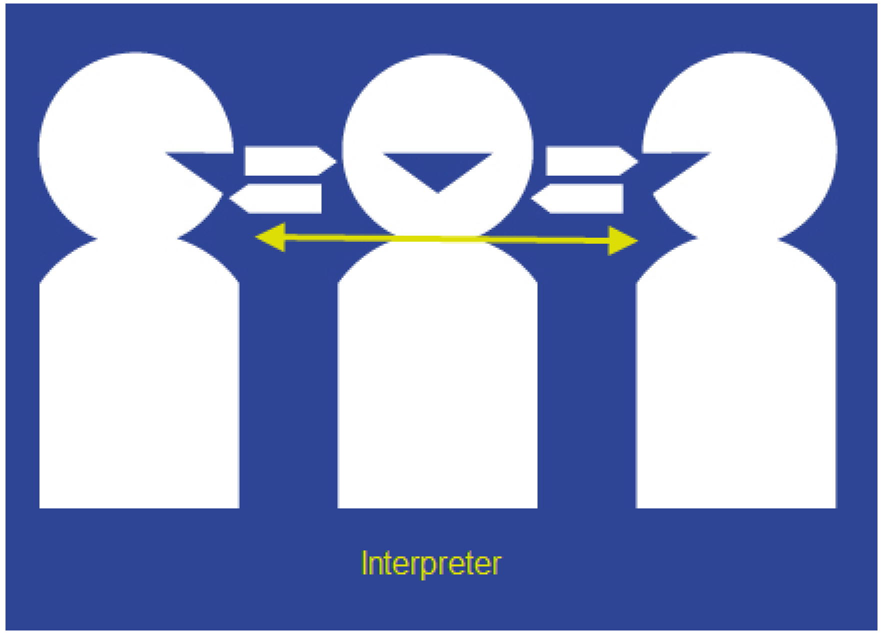
解释器模式Interpreter

波波老师~研发总监/资深架构师

波波微课

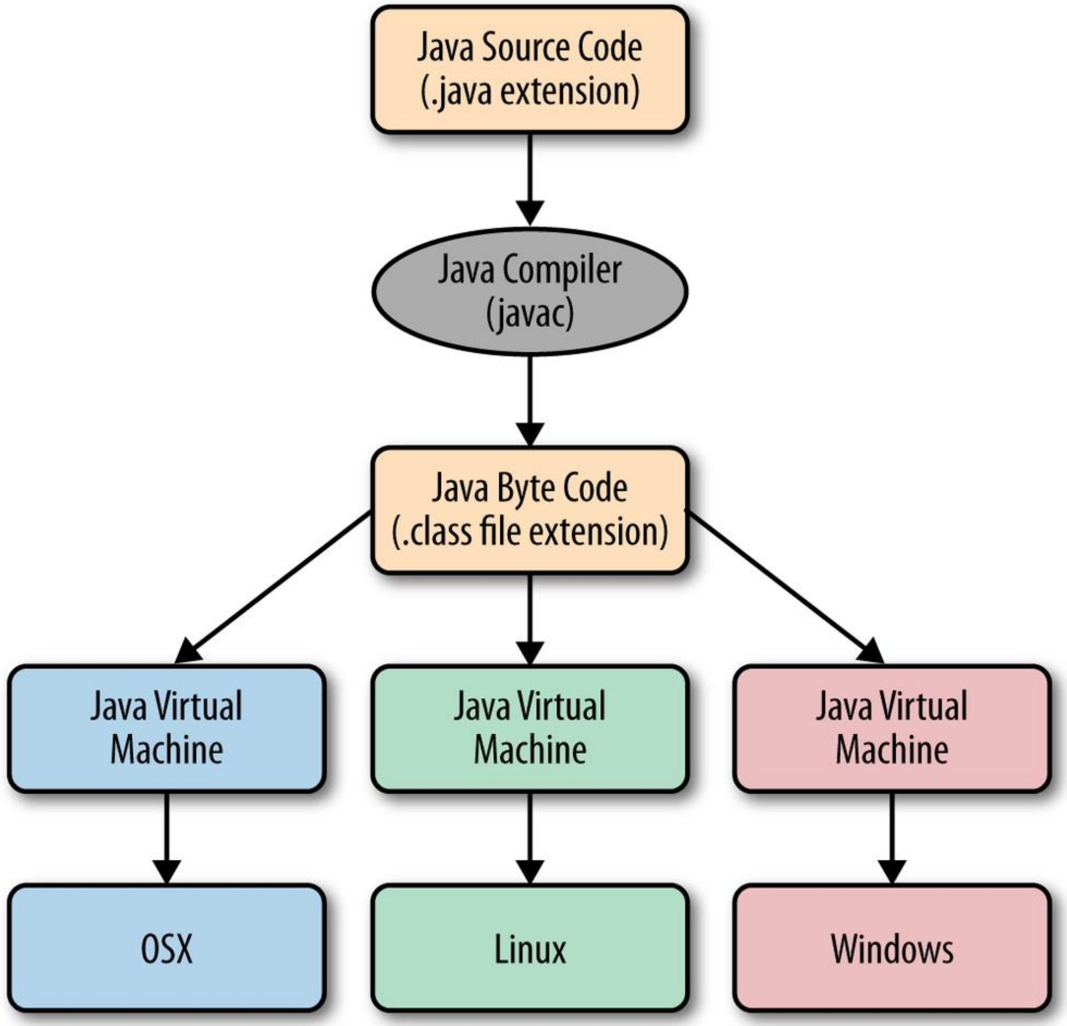
spring2go.com



