МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**КУРСОВАЯ РАБОТА**

по дисциплине: “**Программирование на языке Java**”

на тему: **«Частотный словарь словосочетаний букв алфавита»**

Выполнил**:** студент группы 10702221

Парфенова У. Д.

Руководитель**:** доц. Сидорик В.В.

Минск 2023

МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет информационных технологий и робототехники

Кафедра программного обеспечения информационных систем и технологий

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовой работе

по дисциплине: **“Программирование на языке Java”**

на тему: **«Частотный словарь сочетаний букв алфавита»**

Выполнил**:** студент группы 10702221

Парфенова У. Д.

Руководитель**:** доц. Сидорик В.В.

Минск 2023

Оглавление

[**ВВЕДЕНИЕ** 4](#_Toc153198119)

1. [**ПОСТАНОВКА И ОПИСАНИЕ ЗАДАЧИ** 5](#_Toc153198120)

2. [**ТЕОРЕТИЧЕКИЙ РАЗДЕЛ** 6](#_Toc153198121)

* 1. [Расположение букв на клавиатуре 6](#_Toc153198122)

3. [**МАТЕМАТИЧЕСКАЯ МОДЕЛЬ** 7](#_Toc153198123)

4. [**БИЗНЕС-ЛОГИКА ПРОЕКТА** 9](#_Toc153198124)

5. [**ПРАКТИЧЕСКИЙ РАЗДЕЛ** 11](#_Toc153198125)

[5.1 Описание структуры программы 11](#_Toc153198126)

[5.2 UML-диаграммы классов 11](#_Toc153198127)

[5.3 Структура проекта в Intelij IDEA 11](#_Toc153198128)

[**6. ДЕМОНСТРАЦИЯ РАБОТЫ ПРИЛОЖЕНИЯ** 11](#_Toc153198129)

[**ЗАКЛЮЧЕНИЕ** 19](#_Toc153198130)

[**СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ** 20](#_Toc153198131)

[**ПРИЛОЖЕНИЕ 1** 21](#_Toc153198132)

# **ВВЕДЕНИЕ**

В современном информационном обществе изучение языковых явлений является важным направлением в лингвистике и информационных технологиях. Одним из методов анализа языковых структур является частотный анализ, позволяющий выявить особенности и закономерности в использовании языка. В рамках данной курсовой работы мы обратим внимание на частотный словарь сочетаний букв алфавита, что представляет собой интересное исследование, направленное на выявление предпочтительных комбинаций символов в текстовых данных.

Изучение частотных словарей является ключевым компонентом анализа текстов, так как это позволяет не только понять структуру языка, но и применять эту информацию в различных областях, таких как обработка естественного языка, криптография, создание эффективных алгоритмов сжатия данных и многие другие. Частотный анализ сочетаний букв дает возможность выделить наиболее часто встречающиеся биграммы и триграммы, что имеет важное значение при разработке алгоритмов сжатия, а также в анализе и синтезе текстовых данных.

Программа позволяет анализировать текст, искать сочетание букв и слов в тексте, сохранять полученные результаты, а также на основе результатов строит график для наглядного отображения частоты использования букв в русском и английском алфавите.

# **ПОСТАНОВКА И ОПИСАНИЕ ЗАДАЧИ**

**Условие задачи**: разработать приложение, определяющее частоту использования сочетания (по две) букв русского или английского алфавита. Исходные тексты содержатся в текстовых файлах. Проведите анализ и сравнение с вашей клавиатурой.

**Цель программы**: произвести анализ сочетаний букв русского и английского алфавита в тексте, вывести статистику результата анализа.

**Исходные данные**:

1. Операционная система Windows 10;
2. Язык программирования Java;
3. Библиотека Swing, AWT;
4. Среда разработки Intellij IDEA 2023.2.5

**Исходные данные для графика**: текст для анализа.

**Выходные данные**: файл с результатом анализа, график частот использования букв в алфавитах.

# **ТЕОРЕТИЧЕКИЙ РАЗДЕЛ**

## 2.1 Расположение букв на клавиатуре

Расположение букв на клавиатуре имеет свою историю, и оно было разработано с учетом различных факторов, включая удобство набора, технические ограничения и языковые особенности. Основной стандарт, который используется в современных компьютерных клавиатурах, называется клавиатурой Квёрти (QWERTY).

История клавиатуры Квёрти начинается с машинописных машин, которые были разработаны в конце 19 века. Первой коммерчески успешной машинописной машиной была "Шрифтолог", выпущенная в 1873 году. Она использовала механизм с подвижными металлическими рычагами для нанесения чернил на бумагу. Проблемой было то, что если пользователь быстро набирал текст, то металлические рычаги часто заедали, вызывая замедление и поломки.

Чтобы избежать этой проблемы, изобретатель Кристофер Латамер Шолз в 1868 году предложил устройство, которое распределяло наиболее часто используемые буквы в английском языке так, чтобы они находились на расстоянии друг от друга. Это приводило к уменьшению вероятности заедания механизма. Распределение букв получило имя "QWERTY" по первым шести буквам верхнего ряда клавиш.

Однако, существует и другая теория, согласно которой распределение букв QWERTY было выбрано для того, чтобы умедлить набор и уменьшить вероятность забивания машинописных машин. В то время как эта теория популярна, есть исследования, которые предполагают, что она не является единственной причиной.

С течением времени клавиатура QWERTY стала стандартом и была перенесена на компьютерные клавиатуры. Существует ряд альтернативных раскладок клавиш, таких как Dvorak Simplified Keyboard, которые предлагают более оптимальное распределение букв для увеличения скорости набора и уменьшения усталости рук, но QWERTY остается широко распространенным стандартом, особенно в сфере компьютеров и интернета.

# **МАТЕМАТИЧЕСКАЯ МОДЕЛЬ**

Текст рассматривается как случайный процесс, а единица текста (буквы, буквосочетания, морфемы, словоупотребления, словосочетания) – как случайные события. Наиболее распространённой вероятно-статистической моделью текста является частотный словарь (список) словоформ и слов.

В действительности каждый реальный текст является нестационарным процессом, и распределение вероятностей употребления многих слов и словоформ зависит от темы текста, времени его написания, от индивидуальных особенностей и стилевого вкуса автора.

Математическая модель для частотного словаря сочетаний букв алфавита может быть представлена в виде вероятностной функции. Предположим, что у нас есть конечный алфавит A, состоящий из букв, и мы рассматриваем все возможные сочетания букв длиной n. Обозначим буквенное сочетание как w = (w1, w2, … , wn), где каждый wi принадлежит алфавиту A.

Пусть N - общее количество сочетаний букв длиной n в рассматриваемых текстах, а f(w) - частота появления конкретного сочетания w. Тогда вероятность появления данного сочетания может быть определена как:

.

Где:

- P(w) - вероятность появления сочетания w,

- f(w) - частота появления сочетания w,

- N - общее количество сочетаний букв длиной n в текстах.

Мы также можем рассмотреть условные вероятности для буквенных сочетаний. Например, условная вероятность появления буквы wi после буквы wi-1 может быть выражена как:

.

Где:

- P(wi | wi-1) - условная вероятность появления буквы wi после буквы wi-1,

- f(wi-1wi) - частота появления сочетания wi-1wi,

- f(wi-1) - частота появления сочетания wi-1.

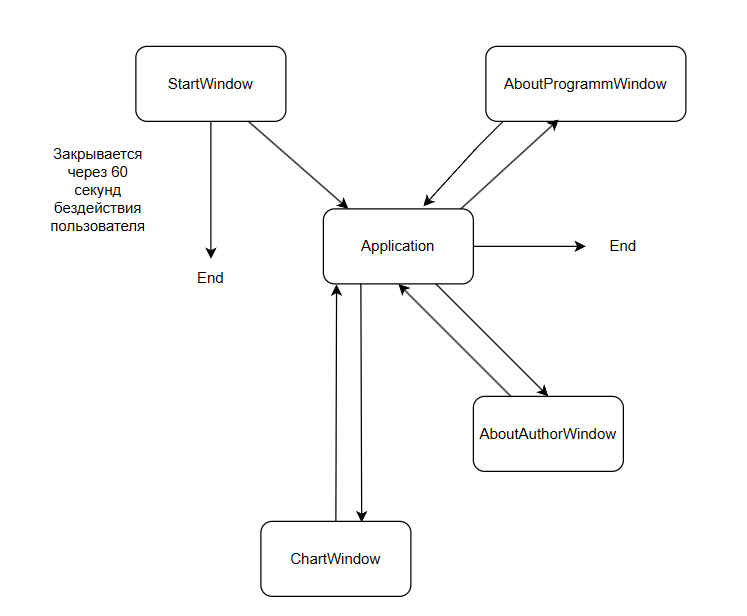
Такая математическая модель позволяет формализовать и анализировать статистические закономерности в частотном словаре буквенных сочетаний алфавита.

# **БИЗНЕС-ЛОГИКА ПРОЕКТА**

***Приложения содержит:***

1. **Окно входа в приложение (StartWindow)**. Окно представляет собой титульный лист курсовой работы с возможностью начала работы с приложением и завершения через 60 секунд при бездействии пользователя, либо при срочной необходимости.
2. **Основное окно (Application)**. Окно для анализа текста, поиска строки/слова/сочетания букв/буквы в тексте. Также построения графика, отражающего частоту употребления букв в тексте.
3. **Окно с информацией об авторе (AboutAuthorWindow)**. Окно, содержащее информацию об авторе, такую как ФИО автора, учебную группу и адрес электронной почты.
4. **Окно с информацией о программе (AboutProgrammWindow)**. Окно, в котором отражено описание функционала программы и версия приложения.
5. **Окно с построением графика (ChartWindow)**. Окно для демонстрации графика, который построен на анализе введенного текста.

