Санкт-Петербургский политехнический университет Петра Великого
Институт компьютерных наук и технологи
Высшая школа интеллектуальных систем и суперкомпьютерных технологий

КУРСОВОЙ ПРОЕКТ

Дисциплина: Алгоритмы и структуры данных

Тема: разработка GUI приложения для восстановления пропущенных знаков в математическом выражении

Выполнил студент гр. 3530901/80003 _____ А.С.Джеус

Преподаватель _____ М.И.Глухих

2019 г.

Санкт-Петербург
2019

Оглавление

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Задан математический пример с положительными целыми числами и возможным использованием скобок (допускаются в том числе и вложенные скобки). Математические знаки примера “потеряны”. Написать программу, восстанавливающую знаки, если это возможно. Знаки могут быть: +, -, *.

Пример: 7 (5 3) 4 = 15
Ответ: 7+(5-3)*4 = 15

GitHub репозиторий проекта: https://github.com/MickeyMouseMouse/Course_Project_Semester3.

2. МЕТОД РЕШЕНИЯ

В программе использована концепция MVC (Model-View-Controller) для отделения бизнес-логики от визуализации. Весь код разбит на пять файлов: GUI.java (View), Controller.java (Controller), GUIDialogs.java (View), SignsFinder.java (Model), PartOfExpression.java (Model).

Класс GUI отвечает за визуальное представление приложения. Он содержит функцию main, задание параметров окна и расположение внутри него всех элементов графического интерфейса, а также события на взаимодействие с этими элементами.

Класс Controller содержит объявление всех графических элементов и методы для работы с ними. Так как работа бизнес-логики инициируется именно в этом классе и в общем случае может занимать длительное время, во избежание зависания графического интерфейса используется объект класса Service, который специально предназначен для решения подобной проблемы.

Класс GUIDialogs предназначен для отображения message box'ов — сообщений с той или иной информацией. В этом классе имеются следующие методы: showInformation, showError, showWarning для показа сообщений с информацией, ошибкой или предупреждением.

Классы SignsFinder и PartOfExpression реализуют бизнес-логику. Рассмотрим, как они работают более подробно: пусть пользователь задал входные данные и запустил процесс нахождения решения.

Для начала программа анализирует входную строку на наличие в ней лишних или, наоборот, недостающих пробелов. Если такие недочеты присутствуют, то они устранаются. Также каждая новая введенная строка сравнивается с предыдущей. Если строки совпадают, то процесс поиска решения не будет запущен повторно, так как ответ уже был найден и выведен ранее.

Далее строка проверяется на соответствие шаблону (используется регулярное выражение). Отдельно проверяется корректность расстановки скобок, если они есть.

После этого строка разбивается на составные части – экземпляры класса PartOfExpression. Каждая такая часть хранит в себе следующие данные: i) для числа – абсолютное значение этого числа, для открывающейся скобки – зарезервированное значение -2, для закрывающейся скобки – зарезервированное значение -1; ii) текущий знак перед числом или скобкой; iii) количество возможных знаков перед числом или скобкой (2:+,- или 3:+,-,*). Пример: 1 2 = 3. Перед единицей может быть только два знака (+ или -), перед двойкой может быть три знака (+,- или *). Также в PartOfExpression определены методы для работы с этими составными частями такие как: установка текущего знака, умножение и сложение, сравнение, инвертирование знака.

После разделения строки на составные части запускается рекурсивный перебор всех возможных расстановок знаков (от всех + до всех *). Для каждой расстановки сразу же считается значение выражения и сравнивается с требуемым результатом. Как только удается найти решение, происходит выход из рекурсии. После чего формируется результирующая строка с правильно расставленными знаками или сообщение, что такой строки не существует.

Рассмотрим методы класса SignsFinder:

- setInputString() - получает строку, которую нужно установить как входную; проверяет правильность форматирования строки и исправляет недочеты; возвращает true, если строка принята, или false, если строка является копией предыдущей строки.
- getInputString() - возвращает текущую установленную входную строку.

- isValid() - проверяет ранее полученную строку на соответсвие регулярному выражению и на корректность расстановки скобок; возвращает соответсвующее boolean значение.
- solve() - главная фунция в классе, инициирующая начало подбора решения; возвращает результирующую строку для вывода на экран.
- split() - разделяет ранее установленную входную строку на составные части – элементы класса PartOfExpression.
- findSolution() - получает номер текущего математического оператора, с которым предстоит работать; устанавливает знак (+, - или *) и, когда очередной набор знаков готов, инициирует вычисление значения выражения, которое потом сравнивает с искомым значением. Перебор всех комбинаций знаков реализован с помощью рекурсивного вызова функции.
- getResult() - получает два индекса, указывающих с какого по какой операнд нужно произвести вычисления; функция создает массив из элементов PartOfExpression, находящихся в одной скобочной группе. Если встречаются вложенные скобки, функция вызывает сама себя для очередной скобочной группы; возвращает значение выражения скобочной группы.
- calculate() - получает массив из элементов PartOfExpression (заведомо известно, что эти элементы состоят в одной скобочной группе); функция производит все необходимые операции (+, -, *); возвращает значение выражения.
- cancel() - функция, предназначенная для прерывания процесса подбора решения (используется для выхода из рекурсии, когда решение уже найдено, или по желанию пользователя).
- isSucceeded() - возвращает boolean значение, указывающее на наличие или отсутствие ошибки во время работы алгоритма на данный момент.
- makeAnswer() - формирует и возвращает результирующую строку.