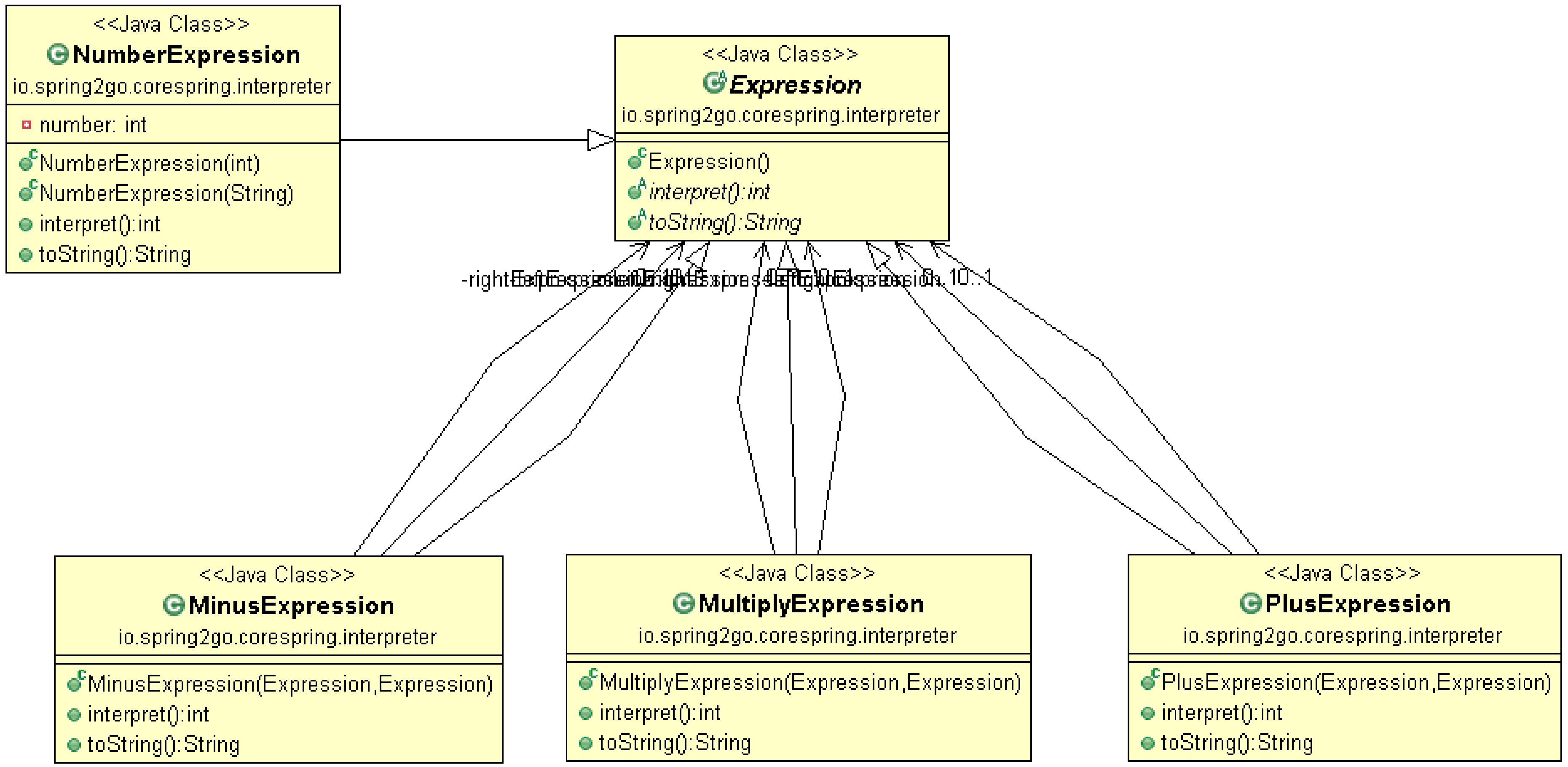
定义

- 给定一个语言和其文法表示，解释器能够翻译该语言中的句子
- 适用于简单文法
- 解释/翻译/转换
- 高级模式

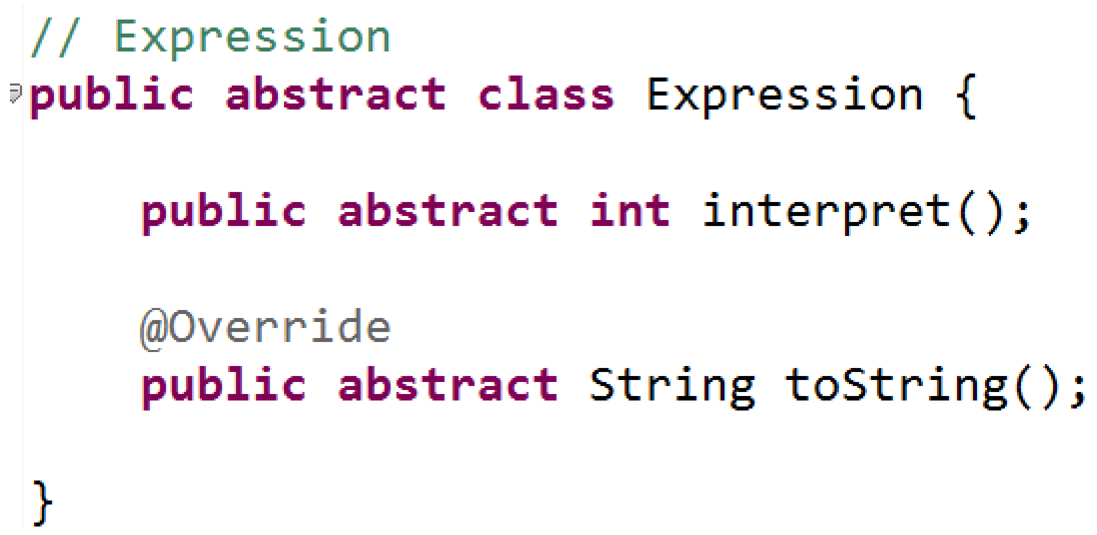
经典案例JVM解释器


Java Source Code
(.java extension)
Java Compiler
(javac)
Java Byte Code
(.class file extension)
Java Virtual Machine
Java Virtual Machine
Java Virtual Machine
OSX
Linux
Windows

