**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ**

**УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ ИМ. ПРОФ. М.А. БОНЧ-БРУЕВИЧА»**

**(СПбГУТ)**

Факультет Инфокоммуникационных сетей и систем

Кафедра Защищенных систем связи

Дисциплина Разработка защищенных сетевых приложений

**Отчет по курсовой работе**

*Разработка приложения для информационной рассылки сервесных почтовых сообщений*

*(тема )*

Студент:

Хохлов А.Е. ИБС-23

*(Ф.И.О., № группы) (подпись)*

Преподаватель:

ассистент кафедры ЗСС

Шариков П.И.

*(уч. степень, уч. звание, Ф.И.О.) (подпись)*

Санкт-Петербург

2023

# Содержание

[Содержание 2](#_Toc153658912)

[Введение 3](#_Toc153658913)

[**1 Описания библиотек и протокола smtp** 5](#_Toc153658914)

[1.1 Swing(javax.swing) 5](#_Toc153658915)

[1.1.1 Основные классы 5](#_Toc153658916)

[1.1.2 Основные методы 5](#_Toc153658917)

[1.2.2 Основные методы 7](#_Toc153658918)

[1.3 java.io 7](#_Toc153658919)

[1.3.1 Основные классы 8](#_Toc153658920)

[1.3.2 Основные методы 8](#_Toc153658921)

[1.4 java.util.Properties 9](#_Toc153658922)

[1.4.1 Основные методы 9](#_Toc153658923)

[1.5 Java Mail API 10](#_Toc153658924)

[1.5.1 Основные классы и интерфейсы 10](#_Toc153658925)

[1.6 SMTP 11](#_Toc153658926)

[2. Листинг программы и UML диаграммы 13](#_Toc153658927)

[2.1 Листинг программы и описания методов 13](#_Toc153658928)

[2.2 UML диаграммы 26](#_Toc153658929)

[3. Примеры и ресурсы 27](#_Toc153658930)

[4. Обработка ошибок 30](#_Toc153658931)

[**Заключение** 34](#_Toc153658932)

[**Список литературы** 35](#_Toc153658933)

# Введение

Объектом исследования в курсовой работе является почта

Предмет исследования: разработка приложения для информационной рассылки сервисных почтовых сообщений

Цель: разработать программное обеспечение в котором пользователь сможет добавлять новые почтовые адреса, удалять их, а также рассылать сообщения.

Проблема заключается в необходимости создания надежного, легко масштабируемого и удобного в использовании приложения, которое позволит пользователям эффективно отправлять информационные сообщения через почтовые сервисы. Это приложение должно обладать функциональностью управления списком адресатов, содержанием сообщений, аутентификацией и безопасностью передачи данных.

Актуальность: эта тема представляет собой важное направление в сфере коммуникаций и информационных технологий. Она актуальна в контексте растущей потребности компаний и организаций в автоматизированных и эффективных средствах для отправки сервисных сообщений.

Задачи, которые требуется решить, чтобы достигнуть поставленной цели:

1. Разработку удобного пользовательского интерфейса для взаимодействия с приложением.
2. Реализацию механизмов добавления и удаления адресатов для рассылки.
3. Обеспечение безопасности хранения учетных данных для доступа к почтовым сервисам.
4. Создание функционала для формирования и отправки информационных сообщений различным адресатам.

# **1 Описания библиотек и протокола smtp**

## 1.1 Swing(javax.swing)

javax.swing - это библиотека для создания графического интерфейса пользователя (GUI) в Java. Она предоставляет множество компонентов для построения пользовательского интерфейса, таких как окна, кнопки, текстовые поля, списки, панели и другие элементы.

### 1.1.1 Основные классы

**1. JFrame**: Класс, представляющий главное окно приложения. Он содержит заголовок, размеры и компоненты, размещаемые в окне.

JFrame frame = new JFrame("Заголовок окна");

**2. JPanel**: Этот класс представляет панель, которая может содержать другие компоненты. Он используется для организации компонентов внутри окна.

JPanel panel = new JPanel();

**3. JButton**: Кнопка, которую пользователь может нажимать для выполнения действия. Вы можете добавить слушателя событий для реагирования на щелчок кнопки.

JButton button = new JButton("Нажми меня");

**4. JList**: Этот класс представляет список элементов, которые могут быть выбраны пользователем.

DefaultListModel<String> listModel = new DefaultListModel<>();  
JList<String> list = new JList<>(listModel);

**5. JScrollPane**: Класс для добавления полос прокрутки к компонентам, которые не вмещаются полностью в область отображения.

JScrollPane scrollPane = new JScrollPane(component);

### 1.1.2 Основные методы

**1. addActionListener(ActionListener listener)**: Метод для добавления слушателя событий к компоненту. Например, к кнопке для обработки нажатия.

button.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 }  
});

**2. setLayout(LayoutManager manager)**: Метод для установки менеджера размещения компонентов на панели.

panel.setLayout(new BorderLayout());

**3. add(Component comp)**: Метод для добавления компонента на контейнер, например, на панель.

panel.add(button);

**4. setDefaultCloseOperation(int operation)**: Метод для установки операции по закрытию окна. Например, JFrame.EXIT\_ON\_CLOSE для завершения приложения при закрытии окна.

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

**5. setVisible(boolean b)**: Метод для установки видимости компонента.

frame.setVisible(true);

Swing предоставляет множество других классов и методов для создания богатого и интерактивного пользовательского интерфейса в Java. Эти компоненты и методы позволяют разработчикам создавать удобные и интуитивно понятные GUI для своих приложений.

1.2 AWT (java.awt)

1.2.1 Основные классы

**1. Frame**: Это класс, представляющий главное окно приложения. Он похож на JFrame из javax.swing.

Frame frame = new Frame("Заголовок окна");

**2. Panel**: Панель, которая может содержать другие компоненты (например, кнопки, текстовые поля и т.д.)

Panel panel = new Panel();

**3. Button**: Элемент интерфейса, который можно нажимать для выполнения определенного действия.

Button button = new Button("Нажми меня");

**4. Label**: Используется для отображения текста на пользовательском интерфейсе.

Label label = new Label("Текст");

**5. TextField**: Поле, в которое пользователь может вводить текст.

TextField textField = new TextField();

**6. Canvas**: Позволяет рисовать графику (например, геометрические фигуры, изображения) на панели.

Canvas canvas = new Canvas();

### 1.2.2 Основные методы

**1. setLayout(LayoutManager manager)**: Устанавливает менеджер размещения компонентов на панели.

panel.setLayout(new BorderLayout());

**2. add(Component comp)**: Добавляет компонент на контейнер (например, на панель).

panel.add(button);

**3. setSize(int width, int height)**: Устанавливает размер окна или компонента.

frame.setSize(400, 300);

**4. setVisible(boolean b)**: Устанавливает видимость компонента

frame.setVisible(true);

**5. setForeground(Color c)** и **setBackground(Color c)**: Устанавливают цвет переднего и фонового плана для компонентов соответственно.

button.setForeground(Color.RED);  
panel.setBackground(Color.WHITE);

## 1.3 java.io

java.io - это библиотека в Java, предназначенная для работы с вводом/выводом данных (Input/Output). Она предоставляет классы и методы для чтения и записи данных из различных источников. Давайте рассмотрим основные классы и методы java.io.