Наглядно посмотреть на бизнес-логику проекта можно увидеть на рисунке 4.1.

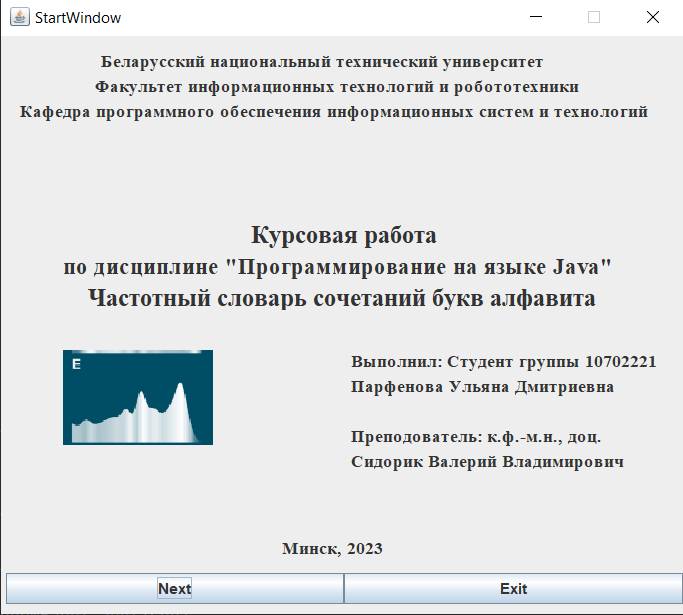


*Рисунок 4.1 – Бизнес-логика проекта*

# **ПРАКТИЧЕСКИЙ РАЗДЕЛ**

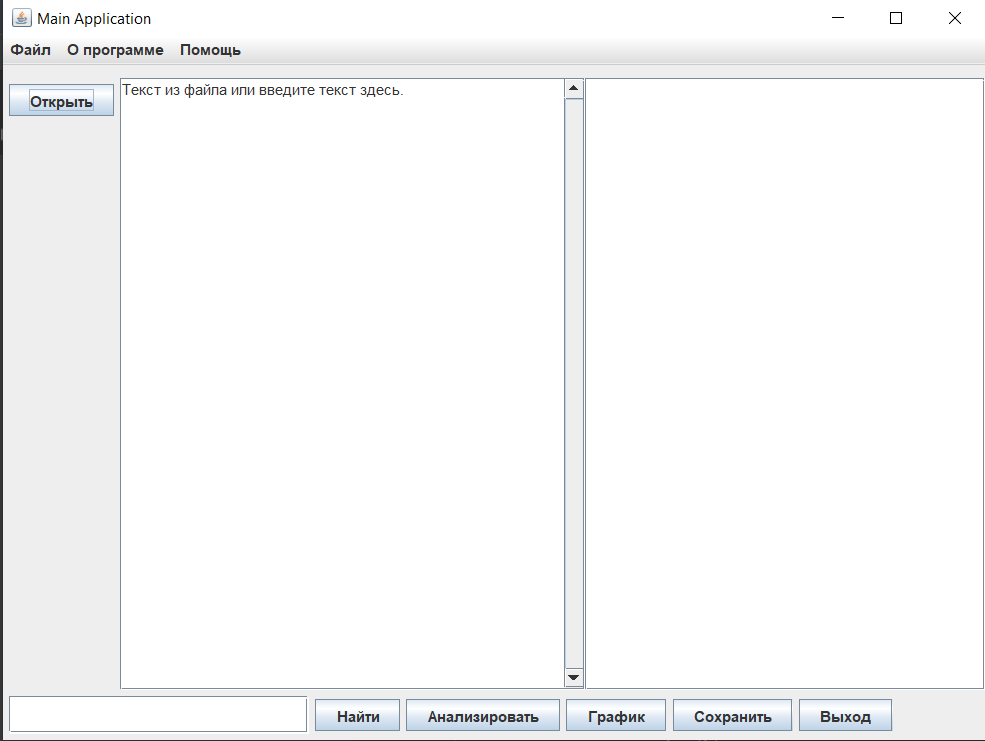
## Описание структуры программы

* + 1. После запуска программы перед пользователем появляется стартовое окно (StartWindow), содержащее основную информацию проекта (название учебного учреждения, кафедры, дисциплины и т.д.). Также присутствует две кнопки: “далее”(вызывающая метод runApplication()) открывает основное окно программы, “выйти”(вызывает метод dispose()) закрывает приложение. Если за 60 секунд пользователь не закрыл стартовое окно, программа завершится автоматически, с помощью класса Timer. Скриншот окна представлен на рисунке 5.1.1.1.



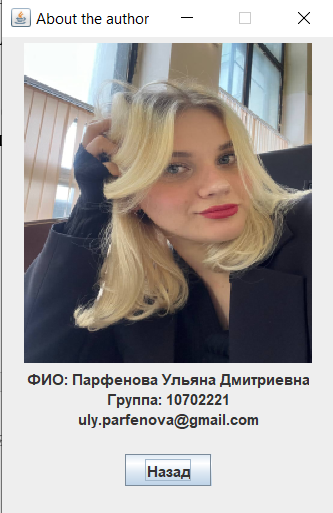
*Рисунок 5.1.1.1 – Стартовое окно программы*

* + 1. Главное окно содержит основную рабочую область. Центральная часть окна отведена под поле вывода текста(textArea), использующее методы класса SearchFunction(), который вводит пользователь. Также большую часть окна занимает поле для вывода результата(resultArea), после анализа текста. Остальную часть окна занимают кнопки: Открыть(openButton), которая использует метод openFile(), Найти(findButton), использующую методы класса SearchFunction, Анализировать(analyzeButton), График(chartButton), использующую метод displayChart класса ChartWindow, Сохранить(saveButton), Выход(exitButton)(рисунок 5.1.2.1).



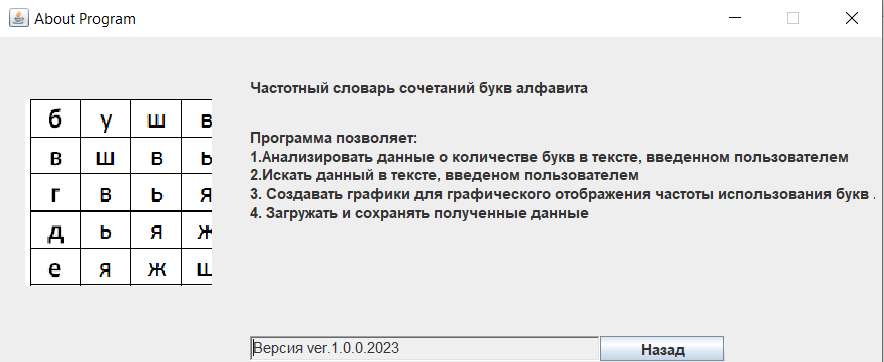
*Рисунок 5.1.2.1 – Основное окно программы*

* + 1. Окно об авторе содержит основную информацию об авторе(ФИО, номер группы, адрес электронной почты)(рисунок 5.1.3.1).



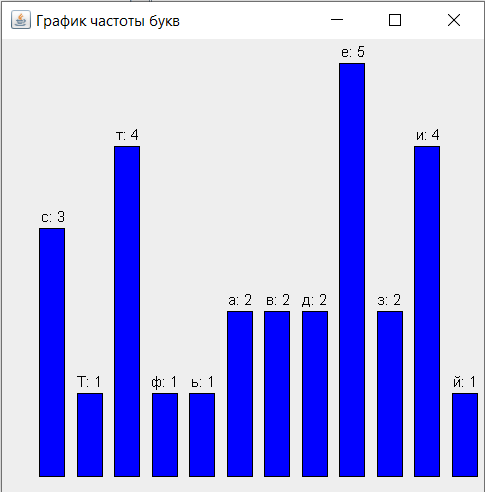
*Рисунок 5.1.3.1 – Окно об авторе*

* + 1. Окно о программе содержит основную информацию о программы: как программа работает, для чего она нужна, какие функции она может реализовать. Скриншот окна представлен на рисунке 5.1.4.1.



*Рисунок 5.1.4.1 – Окно о программе*

* + 1. Окно для вывода графика реализует отрисовку графика с помощью класса drawChart(Graphic g), показывает наглядную частоту использования букв в тексте. Скриншоты результата представлены на рисунке 5.1.5.1.

