Оглавление

Введение

В данной пояснительной записке рассматривается описание приложения «Калькулятор интегралов», разработанного при помощи применения объектно-ориентированного подхода в программировании.

«Программа – это набор объектов, указывающих друг другу, что делать, посредством сообщений», – пишет Брюс Эккель («Философия Java». – С. 42).

То есть, каждый объект обладает определенным состоянием (другими словами, переменными экземпляра, или атрибутами), поведением (методами, или, – следуя терминологии процедурного стиля программирования, – функциями), а также индивидуальностью (например, уникальным адресом в памяти).

Следует заметить: в Java (почти) все является объектом, – и этот факт, несомненно, положительно выделяет этот язык программирования на фоне своих конкурентов из семейства смешанных ЯП. «Хотя Java основан на C++, он является более «чистокровным» объектно-ориентированным языком», – Брюс Эккель («Философия Java». – С. 70).

Благодаря ООП появляется возможность описать решаемую задачу в контексте, собственно, самой задачи, а не в контексте компьютера, на котором будет исполняться решение.

«Чтобы объяснить природу ООП, прибегают к трем волшебным словам: инкапсуляция, наследование и полиморфизм», – Роберт Мартин («Чистая архитектура». – С. 55).

В качестве основного инструмента используется интегрированная среда разработки – IntelliJ IDEA 2023.1.1. Язык программирования – Java SE 11.0.18.

1 Разработка компьютерного приложения

1.1 Разработка технического задания

1.1.1 Полное наименование системы и её условное обозначение

Полное наименование

«Разработка приложения «Калькулятор интегралов» для вычисления интегралов, нахождения корней уравнений и выполнения простейших арифметических операций».

Условное обозначение системы

«Калькулятор интегралов».

1.1.2 Основание для разработки

Основанием для разработки данной информационной системы является приказ от НГТУ им Р. Е. Алексеева на выполнение выпускной квалификационной работы по дисциплине «Специалист по информационным технологиям».

1.1.3 Назначение и цели создания системы

Назначение системы

Программный продукт предназначен для предоставления ответов на такие математические операции, которые требуют особого внимания при ручном счете и нуждаются в машинной проверке, – а также для выполнения простейших арифметических операций.

Цели создания системы

1) Упрощение работы с математическими моделями;

2) Предоставление помощи в изучении некоторых разделов алгебры.

С помощью данного приложения пользователь сможет осуществить вышеперечисленные цели.

Основные задачи разработки

1) Обеспечить возможность редактирования математических выражений;

2) Обеспечить возможность выбора определенной арифметической операции;

3) Обеспечить сохранение истории вычислений при перезапуске приложения.

1.2 Анализ технического задания

1.2.1 Общие положения

Согласно техническому заданию, необходимо разработать приложение для компьютера, которое предназначено для выполнения различных математических функций над заданным выражением.

1.2.2 Требования к системе

Требования к структуре и функционированию системы

Программный продукт, разрабатываемый в рамках выпускной квалификационной работы, должен удовлетворять следующему перечню функциональных требований:

1) Наличие удобного и понятного графического интерфейса;

2) Возможность редактирования математических выражений;

3) Сохранение пользовательской истории при перезапуске приложения.

Входные данные

Входными данными при работе с программным продуктом должны быть нажатия на левую кнопку мыши и, в некоторых случаях, нажатия на клавиатуру.

Выходные данные

Выходными данными при работе настольного приложения являются ответы, полученные применением математической функции к заданному пользователем выражению.

Требования к надёжности

Информационная система должна сохранять работоспособность и обеспечивать восстановление своих функций при возникновении внештатных ситуаций.

Требования к эргономике и технической эстетике

Подсистема ввода данных, подсистема формирования и визуализации

выходных данных должны обеспечивать удобный для конечного пользователя интерфейс.

Главное окно программного продукта должно позволять пользователю видеть прогресс записи выражения и варианты математических операций, которые возможно применить к нему.

