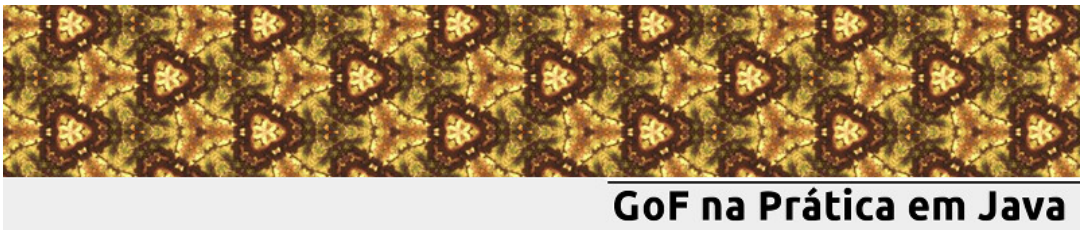
Observer

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Prover uma sincronização, coordenação e consistência entre objetos relacionados. Definir uma dependência "um para muitos" entre os objetos de modo que, caso o objeto mude seu estado, seus dependentes sejam notificados e atualizados.



1 Ficha do Padrão

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

Conhecimentos : Coleções, Classes Abstratas e Classes Concretas.

Consequências : Permite o acoplamento entre sujeito e o observador com suporte a comunicação broadcast.

É usado quando : Uma abstração possui dois aspectos que são dependentes. Encapsulando esses aspectos em objetos separados fará com que se possa variá-los e reusá-los independentemente. Um objeto deve poder notificar outros objetos sem assumir nada sobre eles.

2 Problema

O usuário planeja escrever classes para converter um determinado número inteiro para diversas bases numéricas. Porém deve informá-las quando o valor do número for modificado.

3 Prévia Estrutura de Classes

Classe Numero já construída pelo usuário:

Listagem 1: *Classe Numero*

```
1  class Numero {
2    private int valor;
3
4    public int hashCode() {
5      return valor;
6    }
7    public void setValor(int valor) {
8      this.valor = valor;
9    }
10 }
```

4 Aplicação do Padrão

Classe abstrata para a aplicação do padrão:

Listagem 2: *Classe Observer*

```
abstract class Observer {
  protected Numero num;
  public abstract void notificar();
}
```

Modificação na classe original do usuário para se adequar ao padrão:

Listagem 3: *Classe Numero*

```
class Numero {
  private java.util.List<Observer> observadores = new java.util.ArrayList<Observer>();
  private int valor;

  public int hashCode() {
    return valor;
  }
  public void setValor(int valor) {
    this.valor = valor;
    for (Observer obs : observadores) {
      obs.notificar();
    }
  }
  public void add(Observer obs){
    observadores.add(obs);
  }
}
```

Classe de conversão do número para base hexadecimal:

Listagem 4: *Classe HexObserver*

```
class HexObserver extends Observer {
  public HexObserver(Numero num) {
    this.num = num;
    this.num.add(this);
  }
  public void notificar() {
    System.out.println(num.hashCode() + " em hexadecimal: " +
    Integer.toHexString(num.hashCode()));
  }
}
```

Classe de conversão do número para base octal:

Listagem 5: *Classe OctObserver*

```
class OctObserver extends Observer{
  public OctObserver(Numero num){
    this.num = num;
    this.num.add(this);
  }
  public void notificar() {
    System.out.println(num.hashCode() + " em octal: " + Integer.toOctalString(num.hashCode()));
  }
}
```

Classe exemplo para uma Análise com as expressões:

Listagem 6: *Classe Cliente*

```
public class Cliente {
  public static void main(String[] args) {
    new Cliente().converter();
  }
  public void converter() {
    Numero num = new Numero();
    new HexObserver(num);
    new OctObserver(num);
    num.setValor(15);
    num.setValor(30);
    num.setValor(12);
  }
}
```

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.