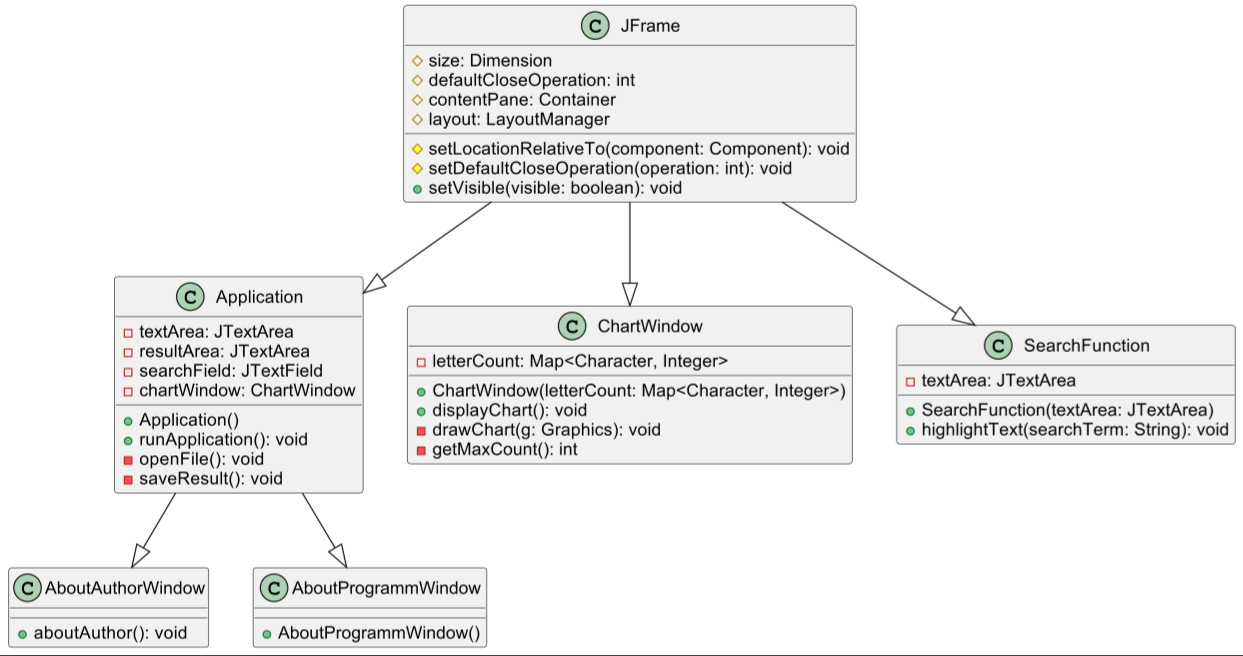


*Рисунок 5.1.5.1 – Окно, реализующее отрисовку графика*

## UML-диаграммы классов

UML (с английского аббревиатура расшифровывается как [Unified Modeling Language](https://www.lucidchart.com/pages/ru/uml) — унифицированный язык моделирования) — это способ наглядно описать архитектуру, проектирование и реализацию комплексных программных систем. Код типичного приложения включает в себя тысячи строк, за связями и иерархиями которых очень непросто уследить. С помощью диаграмм UML структуру программы можно разделить на компоненты и подкомпоненты.

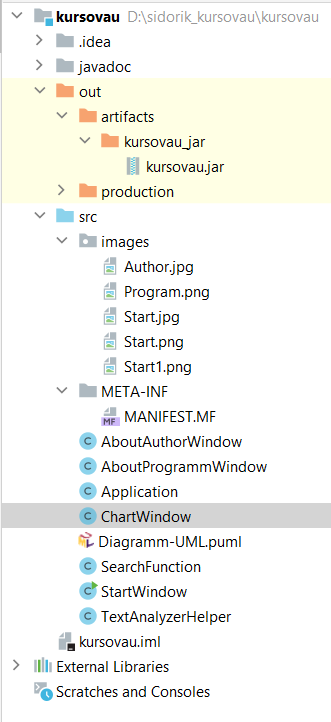
Ниже на рисунке 5.2.1 представлены uml-диаграммы программы «Частотный словарь сочетаний букв в алфавите».



*Рисунок 5.2.1 – UML-диаграмма*

## **Структура проекта в Intelij IDEA**

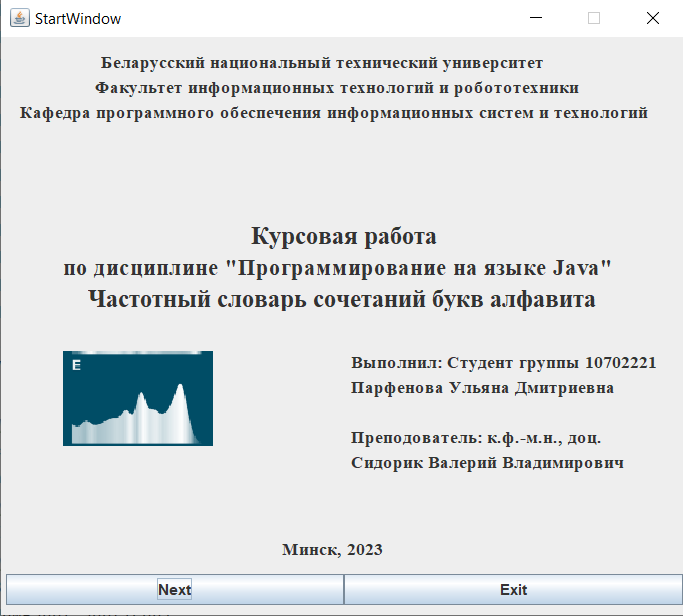
Программа имеет следующую структуру, показанную на рисунке 5.3.1:



*Рисунок 5.3.1 – Структура проекта в Intelij IDEA*

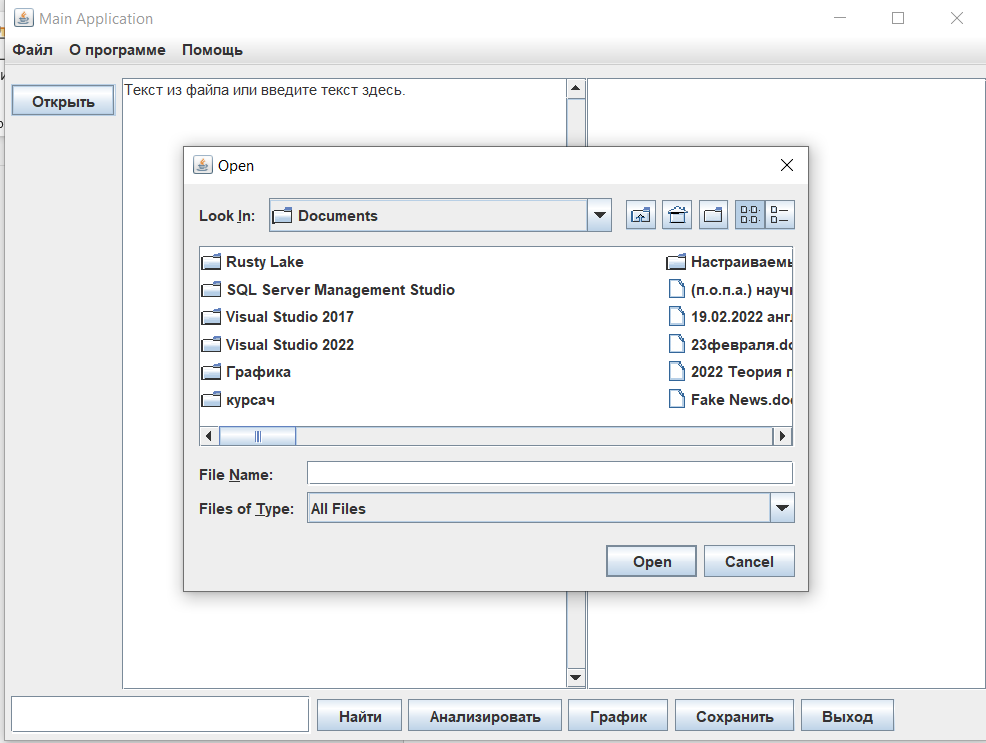
# **ДЕМОНСТРАЦИЯ РАБОТЫ ПРИЛОЖЕНИЯ**

После запуска программы первое окно, что мы видим, это стартовое окно (рисунок 6.1).



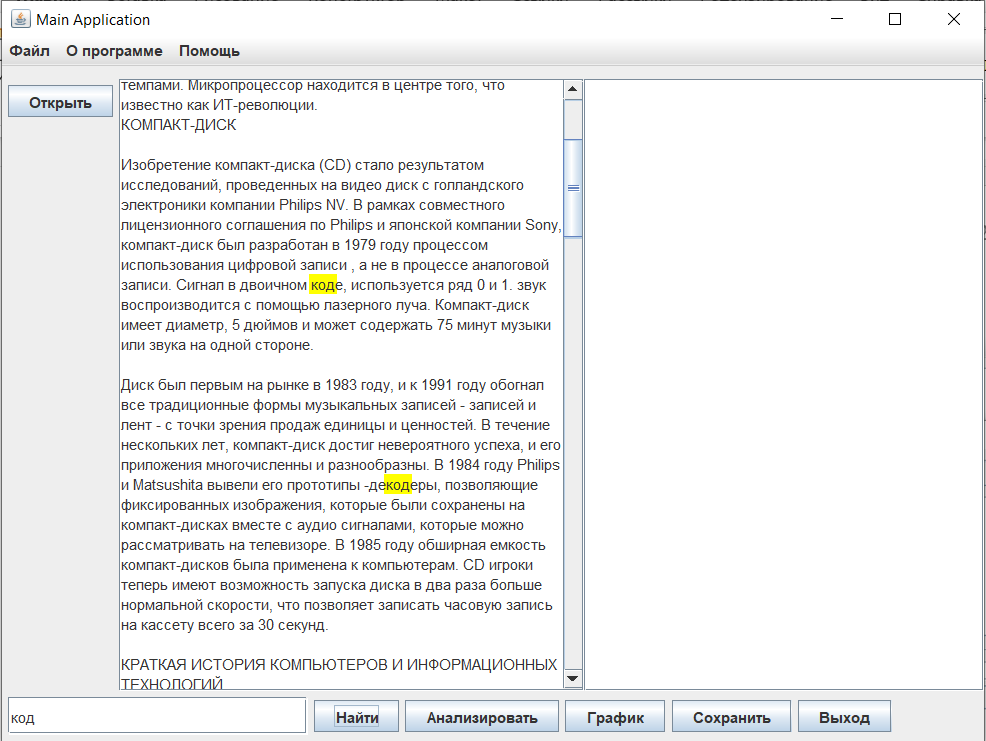
*Рисунок 6.1 – Стартовое окно*

При нажатии на кнопку «Next» в основном окне перед пользователем открывается рабочее окно, содержащее интерфейс для анализа текста (рисунок 6.2).