简单算术解释器

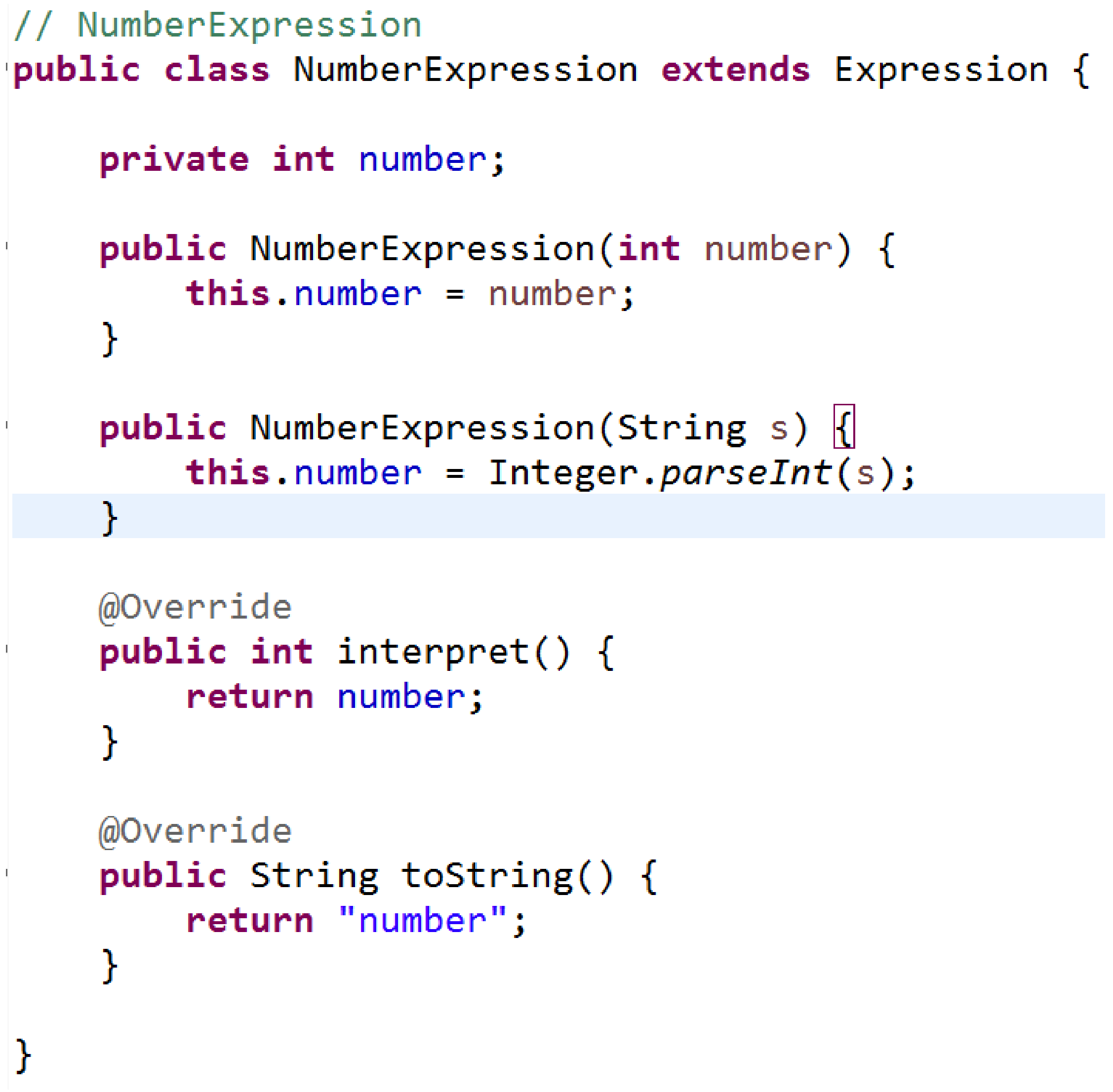

<<Java Class>>
NumberExpression
io.spring2go.corespring.interpreter
number: int
NumberExpression(int)
NumberExpression(String)
interpret():int
toString():String
<<Java Class>>
Expression
io.spring2go.corespring.interpreter
Expression()
interpret():int
toString():String
0..1
<<Java Class>>
MinusExpression
io.spring2go.corespring.interpreter
MinusExpression(Expression,Expression)
interpret():int
toString():String
<<Java Class>>
MultiplyExpression
io.spring2go.corespring.interpreter
MultiplyExpression(Expression,Expression)
interpret():int
toString():String
<<Java Class>>
PlusExpression
io.spring2go.corespring.interpreter
PlusExpression(Expression,Expression)
interpret():int
toString():String

代码~Expression



```java
// Expression
public abstract class Expression {

    public abstract int interpret();

    @Override
    public abstract String toString();

}
```