Рассмотрим методы класса PartOfExpression:

- PartOfExpression() - в классе присутствуют два конструктора. Первый создает экземпляр класса, принимая на вход только знак и значение операнда. Второй контруктор принимает уже существующий экземпляр PartOfExpression и создает его копию.
- setNumberOfSigns() - принимает и устанавливает возможное количество знаков перед операндом.
- getNumberOfSigns() - возвращает возможное количество знаков перед операндом.
- setSign() - принимает 0,1 или 2 и устанавливает операнду соответствующий знак +,- или *.
- getSign() - возвращает код текущего знак операнда (0,1 или 2).
- invertSign() - меняет положительный знак операнда на отрицательный и наоборот.
- getValue() - возвращает текущее абсолютное значение операнда.
- compareTo() - получает объект, с которым необходимо сравнить текущий операнд, сравнивает и возвращает результат (-1 => ‘<’; 0 => ‘=’ ; 1 => ‘>’).
- multiply() - получает объект, на который нужно умножить текущий операнд, умножает и возвращает результат.
- plus() - получает объект, с которым нужно сложить текущий операнд, складывает и возвращает результат.

3. ЛИСТИНГ ПРОГРАММЫ

3.1 GUI.java

```
import javafx.application.Application;
import javafx.application.Platform;
import javafx.event.Event;
import javafx.geometry.HPos;
import javafx.geometry.Insets;
import javafx.scene.control.Tooltip;
import javafx.scene.image.Image;
```

```
import javafx.scene.image.ImageView;
import javafx.scene.input.*;
import javafx.scene.layout.ColumnConstraints;
import javafx.scene.layout.GridPane;
import javafx.scene.layout.Priority;
import javafx.scene.layout.RowConstraints;
import javafx.scene.text.Font;
import javafx.stage.Stage;

public class GUI extends Application {
    public static void main(String[] args) { Application.launch(args); }

    private final Controller cont = new Controller();

    @Override
    public void start(Stage mainStage) {
        cont.setAppIcon(new Image("icon.png"));

        mainStage.getIcons().add(cont.getAppIcon());
        mainStage.setTitle("Signs Finder");
        mainStage.centerOnScreen();
        mainStage.setScene(cont.scene);

        ColumnConstraints column = new ColumnConstraints(600,600, Double.MAX_VALUE);
        column.setHgrow(Priority.ALWAYS);
        cont.gridPane.getColumnConstraints().add(column);

        cont.gridPane.getRowConstraints().add(new RowConstraints(25)); // row for menu bar
        cont.gridPane.getRowConstraints().add(new RowConstraints(30)); // row for info label
        cont.gridPane.getRowConstraints().add(new RowConstraints(60)); // row for input field
        cont.gridPane.getRowConstraints().add(new RowConstraints(25)); // row for button start
        cont.gridPane.getRowConstraints().add(new RowConstraints(60)); // row for output field

        GridPane.setRowIndex(cont.menuBar, 0); // 0 row = menu bar
```

```
        GridPane.setRowIndex(cont.infoLabel, 1); // 1 row = info label
        GridPane.setRowIndex(cont.inputField, 2); // 2 row = input field
        GridPane.setMargin(cont.inputField, new Insets(10));
        GridPane.setRowIndex(cont.buttonStart, 3); // 3 row = button start
        GridPane.setRowIndex(cont.progressIndicator, 3);
        GridPane.setRowIndex(cont.outputField, 4); // 4 row = output field
        GridPane.setMargin(cont.outputField, new Insets(10));
        cont.gridPane.getChildren().addAll(cont.menuBar, cont.infoLabel, cont.inputField,
                cont.buttonStart, cont.progressIndicator, cont.outputField);
        Platform.runLater(cont.inputField::requestFocus); // set focus on inputField
        GridPane.setHalignment(cont.infoLabel, HPos.CENTER);
        GridPane.setHalignment(cont.buttonStart, HPos.CENTER);
        GridPane.setHalignment(cont.progressIndicator, HPos.CENTER);

        mainStage.show();
        mainStage.setMinWidth(mainStage.getWidth());
        mainStage.setMinHeight(mainStage.getHeight());
        mainStage.setMaxHeight(mainStage.getHeight());

        cont.menuBar.getMenus().addAll(cont.menuSignsFinder,
                cont.menuFile, cont.menuProcess);
        cont.menuSignsFinder.getItems().addAll(cont.about, cont.github,
                cont.separator, cont.quit);
        cont.menuFile.getItems().addAll(cont.open, cont.save);
        cont.menuProcess.getItems().addAll(cont.start, cont.cancel);

        cont.about.setOnAction((e) -> cont.showInfo());
        cont.github.setOnAction((e) -> cont.openGitHub());
        cont.quit.setOnAction((e) -> cont.closeApp());

        cont.open.setOnAction((e) -> cont.openFile());
        cont.open.setAccelerator(new KeyCodeCombination(KeyCode.O,
KeyCombination.SHORTCUT_DOWN));
        cont.save.setOnAction((e) -> cont.saveFile());
```

```
        cont.save.setAccelerator(new KeyCodeCombination(KeyCode.S,
KeyCombination.SHORTCUT_DOWN));

        cont.cancel.setDisable(true);

        cont.infoLabel.setFont(Font.font(null, 19));

        cont.inputField.setFont(Font.font(null, 20));
        cont.inputField.setPromptText("Input");

        cont.buttonStart.graphicProperty()
                .setValue(new ImageView(new Image("start.png")));
        cont.buttonStart.setStyle("-fx-padding: 0 0 0 0; -fx-background-radius: 90;");
        cont.buttonStart.setTooltip(new Tooltip("Start solving"));

        cont.progressIndicator.setVisible(false);

        cont.outputField.setFont(Font.font(null, 20));
        cont.outputField.setPromptText("Output");
        cont.outputField.addEventFilter(ContextMenuEvent.CONTEXT_MENU_REQUESTED,
Event::consume);

        // start Drag&Drop
        cont.scene.setOnDragOver((e) -> {
            if (e.getDragboard().hasFiles())
                e.acceptTransferModes(TransferMode.ANY);
        });

        // event on drop new file
        cont.scene.setOnDragDropped((e) ->
            cont.getDragAndDropFile(e.getDragboard().getFiles())
        );

        cont.start.setOnAction((e) -> cont.start());
```

```
        cont.cancel.setOnAction((e) -> cont.cancel());

        cont.inputField.setOnKeyPressed((e) -> {
            if (e.getCode() == KeyCode.ENTER) cont.start();
        });

        cont.buttonStart.setOnMouseClicked((e) -> cont.start());

        // prevent editing of the output field
        cont.outputField.setOnKeyPressed((e) -> {
            if (e.getCode() == KeyCode.BACK_SPACE) e.consume();
        });

        // prevent editing of the output field
        cont.outputField.setOnKeyTyped((e) -> {
            if (e.getCode() != KeyCode.LEFT && e.getCode() != KeyCode.RIGHT) e.consume();
        });
    }
}
```

