Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа интеллектуальных систем и суперкомпьютерных технологий

**КУРСОВОЙ ПРОЕКТ**

**Дисциплина:** Алгоритмы и структуры данных

**Тема:** Разработка GUI приложения для расстановки арифметических знаков в пример, где они отсутствуют.

Выполнил

студент гр. 3530901/90003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Руднев А.К.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Ахин М.Х.

(подпись)

«\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2020 г.

Санкт-Петербург   
2020

**Оглавление**

[**ТЕХНИЧЕСКОЕ ЗАДАНИЕ** 3](#_Toc26066265)

[**МЕТОД РЕШЕНИЯ** 4](#_Toc26066266)

[**СКРИНШОТЫ ПРОГРАММЫ** 7](#_Toc26066281)

Приложение 1:

[**ЛИСТИНГ ПРОГРАММЫ** 8](#_Toc26066267)

[**Main.java** 8](#_Toc26066268)

[**StartController.java** 8](#_Toc26066269)

[**Example.java** 12](#_Toc26066270)

[**CombinationOfSigns.java** 14](#_Toc26066271)

[**PutterSigns.java** 15](#_Toc26066272)

[**Repetitions.java** 17](#_Toc26066273)

[**Calc.java** 19](#_Toc26066274)

[**ТЕСТЫ** 21](#_Toc26066280)

# **ТЕХНИЧЕСКОЕ ЗАДАНИЕ**

Техническое задание:

Задан пример, состоящий из чисел и скобок, а также с ответом.

(Например: 1 1 1=3).

Создать GUI-приложение на Java для расстановки знаков в примере. В окне приложения находятся следующие компоненты: кнопка “решение”, строка для ввода примера, кнопка “помощь”. Пример задает пользователь в строку для записи с помощью клавиатуры. При нажатии на кнопку “решение” проверить написанный в строчку для ввода пример на наличие иных символов и правильность расставления скобок, вычислить какие знаки ‘\*’, ‘+’, ‘-’ должны стоят в примере, чтобы ответ был равен данному значению. Вывести полученный пример со знаками на экран в отдельном сообщении, если невозможно поставить знаки, то следует выдать сообщение о невозможности выполнения такой работы. При нажатии на кнопку “помощь”, пользователю выводится сообщение о работе программы.

**МЕТОД РЕШЕНИЯ**

В проекте использована концепция MVC (Model-View-Controller) для отделения бизнес-логики от визуализации. Код разделен на три пакета классов: view, controller, model. Алгоритм состоит в расстановке комбинаций знаков, расстановке их в примере, переводе в обратную польскую запись, вычисления и замене уже вычисленных действий.

Содержимое пакета view:

* StartView.java – отвечает за отображение меню.

Cодержимое пакета controller:

* StartController.java – отвечает за связь view и model, выполняет проверку входных данных на наличие ошибок или иных символов, также выполняет первоначальный анализ и обработку строк. Передает model все введенные данные и запускает его работу.

Содержимое пакета model:

* Examples.java – получение данных от controller и поиск ответа.
* CombinationOfSigns.java – класс, отвечающий за формирование всех комбинаций знаков, допустимых условием.
* PutterSigns.java – класс, вставляющий знаки в пример.
* PolishNotation.java– класс, который переводит получившиеся примеры со знаками в обратную польскую запись.
* Calc.java – стековый калькулятор.
* Repetitions.java – удаление повторений в процессе подсчета.

Более подробная работа программы:

1.1) **Пакет view** c классом StartView.java с единственным методом StartView(), который определяет набор компонентов в главном окне приложения, а также его размер и название.

2.1) **Пакет controller** содержит класс StartController.java, в который входят:

Основные методы класса StartController.java:

a) toHelp() – отвечает за функциональность кнопки “Help” и выводит сообщение help.txt в отдельном сообщении

б) toSolve() – отвечает за функционал кнопки “Solve” и передает данные в Example.java, а получив результат, выводит его пользователю на экран, если не пришел ответ об некорректности данных или их отсутствии при дальнейших проверках.

Методы, проверяющие введенное выражение на корректность:

a) check() – проверка примера на пустоту, наличие иных символов

б) checkParentheses() – проверка примера на правильность расставленных скобок

в) checkArgument() – проверка первой части примера на наличие иных символов

г) checkAnswer() – проверка второй части примера на наличие иных символов.

д) parse() – разбивает строку на “example” и “answer ” знаком «=».

Методы для первичной парсировки строк (удаление лишних пробелов):

a) deleteSpaces() – удаление лишних пробелов, кроме пробелов между числами.

б) plusSymbols() – добавление символа вместо пробела

г) addSymbols() – добавление символа в конец.

3) Вся логика включена в **пакет model**, который содержит:

3.1) Класс Example.java, содержит методы:

a) countSpaces()) – подсчет пробелов (количества мест под знаки)

б) toPutResponse() – кладет в карту, полученные ответы, генерируемые классом Repetition.java.

в) toPutSigns() – генерирует все комбинации знаков, допустимых условием с помощью класса CombinationOfSigns.java и с помощью класса PutterSigns.java вставляет их в пример.

г) isEquals() – сравнивает ответы в карте и ответ, введенный пользователем.

д) toSolve() – принимает параметры от controller и отдает ответ обратно.

3.2) Класс CombinationOfSigns.java служит для генерации комбинаций знаков рекурсивным способом (для реализации метода toPutSigns() – необходимо иметь комбинации знаков).

3.3) Класс PutterSigns.java вставляет комбинации знаков в примеры и возвращается массив с комбинациями знаков в примерах.

3.4) Класс PolishedNotation.java, который изменяет их согласно обратной польской нотации (“2\*(2+2)” = “2 2 + 2 \*”) имеет несколько методов:

a) isDelimeters() – проверка символа на принадлежность к ‘()+-\*’.

б) isOperator() – проверка символа на принадлежность к ‘+-\*’.

в) priority() – проверка знака на приоритетность.\

г) parse() – перевод примера в обратную польскую запись.

3.5) Класс Calc.java – реализация стекового калькулятора

3.6) Класс Repetition, в котором с помощью стекового калькулятора начинаются подсчеты (объяснение выполения):

1) Выявляется первое действие в примере (например: 2 2 + 2 \*, первое действие = 2 2 +)

2) Идет вычисление результата, поиск и замена таких же действий в других примерах

3) Если в примере есть еще знаки, то цикл запускается с начала.

