**import** java.util.Stack;

/\*\*

\* 利用栈，进行四则运算的类

\* 用两个栈来实现算符优先，一个栈用来保存需要计算的数据numStack,一个用来保存计算优先符priStack

\*

\* 基本算法实现思路为：用当前取得的运算符与priStack栈顶运算符比较优先级：若高于，则因为会先运算，放入栈顶；

\* 若等于，因为出现在后面，所以会后计算，所以栈顶元素出栈，取出操作数运算；

\* 若小于，则同理，取出栈顶元素运算，将结果入操作数栈。各个优先级'(' > '\*' = '/' > '+' = '-' > ')'

\*

\*/

**public** **class** Operate {

**private** Stack<Character> priStack = **new** Stack<Character>();// 操作符栈

**private** Stack<Integer> numStack = **new** Stack<Integer>();;// 操作数栈

/\*\*

\* 传入需要解析的字符串，返回计算结果(此处因为时间问题，省略合法性验证)

\* **@param** str 需要进行技术的表达式

\* **@return** 计算结果

\*/

**public** **int** caculate(String str) {

// 1.判断string当中有没有非法字符

String temp;// 用来临时存放读取的字符

// 2.循环开始解析字符串，当字符串解析完，且符号栈为空时，则计算完成

StringBuffer tempNum = **new** StringBuffer();// 用来临时存放数字字符串(当为多位数时)

StringBuffer string = **new** StringBuffer().append(str);// 用来保存，提高效率

**while** (string.length() != 0) {

temp = string.substring(0, 1);

string.delete(0, 1);

// 判断temp，当temp为操作符时

**if** (!isNum(temp)) {

// 1.此时的tempNum内即为需要操作的数，取出数，压栈，并且清空tempNum

**if** (!"".equals(tempNum.toString())) {

// 当表达式的第一个符号为括号

**int** num = Integer.*parseInt*(tempNum.toString());

numStack.push(num);

tempNum.delete(0, tempNum.length());

}

// 用当前取得的运算符与栈顶运算符比较优先级：若高于，则因为会先运算，放入栈顶；若等于，因为出现在后面，所以会后计算，所以栈顶元素出栈，取出操作数运算；

// 若小于，则同理，取出栈顶元素运算，将结果入操作数栈。

// 判断当前运算符与栈顶元素优先级，取出元素，进行计算(因为优先级可能小于栈顶元素，还小于第二个元素等等，需要用循环判断)

**while** (!compare(temp.charAt(0)) && (!priStack.empty())) {

**int** a = (**int**) numStack.pop();// 第二个运算数

**int** b = (**int**) numStack.pop();// 第一个运算数

**char** ope = priStack.pop();

**int** result = 0;// 运算结果

**switch** (ope) {

// 如果是加号或者减号，则

**case** '+':

result = b + a;

// 将操作结果放入操作数栈

numStack.push(result);

**break**;

**case** '-':

result = b - a;

// 将操作结果放入操作数栈

numStack.push(result);

**break**;

**case** '\*':

result = b \* a;

// 将操作结果放入操作数栈

numStack.push(result);

**break**;

**case** '/':

result = b / a;// 将操作结果放入操作数栈

numStack.push(result);

**break**;

}

}

// 判断当前运算符与栈顶元素优先级， 如果高，或者低于平，计算完后，将当前操作符号，放入操作符栈

**if** (temp.charAt(0) != '#') {

priStack.push(**new** Character(temp.charAt(0)));

**if** (temp.charAt(0) == ')') {// 当栈顶为'('，而当前元素为')'时，则是括号内以算完，去掉括号

priStack.pop();

priStack.pop();

}

}

} **else**

// 当为非操作符时（数字）

tempNum = tempNum.append(temp);// 将读到的这一位数接到以读出的数后(当不是个位数的时候)

}

**return** numStack.pop();

}

/\*\*

\* 判断传入的字符是不是0-9的数字

\*

\* **@param** str

\* 传入的字符串

\* **@return**

\*/

**private** **boolean** isNum(String temp) {

**return** temp.matches("[0-9]");

}

/\*\*

\* 比较当前操作符与栈顶元素操作符优先级，如果比栈顶元素优先级高，则返回true，否则返回false

\*

\* **@param** str 需要进行比较的字符

\* **@return** 比较结果 true代表比栈顶元素优先级高，false代表比栈顶元素优先级低

\*/

**private** **boolean** compare(**char** str) {

**if** (priStack.empty()) {

// 当为空时，显然 当前优先级最低，返回高

**return** **true**;

}

**char** last = (**char**) priStack.lastElement();

// 如果栈顶为'('显然，优先级最低，')'不可能为栈顶。

**if** (last == '(') {

**return** **true**;

}

**switch** (str) {

**case** '#':

**return** **false**;// 结束符

**case** '(':

// '('优先级最高,显然返回true

**return** **true**;

**case** ')':

// ')'优先级最低，

**return** **false**;

**case** '\*': {

// '\*/'优先级只比'+-'高

**if** (last == '+' || last == '-')

**return** **true**;

**else**

**return** **false**;

}

**case** '/': {

**if** (last == '+' || last == '-')

**return** **true**;

**else**

**return** **false**;

}

// '+-'为最低，一直返回false

**case** '+':

**return** **false**;

**case** '-':

**return** **false**;

}

**return** **true**;

}

**public** **static** **void** main(String args[]) {

Operate operate = **new** Operate();

**int** t = operate.caculate("(3+4\*(4\*10-10/2)#");

System.***out***.println(t);

}

}