux";
4   }
5 }
```

Classe concreta de implementação da rede Microsoft:

Listagem 3: *Classe Microsoft*

```
class Microsoft implements Rede {
  public String tipoRede() {
    return "Rede Microsoft";
  }
}
```

4 Aplicação do Padrão

Classe para manter a estrutura circular do WebServer:

Listagem 4: *Classe WebServer*

```
class WebServer {
  private WebServer next;
  private String nome;

  public WebServer(String nome) {
    this.nome = nome;
  }
  public void add(WebServer nextWS) {
    if (next != null)
       next.add(nextWS);
    else
       next = nextWS;
  }
  public void wrapAround(WebServer firstWS) {
    if (next != null)
      next.wrapAround(firstWS);
    else
      next = firstWS;
  }
  public void handle(Rede rede) {
    if ((int)(Math.random()*4) % 2 == 0) {
      System.out.println("WEB Server Conectado - " + rede.tipoRede() + " " + next);
    } else {
      System.out.println("WEB Server " + next + " Ocupado... Tentar outro...");
      next.handle(rede);
    }
  }
  public String toString() {
    return "WebServer: " + nome;
  }
}
```

Classe com um exemplo de uso pelo cliente:

Listagem 5: *Classe Administrador*

```
public class Administrador {
  public static void main(String[] args) {
    new Administrador().gerenciar();
  }
  public void gerenciar() {
    WebServer ws1 = new WebServer("WS Principal");
    WebServer primeiro = new WebServer("WS1");
    ws1.add(primeiro);
```

```
9     ws1.add(new WebServer("WS2"));
10    ws1.wrapAround(primeiro);
11
12    Rede [] redes = { new Linux(), new Linux(), new Microsoft() };
13    for (Rede rede: redes) {
14      ws1.handle(rede);
15    }
16  }
17 }
```

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.