4) Если ответ совпадает – завершает все вычисления и возвращает нужный результат в класс Example(), после этого нужный результат выводится пользователю.

В классе Repetitions.java есть методы:

a) toDelete() – удаляет все повторения в остальных примерах и заменяет их одним числом – результатом

б) toDeleteRepetitions() – подсчет значений примера по действия и удаления повторений с помощью метода toDelete();

г) repetition() – получение данных из класса Example.java, а также сверка результата с ответом, введенным пользователем, возможное завершение вычислений а также возвращение результата назад.

**СКРИНШОТЫ РАБОТЫ ПРОГРАММЫ**

Скриншоты работы программы приведены на рисунках 1 - 4

|  |
| --- |
| Рис. 1- Окно приложения |
| Рис. 2 – Задание примера в строку для ввода |
| Рис. 3 – Выведение результата в отдельном сообщении |
| Рис. 3.1 – Решение примера с выведением результата в сообщении |
| Рис. 3.2 – Решение примера с выведением результата в сообщении |
| Рис. 3.3 – Сообщение об ошибке, так как в овтете есть иные символы |
| Рис. 3.4 – Сообщение об отсутствии решения |
| Рис. 3.5 – Сообщение о некорректном вводе  (так как в примере пристуствуют пустые скобки) |
| Рис. 3.6 – Сообщение о некорректном вводе  (так как неправильно расставлены скобки) |
| Рис. 3.7 – Сообщение о некорректном вводе  (так как в певрой части примера есть иные символы) |
| Рис. 3.8 – Сообщение о некорректном вводе  (так как во второй части примера есть иные символы) |
| Рис. 4 – Выведение сообщения с помощью |

Приложение 1

**ЛИСТИНГ ПРОГРАММЫ**

**Программа:**

**Класс Main.java;**

package lostSigns;  
  
import lostSigns.view.StartView;  
  
  
public class Main {  
 public static void main(String[] args) {  
 new StartView();  
 }  
}

**Пакет controller:**

**Класс StartController.java;**

package lostSigns.controller;  
  
  
import lostSigns.model.Examples;  
  
import javax.swing.\*;  
import java.io.\*;  
import java.util.ArrayList;  
import java.util.List;  
import java.util.PropertyResourceBundle;  
import java.util.ResourceBundle;  
  
public class StartController {  
  
 public void toHelp(JButton help) {  
 help.addActionListener(e -> {  
 try {  
 ResourceBundle res = new PropertyResourceBundle(getClass().getResourceAsStream("help.txt"));  
 JOptionPane.*showMessageDialog*(null, res.getString("help"));  
 } catch (IOException ex) {  
 JOptionPane.*showMessageDialog*(null, ex.getMessage());  
 }  
 });  
 }  
  
 public void toStart(JButton dicision, final JTextField example) {  
 dicision.addActionListener(e -> {  
 switch (*check*((example.getText()))) {  
 case 0:  
 JOptionPane.*showMessageDialog*(null,  
 "Введите пример");  
 break;  
 case 1:  
 JOptionPane.*showMessageDialog*(null,  
 "Введите корректный пример (см. Помощь)");  
 break;  
 case 2:  
 String enteredExample = example.getText();  
 String[] parse = parser(enteredExample);  
 int answer = Integer.*parseInt*(parse[1].trim());  
 String exampleWithoutSigns = *deleteSpaces*(parse[0].trim());  
  
 Examples examples = new Examples(answer, exampleWithoutSigns);  
 String exampleWithSigns = examples.toSolve();  
  
 JOptionPane.*showMessageDialog*(null, exampleWithSigns);  
 break;  
 }  
 });  
 }  
  
  
 private String[] parser(String example) {  
 return example.split("=");  
 }  
  
 public static int check(String example) {  
 if (example.equals("")) return 0;  
 String [] partsOfExample = new String[0];  
  
 if (example.contains("=")) partsOfExample = example.split("=");  
 if (partsOfExample.length > 2 || partsOfExample.length == 1) return 1;  
 else if (!example.contains("=")) return 1;  
 else if (*checkArgument*(partsOfExample[0]) & *checkAnswer*(partsOfExample[1])  
 & *checkParentheses*(partsOfExample[0])) {  
 return 2;  
 } else {  
 return 1;  
 }  
 }  
  
 public static boolean checkParentheses(String exampleWithoutAnswer) {  
 if (!exampleWithoutAnswer.contains("(") && ! exampleWithoutAnswer.contains(")")) return true;  
  
 boolean isOpen = false;  
 int countOpen = 0;  
 int countNumber = 0;  
  
 for (int i = 0; i < exampleWithoutAnswer.length(); i++) {  
 if (exampleWithoutAnswer.charAt(i) == '(') {  
 isOpen = true;  
 countOpen++;  
 }  
 if (*isInteger*(exampleWithoutAnswer.charAt(i))) {  
 if (isOpen && (!*isInteger*(exampleWithoutAnswer.charAt(i - 1)))) countNumber++;  
 }  
 if (exampleWithoutAnswer.charAt(i) == ')') {  
 if (countNumber < 2 && isOpen) return false;  
 countNumber = 0;  
 countOpen--;  
 if (countOpen == -1) return false;  
 isOpen = false;  
 }  
 }  
 return (countOpen == 0);  
 }  
  
 public static boolean isInteger(char s) {  
 boolean isValidInteger = false;  
 try  
 {  
 Integer.*parseInt*(String.*valueOf*(s));  
 isValidInteger = true;  
 }  
 catch (NumberFormatException ignored) {}  
 return isValidInteger;  
 }  
  
 public static boolean checkArgument(String example) {  
 String parse = example.replace(")", "").replace("(", "");  
 return parse.matches("^(\\s\*\\d+\\s+\\d+)+\\s\*\\d\*\\s\*");  
 }  
  
 public static boolean checkAnswer(String example) {  
 return example.matches("^\\s\*\\d+\\s\*");  
 }  
  