3.2 Controller.java

```
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.layout.GridPane;
import javafx.stage.FileChooser;
import javafx.stage.Stage;
import java.awt.Desktop;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.net.URI;
```

```
import java.net.URISyntaxException;
import java.util.*;
import javafx.concurrent.Service;
import javafx.concurrent.Task;

public class Controller {
    final MenuBar menuBar = new MenuBar();

    final Menu menuSignsFinder = new Menu("Signs Finder");
    final MenuItem about = new MenuItem("About");
    final MenuItem github = new MenuItem("View Source On Github");
    final SeparatorMenuItem separator = new SeparatorMenuItem();
    final MenuItem quit = new MenuItem("Quit");

    final Menu menuFile = new Menu("File");
    final MenuItem open = new MenuItem("Open...");
    final MenuItem save = new MenuItem("Save Result As...");

    final Menu menuProcess = new Menu("Process");
    final MenuItem start = new MenuItem("Start");
    final MenuItem cancel = new MenuItem("Cancel");

    final GridPane gridPane = new GridPane();
    final Scene scene = new Scene(gridPane);
    final Label infoLabel = new Label("\"File -> Open...\" / Drag&Drop / Type here to set the input
string");
    final TextField inputField = new TextField();
    final Button buttonStart = new Button();
    final ProgressIndicator progressIndicator = new ProgressIndicator();
    final TextField outputField = new TextField();

    private SignsFinder model = new SignsFinder();
    private Image appIcon;
    private final GUIDialogs dialogs = new GUIDialogs();
```

```
    public void setAppIcon(Image icon) {
        appIcon = icon;
        dialogs.setDialogIcon(icon);
    }

    public Image getAppIcon() { return appIcon; }

    public void showInfo() {
        dialogs.showInformation("Developer: Andrew Jeus\n" +
                "Course 2, Group №3530901/80003");
    }

    public void openGitHub() {
        try {
            Desktop.getDesktop().browse(new
URI("https://github.com/MickeyMouseMouse/Course_Project_Semester3"));
        } catch (URISyntaxException | IOException e) {
            dialogs.showError("Error 1: Can't open the link");
        }
    }

    public void closeApp() { System.exit(0); }

    public void openFile() {
        File file = showOpenFileChooser();
        if (file != null) getInputStringFromFile(file);
    }

    public void saveFile() {
        File file = showSaveFileChooser();

        if (file == null) return;
```

```
        try (FileWriter writer = new FileWriter(file.getAbsolutePath())) {
            writer.write(outputField.getText());
        } catch (IOException e) {
            dialogs.showError("Error 3: File save failed");
        }
    }

    private File showOpenFileChooser() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Open Input File");
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Text file",
"*.txt"));
        return fileChooser.showOpenDialog(new Stage());
    }

    private File showSaveFileChooser() {
        FileChooser fileChooser = new FileChooser();
        fileChooser.setTitle("Save Output File");
        fileChooser.getExtensionFilters().addAll(new FileChooser.ExtensionFilter("Text file",
"*.txt"));
        return fileChooser.showSaveDialog(new Stage());
    }

    public void getDragAndDropFile(List<File> file) {
        if (file.size() != 1) {
            dialogs.showError("Error 4: Only ONE file can be selected");
            return;
        }

        if (file.get(0).getName().matches("(.*)\\.(txt)"))
            getInputStringFromFile(file.get(0));
        else
            dialogs.showError("Error 5: Only *.txt file can be selected");
    }
```

```
private void getInputStringFromFile(File inputFile) {
    String inputString = "";

    try (Scanner scanner = new Scanner(inputFile)) {
        if (scanner.hasNext()) inputString = scanner.nextLine();
    } catch (FileNotFoundException e) {
        dialogs.showError("Error 2: File not found");
    }

    inputField.setText(inputString);
    outputField.setText("");
}

public void start() {
    if (process.isRunning()) return;

    if (inputField.getText().length() == 0) {
        outputField.setText("");
        dialogs.showWarning("Warning 2: The input string is empty");
        return;
    }

    if (!model.setInputString(inputField.getText())) {
        dialogs.showInformation("This is already done. Look at the output field.");
        return;
    }

    outputField.setText("");
    if (!model.isValid()) {
        dialogs.showWarning("Warning 1: The input string isn't valid\n" +
                "Example: 7 ( 5 3 ) 4 = 15 -> 7+( 5-3 )*4 = 15");
    } else {
        setGUISettings(false);
```

```
        process.reset();
        process.start();

        process.setOnSucceeded((e) -> {
            setGUISettings(true);

            if (model.isSucceeded()) {
                outputField.setText((String) process.getValue());
            } else {
                dialogs.showError("Error 6: " + process.getValue() +
                        " (Integer type is used, 4 bytes)");
            }
        });

        process.setOnCancelled((e) -> setGUISettings(true));
    }
}

// The Service class is used to prevent the GUI from freezing
// if it takes a long time to find a solution.
private Service process = new Service() {
    @Override
    protected Task createTask() {
        return new Task() {
            @Override
            protected String call() {
                return model.solve();
            }
        };
    }
};

public void cancel() {
    model.cancel();
```

```
        process.cancel();
    }

    private void setGUISettings(boolean mode) {
        open.setDisable(!mode);
        save.setDisable(!mode);

        start.setDisable(!mode);
        cancel.setDisable(mode);

        inputField.setEditable(mode);
        buttonStart.setVisible(mode);
        progressIndicator.setVisible(!mode);
    }
}
```

