Дудников Кирилл БПИ206.

Каждая ошибка располагается на новом листе.

Формат описания ошибки:

* Строки <номер> или Метод <сигнатура>
* Исключение на тесте
* Код до
* Код после
* Пояснение

Примечания:

* Сначала я поставил условие фаззеру - загружать только строки, соответствующие условиям в методе main класса Calc. Цитата: "Поддерживаются цифры, операции +,-,\*,/,^,% и приоритеты в виде скобок ( и )", то есть строки, соответствующие regex-паттерну “[+\-\*/^%()\d]+”. Так что сначала идут ошибки с таким входом, а потом уже другие.
* В папке fuzzer-crashes есть примеры артефактов, созданных фаззером. Но там есть не все описанные тут ошибки, так как пару раз я случайно target перманентно удалял.

**Строка(и) 126:**

**Исключение на тесте:**

| Calc.calculate(Calc.opn("0-0")); |
| --- |

**Код до:**

| **switch** (sTmp.charAt(1)) { |
| --- |

**Код после:**

| **switch** (sTmp.charAt(0)) { |
| --- |

**Пояснение:**

Берется второй символ из sTmp вместо первого. Длина строки sTmp здесь всегда равна 1.

**Метод isOp(char):**

**Исключение на тесте:**

| Calc.calculate(Calc.opn("333%3")); |
| --- |

**Код до:**

| **private** **static** **boolean** **isOp**(**char** c) {  **switch** (c) {  **case** '-':  **case** '+':  **case** '\*':  **case** '/':  **case** '^':  **case** '!':  **return** **true**;  }  **return** **false**;  } |
| --- |

**Код после:**

| **private** **static** **boolean** **isOp**(**char** c) {  **switch** (c) {  **case** '-':  **case** '+':  **case** '\*':  **case** '/':  **case** '^':  **case** '%':  **case** '!':  **return** **true**;  }  **return** **false**;  } |
| --- |

**Пояснение:**

Забыт оператор %.

**Метод opn(String):**

**Исключение на тесте:**

| Calc.calculate(Calc.opn("2(")); |
| --- |

**Код до:**

| **public** **static** String **opn**(String sIn) **throws** CalcException {  StringBuilder sbStack = **new** StringBuilder(), sbOut = **new** StringBuilder();  **char** cIn, cTmp;   **for** (**int** i = 0; i < sIn.length(); i++) {  cIn = sIn.charAt(i);  **if** (isOp(cIn)) {  **while** (sbStack.length() > 0) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **if** (isOp(cTmp) && (opPrior(cIn) <= opPrior(cTmp))) {  sbOut.append(" ").append(cTmp).append(" ");  sbStack.setLength(sbStack.length()-1);  } **else** {  sbOut.append(" ");  **break**;  }  }  sbOut.append(" ");  sbStack.append(cIn);  } **else** **if** ('(' == cIn) {  sbStack.append(cIn);  } **else** **if** (')' == cIn) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **while** ('(' != cTmp) {  **if** (sbStack.length() < 1) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }  sbOut.append(" ").append(cTmp);  sbStack.setLength(sbStack.length()-1);  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  }  sbStack.setLength(sbStack.length()-1);  } **else** {  *// Если символ не оператор - добавляем в выходную последовательность*  sbOut.append(cIn);  }  }   *// Если в стеке остались операторы, добавляем их в входную строку*  **while** (sbStack.length() > 0) {  sbOut.append(" ").append(sbStack.substring(sbStack.length()-1));  sbStack.setLength(sbStack.length()-1);  }   **return** sbOut.toString();  } |
| --- |