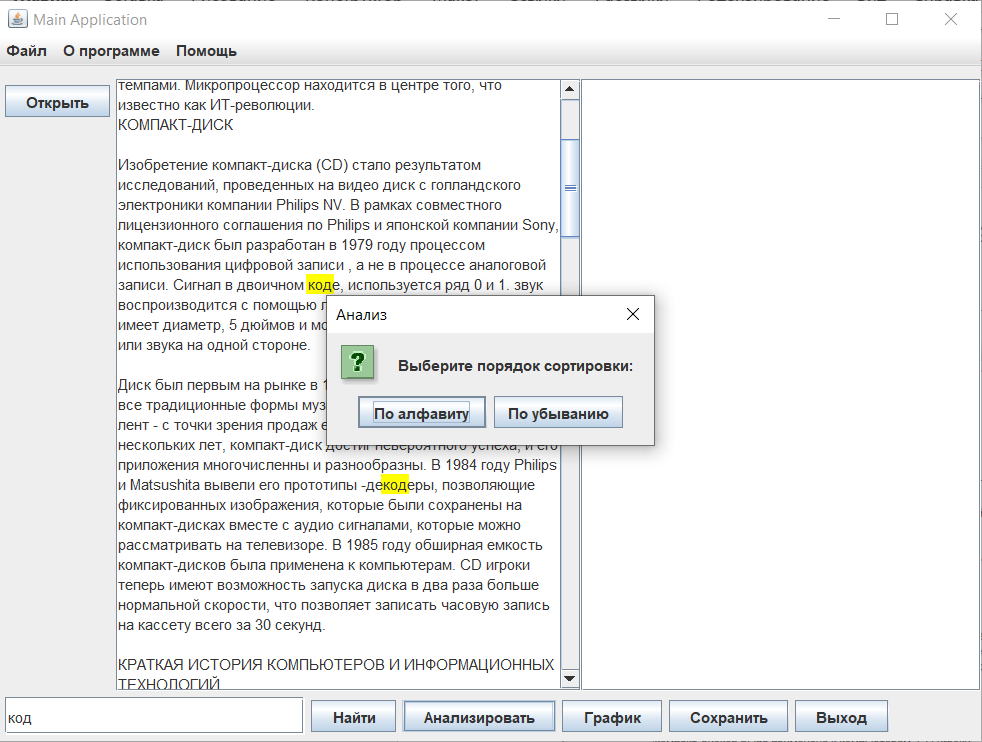
*Рисунок 6.2 – Открытие файла*

По нажатию на кнопку «Найти» пользователь ищет строку в тексте (рисунок 6.3).



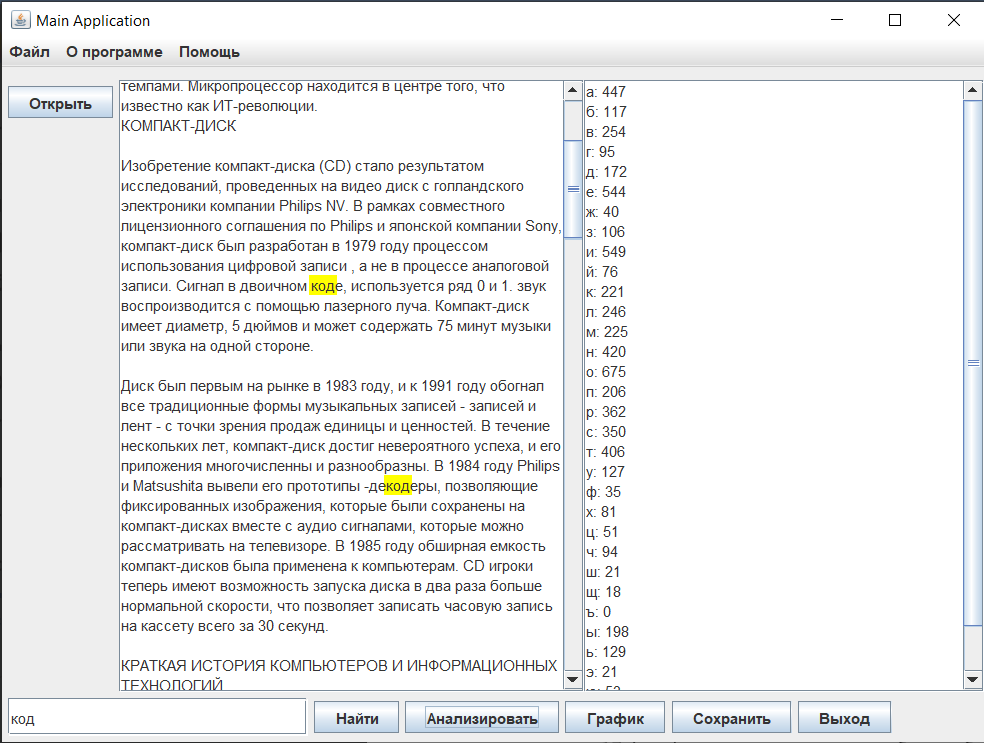
*Рисунок 6.3 – Поиск сроки в файле*

Далее пользователь может анализировать текст, при этом выбирает каким образом сортировать результат (рисунок 6.4).



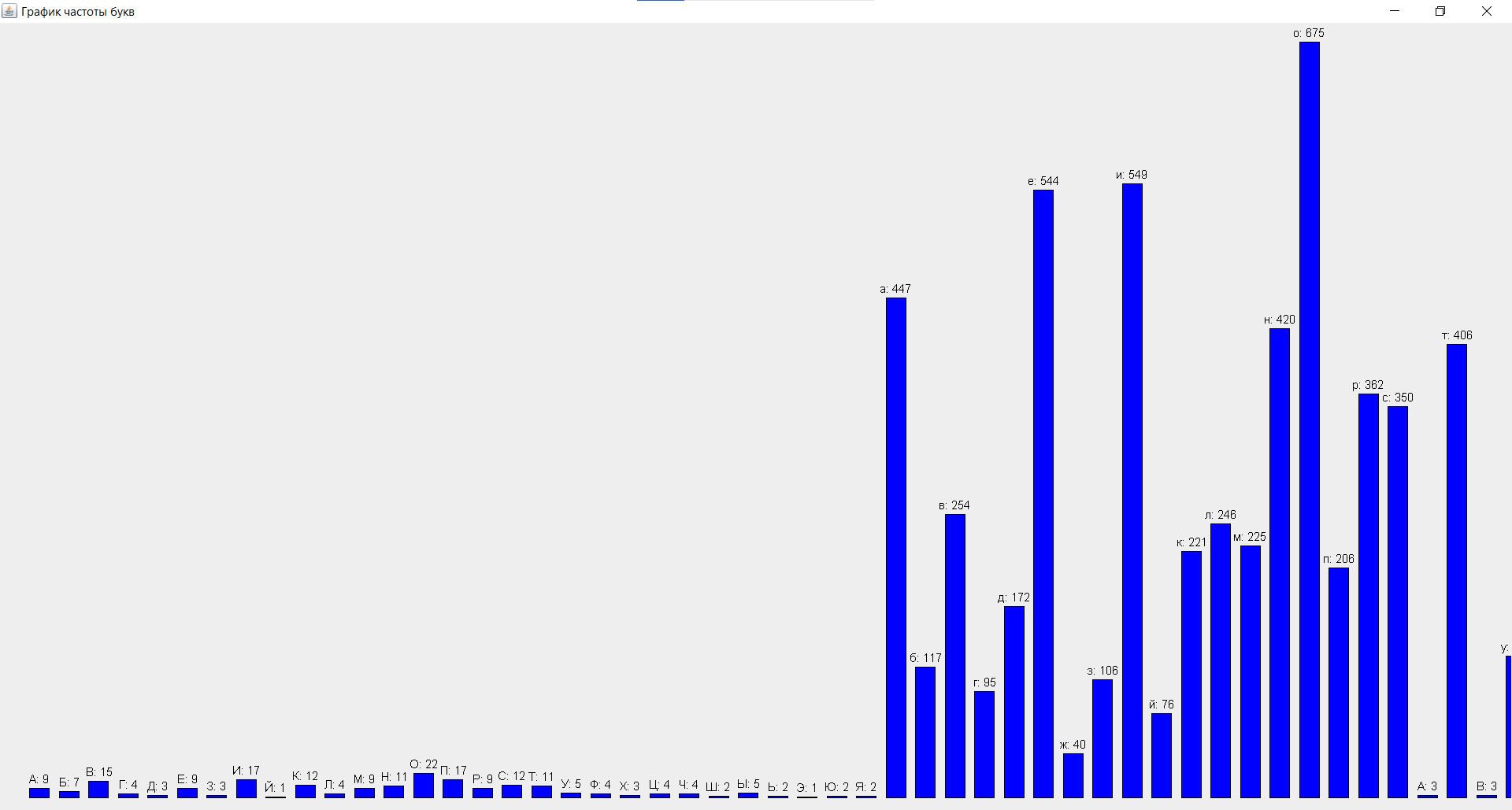
*Рисунок 6.4 – Анализ текста и выбор сортировки*

Результат сортировки показан на рисунке 6.5.