3.3 GUIDialogs.java

```
import javafx.scene.control.Alert;
import javafx.scene.control.DialogPane;
import javafx.scene.image.Image;
import javafx.stage.Stage;

public class GUIDialogs {
    private Alert alert;
    private Image icon;

    public void setDialogIcon(Image icon) { this.icon = icon; }

    public void showError(String errorText) {
        alert = new Alert(Alert.AlertType.ERROR);
        show(errorText);
    }

    public void showWarning(String warningText) {
```

```
        alert = new Alert(Alert.AlertType.WARNING);
        show(warningText);
    }

    public void showInformation(String infoText) {
        alert = new Alert(Alert.AlertType.INFORMATION);
        show(infoText);
    }

    private void show(String text) {
        DialogPane dialogPane = alert.getDialogPane();
        if (icon != null)
            ((Stage) dialogPane.getScene().getWindow()).getIcons().add(icon);
        dialogPane.setStyle("-fx-font-size: 15px;");

        alert.setHeaderText(null);
        alert.setContentText(text);
        alert.showAndWait();
    }
}
```

3.4 SignsFinder.java

```
import java.util.List;
import java.util.ArrayList;

public class SignsFinder {
    private String inputString;
    private List<PartOfExpression> leftPart;
    private PartOfExpression rightPart;

    private boolean done; // use for exit from main recursion
    private boolean solution; // true = there is solution, false = there is no solution
    private boolean isCanceled; // the process may be terminated
    private String error; // remember run time error
```

```
// true = new string has been set; false = new string is duplicate of old one
public boolean setInputString(String inputStr) {
    StringBuilder str = new StringBuilder();

    // remove extra spaces
    boolean space = true;
    for (char ch : inputStr.toCharArray())
        if (ch == ' ') {
            if (!space) space = true;
        } else {
            if (space) {
                str.append(" ");
                space = false;
            }
            str.append(ch);
        }

    // insert missing spaces before/after brackets and equals
    int i = 1;
    while (i < str.length() - 1) {
        if (str.charAt(i) == '(' || str.charAt(i) == ')' || str.charAt(i) == '=') {
            if (str.charAt(i - 1) != ' ') {
                str.insert(i, ' ');
                i++;
            }

            if (str.charAt(i + 1) != ' ') {
                str.insert(i + 1, ' ');
                i++;
            }
        }

        i++;
```

```
    }

    // compare new input string with old
    String result = str.toString();
    if (result.equals(inputString))
        return false;
    else {
        inputString = result;
        return true;
    }
}

// use for tests in Tests.java
public String getInputString() { return inputString; }

public boolean isValid() {
    if (!inputString.matches("( ([0-9]+|\\(|\\)))+( = [0-9]+)$")) {
        inputString = null;
        return false;
    }

    // extra check for brackets (right number of opening and closing brackets)
    int brackets = 0;
    for (int i = 0; i < inputString.length(); i++) {
        if (inputString.charAt(i) == '(') {
            brackets++;
            if (inputString.charAt(i + 2) == ')') {
                brackets = -1;
                break;
            }
        }
        if (inputString.charAt(i) == ')') brackets--;
        if (brackets < 0) break;
    }
```

```
    if (brackets == 0)
        return true;
    else {
        inputString = null;
        return false;
    }
}

// start finding a solution
public String solve() {
    done = false;
    solution = false;
    isCanceled = false;
    error = null;

    split();
    if (isSucceeded()) findSolution(0);
    return makeAnswer();
}

private void split() {
    // left part = right part
    String[] parts = inputString.split("= ");

    // value of expression (right part)
    try {
        rightPart = new PartOfExpression((byte) 0, Integer.parseInt(parts[1]));
    } catch (NumberFormatException e) {
        cancel();
        error = parts[1];
        return;
    }
```

```
// left part
leftPart = new ArrayList<>();
PartOfExpression part;
String str = parts[0];
for (int i = 0; i < str.length() - 1; i++) {
   if (str.charAt(i + 1) == ')') {
      leftPart.add(new PartOfExpression((byte) -1, -1));
      i++;
      continue;
   }

   byte numberOfSigns;