 public static String deleteSpaces(String example) {  
 String exampleWithoutSpaces;  
 String parseExample = example.trim();  
 List<String> exampleInList = new ArrayList<>();  
 for (int i = 0, j = 0; i < parseExample.length(); i++, j++) {  
 if (parseExample.charAt(i) == '(' || parseExample.charAt(i) == ')' || parseExample.charAt(i) == ' ') {  
 *addSymbols*(parseExample, i, j, exampleInList);  
 }  
 if (i != 0 && *isInteger*(parseExample.charAt(i - 1)) && *isInteger*(parseExample.charAt(i))) {  
 j--;  
 *plusSymbols*(parseExample, i, j, exampleInList);  
 } else if(*isInteger*(parseExample.charAt(i))) {  
 *addSymbols*(parseExample, i, j, exampleInList);  
 }  
 }  
 exampleInList.removeIf(i -> i.equals(" "));  
 exampleWithoutSpaces = *toStringList*(exampleInList);  
 return exampleWithoutSpaces;  
 }  
  
 public static List<String> plusSymbols(String example, int symbolsForPut,  
 int placeForPut, List<String> exampleInList) {  
 String iStr = exampleInList.get(placeForPut);  
 exampleInList.remove(placeForPut);  
 String what = String.*valueOf*(example.charAt(symbolsForPut));  
 String need = iStr + what;  
 exampleInList.add(placeForPut, need);  
 return exampleInList;  
 }  
  
 public static List<String> addSymbols(String example, int symbolsForPut,  
 int placeForPut, List<String> exampleInList) {  
 exampleInList.add(placeForPut, String.*valueOf*(example.charAt(symbolsForPut)));  
 return exampleInList;  
 }  
  
 public static String toStringList(List<String> list) {  
 StringBuilder sb = new StringBuilder();  
 for(int i = 0; i < list.size(); i++) {  
 sb.append(list.get(i));  
 if (list.get(i).matches("^\\d+") && ((i + 1) < list.size() && !list.get(i + 1).equals(")")))  
 sb.append(" ");  
 if (list.get(i).equals(")") && ((i + 1) < list.size() && !list.get(i + 1).equals(")")))  
 sb.append(" ");  
 }  
 return sb.toString();  
 }  
}

**Пакет model:**

**Класс Example.java;**

package lostSigns.model;  
  
  
import java.util.\*;  
  
public class Examples {  
 private final int enteredAnswer;  
 private final String example;  
 private ArrayList<String> examplesWithSigns = new ArrayList<>();  
 private final Map<Integer, String> examplesWithAnswers = new HashMap<>();  
  
  
 public Examples(int enteredAnswer, String example) {  
 this.enteredAnswer = enteredAnswer;  
 this.example = example;  
 examplesWithSigns.removeAll(examplesWithSigns);  
 examplesWithAnswers.clear();  
 }  
  
  
 public String toSolve() {  
 List<List<String>> calculatedExamples = new ArrayList<>();  
 examplesWithSigns = toPutSigns(example);  
 List<String> polishNotationForm;  
 for (String in : examplesWithSigns) {  
 PolishNotation polishNotation = new PolishNotation();  
 polishNotationForm = polishNotation.parse(in);  
 boolean flag = PolishNotation.*flag*;  
 if (flag) {  
 calculatedExamples.add(polishNotationForm);  
 }  
 }  
 Repetitions repetitionsInList = new Repetitions(enteredAnswer, true, calculatedExamples);  
 List<List<String>> answersForExamples = repetitionsInList.repetition();  
 answersForExamples.removeIf(i -> i.size() > 1);  
 for(int i = 0; i < answersForExamples.size(); i++) {  
 toPutResponse(Integer.*parseInt*(answersForExamples.get(i).get(0)), examplesWithSigns.get(i));  
 }  
 return isEquals(examplesWithAnswers, enteredAnswer);  
 }  
  
 public ArrayList<String> toPutSigns(String example) {  
 int count = toCountSpaces(example);  
 List<String> signs = CombinationOfSigns.*toCombinationSigns*(count);  
 examplesWithSigns = (ArrayList<String>) PutterSigns.*putSigns*(signs, example);  
 return examplesWithSigns;  
 }  
  
 public int toCountSpaces(String example) {  
 int count = 0;  
 for(int i = 0; i < example.length(); i++) {  
 if (example.charAt(i) == ' ') count++;  
 }  
 return count;  
 }  
  
 public Map<Integer, String> toPutResponse(Integer calc, String in) {  
 examplesWithAnswers.put(calc, in);  
 return examplesWithAnswers;  
 }  
  
 public String isEquals(Map<Integer, String> examplesWithSings, int enteredAnswer) {  
 for (Map.Entry<Integer, String> index : examplesWithSings.entrySet()) {  
 if (index.getKey() == enteredAnswer) return index.getValue() + "=" + enteredAnswer;  
 }  
 return "Решения нет";  
 }  
}

**Класс CombinationOfSigns.java;**

package lostSigns.model;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class CombinationOfSigns {  
 public interface PermuteCallback{  
 public void handle(Object[] snapshot);  
 };  
  
 static void permute(Object[] a, int k, PermuteCallback callback) {  
 int n = a.length;  
 int[] indexes = new int[k];  
 int total = (int) Math.*pow*(n, k);  
 Object[] snapshot = new Object[k];  
  
 while (total-- > 0) {  
 for (int i = 0; i < k; i++){  
 snapshot[i] = a[indexes[i]];  
 }  
 callback.handle(snapshot);  
 for (int i = 0; i < k; i++) {  
 if (indexes[i] >= n - 1) {  
 indexes[i] = 0;  
 } else {  
 indexes[i]++;  
 break;  
 }  
 }  
 }  
 }  
  
 public static List<String> toCombinationSigns(int count) {  
 List<String> list = new ArrayList<>();  
 Object[] chars = {'\*', '+', '-'};  
 PermuteCallback callback = snapshot -> {  
 StringBuilder str = new StringBuilder();  
 for (Object o : snapshot) {  
 str.append(o);  
 }  
 list.add(str.toString());  
 };  
 *permute*(chars, count, callback);  
 return list;  
 }  
 }

**Класс PutterSigns.java;**

package lostSigns.model;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class PutterSigns {  
  
 public static List<String> putSigns(List<String> signs, String example) {  
 List<String> exampleWithSigns = new ArrayList<>();  
 for (String sign : signs) {  
 String ex = example;  
 String str = "";  
 for (int k = 0, j = 0; k < ex.length() && j < sign.length(); k++) {  
 if (ex.charAt(k) == ' ') {  
 str = ex.substring(0, k) + sign.charAt(j) + ex.substring(k + 1);  
 ex = str;  
 k--;  
 j++;  
 }  
 }  
 exampleWithSigns.add(str);  
 }  
 return exampleWithSigns;  
 }  
}