### 1.3.1 Основные классы

**1. File**: Представляет файл или директорию в файловой системе.

File file = new File("file.txt");

**2. InputStream** и **OutputStream**: Абстрактные классы для ввода и вывода байтовых данных.

InputStream inputStream = new FileInputStream("input.txt");  
OutputStream outputStream = new FileOutputStream("output.txt");

**3. Reader** и **Writer**: Абстрактные классы для ввода и вывода символьных данных.

Reader reader = new FileReader("file.txt");  
Writer writer = new FileWriter("file.txt");

**4. BufferedReader** и **BufferedWriter**: Предоставляют буферизированный ввод и вывод для увеличения производительности при работе с символьными данными.

BufferedReader bufferedReader = new BufferedReader(new FileReader("file.txt"));  
BufferedWriter bufferedWriter = new BufferedWriter(new FileWriter("file.txt"));

**5. DataInputStream** и **DataOutputStream**: Позволяют записывать и читать примитивные данные в двоичном формате.

DataInputStream dataInputStream = new DataInputStream(new FileInputStream("data.bin"));  
DataOutputStream dataOutputStream = new DataOutputStream(new FileOutputStream("data.bin"));

### 1.3.2 Основные методы

**1. read()** и **write()**: Чтение и запись данных в потоке.

int data = inputStream.read();  
outputStream.write(data);

**2. read(byte[] buffer)** и **write(byte[] buffer)**: Чтение и запись массива байтов.

byte[] buffer = new byte[1024];  
int bytesRead = inputStream.read(buffer);  
outputStream.write(buffer, 0, bytesRead);

**3. close()**: Закрытие потока после завершения операций ввода/вывода.

inputStream.close();  
outputStream.close();

**4. readLine()** и **write(String str)**: Чтение и запись строки символов.

String line = bufferedReader.readLine();  
bufferedWriter.write(line);

**5. flush()**: Принудительная запись буферизованных данных в поток.

bufferedWriter.flush();

## 1.4 java.util.Properties

### 1.4.1 Основные методы

**1. setProperty(String key, String value)**: Добавляет свойство в список или обновляет значение существующего ключа.

Properties properties = new Properties();  
properties.setProperty("username", "myUsername");

**2. getProperty(String key)**: Возвращает значение, связанное с указанным ключом.

String username = properties.getProperty("username");

**3. load(InputStream inputStream)** и **store(OutputStream outputStream, String comments)**: Загружает и сохраняет свойства в файле или потоке данных.

FileInputStream fileInputStream = new FileInputStream("config.properties");  
properties.load(fileInputStream);  
FileOutputStream fileOutputStream = new FileOutputStream("config.properties");  
properties.store(fileOutputStream, "Application Configuration");

4. stringPropertyNames(): Возвращает набор ключей в виде объекта типа Set<String>.

Set<String> keys = properties.stringPropertyNames();

## 1.5 Java Mail API

### 1.5.1 Основные классы и интерфейсы

**1. Session**: Представляет сеанс почтовой передачи и содержит настройки, такие как параметры подключения (хост, порт), аутентификация и прочие свойства.

Properties props = new Properties();  
props.put("mail.smtp.auth", "true");  
props.put("mail.smtp.starttls.enable", "true");  
props.put("mail.smtp.host", "smtp.example.com");  
props.put("mail.smtp.port", "587");  
  
Session session = Session.getInstance(props, new javax.mail.Authenticator() {  
 protected PasswordAuthentication getPasswordAuthentication() {  
 return new PasswordAuthentication("username", "password");  
 }  
});

**2. Message**: Представляет электронное письмо и содержит информацию о его содержании, получателях, отправителе и т.д.

MimeMessage message = new MimeMessage(session);  
message.setFrom(new InternetAddress("from@example.com"));  
 message.addRecipient(Message.RecipientType.TO, new InternetAddress("to@example.com"));  
 message.setSubject("Subject");  
message.setText("Body of the email");

**3. Transport**: Используется для отправки сообщений через почтовый сервер.

Transport.send(message);

**4. Store**: Используется для получения сообщений из почтового ящика.

Store store = session.getStore("imap");  
store.connect("imap.example.com", "username", "password");  
Folder inbox = store.getFolder("INBOX");  
inbox.open(Folder.READ\_ONLY);  
 Message[] messages = inbox.getMessages();

## 1.6 SMTP

SMTP (Simple Mail Transfer Protocol) - это протокол, используемый для отправки электронной почты через интернет. Он определяет, как почтовые серверы отправляют, принимают и передают письма друг другу по сети. Вот основные концепции и этапы работы SMTP:

**Основные концепции SMTP:**

1. **Клиент-серверная модель:** SMTP использует архитектуру клиент-сервер, где клиентский почтовый сервер отправляет письмо через интернет, используя протокол SMTP, к серверу получателя, который также работает на SMTP.
2. **Порт:** Стандартный порт SMTP - 25 (порт 587 - также используется для отправки через шифрованное соединение).
3. **Простота:** SMTP достаточно прост в реализации и понимании, основной целью является передача текстовых сообщений от отправителя к получателю.

**Этапы работы SMTP:**

1. **Установка соединения:** Клиент (отправитель) устанавливает TCP-соединение с сервером SMTP на порт 25 или 587 (обычно).
2. **Приветствие и авторизация:** Клиент и сервер обмениваются командами, начиная с приветственного сообщения (220 сервис готов), затем происходит процесс авторизации отправителя, если он требуется.
3. **Отправка сообщения:** Клиент передает серверу отправляемое сообщение, указывая команды, такие как FROM (от кого), TO (кому), а также сам текст письма.
4. **Передача данных:** После формирования сообщения, клиент передает данные серверу. Этот процесс включает в себя кодирование, если сообщение содержит прикрепленные файлы или изображения.
5. **Завершение и разрыв соединения:** После успешной передачи данных, сервер отсылает код завершения операции (например, 250 - операция выполнена успешно), а затем разрывает соединение.