   if (i == 0 || str.charAt(i - 1) == '('
      numberOfSigns = 2;
   else
      numberOfSigns = 3;

   if (str.charAt(i + 1) == '(') {
      part = new PartOfExpression((byte) -1, -2);
      i++;
   } else {
      int j = i + 1;
      while (str.charAt(j) != ' ')
         j++;

      try {
      part = new PartOfExpression((byte) -1,
            Integer.parseInt(str.substring(i + 1, j)));
      } catch (NumberFormatException e) {
         cancel();
         error = str.substring(i + 1, j);
         return;
      }
      i = j - 1;
```

```
        }

        part.setNumberOfSigns(numberOfSigns);
        leftPart.add(part);
    }
}

// 0 = '+'; 1 = '-'; 2 = '*'
private void findSolution(int index) {
    for (byte i = 0; i < leftPart.get(index).getNumberOfSigns(); i++) {
        if (done) break; // exit from the recursion
        leftPart.get(index).setSign(i);

        if (index != leftPart.size() - 1)
            findSolution(index + 1);
        else
            if (getResult(0, leftPart.size()).compareTo(rightPart) == 0) {
                done = true;
                solution = true;
            }
    }
}

private PartOfExpression getResult(int start, int stop) {
    List<PartOfExpression> expression = new ArrayList<>();
    for (int i = start; i < stop; i++)
        if (leftPart.get(i).getValue() != -2) { // -2 = '('; -1 = ')'
            expression.add(new PartOfExpression(leftPart.get(i)));
        } else {
            int j = i + 1;
            int brackets = 1;
            while (true) {
                if (leftPart.get(j).getValue() == -2) brackets++;
                if (leftPart.get(j).getValue() == -1)
```

```
                if (--brackets == 0) break;
            j++;
        }

        PartOfExpression part = getResult(i + 1, j);
        if (leftPart.get(i).getSign() == 1) part.invertSign();

        if (leftPart.get(i).getSign() == 2) { // 2 = '*'
            if (part.getSign() == 1) // 1 = '-'
                expression.get(expression.size() - 1).invertSign();
            part.setSign((byte) 2);
        }

        expression.add(part);
        i = j;
    }

    return calculate(expression);
}

private PartOfExpression calculate(List<PartOfExpression> expression) {
    PartOfExpression result = new PartOfExpression((byte) 0, 0);
    for (int i = 0; i < expression.size(); i++) {
        if (i != expression.size() - 1 && expression.get(i + 1).getSign() == 2) { // 2 = '*'
            PartOfExpression productOfNumbers = expression.get(i);
            while (i != expression.size() - 1 && expression.get(i + 1).getSign() == 2)
                if (productOfNumbers.multiply(expression.get(i++ + 1))) {
                    cancel();
                    error = "Overflow";
                    break;
                }

            if (done) break;
```

```
            if (result.plus(productOfNumbers)) {
                cancel();
                error = "Overflow";
                break;
            }
        } else
            if (result.plus(expression.get(i))) {
                cancel();
                error = "Overflow";
                break;
            }
    }

    return result;
}

public void cancel() {
    isCanceled = true;
    done = true;
    inputString = null;
}

public boolean isSucceeded() { return error == null; }

// make the required string from ArrayList<PartOfExpression>
private String makeAnswer() {
    if (!isSucceeded()) return error;
    if (isCanceled) return "";
    if (!solution) return "There is no solution";

    StringBuilder result = new StringBuilder();

    for (int i = 0; i < leftPart.size(); i++) {
        if (leftPart.get(i).getValue() == -1) {
```

```
            result.append(" )");
            continue;
        }

        switch (leftPart.get(i).getSign()) {
            case 0:
                if (i != 0 && leftPart.get(i - 1).getValue() != -2)
                    result.append("+");
                else
                    result.append(" ");
                break;
            case 1:
                result.append("-");
                break;
            case 2:
                result.append("*");
                break;
        }

        if (leftPart.get(i).getValue() == -2)
            result.append("(");
        else
            result.append(leftPart.get(i).getValue());
    }

    result.append(" = ").append(rightPart.getValue());
    return result.toString();
  }
}
```