класс PolishNotation;

package lostSigns.model;  
  
import java.util.\*;  
  
public class PolishNotation {  
 private static final String *operators* = "+-\*";  
 private static final String *delimiters* = "() " + *operators*;  
 public static boolean *flag* = true;  
  
  
 public boolean isDelimiter(String token) {  
 if (token.length() != 1) return false;  
 for (int i = 0; i < *delimiters*.length(); i++) {  
 if (token.charAt(0) == *delimiters*.charAt(i)) return true;  
 }  
 return false;  
 }  
  
 public boolean isOperator(String token) {  
 for (int i = 0; i < *operators*.length(); i++) {  
 if (token.charAt(0) == *operators*.charAt(i)) return true;  
 }  
 return false;  
 }  
  
 public int priority(String token) {  
 if (token.equals("(")) return 1;  
 if (token.equals("+") || token.equals("-")) return 2;  
 if (token.equals("\*")) return 3;  
 return 4;  
 }  
  
 public List<String> parse(String infix) {  
 List<String> postfix = new ArrayList<>();  
 Deque<String> stack = new ArrayDeque<>();  
 StringTokenizer tokenizer = new StringTokenizer(infix, *delimiters*, true);  
 String curr;  
 while (tokenizer.hasMoreTokens()) {  
 curr = tokenizer.nextToken();  
 if (isDelimiter(curr)) {  
 if (curr.equals("(")) stack.push(curr);  
 else if (curr.equals(")")) {  
 while (!stack.peek().equals("(")) {  
 postfix.add(stack.pop()); }  
 stack.pop();  
 } else {  
 while (!stack.isEmpty() && (priority(curr) <= priority(stack.peek()))) {  
 postfix.add(stack.pop());  
 }  
 stack.push(curr);  
 }  
  
 } else {  
 postfix.add(curr);  
 }  
 }  
  
 while (!stack.isEmpty()) {  
 if (isOperator(stack.peek())) postfix.add(stack.pop());  
 }  
 return postfix;  
 }  
}

**Класс Repetitions.java**;

package lostSigns.model;  
  
import java.util.ArrayList;  
import java.util.List;  
  
public class Repetitions {  
 private int answer = 0;  
 private boolean doing = true;  
 private List<List<String>> expression;  
  
 public Repetitions() {  
 }  
  
 public Repetitions(int answer, boolean doing, List<List<String>> expression) {  
 this.answer = answer;  
 this.doing = doing;  
 this.expression = expression;  
 }  
  
 public List<List<String>> repetition() {  
 for(int index = 0; index < expression.size() && doing; index++) {  
 toDeleteRepetition(expression.get(index), index);  
 }  
 return expression;  
 }  
  
 public List<String> toDeleteRepetition(List<String> list, int index) {  
 PolishNotation polishNotation = new PolishNotation();  
 List<String> signNeedToDelete = new ArrayList<>();  
 int count = -2;  
 for (String s : list) {  
 signNeedToDelete.add(s);  
 if (!polishNotation.isOperator(s)) count++;  
 if (polishNotation.isOperator(s) && count == 0) count--;  
 }  
  
 int calculateToPut = Math.*round*(Calc.*calc*(signNeedToDelete));  
 toDelete(expression, signNeedToDelete, calculateToPut, index);  
 if (expression.get(index).size() != 1) toDeleteRepetition(expression.get(index), index);  
 if (expression.get(index).size() == 1 && calculateToPut == answer) {  
 doing = false;  
 return expression.get(index);  
 }  
 return expression.get(index);  
 }  
  
  
 public List<List<String>> toDelete(List<List<String>> expr,List<String> signNeedToDelete,  
 int calculateToPut, int index) {  
  
 String n = String.*valueOf*(calculateToPut);  
 int c = 0;  
 for (; index < expr.size(); index++) {  
 for (int i = 0; i < signNeedToDelete.size() && i < expr.get(index).size(); i++) {  
 if (expr.get(index).get(i).equals(signNeedToDelete.get(i))) c++;  
 }  
 if (c == signNeedToDelete.size()) {  
 expr.get(index).subList(0, signNeedToDelete.size()).clear();  
 expr.get(index).add(0, n);  
 }  
 expression = expr;  
 c = 0;  
 }  
 return expr;  
 }  
}

**Класс Calc.java;**

package lostSigns.model;  
  
import java.util.ArrayDeque;  
import java.util.Deque;  
import java.util.List;  
  
public class Calc {  
 public static Integer calc(List<String> postfix) {  
 Deque<Integer> stack = new ArrayDeque<>();  
 for (String x : postfix) {  
 switch (x) {  
 case "+":  
 stack.push(stack.pop() + stack.pop());  
 break;  
 case "-":  
 Integer b = stack.pop(), a = stack.pop();  
 stack.push(a - b);  
 break;  
 case "\*":  
 stack.push(stack.pop() \* stack.pop());  
 break;  
 default:  
 stack.push(Integer.*valueOf*(x));  
 break;  
 }  
 }  
 return stack.pop();  
 }  
}

пакет view;

класс StartView;

package lostSigns.view;  
  
  
import lostSigns.controller.StartController;  
  
import java.awt.FlowLayout;  
  
import javax.swing.JButton;  
import javax.swing.JFrame;  
import javax.swing.JPanel;  
import javax.swing.JTextField;  
import javax.swing.WindowConstants;  
  
  
public class StartView {  
 public StartView() {  
 StartController startController = new StartController();  
 JPanel panel = new JPanel(new FlowLayout(FlowLayout.*CENTER*));  
 JTextField example = new JTextField(12);  
  
 JButton decision = new JButton("Solve");  
 panel.add(decision);  
 startController.toStart(decision, example);  
  
 example.setToolTipText("Пример");  
 panel.add(example);  
  
 JButton help = new JButton("Help");  
 panel.add(help);  
 startController.toHelp(help);  
  
 JFrame startFrame = new JFrame("Lost signs");  
 startFrame.setContentPane(panel);  
 startFrame.setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*);  
 startFrame.setSize(400, 72);  
 startFrame.setVisible(true);  
 }  
}

