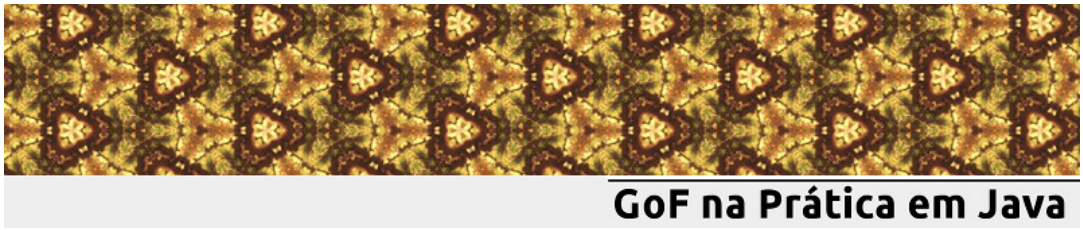
Interpreter

Fernando Anselmo
GoF na Prática em Java

Função deste Padrão

Dada uma linguagem, definir uma representação para sua gramática junto com um interpretador que usa a representação para interpretar sentenças na linguagem.



1 Ficha do Padrão

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

Conhecimentos : Interface e Classes Concretas.

Consequências : Muito fácil de mudar, implementar e extender uma gramática. Adicionar novos caminhos para interpretar expressões.

É usado quando : A gramática é simples e a eficiência não é um ponto crítico.

2 Problema

Nosso usuário está construindo um programa de inteligência artificial, porém ele precisa criar interpretações para os questionamento "E" e "OU".

3 Prévia Estrutura de Classes

Interface com a estrutura para construção das expressões:

Listagem 1: *Interface Expressao*

```
1 interface Expressao {
2   public boolean entender(String contexto);
3 }
```

Classe para a expressão final já construída pelo usuário:

Listagem 2: *Classe ExpressaoFinal*

```
1 class ExpressaoFinal implements Expressao {
2   private String dado;
3
4   public ExpressaoFinal(String dado) {
5     this.dado = dado;
6   }
```

```
  public boolean entender(String contexto) {
    return (contexto.contains(dado));
  }
}
```

4 Aplicação do Padrão

Classe para a interpretação da expressão “E”:

Listagem 3: *Classe ExpressaoE*

```
class ExpressaoE implements Expressao {
  private Expressao expr1 = null;
  private Expressao expr2 = null;

  public ExpressaoE(Expressao expr1, Expressao expr2) {
    this.expr1 = expr1;
    this.expr2 = expr2;
  }
  public boolean entender(String contexto) {
    return expr1.entender(contexto) && expr2.entender(contexto);
  }
}
```

Classe para a interpretação da expressão “OU”:

Listagem 4: *Classe ExpressaoOu*

```
class ExpressaoOu implements Expressao {
  private Expressao expr1 = null;
  private Expressao expr2 = null;

  public ExpressaoOu(Expressao expr1, Expressao expr2) {
    this.expr1 = expr1;
    this.expr2 = expr2;
  }

  public boolean entender(String contexto) {
    return expr1.entender(contexto) || expr2.entender(contexto);
  }
}
```

Classe com um exemplo de uso pelo cliente para análise das expressões:

Listagem 5: *Classe Analise*

```
public class Analise {
  public Expressao carregarAutores(String autor1, String autor2) {
    return new ExpressaoOu(new ExpressaoFinal(autor1), new ExpressaoFinal(autor2));
  }
  public Expressao carregarLinguagem(String pessoa) {
    return new ExpressaoE(new ExpressaoFinal(pessoa), new ExpressaoFinal("Java"));
  }
  public static void main(String[] args) {
    new Analise().verificar();
  }
  public void verificar() {
    Expressao autores = carregarAutores("Fernando", "Anselmo");
    Expressao conhecimento = carregarLinguagem("Fernando");
    // Verificar
```

```
15      System.out.println("Fernando foi o Autor? " + autores.entender("Fernando Autor"));
16      System.out.println("Fernando sabe Java? " + conhecimento.entender("Fernando Java"));
17    }
18  }
```

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.