3.5 PartOfExpression.java

```
public class PartOfExpression implements Comparable {
  private byte numberOfSigns = 1; // 1 = default, 2 = +/-, 3 = +/-/*
  private byte sign; // -1 = no sign, 0 = +, 1 = -, 2 = *
```

```
private int value; // >= 0 => number, -2 = opening bracket, -1 = closing bracket

public PartOfExpression(byte sign, int value) {
    this.sign = sign;
    this.value = value;
}

public PartOfExpression(PartOfExpression part) {
    numberOfSigns = part.numberOfSigns;
    sign = part.sign;
    value = part.value;
}

public void setNumberOfSigns(byte number) { numberOfSigns = number; }

public byte getNumberOfSigns() { return numberOfSigns; }

public void setSign(byte sign) { this.sign = sign; }

public byte getSign() { return sign; }

public void invertSign() { if (sign == 0) sign = 1; else sign = 0; }

public int getValue() { return value; }

// -1 => this < obj; 0 => this == obj; 1 => this > obj
@Override
public int compareTo(Object obj) {
    if (this == obj) return 0;
    if (obj == null) throw new NullPointerException();

    PartOfExpression other = (PartOfExpression) obj;
    int first, second;
    if (sign == 0) first = value; else first = -value;
```

```
    if (other.sign == 0) second = other.value; else second = -other.value;

    return Integer.compare(first, second);
}

// true = overflow; false = no
public boolean multiply(PartOfExpression other) {
    long product = (long) value * (long) other.value;

    if (product <= Integer.MAX_VALUE) {
        value = (int) product;
        return false;
    }

    return true;
}

// true = overflow; false = no
public boolean plus(PartOfExpression other) {
    long first;
    long second;

    if (sign == 0) first = value; else first = -value;
    if (other.sign == 0) second = other.value; else second = -other.value;

    first += second;
    if (first <= Integer.MAX_VALUE) {
        value = Math.abs((int) first);
        if (first >= 0) sign = 0; else sign = 1;
        return false;
    }

    return true;  } }
```

4. ОШИБКИ И ПРЕДУПРЕЖДЕНИЯ

Errors:

Код ошибки	Описание
1	Can’t open the link; не удалось открыть GitHub репозиторий с текущим проектом в браузере.
2	File not found; не удалось открыть/прочитать требуемый файл со входными данными.
3	File save failed; не удалось сохранить результат в требуемый файл.
4	Only ONE file can be selected; пользователь попытался перетащить (Drag&Drop) более одного файла в окно приложения.
5	Only *.txt file can be selected; пользователь попытался перетащить (Drag&Drop) не *.txt файл в окно приложения.
6	‘some value’ or Overflow (Integer type is used, 4 bytes); во входных данных или во время вычисления появилось переполнение в int переменной.

Warnings:

Код предупреждения	Описание
1	Input string isn't valid Example: 7 (5 3) 4 = 15 -> 7+(5-3)*4 = 15; пользователь ввел некорректную входную строку.
2	Input string is empty; пользователь ввел пустую строку.

5. ТЕСТЫ

Все функции в файле Tests.java тестируют методы класса SignsFinder.java, то есть бизнес-логику.

wrongFormattedInputString() проверяет работоспособность функции setInputString(). На вход setInputString() подаются неверно отформатированные строки, которые в результате обработки становятся пригодными для дальнейшего алгоритма.

badInputString() проверяет работоспособность функции isValid(). На вход isValid() подаются некорректные входные данные, проверяется правильность результата работы функции.

thereIsSolution() и thereIsNoSolution() проверяют работоспособность всего алгоритма по поиску решения. Сравниваются полученные решение с ожидаемыми.

overflow() проверяет способность алгоритма выявлять переполнения в заданных входных данных.

overflowDuringCalculation() проверяет способность алгоритма выявлять переполнения, возникающие в процессе вычислений.