1.3 Стандарты

Программный продукт разрабатывается на основании следующих государственных стандартов:

1) ГОСТ 2.103-68 ЕСКД. Стадии разработки;

2) ГОСТ 2.104-68 ЕСКД. Основные надписи;

3) ГОСТ 2.105-95 ЕСКД. Общие требования к текстовым документам;

4) ГОСТ 2.111-68 ЕСКД. Нормоконтроль;

5) ГОСТ 2.118-73 ЕСКД. Техническое предложение;

6) ГОСТ 2.120-73 ЕСКД. Технический проект;

7) ГОСТ 2.316-68 ЕСКД. Правила нанесения на чертежах надписей, технических требований и таблиц;

8) ГОСТ 7.1-2003. Система стандартов по информации, библиотечному и издательскому делу. Библиографическая запись. Библиографическое описание. Общие требования и правила составления.

1.4 Требования к техническим средствам

Пользовательская система должна обеспечивать отображение ГПИ программы, что накладывает требование на установленную версию Java SE Development Kit 11.0.18 (и выше) в пользовательской системе.

Минимальные технические характеристики пользовательской системы:

1) Процессор (ЦП) – Intel Core i5 с тактовой частотой 1,6 ГГц;

2) Оперативная память (ОЗУ) – 1 Гбайт;

3) Свободная память жесткого диска (ПЗУ) – 100 Мбайт;

4) Монитор – разрешение 800x600 пикселей.

2 Разработка модели предметной области

2.1 Анализ предметной области

Программируемое настольное приложение должно позволять пользователю записывать и редактировать выражения, выполнять с ними математические операции и просматривать историю вычислений.

Запись выражения необходимо поддержать двумя способами: выбором мышью встроенных в главное окно приложения цифр и – при работе с функциями – непосредственно вводом с клавиатуры. Редактирование выражения также следует поддержать двумя способами.

Должны быть реализованы все арифметические операции – 6 штук (согласно М. Я. Выгодскому, «Справочник по элементарной математике». – С. 69-72). А также интегрирование на заданном отрезке методом прямоугольников (численный метод, заключающийся в замене подынтегральной функции на многочлен нулевой степени) и нахождение корней уравнения методом бисекции (численный метод для решения уравнений вида $f(x) = 0$).

«Интегральное исчисление возникло из потребности создать общий метод нахождения площадей, объемов и центров тяжести», – М. Я. Выгодский («Справочник по высшей математике». – С. 327).

Реализация сохранения истории вычислений предполагает существование некоторого определенного файла с его фиксированным относительным путем. Фиксированный файловый путь, может быть, и уменьшает гибкость настройки, но увеличивает оперативность процессов просмотра и изменения истории вычислений вручную.

Нефункциональные требования к программному средству:

1) Наличие удобного и интуитивно понятного пользовательского интерфейса;

2) Наличие минимальных зависимостей от аппаратного обеспечения и периферийных средств.

Окно пользовательского интерфейса необходимо выполнить в

фиксированном размере, чтобы улучшить эффективность работы с данным настольным приложением посредством механического запоминания расположения кнопок.

Благодаря переносимости скомпилированного Java-кода, файлы с расширением class (по сути, весь программный продукт) возможно будет запустить на любой ОС.

2.2 Разработка структуры классов

В соответствие с анализом предметной области можно выделить основные классы информационной системы и их атрибуты:

1) Главный публичный класс *Calculator* должен содержать: метод, возвращающий ссылку на экземпляр главного текстового поля калькулятора, – и конструктор, связанный с приватным методом для реализации возможности взаимодействия пользователя с программным продуктом;

2) Класс *FunctionForm* с пакетным доступом должен содержать: приватные переменные экземпляра, являющиеся составной частью этой формы для заполнения, – и конструктор, в котором будет выполняться работа по обработке введенных пользователем данных для численного интегрирования или поиска корней уравнения;

3) Класс *MathematicalAction* с пакетным доступом должен содержать: статические переменные экземпляра и одну приватную финальную переменную, которая является ссылкой на экземпляр главного класса, – и методы, отвечающие за логику взаимодействия с такими компонентами, как цифры и арифметические операции;

4) Класс *TextualAction* с пакетным доступом должен содержать: приватную финальную переменную экземпляра, которая является ссылкой на экземпляр главного класса, – и методы, отвечающие за логику взаимодействия с такими компонентами, как, например, кнопка очищения поля ввода;

5) Утилитный класс *FileHandler* с пакетным доступом должен

содержать: статические методы, позволяющие работать с файлами (в данном случае, с историей вычислений);

6) Перечисление *Operation* с пакетным доступом должно содержать: абстрактный метод для проведения некоторой математической операции, переопределенный в каждом из элементов перечисления (интегрирование, умножение, сумма и пр.).