**Код после:**

| **public** **static** String **opn**(String sIn) **throws** CalcException {  StringBuilder sbStack = **new** StringBuilder(), sbOut = **new** StringBuilder();  **char** cIn, cTmp;  **int** openedParenthesisCount = 0;   **for** (**int** i = 0; i < sIn.length(); i++) {  cIn = sIn.charAt(i);  **if** (isOp(cIn)) {  **while** (sbStack.length() > 0) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **if** (isOp(cTmp) && (opPrior(cIn) <= opPrior(cTmp))) {  sbOut.append(" ").append(cTmp).append(" ");  sbStack.setLength(sbStack.length()-1);  } **else** {  sbOut.append(" ");  **break**;  }  }  sbOut.append(" ");  sbStack.append(cIn);  } **else** **if** ('(' == cIn) {  sbStack.append(cIn);  openedParenthesisCount++;  } **else** **if** (')' == cIn) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **while** ('(' != cTmp) {  **if** (sbStack.length() < 1) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }  sbOut.append(" ").append(cTmp);  sbStack.setLength(sbStack.length()-1);  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  }  sbStack.setLength(sbStack.length()-1);  openedParenthesisCount--;  } **else** {  *// Если символ не оператор - добавляем в выходную последовательность*  sbOut.append(cIn);  }  }   *// Если кол-во левых скобок превышает кол-во правых.*  *// Обратное проверяется выше.*  **if** (openedParenthesisCount > 0) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }   *// Если в стеке остались операторы, добавляем их в входную строку*  **while** (sbStack.length() > 0) {  sbOut.append(" ").append(sbStack.substring(sbStack.length()-1));  sbStack.setLength(sbStack.length()-1);  }   **return** sbOut.toString();  } |
| --- |

**Пояснение:**

Не проверяется равенство количеств открывающих и закрывающих скобок.

**Метод opn(String):**

**Исключение на тесте:**

| Calc.calculate(Calc.opn(")2")); |
| --- |

**Код до:**

| **public** **static** String **opn**(String sIn) **throws** CalcException {  StringBuilder sbStack = **new** StringBuilder(), sbOut = **new** StringBuilder();  **char** cIn, cTmp;  **int** openedParenthesisCount = 0;   **for** (**int** i = 0; i < sIn.length(); i++) {  cIn = sIn.charAt(i);  **if** (isOp(cIn)) {  **while** (sbStack.length() > 0) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **if** (isOp(cTmp) && (opPrior(cIn) <= opPrior(cTmp))) {  sbOut.append(" ").append(cTmp).append(" ");  sbStack.setLength(sbStack.length()-1);  } **else** {  sbOut.append(" ");  **break**;  }  }  sbOut.append(" ");  sbStack.append(cIn);  } **else** **if** ('(' == cIn) {  sbStack.append(cIn);  openedParenthesisCount++;  } **else** **if** (')' == cIn) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **while** ('(' != cTmp) {  **if** (sbStack.length() < 1) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }  sbOut.append(" ").append(cTmp);  sbStack.setLength(sbStack.length()-1);  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  }  sbStack.setLength(sbStack.length()-1);  openedParenthesisCount--;  } **else** {  *// Если символ не оператор - добавляем в выходную последовательность*  sbOut.append(cIn);  }  }   *// Если кол-во левых скобок превышает кол-во правых.*  *// Обратное проверяется выше.*  **if** (openedParenthesisCount > 0) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }   *// Если в стеке остались операторы, добавляем их в входную строку*  **while** (sbStack.length() > 0) {  sbOut.append(" ").append(sbStack.substring(sbStack.length()-1));  sbStack.setLength(sbStack.length()-1);  }   **return** sbOut.toString();  } |
| --- |