# 2. Листинг программы и UML диаграммы

## 2.1 Листинг программы и описания методов

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.\*;  
import java.util.Properties;  
import javax.mail.\*;  
import javax.mail.internet.\*;  
  
public class EmailSender {  
  
 public static void main(String[] args) {  
 EventQueue.*invokeLater*(new Runnable() {  
 public void run() {  
 *createAndShowGUI*();  
 }  
 });  
 }  
  
 private static void createAndShowGUI() {  
 JFrame frame = new JFrame("Email Sender");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 JPanel panel = new JPanel();  
 panel.setLayout(new BorderLayout());  
 frame.getContentPane().add(panel);  
  
 DefaultListModel<String> emailListModel = new DefaultListModel<>();  
 JList<String> emailList = new JList<>(emailListModel);  
 JScrollPane scrollPane = new JScrollPane(emailList);  
 panel.add(scrollPane, BorderLayout.*CENTER*);  
  
 JPanel buttonPanel = new JPanel();  
 panel.add(buttonPanel, BorderLayout.*SOUTH*);  
  
 JButton addButton = new JButton("Добавить");  
 addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 *addEmail*(emailListModel, frame);  
 }  
 });  
 buttonPanel.add(addButton);  
  
 JButton removeButton = new JButton("Удалить");  
 removeButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 *removeEmail*(emailList, emailListModel);  
 }  
 });  
 buttonPanel.add(removeButton);  
  
 JButton sendButton = new JButton("Отправить письма");  
 sendButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 *sendEmails*(emailListModel);  
 }  
 });  
 buttonPanel.add(sendButton);  
  
 *loadEmailsFromFile*("C://Users//mekai//IdeaProjects//untitled//src//emails.txt", emailListModel);  
  
 frame.pack();  
 frame.setLocationRelativeTo(null);  
 frame.setVisible(true);  
 }  
  
 private static void addEmail(DefaultListModel<String> emailListModel, JFrame frame) {  
 String email = JOptionPane.*showInputDialog*(frame, "Введите адрес электронной почты:");  
 if (email != null && !email.trim().isEmpty()) {  
 String name = JOptionPane.*showInputDialog*(frame, "Введите имя для этой почты:");  
 if (name != null && !name.trim().isEmpty()) {  
 emailListModel.addElement(email + "-" + name);  
 *saveEmailsToFile*("C://Users//mekai//IdeaProjects//untitled//src//emails.txt", emailListModel);  
 }  
 }  
 }  
  
 private static void removeEmail(JList<String> emailList, DefaultListModel<String> emailListModel) {  
 int selectedIndex = emailList.getSelectedIndex();  
 if (selectedIndex != -1) {  
 emailListModel.remove(selectedIndex);  
 *saveEmailsToFile*("C://Users//mekai//IdeaProjects//untitled//src//emails.txt", emailListModel);  
 }  
 }  
  
 private static void sendEmails(DefaultListModel<String> emailListModel) {  
 try {  
 BufferedReader cryptReader = new BufferedReader(new FileReader("C://Users//mekai//IdeaProjects//untitled//src//crypt.txt"));  
 String encryptedCredentials = cryptReader.readLine();  
 cryptReader.close();  
  
 // Дешифровка данных с помощью шифра Цезаря  
 int shift = 3;  
 String decryptedCredentials = *decrypt*(encryptedCredentials, shift);  
  
 // Разделение расшифрованной строки на почту и пароль  
 String[] credentials = decryptedCredentials.split(" ");  
  
 if (credentials.length >= 2) {  
 final String username = credentials[0];  
 final String password = credentials[1];  
  
 String smtpHost = "smtp.mail.ru"; // Измените на адрес SMTP вашего почтового сервиса  
  
 Properties props = new Properties();  
 props.put("mail.smtp.auth", "true");  
 props.put("mail.smtp.starttls.enable", "true");  
 props.put("mail.smtp.host", smtpHost);  
 props.put("mail.smtp.port", "587");  
  
 Session session = Session.*getInstance*(props, new javax.mail.Authenticator() {  
 protected PasswordAuthentication getPasswordAuthentication() {  
 return new PasswordAuthentication(username, password);  
 }  
 });  
  
 String filePathMessage = "C://Users//mekai//IdeaProjects//untitled//src//text.txt"; // Путь к файлу с текстом сообщения  
  
 BufferedReader reader = new BufferedReader(new FileReader(filePathMessage));  
 StringBuilder messageContent = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 messageContent.append(line).append("\n");  
 }  
 reader.close();  
  
 for (int i = 0; i < emailListModel.size(); i++) {  
 String[] parts = emailListModel.get(i).split("-");  
 if (parts.length >= 2) {  
 String email = parts[0];  
 String name = parts[1];  
  
 MimeMessage message = new MimeMessage(session);  
 message.setFrom(new InternetAddress(username));  
 message.addRecipient(Message.RecipientType.*TO*, new InternetAddress(email));  
  
 String personalizedMessage = "Добрый день, " + name + "!\n" + messageContent.toString();  
  
 message.setSubject("Добрый день!");  
 message.setText(personalizedMessage);  
  
 Transport.*send*(message);  
 System.*out*.println("Сообщение успешно отправлено на " + email);  
 }  
 }  
 }  
 } catch (MessagingException | IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static String decrypt(String encryptedText, int shift) {  
 StringBuilder decrypted = new StringBuilder();  
  
 for (char c : encryptedText.toCharArray()) {  
 if (Character.*isLetter*(c)) {  
 char base = Character.*isUpperCase*(c) ? 'A' : 'a';  
 decrypted.append((char) (((c - base - shift + 26) % 26) + base));  
 } else {  
 decrypted.append(c);  
 }  
 }  
  
 return decrypted.toString();  
 }  
  
 private static void loadEmailsFromFile(String filePath, DefaultListModel<String> emailListModel) {  
 try {  
 BufferedReader recipientsReader = new BufferedReader(new FileReader(filePath));  
 String recipient;  
 while ((recipient = recipientsReader.readLine()) != null) {  
 emailListModel.addElement(recipient);  
 }  
 recipientsReader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 private static void saveEmailsToFile(String filePath, DefaultListModel<String> emailListModel) {  
 try {  
 BufferedWriter writer = new BufferedWriter(new FileWriter(filePath));  
 for (int i = 0; i < emailListModel.size(); i++) {  
 writer.write(emailListModel.get(i));  
 writer.newLine();  
 }  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
}

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.\*;  
import java.util.Properties;  
import javax.mail.\*;  
import javax.mail.internet

Это блок импортов, где подключаются необходимые библиотеки и классы для работы с пользовательским интерфейсом (javax.swing), вводом-выводом (java.io), почтой (javax.mail), и другими операциями (java.util.Properties, java.awt).

public class EmailSender {  
  
 public static void main(String[] args) {  
 EventQueue.*invokeLater*(new Runnable() {  
 public void run() {  
 *createAndShowGUI*();  
 }  
 });  
 }

Это начало основного класса EmailSender. Метод main вызывает createAndShowGUI(), который запускает графический интерфейс.

private static void createAndShowGUI() {  
 // Создание и настройка главного окна приложения  
 JFrame frame = new JFrame("Email Sender");  
 frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);  
  
 // Создание панели для компонентов  
 JPanel panel = new JPanel();  
 panel.setLayout(new BorderLayout());  
 frame.getContentPane().add(panel);  
  
 // Создание списка для хранения адресов электронной почты  
 DefaultListModel<String> emailListModel = new DefaultListModel<>();  
 JList<String> emailList = new JList<>(emailListModel);  
 JScrollPane scrollPane = new JScrollPane(emailList);  
 panel.add(scrollPane, BorderLayout.CENTER);  
  