Данное приложение необходимо построить, следуя правильным принципам разработки программного обеспечения. Принципы SOLID позволяют создавать программные структуры (модули) простыми и гибкими к изменениям. Эти модули можно использовать в качестве основы для дальнейшей разработки.

Явно хотелось бы выделить самый первый принцип, Single Responsibility Principle. «Модуль должен иметь одну и только одну причину для изменения», – так описывает его Роберт Мартин («Чистая архитектура». – С. 79).

Следуя данному принципу, разработка классов будет проходить качественно, – и в будущем их реализация если и потребует изменений, то только незначительных.

3 Разработка структуры приложения

3.1 Разработка архитектуры

В данном информационном продукте разработаны несколько классов с описанием различных графических форм и инфраструктура классов, отвечающая за арифметические операции, открытие и сохранение файлов.

При открытии приложения пользователь видит интерфейс главного окна класса *Calculator*, содержащий поле ввода, панель со всеми цифрами и панель с арифметическими операциями.

Отдельно выделены операции интегрирования и нахождения корней уравнения. При выборе одной из них открывается дополнительная графическая форма для задания отрезка и записи выражения.

Все операнды и проводимая операция записываются в поля класса *MathematicalAction*, что позволяет перед сохранением выражения и результата в историю вычислений проверить, соответствует ли проведенная только что операция интегрированию или нахождению корней уравнения, – только в данном случае будет произведена запись в файл.

3.2 Проектирование пользовательского интерфейса и взаимодействие с ним

Данное программное обеспечение имеет удобный пользовательский интерфейс, выполненный в теплой цветовой гамме (рис 3.1).

Рис. 3.1 – Пользовательский интерфейс главного окна приложения

Запись и удаление выражения реализованы интуитивно понятными способами (рис 3.2).

Рис. 3.2 – Пользовательский интерфейс главного окна приложения при редактировании выражения

Результат вычисления представляется в одном окне ввода-вывода (рис. 3.3).

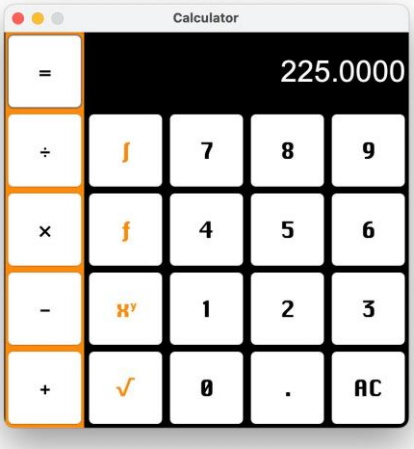

Рис. 3.3 – Пользовательский интерфейс главного окна приложения после применения арифметической операции

Отдельно поддержана функциональность численного интегрирования и поиска корней уравнения. Лаконично выполненные поля ввода позволяют

собрать информацию для дальнейших вычислений (рис 3.4).

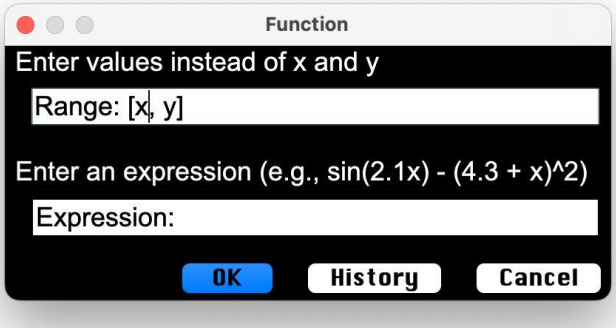

Рис. 3.4 – Пользовательский интерфейс дополнительного окна приложения для численного интегрирования (или для поиска корней уравнения)

Пример использования данной графической формы с введенными значениями (рис 3.5).