**Код после:**

| **public** **static** String **opn**(String sIn) **throws** CalcException {  StringBuilder sbStack = **new** StringBuilder(), sbOut = **new** StringBuilder();  **char** cIn, cTmp;  **int** openedParenthesisCount = 0;   **for** (**int** i = 0; i < sIn.length(); i++) {  cIn = sIn.charAt(i);  **if** (isOp(cIn)) {  **while** (sbStack.length() > 0) {  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **if** (isOp(cTmp) && (opPrior(cIn) <= opPrior(cTmp))) {  sbOut.append(" ").append(cTmp).append(" ");  sbStack.setLength(sbStack.length()-1);  } **else** {  sbOut.append(" ");  **break**;  }  }  sbOut.append(" ");  sbStack.append(cIn);  } **else** **if** ('(' == cIn) {  sbStack.append(cIn);  openedParenthesisCount++;  } **else** **if** (')' == cIn) {  **if** (sbStack.length() < 1) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  **while** ('(' != cTmp) {  sbOut.append(" ").append(cTmp);  sbStack.setLength(sbStack.length()-1);  **if** (sbStack.length() < 1) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }  cTmp = sbStack.substring(sbStack.length()-1).charAt(0);  }  sbStack.setLength(sbStack.length()-1);  openedParenthesisCount--;  } **else** {  *// Если символ не оператор - добавляем в выходную последовательность*  sbOut.append(cIn);  }  }   *// Если кол-во левых скобок превышает кол-во правых.*  *// Обратное проверяется выше.*  **if** (openedParenthesisCount > 0) {  **throw** **new** CalcException("Ошибка разбора скобок. Проверьте правильность выражения.");  }   *// Если в стеке остались операторы, добавляем их в входную строку*  **while** (sbStack.length() > 0) {  sbOut.append(" ").append(sbStack.substring(sbStack.length()-1));  sbStack.setLength(sbStack.length()-1);  }   **return** sbOut.toString();  } |
| --- |

**Пояснение:**

Немного по-другому можно расставить проверку на пустоту стека для получения предыдущей открывающей скобки.

**Метод calculate(String):**

**Исключение на тесте:**

| Calc.calculate(Calc.opn("()")); |
| --- |

**Код до:**

| **public** **static** **double** **calculate**(String sIn) **throws** CalcException {  **double** dA, dB;  String sTmp;  Deque<Double> stack = **new** ArrayDeque<>();  StringTokenizer st = **new** StringTokenizer(sIn);  **while** (st.hasMoreTokens()) {  **try** {  sTmp = st.nextToken().trim();  **if** (1 == sTmp.length() && isOp(sTmp.charAt(0))) {  **if** (stack.size() < 2) {  **throw** **new** CalcException("Неверное количество данных в стеке для операции " + sTmp);  }  dB = stack.pop();  dA = stack.pop();  *// Считываем первый символ*  **switch** (sTmp.charAt(0)) {  **case** '+':  dA += dB;  **break**;  **case** '-':  dA -= dB;  **break**;  **case** '/':  dA /= dB;  **break**;  **case** '\*':  dA \*= dB;  **break**;  **case** '%':  dA %= dB;  **break**;  **case** '^':  dA = Math.pow(dA, dB);  **break**;  **case** '!':  **throw** **new** UnsupportedOperationException("TODO: Не забыть реализовать оператор !");  **default**:  **throw** **new** CalcException("Недопустимая операция " + sTmp);  }  stack.push(dA);  } **else** {  dA = Double.parseDouble(sTmp);  stack.push(dA);  }  } **catch** (CalcException e) {  **throw** **new** CalcException("Недопустимый символ в выражении");  }  }   **if** (stack.size() > 1) {  **throw** **new** CalcException("Количество операторов не соответствует количеству операндов");  }   **return** stack.pop();  } |
| --- |

**Код после:**

| **public** **static** **double** **calculate**(String sIn) **throws** CalcException {  **double** dA, dB;  String sTmp;  Deque<Double> stack = **new** ArrayDeque<>();  StringTokenizer st = **new** StringTokenizer(sIn);  **while** (st.hasMoreTokens()) {  **try** {  sTmp = st.nextToken().trim();  **if** (1 == sTmp.length() && isOp(sTmp.charAt(0))) {  **if** (stack.size() < 2) {  **throw** **new** CalcException("Неверное количество данных в стеке для операции " + sTmp);  }  dB = stack.pop();  dA = stack.pop();  *// Считываем первый символ*  **switch** (sTmp.charAt(0)) {  **case** '+':  dA += dB;  **break**;  **case** '-':  dA -= dB;  **break**;  **case** '/':  dA /= dB;  **break**;  **case** '\*':  dA \*= dB;  **break**;  **case** '%':  dA %= dB;  **break**;  **case** '^':  dA = Math.pow(dA, dB);  **break**;  **case** '!':  **throw** **new** UnsupportedOperationException("TODO: Не забыть реализовать оператор !");  **default**:  **throw** **new** CalcException("Недопустимая операция " + sTmp);  }  stack.push(dA);  } **else** {  dA = Double.parseDouble(sTmp);  stack.push(dA);  }  } **catch** (CalcException e) {  **throw** **new** CalcException("Недопустимый символ в выражении");  }  }   **if** (stack.size() > 1) {  **throw** **new** CalcException("Количество операторов не соответствует количеству операндов");  }   **if** (stack.isEmpty()) {  **throw** **new** CalcException("Нет ни одного оператора или числа.");  }  **return** stack.pop();  } |
| --- |