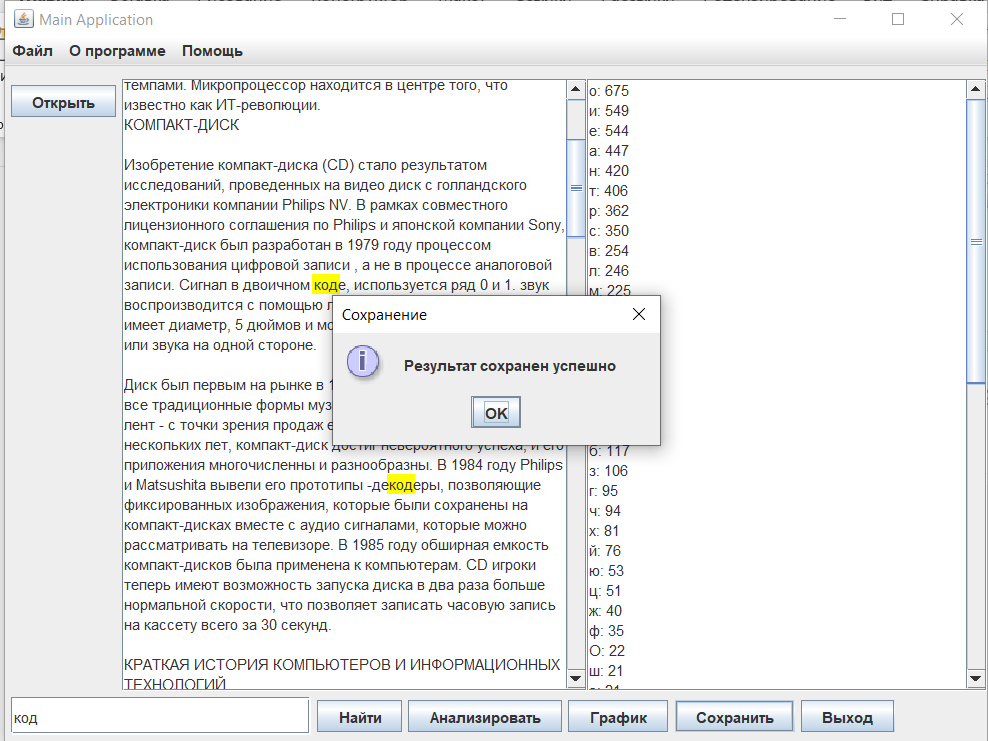
*Рисунок 6.5 – Результат анализа*

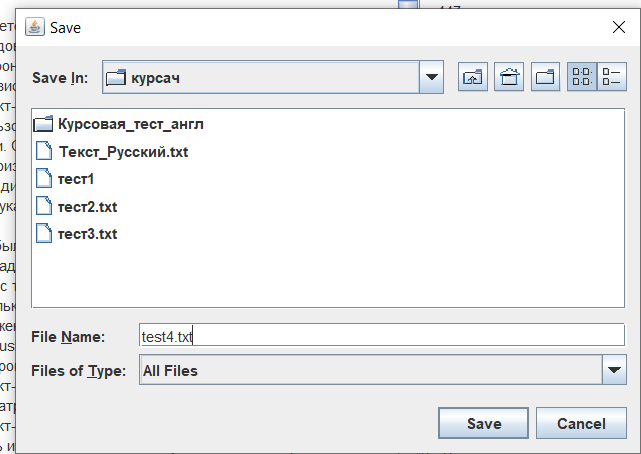
Также пользователь может нарисовать график, опираясь на результаты анализа (рисунок 6.6).

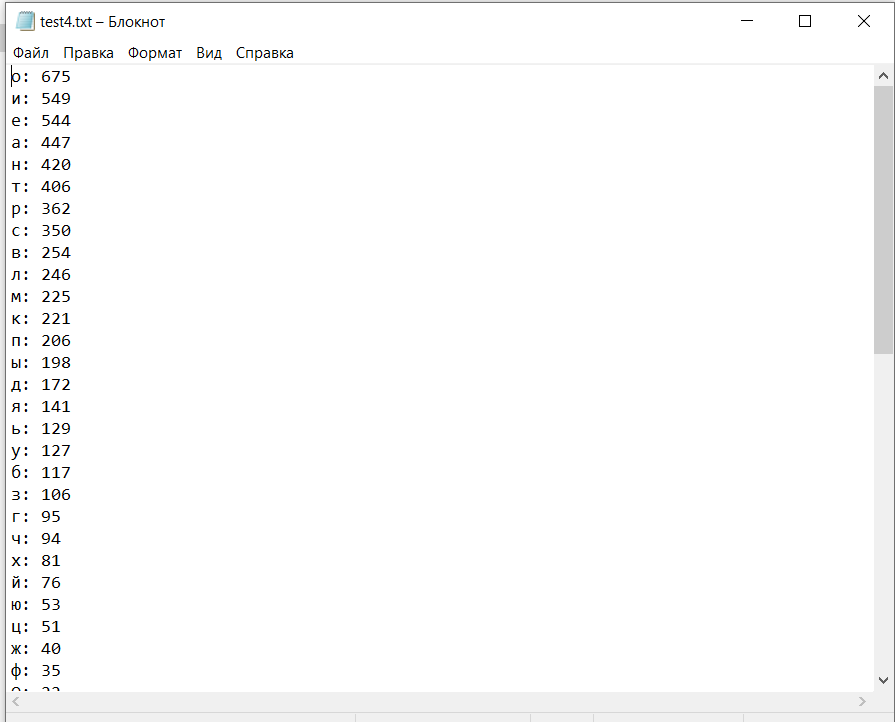


*Рисунок 6.6 – График, отражающий результат анализа*

В интерфейсе приложения у нас есть возможность сохранить, полученный результат (рисунок 6.7-6.9).







*Рисунок 6.7-6.9 – Сохранение результата*

Те же самые операции можно произвести и с текстом на английском языке.

# **ЗАКЛЮЧЕНИЕ**

В результате курсовой работы, мы провели анализ и можно считать, что если условиться, что русский алфавит состоит из 32 букв, то окажется, что самый частый символ – это гласные О (9,42%), Е (7,33%), И (6,72%), А (6,52%) и согласные Н (5,83%), Т (5,56%) (см. рисунок). А реже всего встречаются буквы Ф (0,27%), Ъ (0,03%) и Ё (0,01%). Конечно, в каждом конкретном тексте частоты могут отличаться от приведенных, но эти отклонения будут несущественными.

# **СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ**

1. Блинов, И.Н., Романчик, В.С. "Java. Промышленное программирование: практическое пособие". – Минск : УниверсалПресс, 2007. – 704 с. ISBN 978-985-6699-63-7
2. Рогозин, Д.М. (2019). "Принципы расположения клавиш на клавиатуре". URL: https://uchet-jkh.ru/i/principy-raspolozeniya-klavis-na-klaviature/
3. Скребнев, Ю.И. (2015). "Рождение клавиатуры: от идеи до массового производства". URL: https://isoft.kz/obzori/rozhdenie\_klaviaturi.html
4. Тихомиров, Д.В. (2021). "Keyboard Layouts: Evolution, Implementation, and Efficiency". Higher School of Economics Working Papers, No. WP BRP 236/STI/2021. URL: https://publications.hse.ru/pubs/share/direct/332845895.pdf
5. <https://drive.google.com/drive/shared-with-me> - Методические указания к оформлению курсовой работы, В.В. Сидорик.
6. <https://javadevblog.com/sozdaem-yspolnyaemyj-jar-v-intellij-idea.html> - создание jar-файлов.
7. <https://hr-vector.com/java/swing-graficheskij-interfejs> - графический интерфейс Swing.