 // Создание панели с кнопками  
 JPanel buttonPanel = new JPanel();  
 panel.add(buttonPanel, BorderLayout.SOUTH);  
  
 // Кнопка "Добавить" - вызывает метод addEmail()  
 JButton addButton = new JButton("Добавить");  
 addButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 addEmail(emailListModel, frame);  
 }  
 });  
 buttonPanel.add(addButton);  
  
 // Кнопка "Удалить" - вызывает метод removeEmail()  
 JButton removeButton = new JButton("Удалить");  
 removeButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 removeEmail(emailList, emailListModel);  
 }  
 });  
 buttonPanel.add(removeButton);  
  
 // Кнопка "Отправить письма" - вызывает метод sendEmails()  
 JButton sendButton = new JButton("Отправить письма");  
 sendButton.addActionListener(new ActionListener() {  
 public void actionPerformed(ActionEvent e) {  
 sendEmails(emailListModel);  
 }  
 });  
 buttonPanel.add(sendButton);  
  
 // Загрузка адресов из файла при запуске  
 loadEmailsFromFile("C://Users//mekai//IdeaProjects//untitled//src//emails.txt", emailListModel);  
  
 // Настройка и отображение главного окна  
 frame.pack();  
 frame.setLocationRelativeTo(null);  
 frame.setVisible(true);  
}

Этот метод создает графический интерфейс: главное окно приложения с панелью для списка адресов электронной почты и кнопками для добавления, удаления адресов и отправки писем. Также он загружает адреса из файла.

private static void addEmail(DefaultListModel<String> emailListModel, JFrame frame) {  
 // Получение от пользователя адреса и имени  
 String email = JOptionPane.showInputDialog(frame, "Введите адрес электронной почты:");  
 if (email != null && !email.trim().isEmpty()) {  
 String name = JOptionPane.showInputDialog(frame, "Введите имя для этой почты:");  
 if (name != null && !name.trim().isEmpty()) {  
 // Добавление адреса и имени в список и сохранение в файл  
 emailListModel.addElement(email + "-" + name);  
 saveEmailsToFile("C://Users//mekai//IdeaProjects//untitled//src//emails.txt", emailListModel);  
 }  
 }  
}

Этот метод запрашивает у пользователя адрес электронной почты и имя, добавляет их в список и сохраняет в файл.

private static void removeEmail(JList<String> emailList, DefaultListModel<String> emailListModel) {  
 // Получение индекса выбранного адреса электронной почты  
 int selectedIndex = emailList.getSelectedIndex();  
 if (selectedIndex != -1) {  
 // Удаление выбранного адреса из списка и сохранение изменений в файл  
 emailListModel.remove(selectedIndex);  
 saveEmailsToFile("C://Users//mekai//IdeaProjects//untitled//src//emails.txt", emailListModel);  
 }  
}

Этот метод удаляет выбранный адрес электронной почты из списка и сохраняет обновленный список в файл.

private static void sendEmails(DefaultListModel<String> emailListModel) {  
 try {  
 // Чтение зашифрованных учетных данных из файла  
 BufferedReader cryptReader = new BufferedReader(new FileReader("C://Users//mekai//IdeaProjects//untitled//src//crypt.txt"));  
 String encryptedCredentials = cryptReader.readLine();  
 cryptReader.close();  
  
 // Дешифровка данных с помощью шифра Цезаря  
 int shift = 3;  
 String decryptedCredentials = decrypt(encryptedCredentials, shift);  
  
 // Разделение расшифрованной строки на почту и пароль  
 String[] credentials = decryptedCredentials.split(" ");  
  
 if (credentials.length >= 2) {  
 // Извлечение почты и пароля  
 final String username = credentials[0];  
 final String password = credentials[1];  
  
 String smtpHost = "smtp.mail.ru"; // Адрес SMTP-сервера почтового сервиса  
  
 // Настройка свойств для отправки почты  
 Properties props = new Properties();  
 props.put("mail.smtp.auth", "true");  
 props.put("mail.smtp.starttls.enable", "true");  
 props.put("mail.smtp.host", smtpHost);  
 props.put("mail.smtp.port", "587");  
  
 // Создание сессии для отправки почты  
 Session session = Session.getInstance(props, new javax.mail.Authenticator() {  
 protected PasswordAuthentication getPasswordAuthentication() {  
 return new PasswordAuthentication(username, password);  
 }  
 });  
  
 String filePathMessage = "C://Users//mekai//IdeaProjects//untitled//src//text.txt"; // Путь к файлу с текстом сообщения  
  
 // Чтение текста сообщения из файла  
 BufferedReader reader = new BufferedReader(new FileReader(filePathMessage));  
 StringBuilder messageContent = new StringBuilder();  
 String line;  
 while ((line = reader.readLine()) != null) {  
 messageContent.append(line).append("\n");  
 }  
 reader.close();  
  
 // Отправка персонализированных сообщений каждому адресату  
 for (int i = 0; i < emailListModel.size(); i++) {  
 String[] parts = emailListModel.get(i).split("-");  
 if (parts.length >= 2) {  
 String email = parts[0];  
 String name = parts[1];  
  
 // Создание письма и отправка  
 MimeMessage message = new MimeMessage(session);  
 message.setFrom(new InternetAddress(username));  
 message.addRecipient(Message.RecipientType.TO, new InternetAddress(email));  
  
 // Персонализированное сообщение  
 String personalizedMessage = "Добрый день, " + name + "!\n" + messageContent.toString();  
  
 message.setSubject("Добрый день!");  
 message.setText(personalizedMessage);  
  
 // Отправка письма  
 Transport.send(message);  
 System.out.println("Сообщение успешно отправлено на " + email);  
 }  
 }  
 }  
 } catch (MessagingException | IOException e) {  
 e.printStackTrace();  
 }  
}

Этот метод отправляет персонализированные письма каждому адресату из списка. Он читает зашифрованные учетные данные из файла, дешифрует их, настраивает параметры для отправки почты через SMTP-сервер, читает текст сообщения из файла, создает письмо с персонализированным содержимым и отправляет его каждому адресату.

private static String decrypt(String encryptedText, int shift) {  
 // Дешифровка текста с использованием шифра Цезаря  
 StringBuilder decrypted = new StringBuilder();  
  
 for (char c : encryptedText.toCharArray()) {  
 if (Character.*isLetter*(c)) {  
 char base = Character.*isUpperCase*(c) ? 'A' : 'a';  
 decrypted.append((char) (((c - base - shift + 26) % 26) + base));  
 } else {  
 decrypted.append(c);  
 }  
 }  
  
 return decrypted.toString();  
}

Этот метод дешифрует текст, зашифрованный с использованием шифра Цезаря. Он используется для расшифровки учетных данных перед отправкой почты.

private static void loadEmailsFromFile(String filePath, DefaultListModel<String> emailListModel) {  
 // Загрузка адресов электронной почты из файла  
 try {  
 BufferedReader recipientsReader = new BufferedReader(new FileReader(filePath));  
 String recipient;  
 while ((recipient = recipientsReader.readLine()) != null) {  
 emailListModel.addElement(recipient);  
 }  
 recipientsReader.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Этот метод загружает адреса электронной почты из файла и добавляет их в список.

private static void saveEmailsToFile(String filePath, DefaultListModel<String> emailListModel) {  
 // Сохранение адресов электронной почты в файл  
 try {  
 BufferedWriter writer = new BufferedWriter(new FileWriter(filePath));  
 for (int i = 0; i < emailListModel.size(); i++) {  
 writer.write(emailListModel.get(i));  
 writer.newLine();  
 }  
 writer.close();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Этот метод сохраняет адреса электронной почты из списка в файл для последующего использования.