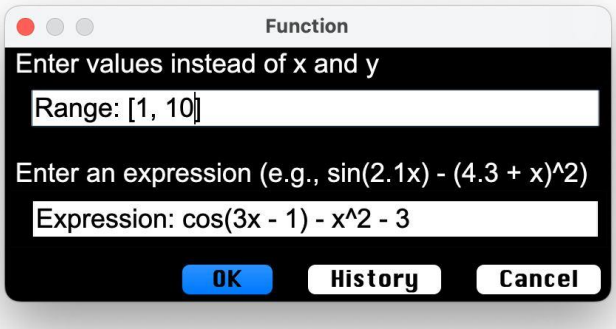

Рис. 3.5 – Пользовательский интерфейс дополнительного окна приложения для численного интегрирования с введенными данными

Открытие истории вычислений представляет собой непосредственное открытие файла (рис 3.6). Данное решение объясняется выгодной гибкостью внесения любых пометок в историю вычислений.

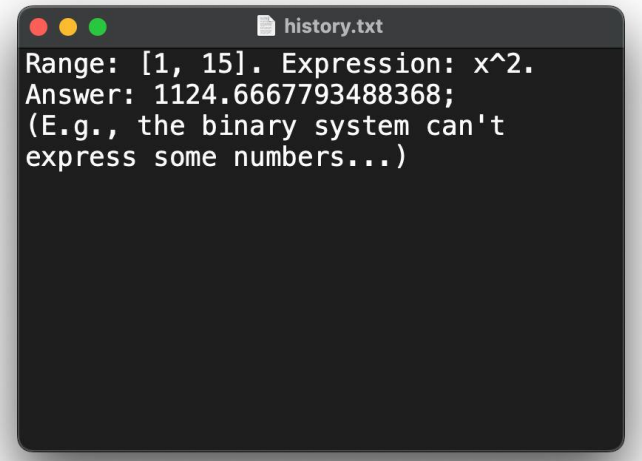

Рис. 3.6 – Интерфейс отображения истории вычислений в виде текстового файла

Основной сценарий взаимодействия пользователя с программным продуктом:

1) Пользователь вводит первый операнд, используя панель цифр в интерфейсе главного окна приложения, затем выбирает арифметическую операцию, вводит следующий операнд, – и получает ответ в общем поле ввода-вывода;

2) При необходимости более сложных вычислений пользователь выбирает одну из встроенных возможностей (интегрирование или поиск корней уравнения), после чего вводит требуемый отрезок и выражение в интерфейсе дополнительно открывшегося окна;

3) При желании просмотреть историю вычислений, пользователь может нажать кнопку «History», которая открывает текстовый документ с существующей историей стандартным текстовым редактором данной ОС.

Заключение

В данной выпускной квалификационной работе было разработано приложение «Калькулятор интегралов» на основе объектно-ориентированного подхода в программировании.

«Объектно-ориентированное программирование позволяет расширять приложение, не затрагивая проверенный ранее и работающий код», – таким образом отзываются об этом подходе Кэти Сьерра и Берт Бейтс («Изучаем Java». – С. 71).

В реализации программного обеспечения широко использовалась инкапсуляция данных, наследование было применено в перечислении, а вытекающий из этого полиморфизм был выгодно использован в алгоритме обработки пользовательских данных.

Дополнительно были написаны модульные тесты. Использовалась методология TDD – полезная, современная и удобная. Невозможно обойти стороной написание тестового кода, т. к. «тестовый код не менее важен, чем код продукта» (согласно Р. Мартину, «Чистый код». – С. 153).

«Процесс познания начинается с ощущений, возникающих в результате непосредственного воздействия предметов и явлений материального мира на органы чувств», – С. Н. Виноградов («Логика: учебник для средней школы». – С. 11). Поэтому отличными решениями при расширении данного программного обеспечения были бы:

1) Внедрение различных тем оформления пользовательского интерфейса;

2) Реализация протоколирования (например, на файловом уровне и на уровне сетевого соединения при возникновении фатальных ошибок);

3) Упаковка настольного приложения в JAR-архив с дальнейшим конвертированием его в EXE-файл;

4) Добавление функциональности интегрирования по двум и более переменным и возможности более гибкой записи выражений;

5) Улучшение быстродействия вычислений, используя целочисленные

примитивные типы данных во всех местах программы, где это возможно (что, с другой стороны, увеличивает количество программного кода).