**Пояснение:**

Обработка пустого (или почти пустого) запроса.

**Метод calculate(String):**

**Исключение на тесте:**

| Calc.calculate(Calc.opn("(4\*4\*4\*4\*4\*4\*4\*)0+00+0")); |
| --- |

**Код до:**

| **public** **static** **double** **calculate**(String sIn) **throws** CalcException {  **double** dA, dB;  String sTmp;  Deque<Double> stack = **new** ArrayDeque<>();  StringTokenizer st = **new** StringTokenizer(sIn);  **while** (st.hasMoreTokens()) {  **try** {  sTmp = st.nextToken().trim();  **if** (1 == sTmp.length() && isOp(sTmp.charAt(0))) {  **if** (stack.size() < 2) {  **throw** **new** CalcException("Неверное количество данных в стеке для операции " + sTmp);  }  dB = stack.pop();  dA = stack.pop();  *// Считываем первый символ*  **switch** (sTmp.charAt(0)) {  **case** '+':  dA += dB;  **break**;  **case** '-':  dA -= dB;  **break**;  **case** '/':  dA /= dB;  **break**;  **case** '\*':  dA \*= dB;  **break**;  **case** '%':  dA %= dB;  **break**;  **case** '^':  dA = Math.pow(dA, dB);  **break**;  **case** '!':  **throw** **new** UnsupportedOperationException("TODO: Не забыть реализовать оператор !");  **default**:  **throw** **new** CalcException("Недопустимая операция " + sTmp);  }  stack.push(dA);  } **else** {  dA = Double.parseDouble(sTmp);  stack.push(dA);  }  } **catch** (CalcException e) {  **throw** **new** CalcException("Недопустимый символ в выражении");  }  }   **if** (stack.size() > 1) {  **throw** **new** CalcException("Количество операторов не соответствует количеству операндов");  }   **if** (stack.isEmpty()) {  **throw** **new** CalcException("Нет ни одного оператора или числа.");  }  **return** stack.pop();  } |
| --- |

**Код после:**

| **public** **static** **double** **calculate**(String sIn) **throws** CalcException {  **double** dA, dB;  String sTmp;  Deque<Double> stack = **new** ArrayDeque<>();  StringTokenizer st = **new** StringTokenizer(sIn);  **while** (st.hasMoreTokens()) {  **try** {  sTmp = st.nextToken().trim();  **if** (1 == sTmp.length() && isOp(sTmp.charAt(0))) {  **if** (stack.size() < 2) {  **throw** **new** CalcException("Неверное количество данных в стеке для операции " + sTmp);  }  dB = stack.pop();  dA = stack.pop();  *// Считываем первый символ*  **switch** (sTmp.charAt(0)) {  **case** '+':  dA += dB;  **break**;  **case** '-':  dA -= dB;  **break**;  **case** '/':  dA /= dB;  **break**;  **case** '\*':  dA \*= dB;  **break**;  **case** '%':  dA %= dB;  **break**;  **case** '^':  dA = Math.pow(dA, dB);  **break**;  **case** '!':  **throw** **new** UnsupportedOperationException("TODO: Не забыть реализовать оператор !");  **default**:  **throw** **new** CalcException("Недопустимая операция " + sTmp);  }  stack.push(dA);  } **else** {  dA = Double.parseDouble(sTmp);  stack.push(dA);  }  } **catch** (NumberFormatException e) {  **throw** **new** CalcException("Недопустимый символ в выражении");  }  }   **if** (stack.size() > 1) {  **throw** **new** CalcException("Количество операторов не соответствует количеству операндов");  }   **if** (stack.isEmpty()) {  **throw** **new** CalcException("Нет ни одного оператора или числа.");  }  **return** stack.pop();  } |
| --- |

**Пояснение:**

Логично выбрасывать исключение с сообщением “Недопустимый символ в выражении”, когда появляется NumberFormatException, а не CalcException.