代码~NumberExpression

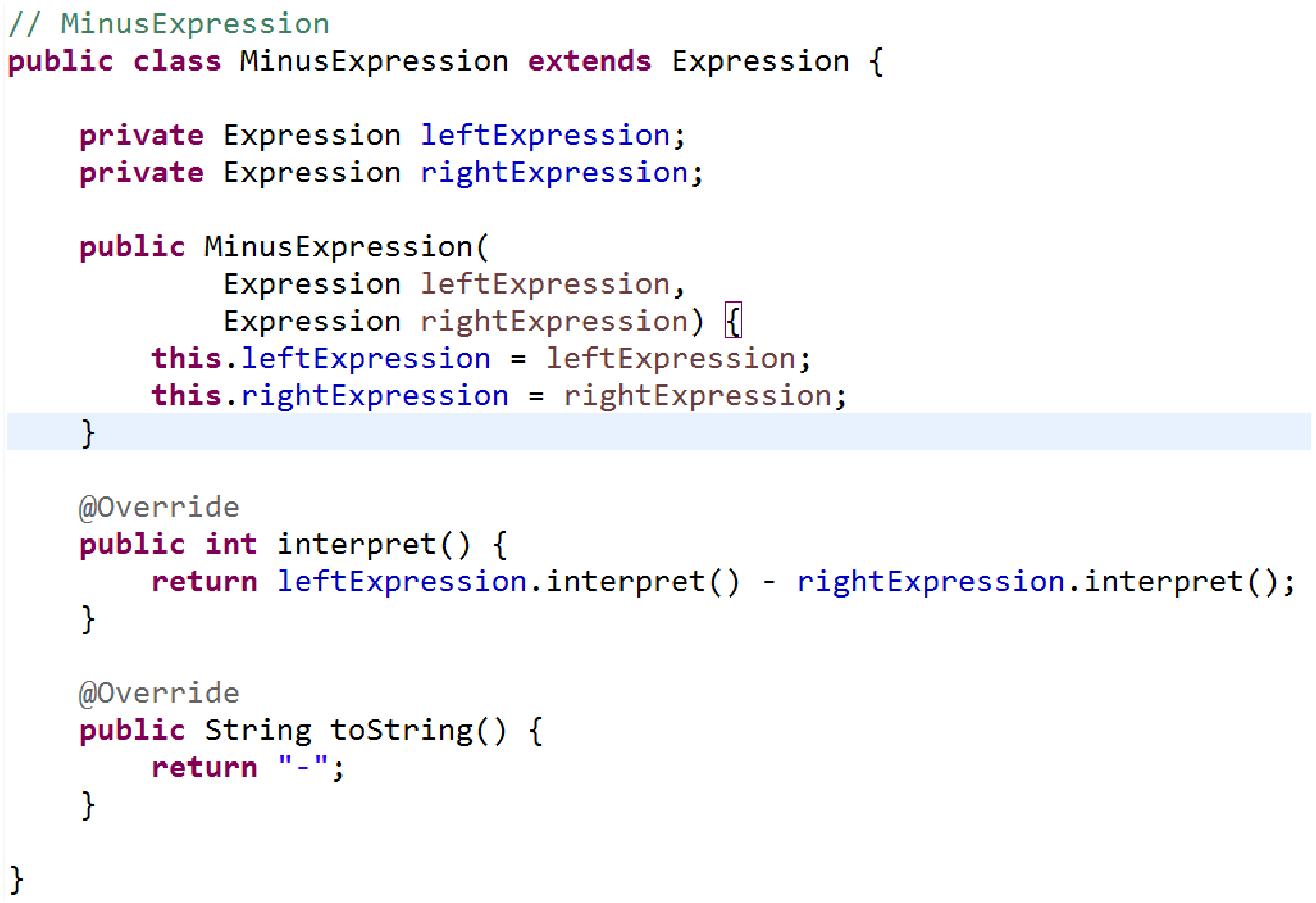


```java
// NumberExpression
public class NumberExpression extends Expression {

    private int number;

    public NumberExpression(int number) {
        this.number = number;
    }

    public NumberExpression(String s) {
        this.number = Integer.parseInt(s);
    }

    @Override
    public int interpret() {
        return number;
    }

    @Override
    public String toString() {
        return "number";
    }

}
```