Это основные методы программы, которые позволяют добавлять, удалять, сохранять адреса электронной почты, отправлять персонализированные сообщения через почтовый сервер и обрабатывать данные из файлов.

## 2.2 UML диаграммы

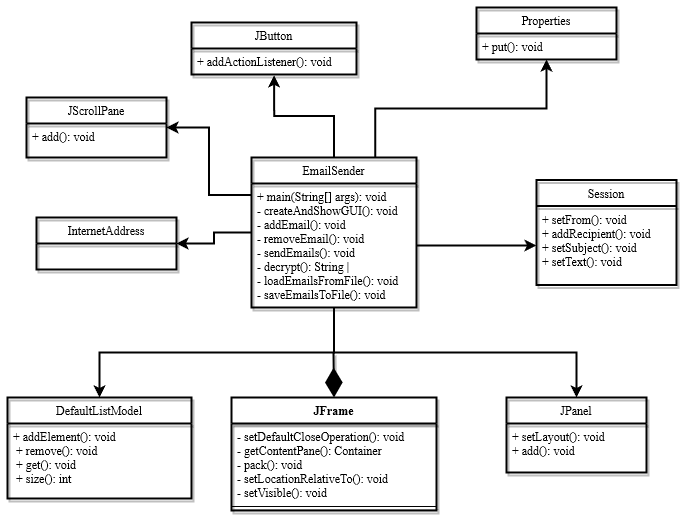


Рис. 1 - UML диаграмма классов

Совокупность всех возможных вариантов использования описывает функциональные возможности системы в целом.