**Тесты**

**CalcTest.java;**

import lostSigns.model.Calc;  
import org.junit.Test;  
  
import java.util.Arrays;  
import java.util.List;  
  
import static org.junit.Assert.*assertEquals*;  
  
public class CalcTest {  
  
 @Test  
 public void calc() {  
 List<String> postfix = Arrays.*asList*("2", "2", "+", "2", "\*");  
 Integer result = 8;  
 *assertEquals*(result, Calc.*calc*(postfix));  
  
 postfix = Arrays.*asList*("3", "2", "\*", "2", "-");  
 result = 4;  
 *assertEquals*(result, Calc.*calc*(postfix));  
  
 postfix = Arrays.*asList*("2", "2", "+", "2", "+");  
 result = 6;  
 *assertEquals*(result, Calc.*calc*(postfix));  
  
 postfix = Arrays.*asList*("1", "1", "\*", "1", "\*");  
 result = 1;  
 *assertEquals*(result, Calc.*calc*(postfix));  
 }  
}

**StartControllerTest.java;**

import lostSigns.controller.StartController;  
  
import org.junit.Test;  
import org.junit.jupiter.api.Tag;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.Assert.\*;  
  
public class CheckStartTest {  
   
 @Test  
 @Tag("General")  
 public void checkTest() {  
 int check = StartController.*check*("2 2 2 = 2");  
 *assertEquals*(2, check);  
  
 check = StartController.*check*("2 2222 2 = 2");  
 *assertEquals*(2, check);  
  
 check = StartController.*check*("2 (2 2) = 2");  
 *assertEquals*(2, check);  
  
 check = StartController.*check*("2 2 2 = 2");  
 *assertEquals*(2, check);  
  
 check = StartController.*check*("(((2 (2 2)))) = 2");  
 *assertEquals*(2, check);  
  
 check = StartController.*check*(")2 (2 2) = 2");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("2 2 = 2 = 2");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("2 2 2 = 2k");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("2ss 2 2 = 2");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("2 2 2() = 2");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("2 2 2()");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("2 2 2 =");  
 *assertEquals*(1, check);  
  
 check = StartController.*check*("");  
 *assertEquals*(0, check);  
 }  
  
 @Test  
 public void checkEmptyTest() {  
 int check = StartController.*check*("");  
 *assertEquals*(0, check);  
 }  
  
 @Test  
 public void checkWithoutEquallyTest() {  
 int check = StartController.*check*("2 2 2");  
 *assertEquals*(1, check);  
 }  
  
 @Test  
 public void checkParenthesesTest() {  
 boolean check = StartController.*checkParentheses*("22 22 2");  
 *assertTrue*(check);  
  
 check = StartController.*checkParentheses*("((22 22) 2)");  
 *assertTrue*(check);  
  
 check = StartController.*checkParentheses*("((22 22) 2)");  
 *assertTrue*(check);  
  
 check = StartController.*checkParentheses*("(2(2 2)2) 2");  
 *assertTrue*(check);  
  
 check = StartController.*checkParentheses*("(2 (22 22)) 2");  
 *assertTrue*(check);  
  
 check = StartController.*checkParentheses*("2 (22 22)) 2");  
 *assertFalse*(check);  
  
 check = StartController.*checkParentheses*("(2) (22 22)( 2");  
 *assertFalse*(check);  
  
 check = StartController.*checkParentheses*("((2 (22 22)) 2");  
 *assertFalse*(check);  
 }  
  
 @Test  
 public void isIntegerTest() {  
 boolean check = StartController.*isInteger*('2');  
 *assertTrue*(check);  
  
 check = StartController.*isInteger*('5');  
 *assertTrue*(check);  
  
 check = StartController.*isInteger*('0');  
 *assertTrue*(check);  
  
 check = StartController.*isInteger*('(');  
 *assertFalse*(check);  
  
 check = StartController.*isInteger*(')');  
 *assertFalse*(check);  
  
 check = StartController.*isInteger*('=');  
 *assertFalse*(check);  
 }  
  
 @Test  
 public void checkArgumentTest() {  
 boolean check = StartController.*checkArgument*(" 2 2 2 2 2 2 2 2 2 ");  
 *assertTrue*(check);  
  
 check = StartController.*checkArgument*(" 2 2 2 2 + 2 2 2 2 2 ");  
 *assertFalse*(check);  
  
 check = StartController.*checkArgument*(" 2 2 2 2 { 2 2 2 2 2 ");  
 *assertFalse*(check);  
  
 check = StartController.*checkArgument*(" 2 2 2 2 2 1111111112 2 2 2 ");  
 *assertTrue*(check);  
  
 check = StartController.*checkArgument*(" 2w 2 2 2k 2 2e 2 2 2 ");  
 *assertFalse*(check);  
 }  
  
 @Test  
 public void checkAnswerTest() {  
 boolean check = StartController.*checkAnswer*(" 2 ");  
 *assertTrue*(check);  
  
 check = StartController.*checkAnswer*("1 1");  
 *assertFalse*(check);  
  
 check = StartController.*checkAnswer*("=");  
 *assertFalse*(check);  
  
 check = StartController.*checkAnswer*("(");  
 *assertFalse*(check);  
 }  
  
 @Test  
 public void deleteSpacesTest() {  
 *assertEquals*("22 22 22",  
 StartController.*deleteSpaces*(" 22 22 22 "));  
 *assertEquals*("(22 2) 2 22",  
 StartController.*deleteSpaces*(" ( 22 2)2 22 "));  
 *assertEquals*("(22 (22 22))",  
 StartController.*deleteSpaces*(" ( 22 (22 22) ) "));  
 }  
  
 @Test  
 public void addSymbolsTest() {  
 List<String> list = new ArrayList<>();  
 List<String> listTest = new ArrayList<>();  
  
 String example = "222(";  
 list.add("2");  
 list.add("2");  
 listTest.add("2");  
 listTest.add("2");  
 listTest.add("2");  
  
 *assertEquals*(listTest, StartController.*addSymbols*(example, 2, 2, list));  
  
 listTest.add("(");  
 *assertEquals*(listTest, StartController.*addSymbols*(example, 3, 3, list));  
 }  
  
 @Test  
 public void plusSymbols() {  
 List<String> list = new ArrayList<>();  
 List<String> listTest = new ArrayList<>();  
 String example = "2222";  
  
 StartController.*addSymbols*(example, 0, 0, list);  
 listTest.add("2222");  
  
 StartController.*plusSymbols*(example, 1, 0, list);  
 StartController.*plusSymbols*(example, 2, 0, list);  
  
