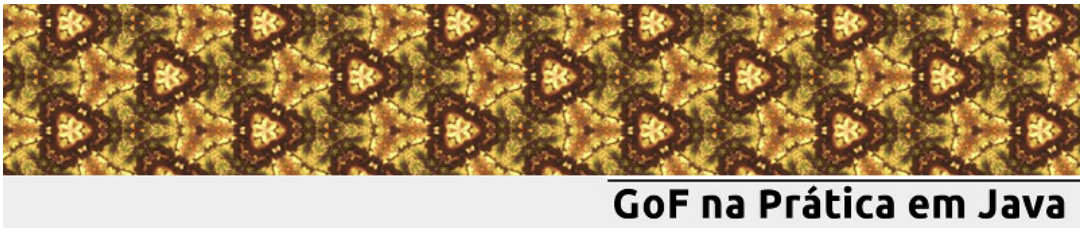
State

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Permitir a um objeto modificar seu comportamento quando seu estado interno se modifica, mudando, efetivamente, a classe do objeto. Distribuir uma operação para que cada classe represente um estado diferente.



1 Ficha do Padrão

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

Conhecimentos : Interface e Classes Concretas.

Consequências : Colocar todo o comportamento associado a um estado em um objeto particular. Realizar a transição de estados explicitamente.

É usado quando : O comportamento de um objeto depende de seu estado. Este deve ser modificado em tempo de execução conforme as mudanças desse estado.

2 Problema

Uma ação pode atingir os seguintes estados: Alta (valor acima de 20.0), média (valor entre 4.0 e 20.0) e baixa (valor abaixo de 4.0). Sempre deve oscilar entre estes estados: "Baixa" que pode variar para "Média" que pode variar para "Alta" que pode variar para "Média" que pode variar para "Baixa", garantir esta oscilação para que não ocorra saltos entre "Baixa" para "Alta" ou vice-versa.

3 Prévia Estrutura de Classes

Classe para representar uma Ação já construída pelo usuário:

Listagem 1: *Classe Acao*

```
1  class Acao {
2    private String empresa;
3    private float valor;
4
5    public Acao(String empresa) {
6      this.empresa = empresa;
7    }
8    public String toString() {
9      return empresa + " - " + valor;
10   }
```

```
  public void setValor(float valor) {
    this.valor = valor;
  }
}
```

4 Aplicação do Padrão

Interface básica para manter o padrão:

Listagem 2: *Interface State*

```
interface State {
  byte getNivel();
  boolean isValor(float valor);
}
```

Classe para representar o estado de baixa:

Listagem 3: *Classe AcaoEmBaixa*

```
class AcaoEmBaixa implements State {
  public byte getNivel() {
    return (byte)1;
  }
  public boolean isValor(float valor) {
    return valor < 4;
  }
}
```

Classe para representar o estado de média:

Listagem 4: *Classe AcaoNaMedia*

```
class AcaoNaMedia implements State {
  public byte getNivel() {
    return (byte)2;
  }
  public boolean isValor(float valor) {
    return valor >= 4 && valor =< 20;
  }
}
```

Classe para representar o estado de alta:

Listagem 5: *Classe AcaoEmAlta*

```
class AcaoEmAlta implements State {
  public byte getNivel() {
    return (byte)3;
  }
  public boolean isValor(float valor) {
    return valor > 20;
  }
}
```

Classe com as regras de variação:

Listagem 6: *Classe Variacao*

```
class Variacao {
  private State atual;
```

```
  public Variacao() {
    atual = new AcaoEmBaixa();
  }
  public boolean isTroca(float valor) {
    if (atual.isValor(valor)) {
      return true;
    }
    State novo = trocarAtual(valor);
    if (novo.getNivel() == atual.getNivel()+1 || novo.getNivel() == atual.getNivel()-1) {
      atual = novo;
      return true;
    }
    return false;
  }
  private State trocarAtual(float valor) {
    if (new AcaoEmAlta().isValor(valor))
      return new AcaoEmAlta();
    else if (new AcaoEmBaixa().isValor(valor))
      return new AcaoEmBaixa();
    else
      return new AcaoNaMedia();
  }
}
```

Modificação na classe que representa a Ação para sua adequação ao padrão:

Listagem 7: *Classe Acao*

```
class Acao {
  private String empresa;
  private float valor;
  private Variacao variacao = new Variacao();

  public Acao(String empresa) {
    this.empresa = empresa;
  }
  public String toString() {
    return empresa + " - " + valor;
  }
  public void setValor(float valor) {
    if (variacao.isTroca(valor)) {
      this.valor = valor;
    } else
      System.out.println("Troca de Valor Indevida");
  }
}
```

Classe exemplo com um teste da modificação dos valores:

Listagem 8: *Classe Bolsa*

```
public class Bolsa {
  public static void main(String [] args) {
    Acao padrao = new Acao("Patterns Inc.");
    padrao.setValor(1.0f);
    System.out.println(padrao);
    padrao.setValor(8.0f);
    System.out.println(padrao);
    padrao.setValor(25.0f);
    System.out.println(padrao);
    // Tentando alterar indevidamente o valor
```

```
11      padrao.setValor(1.0f);
12      System.out.println(padrao);
13    }
14  }
```

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.