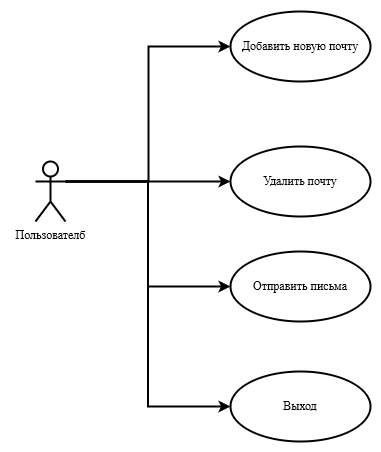


Рис.2 - Диаграмма вариантов использования

# 3. Примеры и ресурсы

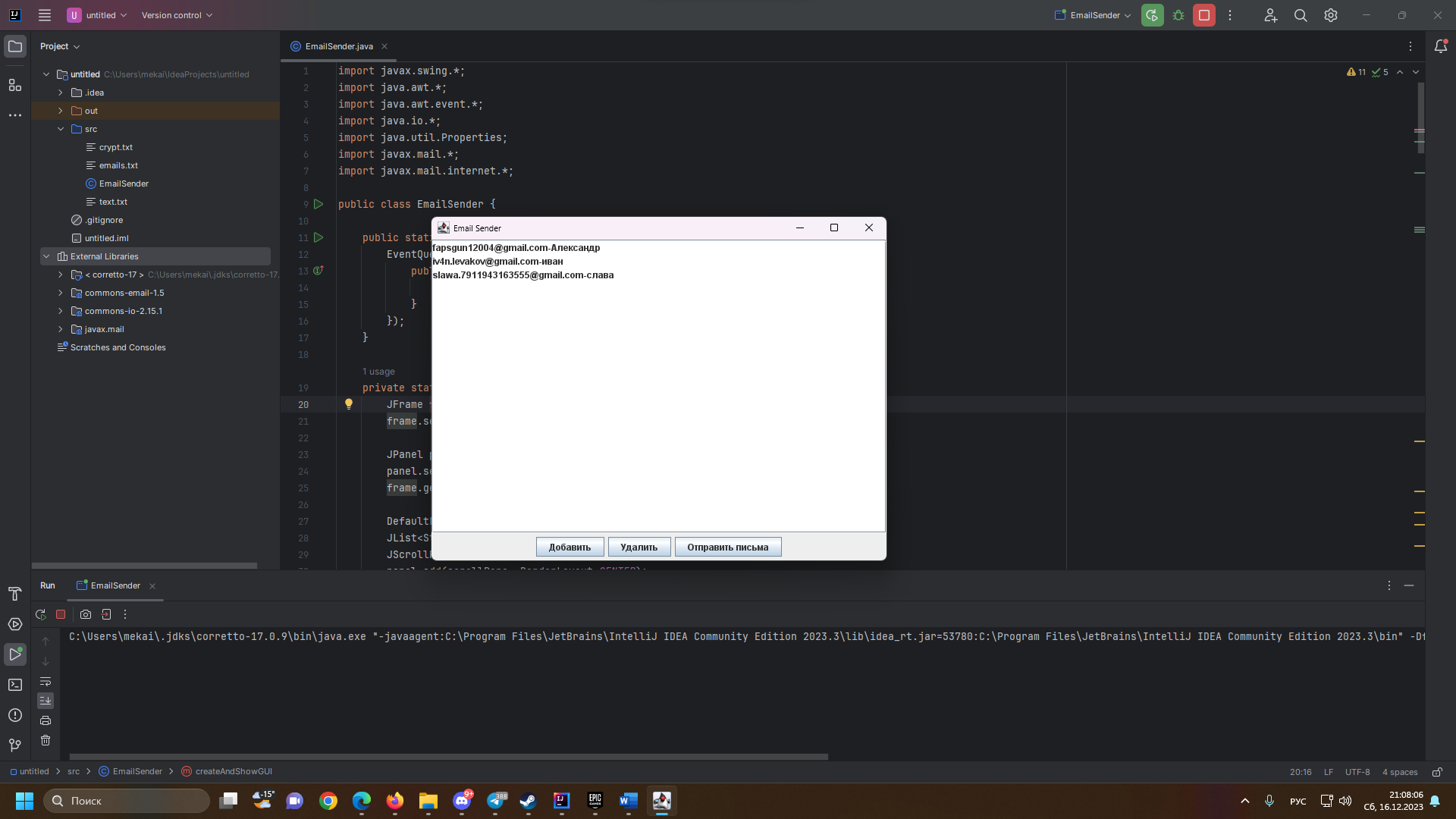


Рис.3 Интерфейс приложения

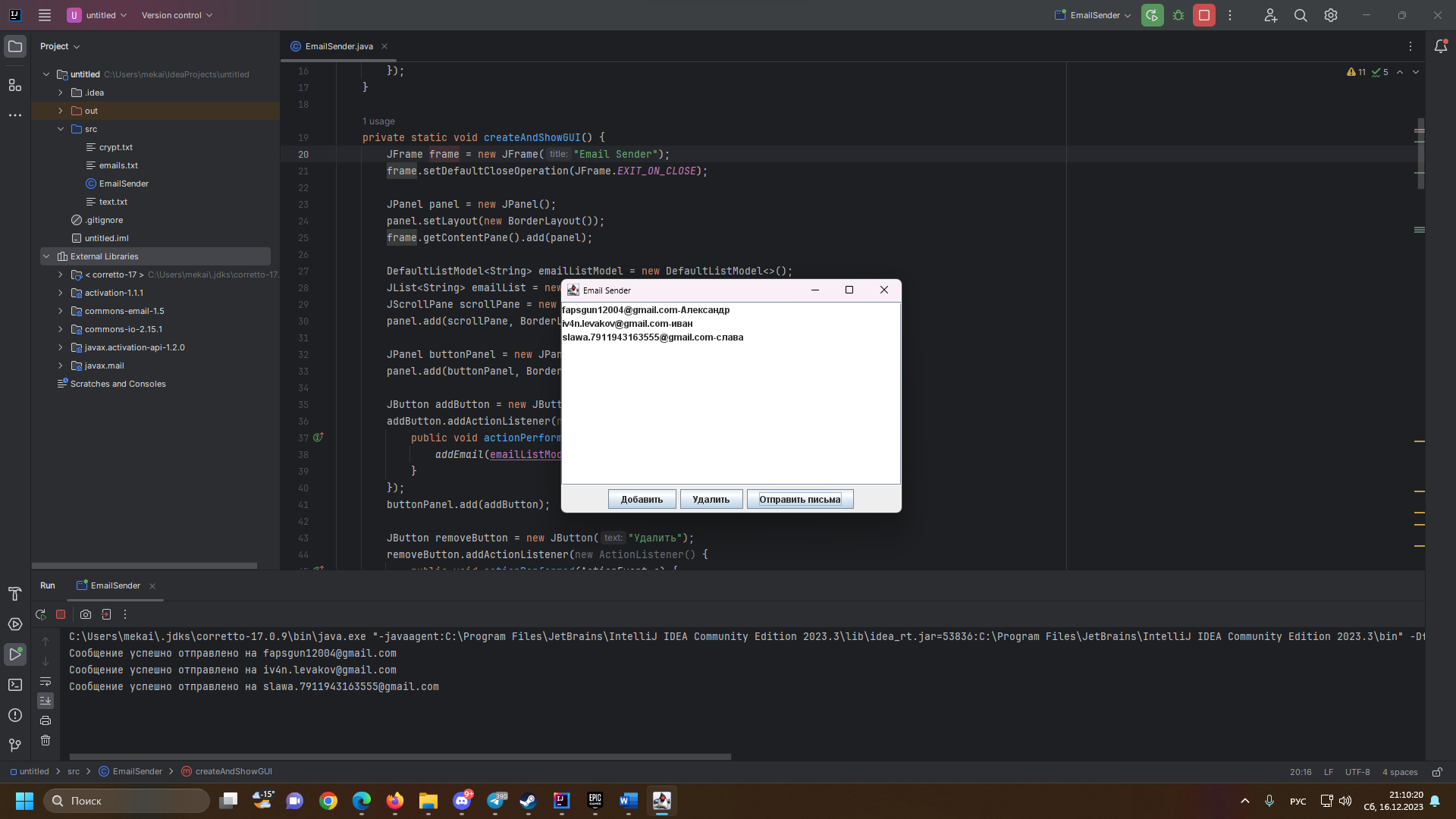


Рис.4 Успешное отправление писем на адреса

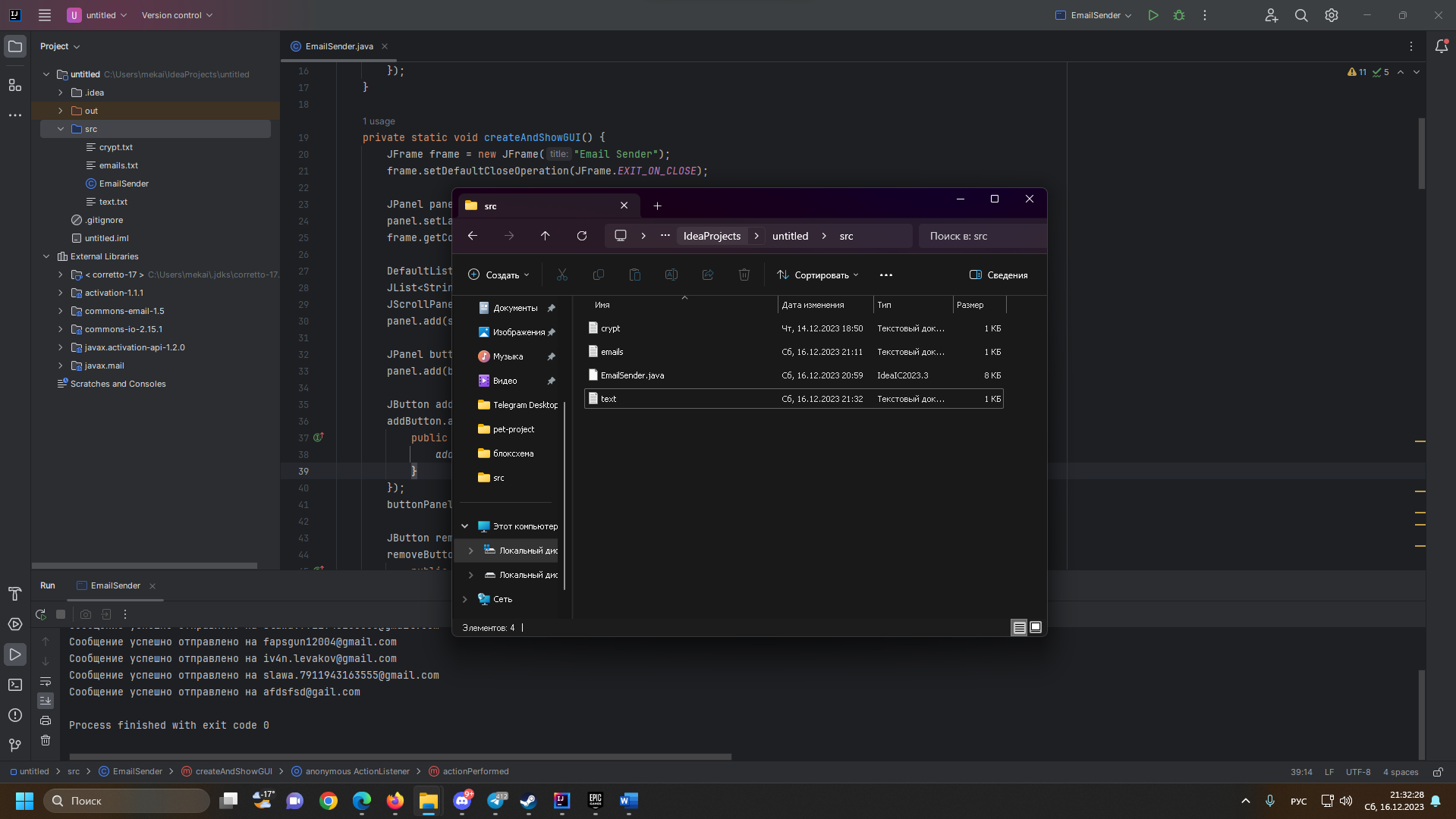


Рис.5 Файлы

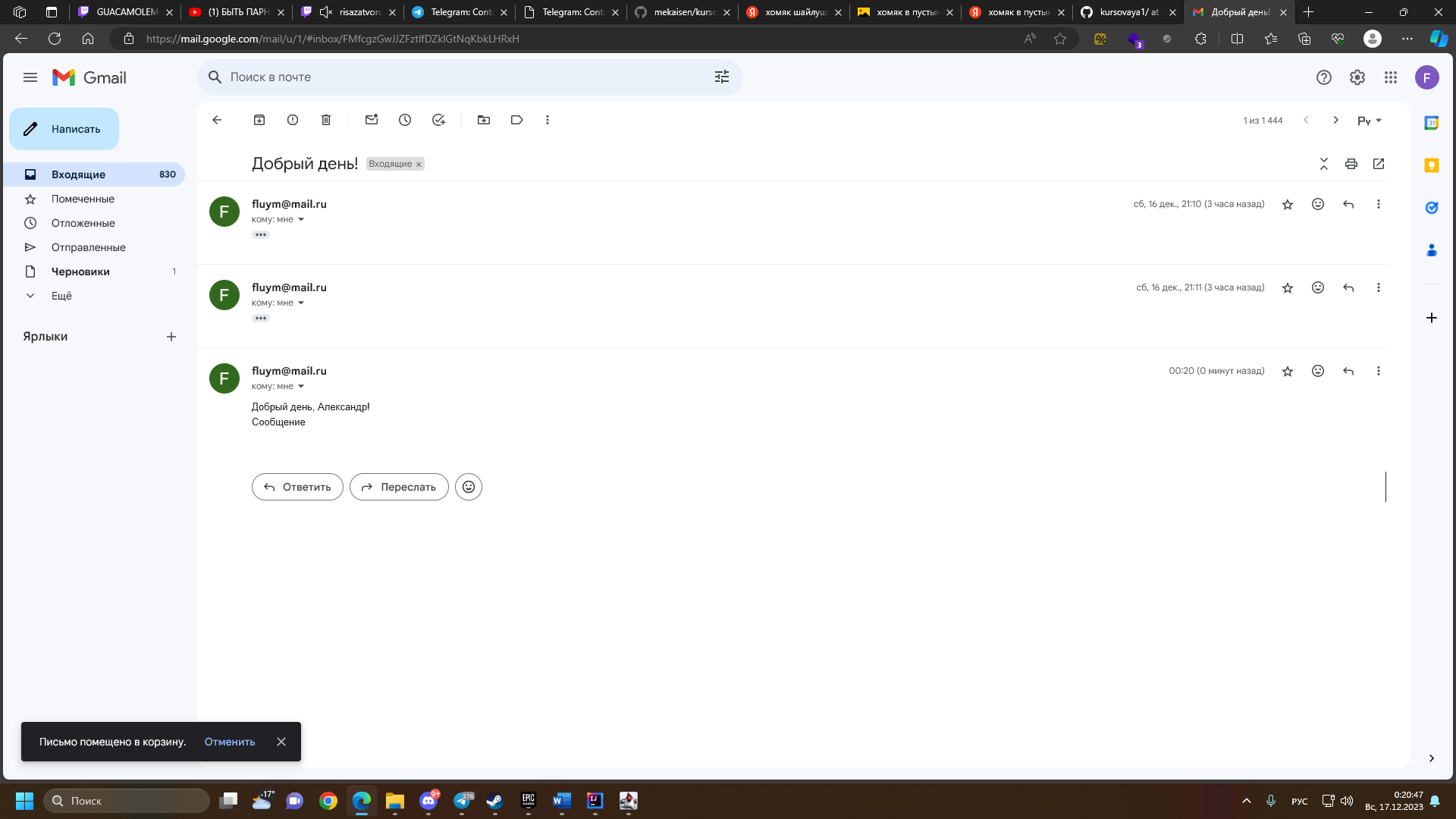


Рис. 6 сообщение пришло на почту

При разработке приложения для информационной рассылки сервисных почтовых сообщений использовались различные онлайн-ресурсы для изучения технологий и инструментов. В процессе подготовки были использованы следующие источники информации:

JavaRush (javarush.ru):

На платформе JavaRush были изучены основы Java и примеры использования JavaMail API. Здесь представлены учебные материалы и квесты, включающие работу с JavaMail.

Metanit (metanit.com):

Учебник по Java на сайте Metanit содержит раздел, посвященный работе с почтой на Java. Были изучены примеры кода и основные концепции работы с JavaMail API.

Oracle Documentation (javaee.github.io/javamail):

Официальная документация Oracle по JavaMail API была использована для более глубокого понимания классов, методов и использования API для отправки почтовых сообщений.

Также были просмотрены ютуб курсы :

<https://www.youtube.com/watch?v=lsat4guY1kc&t=486s>

https://www.youtube.com/watch?v=gUn8-jvNZrE&list=PLS1QulWo1RIaAq9jt-tOCyhh6VpMZeosW

<https://www.youtube.com/watch?v=Yc1wPreAApg>

# 4. Обработка ошибок

В случае если файл “email.txt” отсутствует, то программа не завершится, в консоль выдаст ошибку, что не был найден файл, пользователь с помощью кнопки добавить сможет внести новую почту и файл “email.txt” создастся автоматически.