# **ПРИЛОЖЕНИЕ 1**

**Класс StartWindow**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
*/\*\*  
 \* Класс, представляющий стартовое окно приложения.  
 \*/*public class StartWindow extends JFrame {  
 */\*\*  
 \* Конструктор класса Start, инициализирующий стартовое окно приложения.  
 \*/* public StartWindow() {  
 super("StartWindow");  
 this.setSize(560, 500);  
 *// Установка операции закрытия окна по умолчанию* this.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 *// Запрет изменения размера окна пользователем* this.setResizable(false);  
 Timer timer = new Timer(60000, e -> dispose());*// Таймер для автоматического закрытия окна через 60 секунд* timer.start();  
 setLocationRelativeTo(null);  
 *// Инициализация и настройка меток* JLabel startWindowLabelData\_1 = new JLabel("Беларусский национальный технический университет");  
 JLabel startWindowLabelData\_2 = new JLabel("Факультет информационных технологий и робототехники");  
 JLabel startWindowLabelData\_3 = new JLabel("Кафедра программного обеспечения информационных систем и технологий");  
 JLabel startWindowLabelData\_4 = new JLabel("Курсовая работа");  
 JLabel startWindowLabelData\_5 = new JLabel("по дисциплине \"Программирование на языке Java\"");  
 JLabel startWindowLabelData\_6 = new JLabel("Частотный словарь сочетаний букв алфавита");  
 JLabel startWindowLabelData\_7 = new JLabel("Выполнил: Студент группы 10702221");  
 JLabel startWindowLabelData\_8 = new JLabel("Парфенова Ульяна Дмитриевна");  
 JLabel startWindowLabelData\_9 = new JLabel("Преподователь: к.ф.-м.н., доц.");  
 JLabel startWindowLabelData\_10 = new JLabel("Сидорик Валерий Владимирович");  
 JLabel startWindowLabelData\_11 = new JLabel("Минск, 2023");  
 *// Настройка шрифта для меток* Font startWindowLabelFont\_1 = new Font("Times New Roman", Font.*BOLD*, 14);  
 Font startWindowLabelFont\_2 = new Font("Times New Roman", Font.*BOLD*, 20);  
 Font startWindowLabelFont\_3 = new Font("Times New Roman", Font.*BOLD*, 18);  
 *// Установка шрифта для меток* startWindowLabelData\_1.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_2.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_3.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_4.setFont(startWindowLabelFont\_2);  
 startWindowLabelData\_5.setFont(startWindowLabelFont\_3);  
 startWindowLabelData\_6.setFont(startWindowLabelFont\_2);  
 startWindowLabelData\_7.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_8.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_9.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_10.setFont(startWindowLabelFont\_1);  
 startWindowLabelData\_11.setFont(startWindowLabelFont\_1);  
  
 *// Создание кнопок* JButton start = new JButton("Next");  
 JButton stop = new JButton("Exit");  
  
 *// Создание панели для компоновки компонентов* JPanel startWindowPanel = new JPanel();  
  
 *// Установка менеджера компоновки в null, что позволяет явно задавать позиции компонентов* startWindowPanel.setLayout(null);  
  
 *// Установка расположения меток* startWindowLabelData\_1.setBounds(80, 10, 370, 20);  
 startWindowLabelData\_2.setBounds(75, 30, 400, 20);  
 startWindowLabelData\_3.setBounds(15, 50, 520, 20);  
 startWindowLabelData\_4.setBounds(200, 150, 150, 20);  
 startWindowLabelData\_5.setBounds(50, 175, 450, 20);  
 startWindowLabelData\_6.setBounds(70, 200, 420, 20);  
 startWindowLabelData\_7.setBounds(280, 250, 250, 20);  
 startWindowLabelData\_8.setBounds(280, 270, 280, 20);  
 startWindowLabelData\_9.setBounds(280, 310, 250, 20);  
 startWindowLabelData\_10.setBounds(280, 330, 250, 20);  
 startWindowLabelData\_11.setBounds(225, 400, 100, 20);  
 start.setBounds(4, 430, 271, 25);  
 stop.setBounds(275, 430, 271, 25);  
 *// Добавление меток в панель* startWindowPanel.add(startWindowLabelData\_1);  
 startWindowPanel.add(startWindowLabelData\_2);  
 startWindowPanel.add(startWindowLabelData\_3);  
 startWindowPanel.add(startWindowLabelData\_4);  
 startWindowPanel.add(startWindowLabelData\_5);  
 startWindowPanel.add(startWindowLabelData\_6);  
 startWindowPanel.add(startWindowLabelData\_7);  
 startWindowPanel.add(startWindowLabelData\_8);  
 startWindowPanel.add(startWindowLabelData\_9);  
 startWindowPanel.add(startWindowLabelData\_10);  
 startWindowPanel.add(startWindowLabelData\_11);  
 startWindowPanel.add(start);  
 startWindowPanel.add(stop);  
 this.add(startWindowPanel);  
 this.setVisible(true);  
  
 *// Создание метки для отображения изображения* JLabel icon = new JLabel();  
 ImageIcon img = new ImageIcon("src/images/Start1.png");  
 *// Установка изображения для метки* icon.setIcon(img);  
 *// Установка позиции и размеров метки на панели* icon.setBounds(50, 240, 190, 100);  
 startWindowPanel.add(icon);  
  
 *// Добавление обработчиков событий для кнопок* start.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// Скрыть стартовое окно* setVisible(false);  
 Application.*runApplication*();  
 }  
 });  
 stop.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// Закрыть стартовое окно* dispose();  
 }  
 });  
  
 }  
 */\*\*  
 \* Точка входа в приложение. Создает экземпляр StartWindow.  
 \*  
 \* @param args Аргументы командной строки (не используются).  
 \*/* public static void main(String[] args) {  
 new StartWindow();  
 }  
}

**Класс Application**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
import java.io.\*;  
  
*/\*\*  
 \* Главный класс приложения, представляющий текстовый анализатор.  
 \*/*public class Application extends JFrame {  
  
 private JTextArea textArea;  
 private JTextArea resultArea;  
 private JTextField searchField;  
 private ChartWindow chartWindow;  
  
 */\*\*  
 \* Конструктор класса Application, инициализирующий главное окно приложения.  
 \*/* public Application() {  
 super("Main Application");  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setSize(800, 600);  
 setLayout(new BorderLayout());  
  
 *// Инициализация верхней панели* JPanel topPanel = new JPanel();  
 topPanel.setLayout(new FlowLayout(FlowLayout.*LEFT*));  
  
 *// Инициализация меню* JMenuBar menuBar = new JMenuBar();  
 setJMenuBar(menuBar);  
  
 *// Меню "Файл"* JMenu fileMenu = new JMenu("Файл");  
 menuBar.add(fileMenu);  
  
 JMenuItem exitMenuItem = new JMenuItem("Выход");  
 exitMenuItem.addActionListener(e -> System.*exit*(0));  
 fileMenu.add(exitMenuItem);  
  
 *// Меню "О программе"* JMenu aboutMenu = new JMenu("О программе");  
 menuBar.add(aboutMenu);  
  
 JMenuItem aboutProgramMenuItem = new JMenuItem("О программе");  
 aboutProgramMenuItem.addActionListener(e -> {  
 AboutProgrammWindow aboutProgramMenu = new AboutProgrammWindow();  
 setVisible(false);  
 });  
 aboutMenu.add(aboutProgramMenuItem);  
  
 JMenuItem aboutAuthorMenuItem = new JMenuItem("Об авторе");  
 aboutAuthorMenuItem.addActionListener(e -> {  
 AboutAuthorWindow aboutAuthorMenu = new AboutAuthorWindow();  
 aboutAuthorMenu.aboutAuthor();  
 setVisible(false);  
 });  
 aboutMenu.add(aboutAuthorMenuItem);  
  
 *// Меню "Помощь"* JMenu HelpMenu = new JMenu("Помощь");  
 menuBar.add(HelpMenu);  
  
 *// Добавление верхней панели* add(topPanel, BorderLayout.*NORTH*);  
  
 *// Инициализация текстовой области* textArea = new JTextArea();  
 textArea.setLineWrap(true);  
 textArea.setWrapStyleWord(true);  
 JScrollPane textScrollPane = new JScrollPane(textArea, JScrollPane.*VERTICAL\_SCROLLBAR\_ALWAYS*, JScrollPane.*HORIZONTAL\_SCROLLBAR\_NEVER*);  
 add(textScrollPane, BorderLayout.*CENTER*);  
  
 *// Инициализация области результата* resultArea = new JTextArea();  
 JScrollPane resultScrollPane = new JScrollPane(resultArea);  
 resultScrollPane.setPreferredSize(new Dimension(getWidth() \* 40 / 100, getHeight() \* 40 / 100));  
  
 *// Панель для кнопки "Open"* JPanel openPanel = new JPanel();  
 JButton openButton = new JButton("Открыть");  
 openPanel.add(openButton);  
  
 *// Панель для кнопки "Save"* JPanel savePanel = new JPanel();  
 JButton saveButton = new JButton("Сохранить");  
 savePanel.add(saveButton);  
  
 *// Панель с кнопками "Find", "Analyze", "График"* JPanel bottomPanel = new JPanel();  
 bottomPanel.setLayout(new FlowLayout(FlowLayout.*LEFT*));  
  
 JButton findButton = new JButton("Найти");  
 JButton analyzeButton = new JButton("Анализировать");  
 JButton chartButton = new JButton("График");  
 JButton exitButton = new JButton("Выход");  
  
 searchField = new JTextField();  
 searchField.setPreferredSize(new Dimension(getWidth() \* 30 / 100, 30));  
  
 bottomPanel.add(searchField);  
 bottomPanel.add(findButton);  
 bottomPanel.add(analyzeButton);  
 bottomPanel.add(chartButton);  
 bottomPanel.add(saveButton);  
 bottomPanel.add(exitButton);  
  
 *// Добавление компонентов на окно* add(topPanel, BorderLayout.*NORTH*);  
 add(textScrollPane, BorderLayout.*CENTER*);  
 add(openPanel, BorderLayout.*WEST*);  
 add(resultScrollPane, BorderLayout.*EAST*);  
 add(bottomPanel, BorderLayout.*SOUTH*);  
  
 *// Обработчики событий* openButton.addActionListener(e -> openFile());  
 saveButton.addActionListener(e -> saveResult());  
 exitButton.addActionListener(e -> System.*exit*(0));  
  
 textArea.setText("Текст из файла или введите текст здесь.");  
  