 *assertEquals*(listTest, StartController.*plusSymbols*(example, 3, 0, list));  
 }  
  
 @Test  
 public void toStringListTest() {  
 List<String> list = new ArrayList<>();  
  
 list.add("2");  
 list.add("(");  
 list.add("2");  
 list.add("2");  
 list.add(")");  
 list.add("2");  
  
 *assertEquals*("2 (2 2) 2", StartController.*toStringList*(list));  
  
 list.add("2");  
 list.add("2");  
  
 *assertEquals*("2 (2 2) 2 2 2", StartController.*toStringList*(list));  
 }  
}

**CombinationOfSignsTest.java;**

import lostSigns.model.CombinationOfSigns;  
import org.junit.Test;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.Assert.*assertEquals*;  
  
public class CombinationOfSignsTest {  
  
 @Test  
 public void toCombinationSignsTest() {  
 List<String> combinationOfSignsFirst = new ArrayList<>();  
 combinationOfSignsFirst.add("\*\*");  
 combinationOfSignsFirst.add("+\*");  
 combinationOfSignsFirst.add("-\*");  
 combinationOfSignsFirst.add("\*+");  
 combinationOfSignsFirst.add("++");  
 combinationOfSignsFirst.add("-+");  
 combinationOfSignsFirst.add("\*-");  
 combinationOfSignsFirst.add("+-");  
 combinationOfSignsFirst.add("--");  
  
 *assertEquals*(combinationOfSignsFirst, CombinationOfSigns.*toCombinationSigns*(2));  
  
 List<String> combinationOfSignsSecond = new ArrayList<>();  
 combinationOfSignsSecond.add("\*");  
 combinationOfSignsSecond.add("+");  
 combinationOfSignsSecond.add("-");  
  
 *assertEquals*(combinationOfSignsSecond, CombinationOfSigns.*toCombinationSigns*(1));  
 }  
}

**ExamplesTest.java;**

import lostSigns.model.Examples;  
import org.junit.Test;  
import org.junit.jupiter.api.Tag;  
  
import java.util.ArrayList;  
import java.util.HashMap;  
import java.util.List;  
import java.util.Map;  
  
import static org.junit.Assert.*assertEquals*;  
  
public class ExamplesTest {  
  
 @Test  
 @Tag("General")  
 public void toSolve() {  
 Examples examples = new Examples(2, "2 2 2");  
 *assertEquals*("2-2+2=2", examples.toSolve());  
 examples = new Examples(8, "(2 2) 2");  
 *assertEquals*("(2\*2)\*2=8", examples.toSolve());  
 examples = new Examples(10, "(3 2) 2");  
 *assertEquals*("(3+2)\*2=10", examples.toSolve());  
 examples = new Examples(14, "((2 6) 2) 2");  
 *assertEquals*("((2+6)\*2)-2=14", examples.toSolve());  
 examples = new Examples(2, "(((2 2)) 2)");  
 *assertEquals*("(((2-2))+2)=2", examples.toSolve());  
 }  
  
 @Test  
 public void toPutSigns() {  
 String exampleFirst = "2 2 2";  
 String exampleSecond = "((2 2))";  
  
 List<String> expectedForFirst = new ArrayList<>();  
 expectedForFirst.add("2\*2\*2");  
 expectedForFirst.add("2+2\*2");  
 expectedForFirst.add("2-2\*2");  
 expectedForFirst.add("2\*2+2");  
 expectedForFirst.add("2+2+2");  
 expectedForFirst.add("2-2+2");  
 expectedForFirst.add("2\*2-2");  
 expectedForFirst.add("2+2-2");  
 expectedForFirst.add("2-2-2");  
  
 Examples examples = new Examples(2, "");  
  
 *assertEquals*(expectedForFirst, examples.toPutSigns(exampleFirst));  
  
 List<String> expectedForSecond = new ArrayList<>();  
 expectedForSecond.add("((2\*2))");  
 expectedForSecond.add("((2+2))");  
 expectedForSecond.add("((2-2))");  
  
 *assertEquals*(expectedForSecond, examples.toPutSigns(exampleSecond));  
 }  
  
 @Test  
 public void toCountSpacesTest() {  
 Examples examples = new Examples(2, "2 2 2");  
 *assertEquals*(3, examples.toCountSpaces("2 2 2 2"));  
 *assertEquals*(1, examples.toCountSpaces("22 22"));  
 *assertEquals*(3, examples.toCountSpaces("(2 2) 2 2"));  
 *assertEquals*(2, examples.toCountSpaces("22 (2 2)"));  
 }  
  
 @Test  
 public void toPutResponseTest() {  
 Map<Integer, String> examplesWithAnswers = new HashMap<>();  
 examplesWithAnswers.put(0,"2-2");  
 examplesWithAnswers.put(2, "2+2-2");  
 examplesWithAnswers.put(8, "(2+2)\*2");  
  
 Examples examples = new Examples(2, "");  
  
 Map<Integer, String> examplesWithSigns;  
 examples.toPutResponse(0, "2-2");  
 examples.toPutResponse(2, "2+2-2");  
 examplesWithSigns = examples.toPutResponse(8, "(2+2)\*2");  
 *assertEquals*(examplesWithAnswers, examplesWithSigns);  
 }  
  
 @Test  
 public void isEqualsTest() {  
 Map<Integer, String> examplesWithSigns = new HashMap<>();  
 examplesWithSigns.put(0, "2-2");  
 examplesWithSigns.put(1, "2-2+2-2+1");  
 examplesWithSigns.put(2, "2+2-2");  
 examplesWithSigns.put(3, "2+2+1");  
 examplesWithSigns.put(4, "2+2");  
 examplesWithSigns.put(5, "2+2+1");  
 examplesWithSigns.put(6, "(2+2)+2");  
 examplesWithSigns.put(7, "(2+2)\*2-1");  
 examplesWithSigns.put(8, "(2+2)\*2");  
  
  
 Examples examples = new Examples(2, "");  
  