При этом в данной работе удалось реализовать все поставленные задачи:

1) Обеспечить возможность редактирования математических выражений;

2) Обеспечить возможность выбора определенной арифметической операции;

3) Обеспечить сохранение истории вычислений при перезапуске приложения.

Данный информационный продукт упрощает процесс работы с математическими моделями и способствует плодотворному изучению некоторых разделов алгебры.

В итоге, было реализовано пользовательское приложение «Калькулятор интегралов». Оно предназначено для предоставления ответов на такие математические операции, которые требуют особого внимания при ручном счете и нуждаются в машинной проверке, – а также оно предназначено для выполнения простейших арифметических операций.

Список литературы

1) Виноградов С. Н., Кузьмин А. Ф. – Логика: Учебник для средней школы – Москва: Концептуал, 2020. – 176 с.

2) Выгодский М. Я. – Справочник по высшей математике – Москва: Издательство АСТ, 2022. – 703 с.

3) Выгодский М. Я. – Справочник по элементарной математике – Москва: Издательство АСТ, 2022. – 509 с.

4) Сьерра К., Бейтс Б. – Изучаем Java – 2-е изд. – Москва: Эксмо, 2022. – 720 с.

5) Мартин Р. – Чистая архитектура. Искусство разработки программного обеспечения – СПб.: Питер, 2022. – 352 с.

6) Мартин Р. – Чистый код: создание, анализ и рефакторинг – СПб.: Питер, 2022. – 464 с.

7) Эккель Б. – Философия Java. 4-е полное изд. – СПб.: Питер, 2023. – 1168 с.

8) https://stackoverflow.com/

9) https://wikipedia.org/

10) https://www.baeldung.com/

11) https://www.oracle.com/

Приложения

Приложение 1: код класса *Calculator*

```
package com.nntu;

import javax.swing.*;

/**
 * Represents the calculator's desktop window.
 */
public class Calculator extends JFrame {

    private JPanel mainPanel;
    private JLabel IOField;

    /* Buttons for mathematical actions */
    private JButton zero;
    private JButton one;
    private JButton two;
    private JButton three;
    private JButton four;
    private JButton five;
    private JButton six;
    private JButton seven;
    private JButton eight;
    private JButton nine;
    private JButton additionSign;
    private JButton divisionSign;
    private JButton exponentiationSign;
    private JButton integralSign;
    private JButton multiplicationSign;
    private JButton rootOfEquationSign;
    private JButton rootSign;
    private JButton subtractionSign;
    private JButton equalsSign;

    /* Buttons for textual actions */
    private JButton allClearSign;
    private JButton pointSign;

    /**
     * Initializes a frame for the application.
     * Shows the full application window to a user.
     */
    public Calculator() {
        setContentPane(mainPanel);
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setResizable(false);
        setSize(410,428);
        setTitle("Calculator");
        setVisible(true);
```

```
        initializeListeners();
    }

    /**
     * Initializes all listeners for this application.
     */
    private void initializeListeners() {
        /* Handlers for actions */
        MathematicalAction mathematicalAction = new MathematicalAction(this);
        TextualAction textualAction = new TextualAction(this);

        /* Mathematical actions */
        zero.addActionListener(mathematicalAction::digitAction);
        one.addActionListener(mathematicalAction::digitAction);
        two.addActionListener(mathematicalAction::digitAction);
        three.addActionListener(mathematicalAction::digitAction);
        four.addActionListener(mathematicalAction::digitAction);
        five.addActionListener(mathematicalAction::digitAction);
        six.addActionListener(mathematicalAction::digitAction);
        seven.addActionListener(mathematicalAction::digitAction);
        eight.addActionListener(mathematicalAction::digitAction);
        nine.addActionListener(mathematicalAction::digitAction);

        additionSign.addActionListener(mathematicalAction::operationAction);
        divisionSign.addActionListener(mathematicalAction::operationAction);
        exponentiationSign.addActionListener(mathematicalAction::operationAction);
        integralSign.addActionListener(mathematicalAction::operationAction);
        multiplicationSign.addActionListener(mathematicalAction::operationAction);
        rootOfEquationSign.addActionListener(mathematicalAction::operationAction);
        rootSign.addActionListener(mathematicalAction::operationAction);
        subtractionSign.addActionListener(mathematicalAction::operationAction);

        equalsSign.addActionListener(mathematicalAction::resultAction);

        /* Textual actions */
        allClearSign.addActionListener(textualAction::allClearAction);

        pointSign.addActionListener(textualAction::pointAction);
    }

    /**
     * Returns the main field of the calculator.
     *
     * @return the calculator's main field
     */
    JLabel getIOField() {
        return IOField;
    }
}
```