 SearchFunction search = new SearchFunction(textArea);  
 findButton.addActionListener(e -> {  
 String searchTerm = searchField.getText();  
 search.highlightText(searchTerm);  
 });  
  
 analyzeButton.addActionListener(e -> {  
 String[] options = {"По алфавиту", "По убыванию"};  
 int choice = JOptionPane.*showOptionDialog*(  
 Application.this,  
 "Выберите порядок сортировки:",  
 "Анализ",  
 JOptionPane.*DEFAULT\_OPTION*,  
 JOptionPane.*QUESTION\_MESSAGE*,  
 null,  
 options,  
 options[0]  
 );  
 String text = textArea.getText();  
 String result;  
  
 if (choice == 0) {  
 result = TextAnalyzerHelper.*analyzeAlphabeticalOrder*(text);  
 } else if (choice == 1) {  
 result = TextAnalyzerHelper.*analyzeReverseOrder*(text);  
 } else {  
 result = ""; *// Добавьте обработку других вариантов* }  
 resultArea.setText(result);  
 });  
  
 chartButton.addActionListener(e -> {  
 if (chartWindow == null) {  
 chartWindow = new ChartWindow(TextAnalyzerHelper.*countLetters*(textArea.getText()));  
 chartWindow.displayChart();  
 }  
 });  
 }  
  
 */\*\*  
 \* Метод для открытия файла и загрузки его содержимого в текстовую область.  
 \*/* private void openFile() {  
 JFileChooser fileChooser = new JFileChooser();  
 int result = fileChooser.showOpenDialog(Application.this);  
  
 if (result == JFileChooser.*APPROVE\_OPTION*) {  
 File selectedFile = fileChooser.getSelectedFile();  
  
 try (BufferedReader reader = new BufferedReader(new FileReader(selectedFile))) {  
 StringBuilder content = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 content.append(line).append("\n");  
 }  
 textArea.setText(content.toString());  
 } catch (IOException e) {  
 e.printStackTrace();  
 JOptionPane.*showMessageDialog*(  
 Application.this,  
 "Ошибка при чтении файла",  
 "Ошибка",  
 JOptionPane.*ERROR\_MESSAGE* );  
 }  
 }  
 }  
  
 */\*\*  
 \* Метод для сохранения содержимого результата в файл.  
 \*/* private void saveResult() {  
 JFileChooser fileChooser = new JFileChooser();  
 int result = fileChooser.showSaveDialog(Application.this);  
  
 if (result == JFileChooser.*APPROVE\_OPTION*) {  
 File selectedFile = fileChooser.getSelectedFile();  
  
 try (PrintWriter writer = new PrintWriter(selectedFile)) {  
 writer.print(resultArea.getText());  
 JOptionPane.*showMessageDialog*(  
 Application.this,  
 "Результат сохранен успешно",  
 "Сохранение",  
 JOptionPane.*INFORMATION\_MESSAGE* );  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 JOptionPane.*showMessageDialog*(  
 Application.this,  
 "Ошибка при сохранении файла",  
 "Ошибка",  
 JOptionPane.*ERROR\_MESSAGE* );  
 }  
 }  
 }  
  
 */\*\*  
 \* Статический метод для запуска приложения, вызывается из класса StartWindow.  
 \*/* public static void runApplication() {  
 SwingUtilities.*invokeLater*(() -> {  
 Application application = new Application();  
 application.setVisible(true);  
 });  
 }  
}

**Класс AboutAuthorWindow**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
*/\*\*  
 \* Класс AboutAuthorWindow представляет окно с информацией об авторе.  
 \*/*public class AboutAuthorWindow {  
  
 */\*\*  
 \* Отображает окно с информацией об авторе.  
 \*/* public void aboutAuthor() {  
 JFrame frame = new JFrame("About the author");  
  
 *// Установка менеджера компоновки* frame.setLayout(new BorderLayout());  
  
 *// Добавление фотографии* JPanel imagePanel = new JPanel(new FlowLayout(FlowLayout.*CENTER*));  
 ImageIcon icon = new ImageIcon("src/images/Author.jpg");  
 icon = new ImageIcon(icon.getImage().getScaledInstance(230, 256, Image.*SCALE\_SMOOTH*));  
 JLabel imageLabel = new JLabel(icon);  
 imagePanel.add(imageLabel);  
 frame.add(imagePanel, BorderLayout.*NORTH*);  
  
 *// Создание панели для компонентов* JPanel componentsPanel = new JPanel();  
 componentsPanel.setLayout(new BoxLayout(componentsPanel, BoxLayout.*Y\_AXIS*));  
 componentsPanel.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
  
 *// Full name* JLabel nameLabel = new JLabel("ФИО: Парфенова Ульяна Дмитриевна");  
 nameLabel.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 componentsPanel.add(nameLabel);  
  
 *// Номер учебной группы* JLabel groupLabel = new JLabel("Группа: 10702221");  
 groupLabel.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 componentsPanel.add(groupLabel);  
  
 *// Mail* JLabel mailLabel = new JLabel("uly.parfenova@gmail.com");  
 mailLabel.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 componentsPanel.add(mailLabel);  
  
 frame.add(componentsPanel, BorderLayout.*CENTER*);  
 componentsPanel.add(Box.*createVerticalStrut*(20));  
  
 *// Button* JButton backButton = new JButton("Назад");  
 backButton.setAlignmentX(Component.*CENTER\_ALIGNMENT*);  
 backButton.addActionListener(e -> frame.dispose());  
 componentsPanel.add(backButton);  
  
 backButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// При нажатии на "Назад" делаем окно "О программе" невидимым* frame.setVisible(false);  
 Application.*runApplication*();  
 }  
 });  
  
 *// Установка свойств окна* frame.setSize(280, 420);  
 frame.setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
 frame.setResizable(false);  
 frame.setLocationRelativeTo(null);  
 frame.setVisible(true);  
 }  
}

**Класс AboutProgrammWindow**

*/\*\*  
 \* Класс AboutProgrammWindow представляет окно "О программе" с информацией о программе.  
 \* @version 1.0  
 \* @author Парфенова Ульяна Дмитриевна  
 \*/*import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
*/\*\*  
 \* Класс, представляющий окно "О программе" с информацией о программе.  
 \*/*public class AboutProgrammWindow extends JFrame {  
  
 */\*\*  
 \* Конструктор класса AboutProgrammWindow, инициализирующий окно "О программе".  
 \*/* public AboutProgrammWindow() {  
 super("About Program");  
 setSize(720, 300);  
 setResizable(false);  
 setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*); *// Close only this window* setLayout(null);  
 setLocationRelativeTo(null);  
  
 Font aboutProgramWindowFont\_1 = new Font("Times New Roman", Font.*BOLD*, 14);  
  
 JLabel lblNameOfProgram = new JLabel("Частотный словарь сочетаний букв алфавита");  
 JLabel lblAboutProgramAbility = new JLabel("Программа позволяет:");  
 JLabel lblAboutEaseEnteringData = new JLabel("1.Анализировать данные о количестве букв в тексте, введенном пользователем ");  
 JLabel lblAboutEaseContainData = new JLabel("2.Искать данный в тексте, введеном пользователем");  
 JLabel lblAboutEaseGraphicData = new JLabel("3. Создавать графики для графического отображения частоты использования букв в тексте");  
 JLabel lblAboutEaseDownloadData = new JLabel("4. Загружать и сохранять полученные данные");  
  
 setFont(aboutProgramWindowFont\_1);  
  
 lblNameOfProgram.setBounds(200, 30, 300, 20);  
 lblAboutProgramAbility.setBounds(200, 70, 500, 20);  
 lblAboutEaseEnteringData.setBounds(200, 85, 500, 20);  
 lblAboutEaseContainData.setBounds(200, 100, 500, 20);  
 lblAboutEaseGraphicData.setBounds(200, 115, 500, 20);  
 lblAboutEaseDownloadData.setBounds(200,130,500,20);  
  
 JLabel lblIcon = new JLabel();  
 ImageIcon img = new ImageIcon("src/images/Program.png");  
 lblIcon.setIcon(img);  
 lblIcon.setBounds(20, 50, 150, 150);  
  
 TextArea textVersion = new TextArea();  
 textVersion.setEditable(false);  
 textVersion.setText("Версия ver.1.0.0.2023");  
 textVersion.setBounds(200, 240, 280, 20);  
  
 JButton btnBack = new JButton("Назад");  
 btnBack.setBounds(480, 240, 100, 20);  
 btnBack.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 *// При нажатии на "Назад" делаем окно "О программе" невидимым* setVisible(false);  
 Application.*runApplication*();  
 }  
 });  
  
 add(lblNameOfProgram);  
 add(lblAboutProgramAbility);  
 add(lblAboutEaseEnteringData);  
 add(lblAboutEaseContainData);  
 add(lblAboutEaseGraphicData);  
 add(lblAboutEaseDownloadData);  
 add(lblIcon);  
 add(textVersion);  
 add(btnBack);  
 setVisible(true);  
 }  
}

**Класс ChartWindow**

import javax.swing.\*;  
import java.awt.\*;  
import java.util.Map;  
  
*/\*\*  
 \* Класс ChartWindow представляет окно с графиком частоты букв.  
 \*/*public class ChartWindow extends JFrame {  
  
 private Map<Character, Integer> letterCount;  
  