  
 String check = examples.isEquals(examplesWithSigns, 2);  
 *assertEquals*("2+2-2=2", check);  
  
 check = examples.isEquals(examplesWithSigns, 3);  
 *assertEquals*("2+2+1=3", check);  
  
 check = examples.isEquals(examplesWithSigns, 4);  
 *assertEquals*("2+2=4", check);  
  
 check = examples.isEquals(examplesWithSigns, 5);  
 *assertEquals*("2+2+1=5", check);  
  
 check = examples.isEquals(examplesWithSigns, 6);  
 *assertEquals*("(2+2)+2=6", check);  
  
 check = examples.isEquals(examplesWithSigns, 7);  
 *assertEquals*("(2+2)\*2-1=7", check);  
  
 check = examples.isEquals(examplesWithSigns, 8);  
 *assertEquals*("(2+2)\*2=8", check);  
  
 check = examples.isEquals(examplesWithSigns, 10);  
 *assertEquals*("Решения нет", check);  
 }  
}

**PolishNotationTest.java;**

import lostSigns.model.PolishNotation;  
import org.junit.Test;  
import org.junit.jupiter.api.Tag;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.Assert.\*;  
  
  
public class PolishNotationTest {  
  
 @Test  
 public void isDelimetersTest() {  
 PolishNotation polishNotation = new PolishNotation();  
  
 String token = "(";  
 boolean i = polishNotation.isDelimiter(token);  
 *assertTrue*(i);  
  
 token = ")";  
 i = polishNotation.isDelimiter(token);  
 *assertTrue*(i);  
  
 token = "()";  
 i = polishNotation.isDelimiter(token);  
 *assertFalse*(i);  
  
 token = "&";  
 i = polishNotation.isDelimiter(token);  
 *assertFalse*(i);  
 }  
  
 @Test  
 public void isOperatorTest() {  
 PolishNotation polishNotation = new PolishNotation();  
  
 String token = "+";  
 boolean i = polishNotation.isOperator(token);  
 *assertTrue*(i);  
  
 token = "-";  
 i = polishNotation.isDelimiter(token);  
 *assertTrue*(i);  
  
 token = "\*";  
 i = polishNotation.isDelimiter(token);  
 *assertTrue*(i);  
  
 token = "()))";  
 i = polishNotation.isDelimiter(token);  
 *assertFalse*(i);  
 }  
  
 @Test  
 public void priorityTest() {  
 PolishNotation polishNotation = new PolishNotation();  
  
 int test = polishNotation.priority("(");  
 *assertEquals*(1, test);  
  
 test = polishNotation.priority("+");  
 *assertEquals*(2, test);  
  
 test = polishNotation.priority("-");  
 *assertEquals*(2, test);  
  
 test = polishNotation.priority("\*");  
 *assertEquals*(3, test);  
  
 test = polishNotation.priority(")");  
 *assertEquals*(4, test);  
 }  
  
 @Test  
 @Tag("General")  
 public void parseTest() {  
 PolishNotation polishNotation = new PolishNotation();  
 List<String> result = polishNotation.parse("2+2+2");  
 List<String> needInf = new ArrayList<>();  
 needInf.add("2");  
 needInf.add("2");  
 needInf.add("+");  
 needInf.add("2");  
 needInf.add("+");  
 *assertEquals*(needInf, result);  
  
 result = polishNotation.parse("(2+2)\*2");  
 needInf = new ArrayList<>();  
 needInf.add("2");  
 needInf.add("2");  
 needInf.add("+");  
 needInf.add("2");  
 needInf.add("\*");  
 *assertEquals*(needInf, result);  
  
 result = polishNotation.parse("(2+2)\*2+1");  
 needInf = new ArrayList<>();  
 needInf.add("2");  
 needInf.add("2");  
 needInf.add("+");  
 needInf.add("2");  
 needInf.add("\*");  
 needInf.add("1");  
 needInf.add("+");  
 *assertEquals*(needInf, result);  
 }  
}

**PutterSignsTest.java;**

import lostSigns.model.PutterSigns;  
import org.junit.Test;  
  
import java.util.ArrayList;  
import java.util.List;  
import static org.junit.Assert.*assertEquals*;  
  
public class PutterSignsTest {  
  
 @Test  
 public void putSigns() {  
  
 List<String> signs = new ArrayList<>();  
 signs.add("\*\*");  
 signs.add("\*+");  
 signs.add("\*-");  
 signs.add("+\*");  
 String example = "2 2 2";  
  
 List<String> expected = new ArrayList<>();  
 expected.add("2\*2\*2");  
 expected.add("2\*2+2");  
 expected.add("2\*2-2");  
 expected.add("2+2\*2");  
  
 *assertEquals*(expected, PutterSigns.*putSigns*(signs, example));  
 }  
}

**RepetitionsTest.java;**

import lostSigns.model.Repetitions;  
import org.junit.Test;  
  
import java.util.ArrayList;  
import java.util.List;  
  
import static org.junit.Assert.*assertEquals*;  
  
  
public class RepetitionInListTest {  
  
 @Test  
 public void repetitionTest() {  
 List<String> partOneInTestList = new ArrayList<>();  
 partOneInTestList.add("2");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 List<String> partTwoInTestList = new ArrayList<>();  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("+");  
 List<String> partThreeInTestList = new ArrayList<>();  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("+");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
  
 List<List<String>> testList = new ArrayList<>();  
 testList.add(partOneInTestList);  
 testList.add(partTwoInTestList);  
 testList.add(partThreeInTestList);  
  
 List<List<String>> expectedList = new ArrayList<>();  
 List<String> partOneInExpected = new ArrayList<>();  
 partOneInExpected.add("16");  
 List<String> partTwoInExpected = new ArrayList<>();  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("\*");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("\*");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("+");  
 List<String> partThreeInExpected = new ArrayList<>();  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("\*");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("+");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("\*");  
 expectedList.add(partOneInExpected);  
 expectedList.add(partTwoInExpected);  
 expectedList.add(partThreeInExpected);  
  
 Repetitions repetitionsInList = new Repetitions(16, true, testList);  
 *assertEquals*(expectedList, repetitionsInList.repetition());  
  
  
 partOneInTestList = new ArrayList<>();  
 partOneInTestList.add("2");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partTwoInTestList = new ArrayList<>();  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("+");  
 partThreeInTestList = new ArrayList<>();  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("+");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
  
 testList = new ArrayList<>();  
 testList.add(partOneInTestList);  
 testList.add(partTwoInTestList);  
 testList.add(partThreeInTestList);  
  
 expectedList = new ArrayList<>();  
 partOneInExpected = new ArrayList<>();  
 partOneInExpected.add("16");  
 partTwoInExpected = new ArrayList<>();  
 partTwoInExpected.add("10");  
 partThreeInExpected = new ArrayList<>();  
 partThreeInExpected.add("12");  
 expectedList.add(partOneInExpected);  
 expectedList.add(partTwoInExpected);  
 expectedList.add(partThreeInExpected);  
  
 repetitionsInList = new Repetitions(2, true, testList);  
 *assertEquals*(expectedList, repetitionsInList.repetition());  
 }  
  
