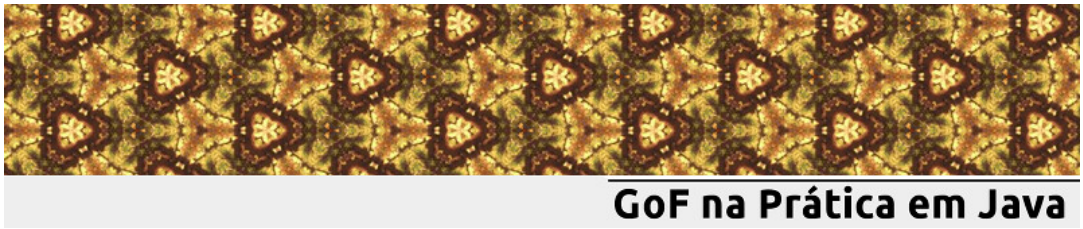
Strategy

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Definir uma família de algoritmos, encapsulando cada um deles, e torna a escolha de qual deles deve ser utilizado flexível. Desligar os algoritmos dos clientes que os usa. Encapsula uma operação fazendo com que as implementações sejam inter cambiais.



1 Ficha do Padrão

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

Conhecimentos : Interface e Classes Concretas.

Consequências : Famílias de algoritmos relacionados como uma alternativa a subclasse. Eliminar comandos condicionais.

É usado quando : Prover uma maneira de configurar uma classe com vários comportamentos possíveis. É necessário em diferentes variações de um algoritmo que são implementadas como uma classe hierárquica.

2 Problema

Com base na classe MeuArray o usuário deseja formar estratégias para mostrar esse de diferentes maneiras, porém sem criar vários métodos para chamadas diferentes.

3 Prévia Estrutura de Classes

Classe que representa o Array de tipos inteiros já construída pelo usuário:

Listagem 1: *Classe MeuArray*

```
1  class MeuArray {
2    private int[] array;
3    private int tam;
4
5    public MeuArray(int tam) {
6      array = new int[tam];
7    }
8    public void addValorEmPos(int valor, int pos) {
9      array[pos] = valor;
10   }
11 }
```

4 Aplicação do Padrão

Interface básica para construção das estratégias de formatação:

Listagem 2: *Interface ArrayFormata*

```
interface ArrayFormata {
  public void mostrar(int[] arr);
}
```

Classe para a estratégia do formato padrão:

Listagem 3: *Classe FormatoPadrao*

```
class FormatoPadrao implements ArrayFormata {
  public void mostrar(int[] arr) {
    System.out.print("{ ");
    for (int n = 0; n < arr.length - 1; n++)
      System.out.print(arr[n] + ", ");
    System.out.println(arr[arr.length-1] + " }");
  }
}
```

Classe para a estratégia do formato posicional:

Listagem 4: *Classe FormatoPosicional*

```
class FormatoPosicional implements ArrayFormata {
  public void mostrar(int[] arr) {
    for (int i = 0; i < arr.length; i++)
      System.out.println("Arr[ " + i + " ] = " + arr[i]);
  }
}
```

Mudança na classe do usuário para refletir o padrão de estratégia:

Listagem 5: *Classe MeuArray*

```
class MeuArray {
  private int[] array;
  private int tam;
  ArrayFormata formata;

  public MeuArray(int tam) {
    array = new int[tam];
  }
  public void addValorEmPos(int valor, int pos) {
    array[pos] = valor;
  }
  public void setEstrategia(ArrayFormata st) {
    formata = st;
    formata.mostrar(array);
  }
}
```

Classe exemplo para o uso de diferentes formatações:

Listagem 6: *Classe Show*

```
public class Show {
  public static void main (String[] arg) {
    new Show().padronizar();
```

```
  }
  public void padronizar() {
    MeuArray m = new MeuArray(10);
    m.addValorEmPos(6, 1);
    m.addValorEmPos(8, 0);
    m.addValorEmPos(1, 4);
    m.addValorEmPos(7, 9);
    System.out.println("No formato corrente:");
    m.setEstrategia(new FormatoPadrao());
    System.out.println("No formato posicional:");
    m.setEstrategia(new FormatoPosicional());
  }
}
```

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.