Приложение 2: код класса *FunctionForm*

```
package com.nntu;

import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;

/**
 * Represents the form's desktop window to get some information for future calculations.
 */
class FunctionForm extends JDialog {

    static final Path HISTORY_FILE = Path.of("history.txt");

    private JPanel mainPanel;
    private JButton cancelButton;
    private JButton historyButton;
    private JButton okButton;
    private JTextField expressionField;
    private JTextField rangeField;

    FunctionForm() {
        setContentPane(mainPanel);
        setDefaultCloseOperation(DO_NOTHING_ON_CLOSE);
        setModal(true);
        getRootPane().setDefaultButton(okButton);
        setResizable(false);
        setSize(410,200);
        setTitle("Function");

        cancelButton.addActionListener((actionEvent) -> closeAction());
        historyButton.addActionListener((actionEvent) -> {
            try {
                FileHandler.openFile(HISTORY_FILE);
            } catch (IOException ioe) {
                try {
                    Files.createFile(HISTORY_FILE);
                    FileHandler.openFile(HISTORY_FILE);
                } catch (IOException e) {
                    System.err.println("Couldn't open and/or create a history file.");
                    e.printStackTrace();
                }
            }
        });
        okButton.addActionListener((actionEvent) -> {
            int indexOfComma = rangeField.getText().lastIndexOf(',');
            int indexOfFirstBrace = 7;
```

```
            int indexOfLastBrace = rangeField.getText().lastIndexOf(']');

            MathematicalAction.firstNumber = rangeField.getText().substring(indexOfFirstBrace +
1, indexOfComma);
            MathematicalAction.secondNumber = rangeField.getText().substring(indexOfComma +
2, indexOfLastBrace);

            int lengthOfSentence = 11;
            MathematicalAction.expression = expressionField.getText().substring(lengthOfSentence
+ 1);

            dispose();
        });

        addWindowListener(new WindowAdapter() {
            @Override
            public void windowClosing(WindowEvent windowEvent) {
                closeAction();
            }
        });

        // Process the tapped ESCAPE
        mainPanel.registerKeyboardAction((actionEvent) -> closeAction(),
                KeyStroke.getKeyStroke(KeyEvent.VK_ESCAPE, 0),
JComponent.WHEN_ANCESTOR_OF_FOCUSED_COMPONENT);
    }

    /**
     * Closes the form's desktop window. Nothing will be saved.
     */
    private void closeAction() {
        dispose();
    }

    public static void main(String[] args) {
        new FunctionForm().setVisible(true);
    }
}
```

