Interpreter Pattern

An Interpreter Pattern says that **"to define a representation of grammar of a given language, along with an interpreter that uses this representation to interpret sentences in the language".**

Basically the Interpreter pattern has limited area where it can be applied. We can discuss the Interpreter pattern only in terms of formal grammars but in this area there are better solutions that is why it is not frequently used.

This pattern can applied for parsing the expressions defined in simple grammars and sometimes in simple rule engines.

SQL Parsing uses interpreter design pattern.

Advantage of Interpreter Pattern

* It is easier to change and extend the grammar.
* Implementing the grammar is straightforward.

Usage of Interpreter pattern:

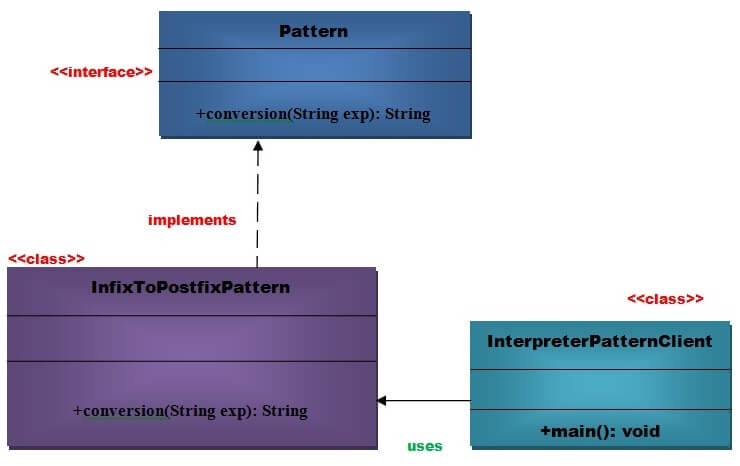
It is used:

* When the grammar of the language is not complicated.
* When the efficiency is not a priority.

Example of Interpreter Pattern

Let's understand the example of Interpreter Pattern by the above UML diagram.

UML for Interpreter Pattern:



Implementation of above UML

Step 1

Create a **Pattern** interface.

1. **public** **interface** Pattern {
2. **public** String conversion(String exp);
3. }

Step 2

Create a **InfixToPostfixPattern** class that will allow what kind of pattern you want to convert.

*File: InfixToPostfixPattern.java*

1. **import** java.util.Stack;
2. **public** **class** InfixToPostfixPattern **implements** Pattern{
3. @Override
4. **public** String conversion(String exp) {
5. **int** priority = 0;// for the priority of operators.
6. String postfix = "";
7. Stack<Character> s1 = **new** Stack<Character>();
8. **for** (**int** i = 0; i < exp.length(); i++)
9. {
10. **char** ch = exp.charAt(i);
11. **if** (ch == '+' || ch == '-' || ch == '\*' || ch == '/'||ch=='%')
12. {
13. // check the precedence
14. **if** (s1.size() <= 0)
15. s1.push(ch);
16. }
17. **else**
18. {
19. Character chTop = (Character) s1.peek();
20. **if** (chTop == '\*' || chTop == '/')
21. priority = 1;
22. **else**
23. priority = 0;
24. **if** (priority == 1)
25. {
26. **if** (ch == '\*' || ch == '/'||ch=='%')
27. {
28. postfix += s1.pop();
29. i--;
30. }
31. **else**
32. { // Same
33. postfix += s1.pop();
34. i--;
35. }
36. }
37. **else**
38. {
39. **if** (ch == '+' || ch == '-')
40. {
41. postfix += s1.pop();
42. s1.push(ch);
43. }
44. **else**
45. s1.push(ch);
46. }
47. }
48. }
49. **else**
50. {
51. postfix += ch;
52. }
53. }
54. **int** len = s1.size();
55. **for** (**int** j = 0; j < len; j++)
56. postfix += s1.pop();
57. **return** postfix;
59. }
60. }// End of the InfixToPostfixPattern class.

Step 3

Create a **InterpreterPatternClient** class that will use InfixToPostfix Conversion.

*File: InterpreterPatternClient.java*

1. **public** **class** InterpreterPatternClient {
2. **public** **static** **void** main(String[] args)
3. {
4. String infix = "a+b\*c";
6. InfixToPostfixPattern ip=**new** InfixToPostfixPattern();
8. String postfix = ip.conversion(infix);
9. System.out.println("Infix:   " + infix);
10. System.out.println("Postfix: " + postfix);
11. }
12. }

Output

1. Infix:   a+b\*c
2. Postfix: abc\*+