代码~PlusExpression

```java
// PlusExpression
public class PlusExpression extends Expression {

    private Expression leftExpression;
    private Expression rightExpression;

    public PlusExpression(
            Expression leftExpression,
            Expression rightExpression) {
        this.leftExpression = leftExpression;
        this.rightExpression = rightExpression;
    }

    @Override
    public int interpret() {
        return leftExpression.interpret() + rightExpression.interpret();
    }

    @Override
    public String toString() {
        return "+";
    }

}
```

代码~MinusExpression



```java
// MinusExpression
public class MinusExpression extends Expression {

    private Expression leftExpression;
    private Expression rightExpression;

    public MinusExpression(
            Expression leftExpression,
            Expression rightExpression) {
        this.leftExpression = leftExpression;
        this.rightExpression = rightExpression;
    }

    @Override
    public int interpret() {
        return leftExpression.interpret() - rightExpression.interpret();
    }

    @Override
    public String toString() {
        return "-";
    }

}
```

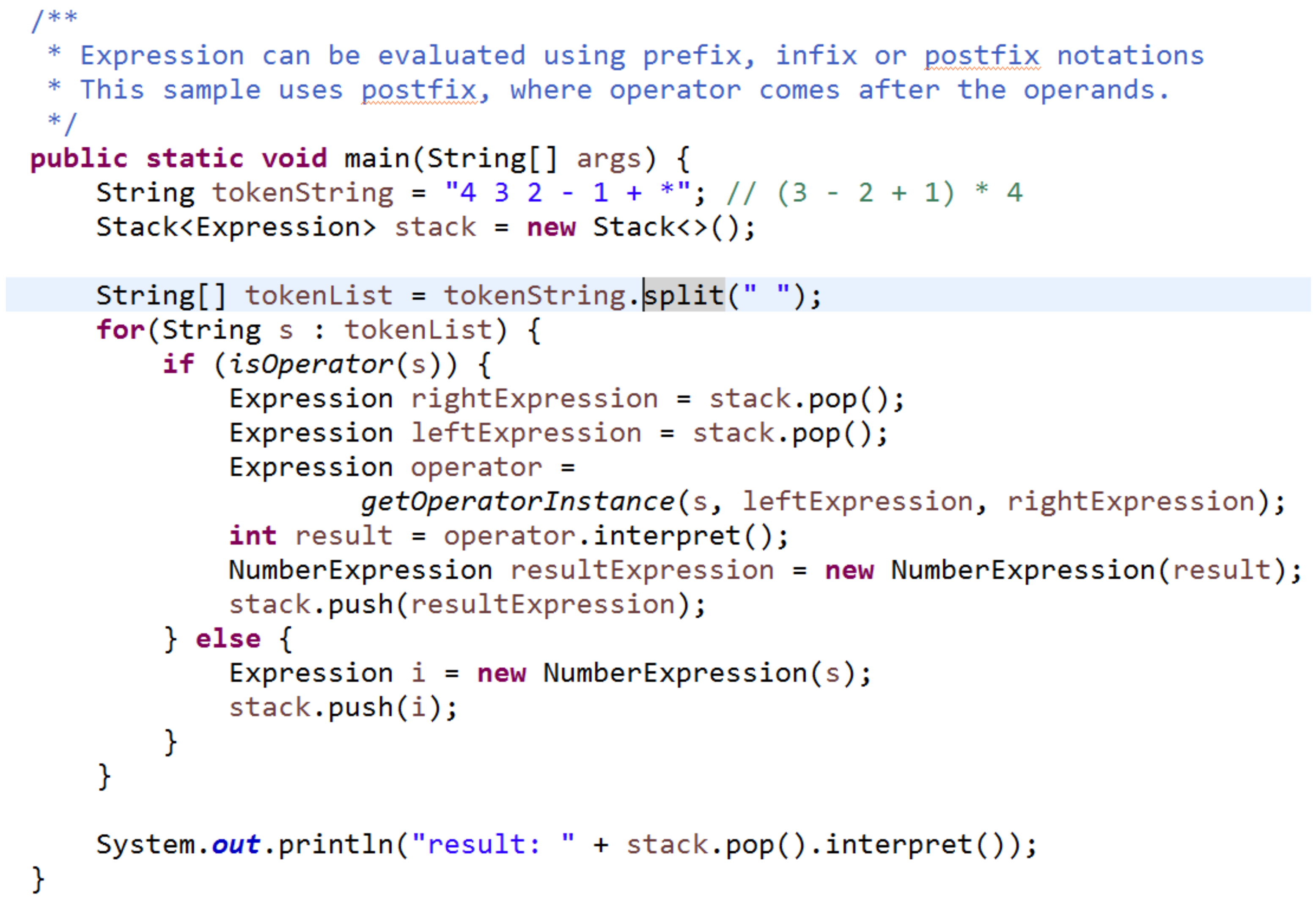
代码~MultiplyExpression

```java
// MultiplyExpression
public class MultiplyExpression extends Expression {

    private Expression leftExpression;
    private Expression rightExpression;

    public MultiplyExpression(
            Expression leftExpression,
            Expression rightExpression) {
        this.leftExpression = leftExpression;
        this.rightExpression = rightExpression;
    }

    @Override
    public int interpret() {
        return leftExpression.interpret() * rightExpression.interpret();
    }

    @Override
    public String toString() {
        return "*";
    }

}
```