Tests.java

```
import org.junit.Test;

import static org.junit.Assert.assertEquals;
import static org.junit.Assert.assertFalse;

public class Tests {
   private SignsFinder model = new SignsFinder();

   @Test
   public void wrongFormattedInputString() {
      model.setInputString("1=1");
      assertEquals(" 1 = 1", model.getInputString());

      model.setInputString("1    2  3=   5");
      assertEquals(" 1 2 3 = 5", model.getInputString());

      model.setInputString(" (((1 2)3   4) 5)=6");
      assertEquals(" ( ( ( 1 2 ) 3 4 ) 5 ) = 6", model.getInputString());
   }

   @Test
   public void badInputString() {
      model.setInputString("");
      assertFalse(model.isValid());

      model.setInputString("1 2 3");
      assertFalse(model.isValid());

      model.setInputString("( 1 2 ( 3 ) = 4");
      assertFalse(model.isValid());

      model.setInputString("( 1 2 ( 3 ) ) = ");
      assertFalse(model.isValid());

      model.setInputString("1 2 3 4 p = 24");
      assertFalse(model.isValid());
```

```
        model.setInputString("1 2 3 4 + 3 = 24");
        assertFalse(model.isValid());

        model.setInputString("1 2 3 4 3 = -24");
        assertFalse(model.isValid());
    }

    @Test
    public void thereIsSolution() {
        model.setInputString("2 ( 2 3 4 ( 4 5 ) 3 ) = 15");
        if (model.isValid())
            assertEquals(" 2+( 2+3+4+(-4+5 )+3 ) = 15", model.solve());
        else
            throw new RuntimeException("Input string isn't valid");
    }

    @Test
    public void thereIsNoSolution() {
        model.setInputString("2 3 3 ( 5 ( 6 3 ) ) = 28");
        if (model.isValid())
            assertEquals("There is no solution", model.solve());
        else
            throw new RuntimeException("Input string isn't valid");
    }

    @Test
    public void overflow() {
        model.setInputString("1 10000000000 1 = 1"); // 10 000 000 000 is a very large number for 4
bytes
        if (model.isValid())
            assertEquals("10000000000", model.solve());
        else
            throw new RuntimeException("Input string isn't valid");
    }

    @Test
    public void overflowDuringCalculation() {
        model.setInputString("100000 100000 = 1"); // 100 000 * 100 000 = 10 000 000 000
        if (model.isValid())
            assertEquals("Overflow", model.solve());
        else
            throw new RuntimeException("Input string isn't valid");
    }
}
```

6. СКРИНШОТЫ ПРОГРАММЫ

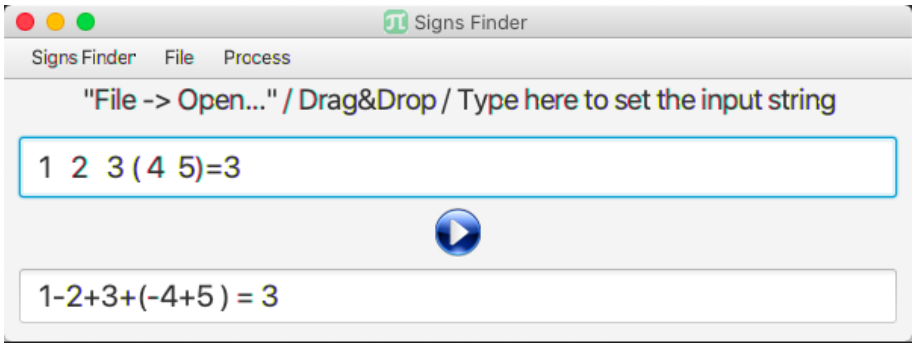

Рис. 1 Решение успешно найдено

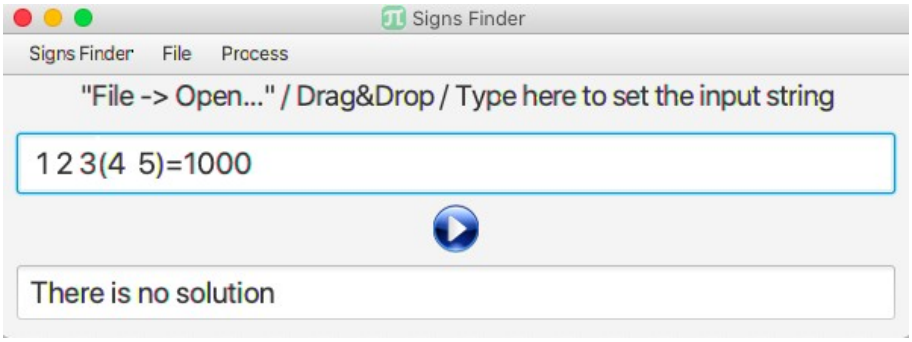

Рис. 2 Решение не существует

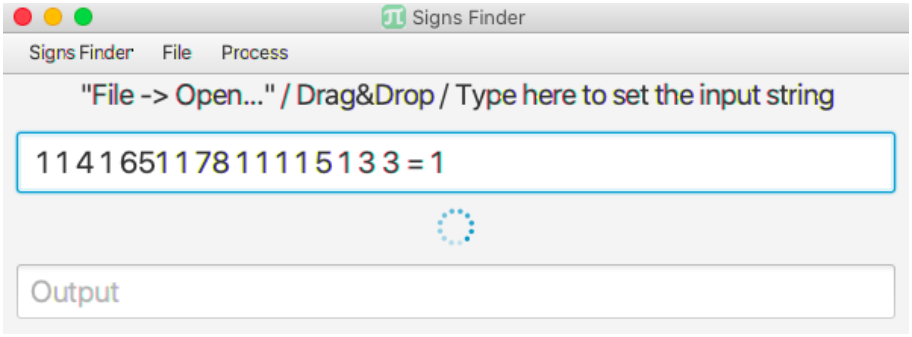

Рис. 3 Процесс нахождения решения (тот случай, когда перебор занимает ощутимое время)

