

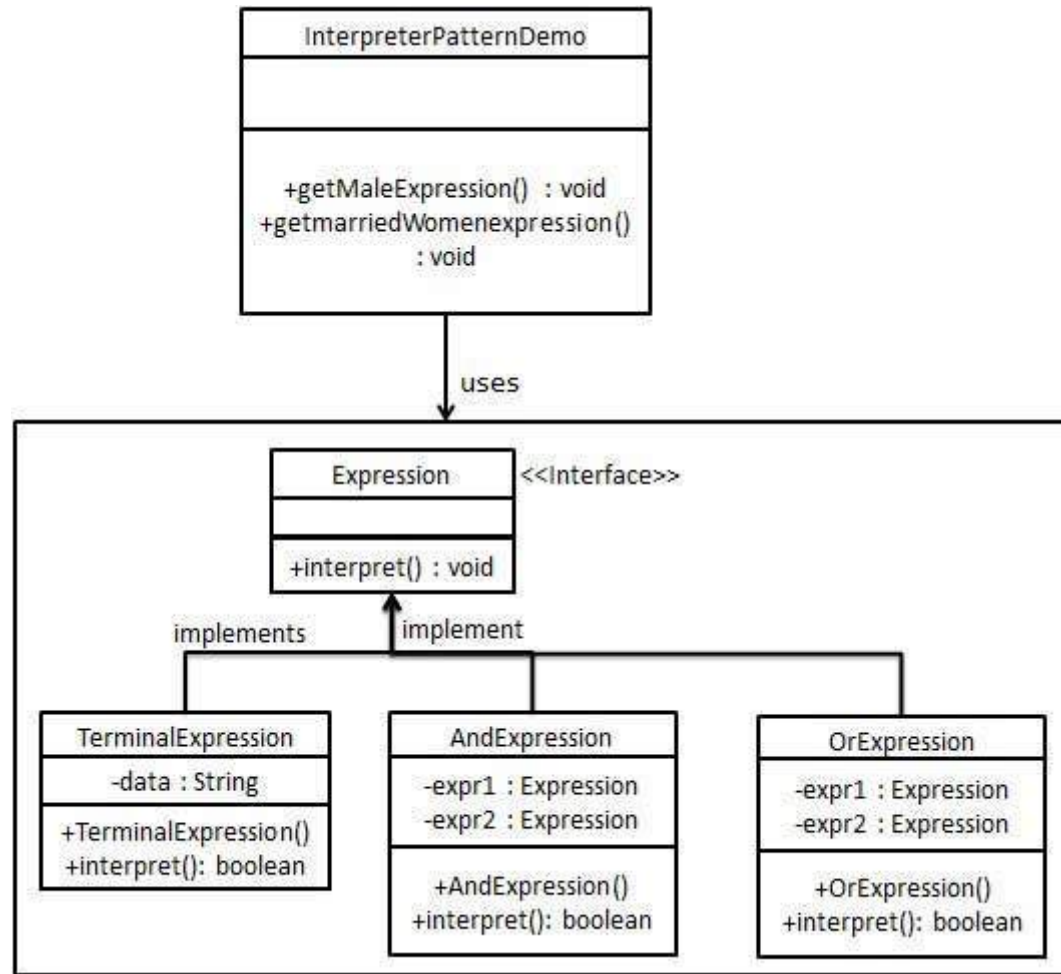
Design Patterns - Interpreter Pattern

Interpreter pattern provides a way to evaluate language grammar or expression. This type of pattern comes under behavioral pattern. This pattern involves implementing an expression interface which tells to interpret a particular context. This pattern is used in SQL parsing, symbol processing engine etc.

Implementation

We are going to create an interface *Expression* and concrete classes implementing the *Expression* interface. A class *TerminalExpression* is defined which acts as a main interpreter of context in question. Other classes *OrExpression*, *AndExpression* are used to create combinational expressions.

InterpreterPatternDemo, our demo class, will use *Expression* class to create rules and demonstrate parsing of expressions.



Step 1

Create an expression interface.

Expression.java

```

public interface Expression {
    public boolean interpret(String context);
}
  
```

Step 2

Create concrete classes implementing the above interface.

TerminalExpression.java

```
public class TerminalExpression implements Expression {

    private String data;

    public TerminalExpression(String data){
        this.data = data;
    }

    @Override
    public boolean interpret(String context) {

        if(context.contains(data)){
            return true;
        }
        return false;
    }
}
```

OrExpression.java

```
public class OrExpression implements Expression {

    private Expression expr1 = null;
    private Expression expr2 = null;

    public OrExpression(Expression expr1, Expression expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }
}
```

```

    }

    @Override
    public boolean interpret(String context) {
        return expr1.interpret(context) || expr2.interpret(context);
    }
}

```

AndExpression.java

```

public class AndExpression implements Expression {

    private Expression expr1 = null;
    private Expression expr2 = null;

    public AndExpression(Expression expr1, Expression expr2) {
        this.expr1 = expr1;
        this.expr2 = expr2;
    }

    @Override
    public boolean interpret(String context) {
        return expr1.interpret(context) && expr2.interpret(context);
    }
}

```

Step 3

InterpreterPatternDemo uses *Expression* class to create rules and then parse them.

InterpreterPatternDemo.java

```

public class InterpreterPatternDemo {

```

```

//Rule: Robert and John are male
public static Expression getMaleExpression(){
    Expression robert = new TerminalExpression("Robert");
    Expression john = new TerminalExpression("John");
    return new OrExpression(robert, john);
}

//Rule: Julie is a married women
public static Expression getMarriedWomanExpression(){
    Expression julie = new TerminalExpression("Julie");
    Expression married = new TerminalExpression("Married");
    return new AndExpression(julie, married);
}

public static void main(String[] args) {
    Expression isMale = getMaleExpression();
    Expression isMarriedWoman = getMarriedWomanExpression();

    System.out.println("John is male? " + isMale.interpret("John"));
    System.out.println("Julie is a married women? " + isMarriedWoman.interpret("Married Julie"));
}
}

```

Step 4

Verify the output.

```

John is male? true
Julie is a married women? true

```