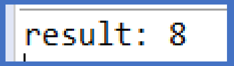
代码~工具方法

```java
public static boolean isOperator(String s) {
    return s.equals("+") || s.equals("-") || s.equals("*");
}

// Get expression for string
public static Expression getOperatorInstance(
        String s, Expression left, Expression right) {
    switch (s) {
    case "+":
        return new PlusExpression(left, right);
    case "-":
        return new MinusExpression(left, right);
    case "*":
        return new MultiplyExpression(left, right);
    default:
        return new MultiplyExpression(left, right);
    }
}
```

代码~解释器主程序



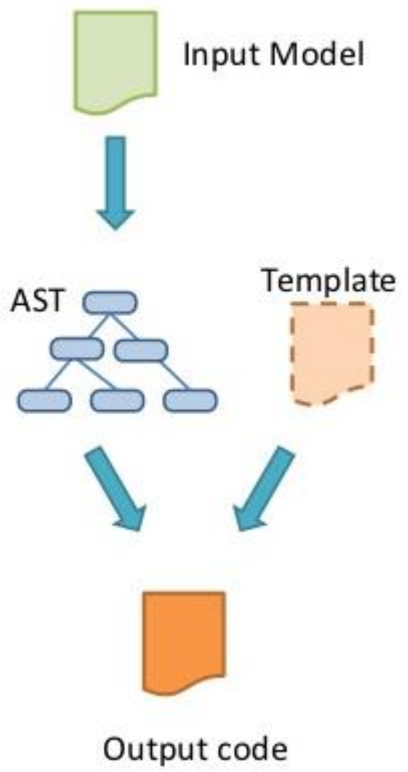
```java
/**
 * Expression can be evaluated using prefix, infix or postfix notations
 * This sample uses postfix, where operator comes after the operands.
 */
public static void main(String[] args) {
    String tokenString = "4 3 2 - 1 + *"; // (3 - 2 + 1) * 4
    Stack<Expression> stack = new Stack<>();

    String[] tokenList = tokenString.split(" ");
    for(String s : tokenList) {
        if (isOperator(s)) {
            Expression rightExpression = stack.pop();
            Expression leftExpression = stack.pop();
            Expression operator =
                    getOperatorInstance(s, leftExpression, rightExpression);
            int result = operator.interpret();
            NumberExpression resultExpression = new NumberExpression(result);
            stack.push(resultExpression);
        } else {
            Expression i = new NumberExpression(s);
            stack.push(i);
        }
    }

    System.out.println("result: " + stack.pop().interpret());
}
```



```
result: 8
```

应用

- java.util.Pattern
- java.text.Normalizer
- java.text.Format子类
- Spring Expression Language(SpEL)
- 代码生成器



课后练习

- 修改代码支持除法(/)操作

参考

- Interpreter Design Pattern in Java
  - https://www.journaldev.com/1635/interpreter-design-pattern-java
- Design Patterns – Interpreter Pattern
  - http://www.tutorialspoint.com/design_pattern/interpreter_pattern.htm

代码

- https://github.com/spring2go/core-spring-patterns

波波微课
spring2go.com

波波微课

spring2go.com