Приложение 3: код класса *MathematicalAction*

```
package com.nntu;

import net.objecthunter.exp4j.tokenizer.UnknownFunctionOrVariableException;

import java.awt.event.ActionEvent;
import java.io.IOException;

/**
 * Represents the mathematical logic for a user's input.
 */
class MathematicalAction {
```

```
static String firstNumber;
static String expression;
static String secondNumber;
static Operation operation;

private final Calculator calculator;

MathematicalAction(Calculator calculator) {
   this.calculator = calculator;
}

/**
 * Prints a specified digit to the calculator's field.
 *
 * @param actionEvent an event
 */
void digitAction(ActionEvent actionEvent) {
   String tappedDigit = actionEvent.getActionCommand();
   String currentNumber = calculator.getIOField().getText();

   // Deal with an empty calculator's field
   // And with behavior after user's selected operation
   if (currentNumber.equals("0")
         || operation != null && currentNumber.equals(firstNumber)) {
      calculator.getIOField().setText(tappedDigit);
   } else calculator.getIOField().setText(currentNumber + tappedDigit);
}

/**
 * Remembers a specified operation for next calculations.
 *
 * @param actionEvent an event
 */
void operationAction(ActionEvent actionEvent) {
   firstNumber = calculator.getIOField().getText();

   switch (actionEvent.getActionCommand()) {
      case "+":
         operation = Operation.ADDITION;
         break;
      case "÷":
         operation = Operation.DIVISION;
         break;
      case "∫":
         operation = Operation.INTEGRATION;
         FunctionForm.main(new String[0]);
         calculator.getIOField().setText("press =");
         break;
      case "×":
         operation = Operation.MULTIPLICATION;
         break;
      case "xʸ":
```

```
            operation = Operation.EXPONENTIATION;
            break;
        case "√":
            operation = Operation.ROOT;
            break;
        case "f":
            operation = Operation.ROOT_OF_EQUATION;
            FunctionForm.main(new String[0]);
            calculator.getIOField().setText("press =");
            break;
        case "-":
            operation = Operation.SUBTRACTION;
            break;
        default:
            throw new UnsupportedOperationException("There's no such mathematical
operation.");
      }
   }

   /**
    * Prints the result to the calculator's field.
    *
    * @param actionEvent an event
    */
   void resultAction(ActionEvent actionEvent) {
      if (operation != null) {
         if (operation != Operation.INTEGRATION && operation !=
Operation.ROOT_OF_EQUATION) {
            secondNumber = calculator.getIOField().getText();
         }

         double result = 0;
         try {
            result = operation.calculate(
                  Double.parseDouble(firstNumber), Double.parseDouble(secondNumber)
            );
            calculator.getIOField().setText(String.format("%.3f", result));

            // Remember the result for next calculations
            firstNumber = calculator.getIOField().getText();

            // Save the expression to a history if it's needed
            if (operation == Operation.INTEGRATION || operation ==
Operation.ROOT_OF_EQUATION) {
               FileHandler.saveExpression(FunctionForm.HISTORY_FILE, firstNumber,
secondNumber,
                     expression, String.valueOf(result));
            }
         } catch (ArithmeticException ae) {
            calculator.getIOField().setText(ae.getMessage());
         } catch (IOException ioe) {
            // Try again
            try {
```

```
                FileHandler.saveExpression(FunctionForm.HISTORY_FILE, firstNumber,
secondNumber,
                        expression, String.valueOf(result));
                } catch (IOException e) {
                    System.err.println("Couldn't save the expression to a history file.\n");
                    e.printStackTrace();
                }
            } catch (UnknownFunctionOrVariableException ufove) {
                calculator.getIOField().setText("Invalid expression.");
            }
        }
    }
}
```

Приложение 4: код класса *TextualAction*

```
package com.nntu;

import java.awt.event.ActionEvent;

/**
 * Represents the textual logic for a user's input.
 */
class TextualAction {

    private final Calculator calculator;

    TextualAction(Calculator calculator) {
        this.calculator = calculator;
    }

    /**
     * Prints a point to the calculator's field.
     *
     * @param actionEvent an event
     */
    void pointAction(ActionEvent actionEvent) {
        String currentNumber = calculator.getIOField().getText();
        if (!currentNumber.contains(".")) calculator.getIOField().setText(currentNumber + ".");
    }

    /**
     * Clears everything from the calculator's field.
     *
     * @param actionEvent an event
     */
    void allClearAction(ActionEvent actionEvent) {
        calculator.getIOField().setText("0");
        MathematicalAction.firstNumber = null;
        MathematicalAction.expression = null;
        MathematicalAction.secondNumber = null;
        MathematicalAction.operation = null;
```

```
    }
}
```

