学号 E21514033 专业 软件工程 姓名

实验日期 **6.7**  教师签字 成绩

实验报告

【实验名称】 解释器模式编程实现

【实验目的】

1. 熟悉解释器模式的使用
2. 能够给定表达式来输出结果。

【实验原理】

1. 给定一个语言，定义它的文法的一种表示，并定义一个解释器，这个解释器使用该表示来解释语言中的句子。

2. 通过实现每个字符的处理来实现整个字符串的处理。

【实验内容】

表达式总类：

Expression类：

其中的参数in，用于传入最外层对象。然后通过对比，如果时最外层那就打印结果。

abstract public class Expression {

abstract public double Interpret(Expression in);

}

浮点数类：

DoubleExpression类：

只需要简单的打印自身并向上传值。

public class DoubleExpression extends Expression{

double result;

DoubleExpression(double hmc){

result = hmc;

}

@Override

public double Interpret(Expression in) {

System.out.print(result);

return result;

}

}

操作运算符类：

OpExpression:

继承与Expression，定义了left和right。

public class OpExpression extends Expression {

Expression left;

Expression right;

@Override

public double Interpret(Expression in) {

return 0;

}

}

加法类：

AddExpression:

通过左边自行调用函数处理，右边自行调用函数处理。

通过判断，传入的最外层对象来决定是否输出最后的等于号。

public class AddExpression extends OpExpression{

AddExpression(Expression left,Expression right){

this.left = left;

this.right = right;

}

AddExpression(){}

@Override

public double Interpret(Expression in) {

double left = this.left.Interpret(in);

System.out.print(" + ");

double right = this.right.Interpret(in);

double temp = left + right;

if (this == in){

System.out.print(" = ");

System.out.print(temp);

}

return temp;

}

}

括号类：

public class BracketExpression extends OpExpression {

BracketExpression(){}

BracketExpression(Expression in){

this.left = in;

this.right = in;

}

@Override

public double Interpret(Expression in) {

System.out.print("(");

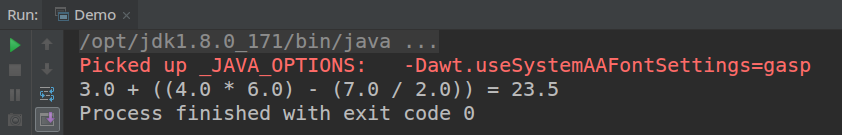
double inter = this.left.Interpret(in);

System.out.print(")");

return inter;

}

}



实验截图：

【小结或讨论】

1. 在函数中同时实现输出和处理。
2. 一开始格式化的在内层表达式两边输出括号，后通过括号类来控制括号输出。
3. 一开始通过函数重载实现区分最外层，后来直接传入最外层对象。使用判断来输出最外层的等于号。