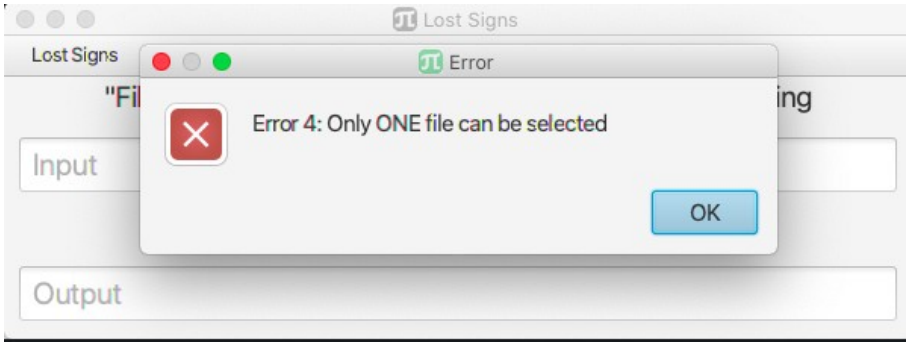

Рис. 4 Ошибка, вызванная попыткой перетащить (Drag&Drop) более одного файла в окно программы

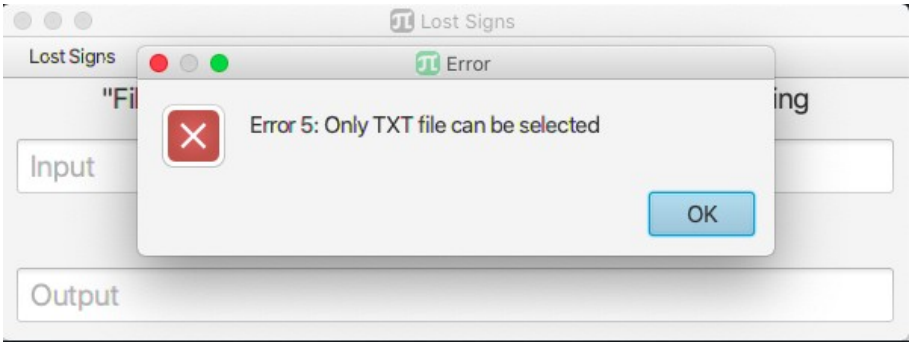

Рис. 5 Ошибка, вызванная попыткой перетащить (Drag&Drop) не *.txt файл в окно программы

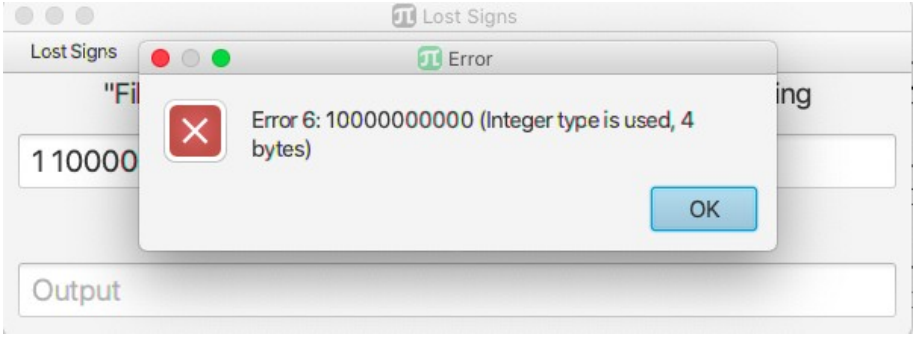

Рис. 6 Ошибка, вызванная переполнением во входной строке

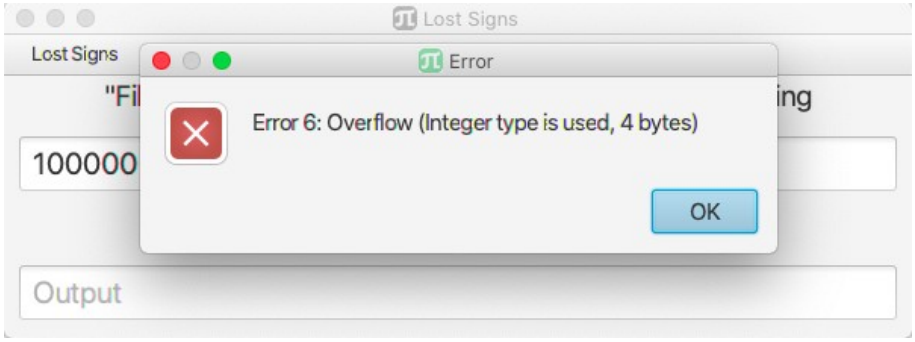

Рис. 7 Ошибка, вызванная переполнением во время вычисления

Рис. 8 Предупреждение о неверном формате входной строки

Рис. 9 Предупреждение о том, что задана пустая строка