Приложение 5: код класса *FileHandler*

```
package com.nntu;

import java.io.FileWriter;
import java.io.IOException;
import java.nio.charset.StandardCharsets;
import java.nio.file.Path;

/**
 * Represents a utility handler for writing, opening and saving files or expressions.
 */
class FileHandler {

    /**
     * Opens in a new desktop window the specified file.
     *
     * @param path file to open
     * @throws IOException if an I/O error occurs
     */
    static void openFile(Path path) throws IOException {
        new ProcessBuilder()
                .command("open", path.toString())
                .start();
    }

    /**
     * Saves to the specified file
     * the specified mathematical expression with its range and its result.
     *
     * @param path file to save to
     * @param lowerBoundOfRange the left border of the range
     * @param upperBoundOfRange the right border of the range
     * @param expression the mathematical expression
     * @param answer the result of calculating
     *
     * @throws IOException if an I/O error occurs
     */
    static void saveExpression(Path path, String lowerBoundOfRange, String
upperBoundOfRange,
                               String expression, String answer) throws IOException {
        try (FileWriter writer = new FileWriter(path.toString(), StandardCharsets.UTF_8, true)) {
            writer.write("Range: [" + lowerBoundOfRange + ", " + upperBoundOfRange + "]. ");
            writer.write("Expression: " + expression + ". ");
            writer.write("Answer: " + answer + ";\n");
        }
    }
}
```

Приложение 6: код перечисления *Operation*

```
package com.nntu;

import net.objecthunter.exp4j.Expression;
import net.objecthunter.exp4j.ExpressionBuilder;

/**
 * Provides arithmetic operations.
 */
enum Operation {

    ADDITION {
        @Override
        double calculate(double first, double second) {
            return first + second;
        }
    },
    DIVISION {
        @Override
        double calculate(double first, double second) {
            return first / second;
        }
    },
    EXPONENTIATION {
        @Override
        double calculate(double first, double second) {
            return Math.pow(first, second);
        }
    },
    INTEGRATION {
        /**
         * Calculates the integral at a specified interval.
         *
         * @param start the start of an interval
         * @param end the end of an interval
         */
        @Override
        double calculate(double start, double end) {
            // maintain integration in both directions
            boolean isRangeReversed = false;

            if (end < start) {
                isRangeReversed = true;
                double temp = end;
                end = start;
                start = temp;
            }

            Expression expression = new ExpressionBuilder(MathematicalAction.expression)
                    .variable("x").build();
```

```
            double step = 1e-3;
            double sum = 0;
            while (start <= end) {
                sum += step * expression.setVariable("x", start).evaluate();
                start += step;
            }

            return isRangeReversed
                    ? sum * (-1)
                    : sum;
        }
    },
    MULTIPLICATION {
        @Override
        double calculate(double first, double second) {
            return first * second;
        }
    },
    ROOT {
        @Override
        double calculate(double first, double second) {
            return Math.pow(first, 1 / second);
        }
    },
    ROOT_OF_EQUATION {
        private double lowerBound;
        private double upperBound;

        /**
         * Calculates the root of equation at a specified interval using bisection method.
         *
         * @param start the start of an interval
         * @param end the end of an interval
         */
        @Override
        double calculate(double start, double end) {
            if (end < start) throw new ArithmeticException("Wrong range.");
            lowerBound = start;
            upperBound = end;
            return calculateRoot(start, end);
        }

        /**
         * Recursively finds the root of equation at a specified interval.
         */
        private double calculateRoot(double start, double end) {
            double epsilon = 1e-3;
            if (end - start <= epsilon) return start;

            double x = start + (end - start) / 2;

            if (Math.abs(lowerBound - x) <= epsilon || Math.abs(upperBound - x) <= epsilon)
                throw new ArithmeticException("There's no root.");
```

```
            Expression expression = new ExpressionBuilder(MathematicalAction.expression)
                    .variable("x").build();

            if (expression.setVariable("x", start).evaluate()
                    * expression.setVariable("x", x).evaluate() > 0) {
                return calculateRoot(x, end);
            } else return calculateRoot(start, x);
        }
    },
    SUBTRACTION {
        @Override
        double calculate(double first, double second) {
            return first - second;
        }
    };

    /**
     * Calculates some result using two specified numbers.
     *
     * @param first the first number
     * @param second the second number
     * @return the result
     */
    abstract double calculate(double first, double second);
}
```