 */\*\*  
 \* Конструктор класса ChartWindow.  
 \*  
 \* @param letterCount Карта, содержащая частоту каждой буквы.  
 \*/* public ChartWindow(Map<Character, Integer> letterCount) {  
 this.letterCount = letterCount;  
  
 setTitle("График частоты букв");  
 setSize(400, 400); *// Увеличил высоту окна* setDefaultCloseOperation(JFrame.*DISPOSE\_ON\_CLOSE*);  
  
 JPanel chartPanel = new JPanel() {  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 drawChart(g);  
 }  
 };  
  
 chartPanel.setPreferredSize(new Dimension(400, 300));  
 add(chartPanel);  
 }  
  
 */\*\*  
 \* Отображает окно с графиком частоты букв.  
 \*/* public void displayChart() {  
 SwingUtilities.*invokeLater*(() -> {  
 setVisible(true);  
 repaint();  
 });  
 }  
  
 */\*\*  
 \* Рисует график частоты букв.  
 \*  
 \* @param g Графический контекст.  
 \*/* private void drawChart(Graphics g) {  
 if (letterCount != null) {  
 int maxHeight = getHeight() - 70;  
 int barWidth = 20;  
 int x = 30;  
  
 for (Map.Entry<Character, Integer> entry : letterCount.entrySet()) {  
 char letter = entry.getKey();  
 int count = entry.getValue();  
  
 int barHeight = (int) ((double) count / getMaxCount() \* maxHeight);  
  
 g.setColor(Color.*BLUE*);  
 g.fillRect(x, getHeight() - barHeight - 50, barWidth, barHeight);  
  
 g.setColor(Color.*BLACK*);  
 g.drawRect(x, getHeight() - barHeight - 50, barWidth, barHeight);  
  
 *// Добавляем подпись для каждого столбца* String label = String.*format*("%s: %d", letter, count);  
 int labelX = x + barWidth / 2 - g.getFontMetrics().stringWidth(label) / 2;  
 int labelY = getHeight() - barHeight - 55;  
 g.drawString(label, labelX, labelY);  
  
 x += barWidth + 10;  
 }  
  
 *// Добавляем легенду* int legendX = 30;  
 int legendY = getHeight() - 20;  
 for (Map.Entry<Character, Integer> entry : letterCount.entrySet()) {  
 char letter = entry.getKey();  
 g.drawString(String.*valueOf*(letter), legendX, legendY);  
 legendX += barWidth + 10;  
 }  
 }  
 }  
  
 */\*\*  
 \* Возвращает максимальное значение частоты в карте.  
 \*  
 \* @return Максимальное значение частоты.  
 \*/* private int getMaxCount() {  
 int maxCount = 0;  
 for (int count : letterCount.values()) {  
 if (count > maxCount) {  
 maxCount = count;  
 }  
 }  
 return maxCount;  
 }  
}

**Класс SearchFunction**

import javax.swing.text.DefaultHighlighter;  
import javax.swing.\*;  
import java.awt.\*;  
  
*/\*\*  
 \* Класс SearchFunction предоставляет функциональность выделения текста в JTextArea.  
 \*/*public class SearchFunction extends JFrame{  
  
 private JTextArea textArea;  
  
 */\*\*  
 \* Конструктор класса SearchFunction.  
 \*  
 \* @param textArea JTextArea, в котором будет производиться поиск и выделение текста.  
 \*/* public SearchFunction(JTextArea textArea) {  
 this.textArea = textArea;  
 }  
  
 */\*\*  
 \* Выделяет все вхождения указанного текста в JTextArea цветом.  
 \*  
 \* @param searchTerm Текст для поиска и выделения.  
 \*/* public void highlightText(String searchTerm) {  
 if (searchTerm.isEmpty()) {  
 clearHighlights();  
 return;  
 }  
  
 String text = textArea.getText();  
 clearHighlights();  
  
 int index = text.indexOf(searchTerm);  
 while (index >= 0) {  
 try {  
 textArea.getHighlighter().addHighlight(index, index + searchTerm.length(),  
 new DefaultHighlighter.DefaultHighlightPainter(Color.*YELLOW*));  
 index = text.indexOf(searchTerm, index + 1);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 */\*\*  
 \* Очищает все выделения в JTextArea.  
 \*/* private void clearHighlights() {  
 textArea.getHighlighter().removeAllHighlights();  
 }  
}

**Класс TextAnalyzerHelper**

import java.util.\*;  
import java.util.stream.Collectors;  
  
*/\*\*  
 \* Класс {@code TextAnalyzerHelper} предоставляет утилитарные методы для анализа текста и подсчета частоты букв.  
 \*/*public class TextAnalyzerHelper {  
  
 */\*\*  
 \* Анализирует заданный текст и возвращает строковое представление частоты букв в алфавитном порядке.  
 \*  
 \* @param text Текст для анализа.  
 \* @return Строковое представление частоты букв в алфавитном порядке.  
 \*/* public static String analyzeAlphabeticalOrder(String text) {  
 String language = *detectLanguage*(text);  
 Map<Character, Integer> letterCount = *countLetters*(text, language);  
 return *buildResult*(letterCount, language);  
 }  
  
 */\*\*  
 \* Анализирует заданный текст и возвращает строковое представление частоты букв в обратном порядке.  
 \*  
 \* @param text Текст для анализа.  
 \* @return Строковое представление частоты букв в обратном порядке.  
 \*/* public static String analyzeReverseOrder(String text) {  
 String language = *detectLanguage*(text);  
 Map<Character, Integer> letterCount = *countLetters*(text, language);  
  
 *// Сортировка по убыванию частот* List<Map.Entry<Character, Integer>> sortedEntries = letterCount.entrySet()  
 .stream()  
 .sorted(Collections.*reverseOrder*(Map.Entry.*comparingByValue*()))  
 .collect(Collectors.*toList*());  
  
 return *buildResult*(sortedEntries, language);  
 }  
  
 */\*\*  
 \* Определяет язык текста (русский или английский) на основе используемого набора символов.  
 \*  
 \* @param text Текст для анализа.  
 \* @return Определенный язык ("ru" для русского, "en" для английского).  
 \*/* private static String detectLanguage(String text) {  
 for (char ch : text.toCharArray()) {  
 if (Character.UnicodeBlock.*of*(ch) == Character.UnicodeBlock.*CYRILLIC*) {  
 return "ru";  
 }  
 }  
 return "en";  
 }  
  
 */\*\*  
 \* Подсчитывает количество вхождений букв в заданном тексте для конкретного языка.  
 \*  
 \* @param text Текст для анализа.  
 \* @param language Язык текста ("ru" для русского, "en" для английского).  
 \* @return Карта, содержащая частоту букв.  
 \*/* public static Map<Character, Integer> countLetters(String text, String language) {  
 Map<Character, Integer> letterCount = new HashMap<>();  
  
 for (char ch : text.toCharArray()) {  
 if (Character.*isLetter*(ch)) {  
 if ((language.equals("ru") && Character.UnicodeBlock.*of*(ch) == Character.UnicodeBlock.*CYRILLIC*)  
 || (language.equals("en") && Character.UnicodeBlock.*of*(ch) == Character.UnicodeBlock.*BASIC\_LATIN*)) {  
 letterCount.put(ch, letterCount.getOrDefault(ch, 0) + 1);  
 }  
 }  
 }  
  
 return letterCount;  
 }  
  
 */\*\*  
 \* Подсчитывает количество вхождений букв в заданном тексте.  
 \*  
 \* @param text Текст для анализа.  
 \* @return Карта, содержащая частоту букв.  
 \*/* public static Map<Character, Integer> countLetters(String text) {  
 Map<Character, Integer> letterCount = new HashMap<>();  
  
 for (char ch : text.toCharArray()) {  
 if (Character.*isLetter*(ch)) {  
 letterCount.put(ch, letterCount.getOrDefault(ch, 0) + 1);  
 }  
 }  
  
 return letterCount;  
 }  
  
 */\*\*  
 \* Строит строковое представление частоты букв на основе предоставленной карты и языка.  
 \*  
 \* @param letterCount Карта, содержащая частоту букв.  
 \* @param language Язык текста ("ru" для русского, "en" для английского).  
 \* @return Форматированное строковое представление частоты букв.  
 \*/* private static String buildResult(Map<Character, Integer> letterCount, String language) {  
 StringBuilder result = new StringBuilder();  
  
 char start, end;  
 if (language.equals("ru")) {  
 start = 'а';  
 end = 'я';  
 } else {  
 start = 'a';  
 end = 'z';  
 }  
  
 for (char ch = start; ch <= end; ch++) {  
 result.append(ch).append(": ").append(letterCount.getOrDefault(ch, 0)).append("\n");  
 }  
  
 return result.toString();  
 }  
  
 */\*\*  
 \* Строит строковое представление частоты букв на основе списка отсортированных записей и языка.  
 \*  
 \* @param sortedEntries Список отсортированных записей карты.  
 \* @param language Язык текста ("ru" для русского, "en" для английского).  
 \* @return Форматированное строковое представление частоты букв.  
 \*/* private static String buildResult(List<Map.Entry<Character, Integer>> sortedEntries, String language) {  
 StringBuilder result = new StringBuilder();  
  
 for (Map.Entry<Character, Integer> entry : sortedEntries) {  
 result.append(entry.getKey()).append(": ").append(entry.getValue()).append("\n");  
 }  
  
 return result.toString();  
 }  
}