 @Test  
 public void toDeleteRepetitionTest() {  
  
 List<String> partOneInTestList = new ArrayList<>();  
 partOneInTestList.add("2");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 List<String> partTwoInTestList = new ArrayList<>();  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("+");  
 List<String> partThreeInTestList = new ArrayList<>();  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("+");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
  
 List<List<String>> testList = new ArrayList<>();  
 testList.add(partOneInTestList);  
 testList.add(partTwoInTestList);  
 testList.add(partThreeInTestList);  
  
 List<String> partTwoExpected = new ArrayList<>();  
 partTwoExpected.add("16");  
 List<String> partOneExpected = new ArrayList<>();  
 partOneExpected.add("10");  
 List<String> partThreeExpected = new ArrayList<>();  
 partThreeExpected.add("12");  
  
 Repetitions repetitionInList = new Repetitions(16, true, testList);  
 *assertEquals*(partTwoExpected, repetitionInList.toDeleteRepetition(partOneInTestList, 0));  
 repetitionInList = new Repetitions(10, true, testList);  
 *assertEquals*(partOneExpected, repetitionInList.toDeleteRepetition(partTwoInTestList, 1));  
 repetitionInList = new Repetitions(12, true, testList);  
 *assertEquals*(partThreeExpected, repetitionInList.toDeleteRepetition(partThreeInTestList, 2));  
 }  
  
  
 @Test  
 public void toDeleteTest() {  
  
 List<String> signNeedToDelete = new ArrayList<>();  
 signNeedToDelete.add("2");  
 signNeedToDelete.add("2");  
 signNeedToDelete.add("\*");  
  
 List<String> partOneInTestList = new ArrayList<>();  
 partOneInTestList.add("2");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 List<String> partTwoInTestList = new ArrayList<>();  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("+");  
 List<String> partThreeInTestList = new ArrayList<>();  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("+");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
  
 List<List<String>> testList = new ArrayList<>();  
 testList.add(partOneInTestList);  
 testList.add(partTwoInTestList);  
 testList.add(partThreeInTestList);  
  
 List<String> partOneInExpected = new ArrayList<>();  
 partOneInExpected.add("4");  
 partOneInExpected.add("2");  
 partOneInExpected.add("\*");  
 partOneInExpected.add("2");  
 partOneInExpected.add("\*");  
 List<String> partTwoInExpected = new ArrayList<>();  
 partTwoInExpected.add("4");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("\*");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("+");  
 List<String> partThreeInExpected = new ArrayList<>();  
 partThreeInExpected.add("4");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("+");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("\*");  
  
 List<List<String>> expectedList = new ArrayList<>();  
 expectedList.add(partOneInExpected);  
 expectedList.add(partTwoInExpected);  
 expectedList.add(partThreeInExpected);  
  
  
 Repetitions repetitionInList = new Repetitions();  
 int numberToPut = 4;  
 *assertEquals*(expectedList, repetitionInList.toDelete(testList, signNeedToDelete, numberToPut, 0));  
  
  
 signNeedToDelete = new ArrayList<>();  
 signNeedToDelete.add("8");  
 signNeedToDelete.add("2");  
 signNeedToDelete.add("\*");  
  
 numberToPut = 16;  
  
 testList = new ArrayList<>();  
 partOneInTestList = new ArrayList<>();  
 partOneInTestList.add("8");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partTwoInTestList = new ArrayList<>();  
 partTwoInTestList.add("8");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("+");  
 partThreeInTestList = new ArrayList<>();  
 partThreeInTestList.add("4");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("+");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
  
 testList.add(partOneInTestList);  
 testList.add(partTwoInTestList);  
 testList.add(partThreeInTestList);  
  
 expectedList = new ArrayList<>();  
 partOneInExpected = new ArrayList<>();  
 partOneInExpected.add("16");  
 partTwoInExpected = new ArrayList<>();  
 partTwoInExpected.add("8");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("+");  
 partThreeInExpected = new ArrayList<>();  
 partThreeInExpected.add("4");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("+");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("\*");  
 expectedList.add(partOneInExpected);  
 expectedList.add(partTwoInExpected);  
 expectedList.add(partThreeInExpected);  
  
 *assertEquals*(expectedList, repetitionInList.toDelete(testList, signNeedToDelete, numberToPut, 0));  
  
 signNeedToDelete = new ArrayList<>();  
 signNeedToDelete.add("2");  
 signNeedToDelete.add("2");  
 signNeedToDelete.add("2");  
 signNeedToDelete.add("\*");  
 signNeedToDelete.add("+");  
  
 numberToPut = 8;  
  
 testList = new ArrayList<>();  
 partOneInTestList = new ArrayList<>();  
 partOneInTestList.add("2");  
 partOneInTestList.add("2");  
 partOneInTestList.add("2");  
 partOneInTestList.add("\*");  
 partOneInTestList.add("+");  
 partTwoInTestList = new ArrayList<>();  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("2");  
 partTwoInTestList.add("\*");  
 partTwoInTestList.add("-");  
 partThreeInTestList = new ArrayList<>();  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("+");  
 partThreeInTestList.add("2");  
 partThreeInTestList.add("\*");  
  
 testList.add(partOneInTestList);  
 testList.add(partTwoInTestList);  
 testList.add(partThreeInTestList);  
  
 expectedList = new ArrayList<>();  
 partOneInExpected = new ArrayList<>();  
 partOneInExpected.add("8");  
 partTwoInExpected = new ArrayList<>();  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("2");  
 partTwoInExpected.add("\*");  
 partTwoInExpected.add("-");  
 partThreeInExpected = new ArrayList<>();  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("+");  
 partThreeInExpected.add("2");  
 partThreeInExpected.add("\*");  
 expectedList.add(partOneInExpected);  
 expectedList.add(partTwoInExpected);  
 expectedList.add(partThreeInExpected);  
  
 *assertEquals*(expectedList, repetitionInList.toDelete(testList, signNeedToDelete, numberToPut, 0));  
 }  
}