**Метод calculate(String):**

**Исключение на тесте:**

| Calc.calculate(Calc.opn("1!1")); |
| --- |

**Код до:**

| **public** **static** **double** **calculate**(String sIn) **throws** CalcException {  **double** dA, dB;  String sTmp;  Deque<Double> stack = **new** ArrayDeque<>();  StringTokenizer st = **new** StringTokenizer(sIn);  **while** (st.hasMoreTokens()) {  **try** {  sTmp = st.nextToken().trim();  **if** (1 == sTmp.length() && isOp(sTmp.charAt(0))) {  **if** (stack.size() < 2) {  **throw** **new** CalcException("Неверное количество данных в стеке для операции " + sTmp);  }  dB = stack.pop();  dA = stack.pop();  *// Считываем первый символ*  **switch** (sTmp.charAt(0)) {  **case** '+':  dA += dB;  **break**;  **case** '-':  dA -= dB;  **break**;  **case** '/':  dA /= dB;  **break**;  **case** '\*':  dA \*= dB;  **break**;  **case** '%':  dA %= dB;  **break**;  **case** '^':  dA = Math.pow(dA, dB);  **break**;  **case** '!':  **throw** **new** UnsupportedOperationException("TODO: Не забыть реализовать оператор !");  **default**:  **throw** **new** CalcException("Недопустимая операция " + sTmp);  }  stack.push(dA);  } **else** {  dA = Double.parseDouble(sTmp);  stack.push(dA);  }  } **catch** (NumberFormatException e) {  **throw** **new** CalcException("Недопустимый символ в выражении");  }  }   **if** (stack.size() > 1) {  **throw** **new** CalcException("Количество операторов не соответствует количеству операндов");  }   **if** (stack.isEmpty()) {  **throw** **new** CalcException("Нет ни одного оператора или числа.");  }  **return** stack.pop();  } |
| --- |

**Код после:**

| **public** **static** **double** **calculate**(String sIn) **throws** CalcException {  **double** dA, dB;  String sTmp;  Deque<Double> stack = **new** ArrayDeque<>();  StringTokenizer st = **new** StringTokenizer(sIn);  **while** (st.hasMoreTokens()) {  **try** {  sTmp = st.nextToken().trim();  **if** (1 == sTmp.length() && isOp(sTmp.charAt(0))) {  **if** (stack.size() < 2) {  **throw** **new** CalcException("Неверное количество данных в стеке для операции " + sTmp);  }  dB = stack.pop();  dA = stack.pop();  *// Считываем первый символ*  **switch** (sTmp.charAt(0)) {  **case** '+':  dA += dB;  **break**;  **case** '-':  dA -= dB;  **break**;  **case** '/':  dA /= dB;  **break**;  **case** '\*':  dA \*= dB;  **break**;  **case** '%':  dA %= dB;  **break**;  **case** '^':  dA = Math.pow(dA, dB);  **break**;  **default**:  **throw** **new** CalcException("Недопустимая операция " + sTmp);  }  stack.push(dA);  } **else** {  dA = Double.parseDouble(sTmp);  stack.push(dA);  }  } **catch** (NumberFormatException e) {  **throw** **new** CalcException("Недопустимый символ в выражении");  }  }   **if** (stack.size() > 1) {  **throw** **new** CalcException("Количество операторов не соответствует количеству операндов");  }   **if** (stack.isEmpty()) {  **throw** **new** CalcException("Нет ни одного оператора или числа.");  }  **return** stack.pop();  } |
| --- |

**Пояснение:**

Не знаю, наверное можно просто удалить блок case для операции “!”. В операциях, упомянутых в main (т.е. в приложении), нет факториала. Можно еще удалить факториал из метода isOp(char), но я не стал этого делать.