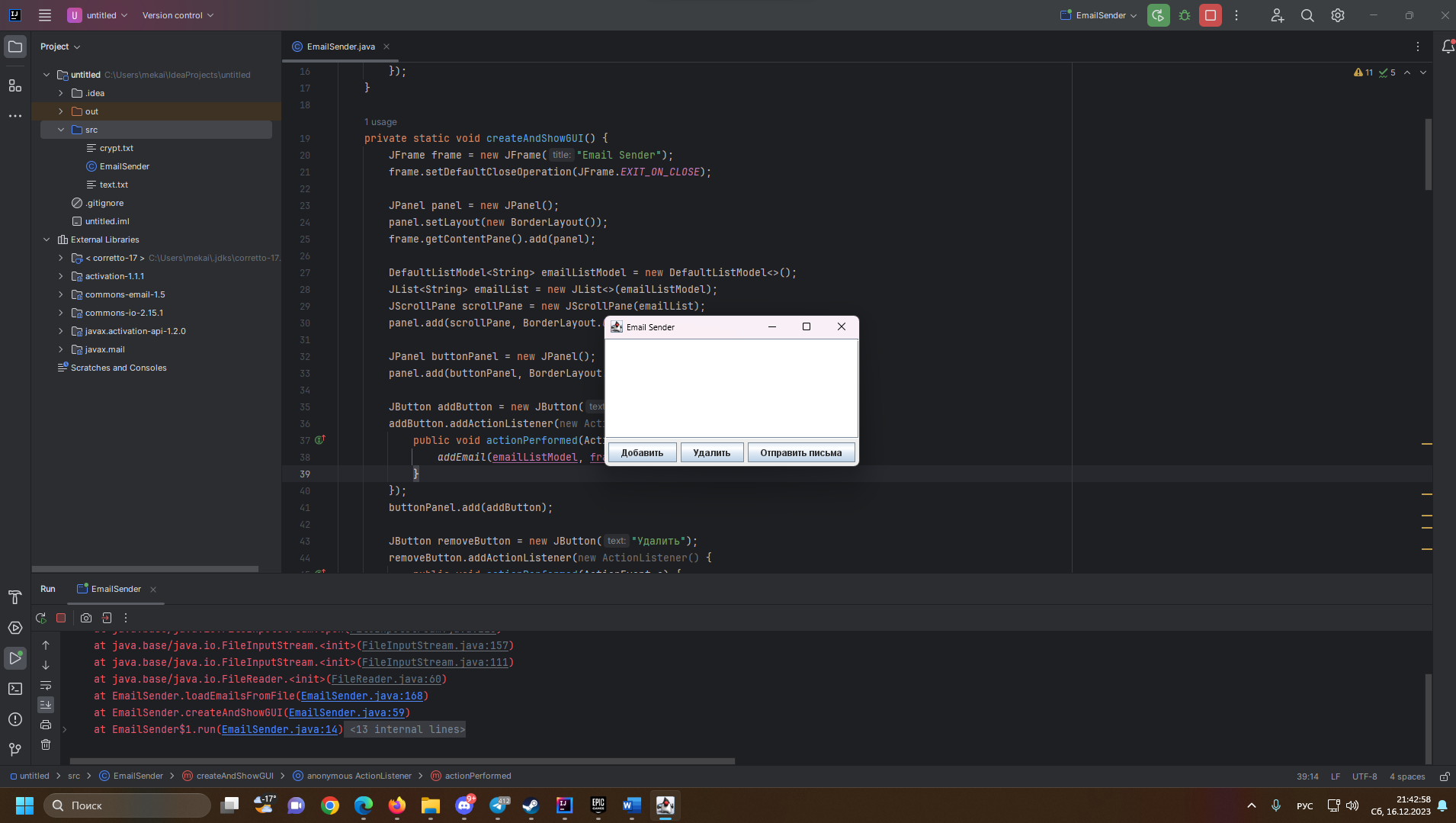


Рис.7 Ошибка в консоли

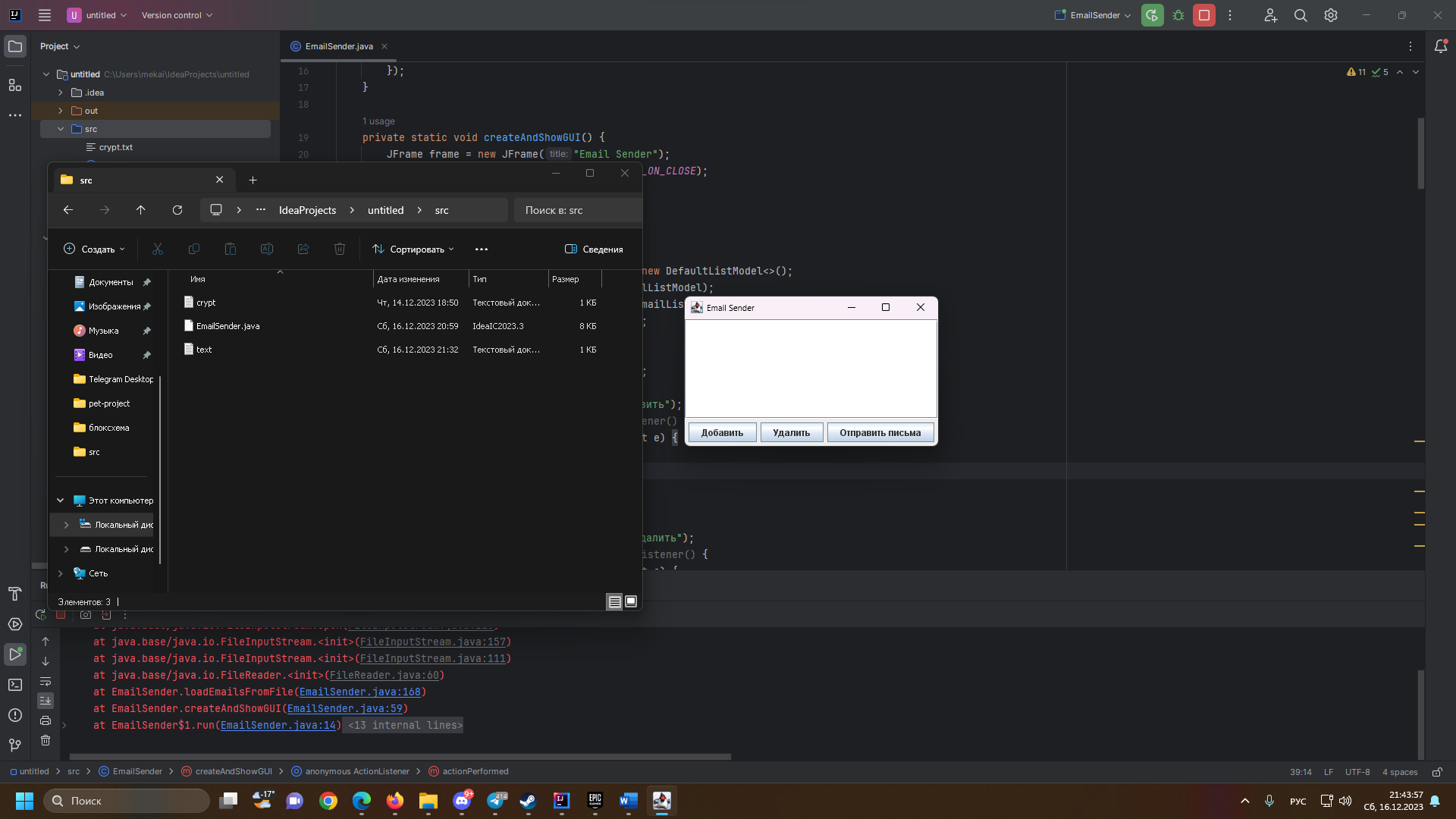


Рис.8 Отсутствие файла “email.txt”

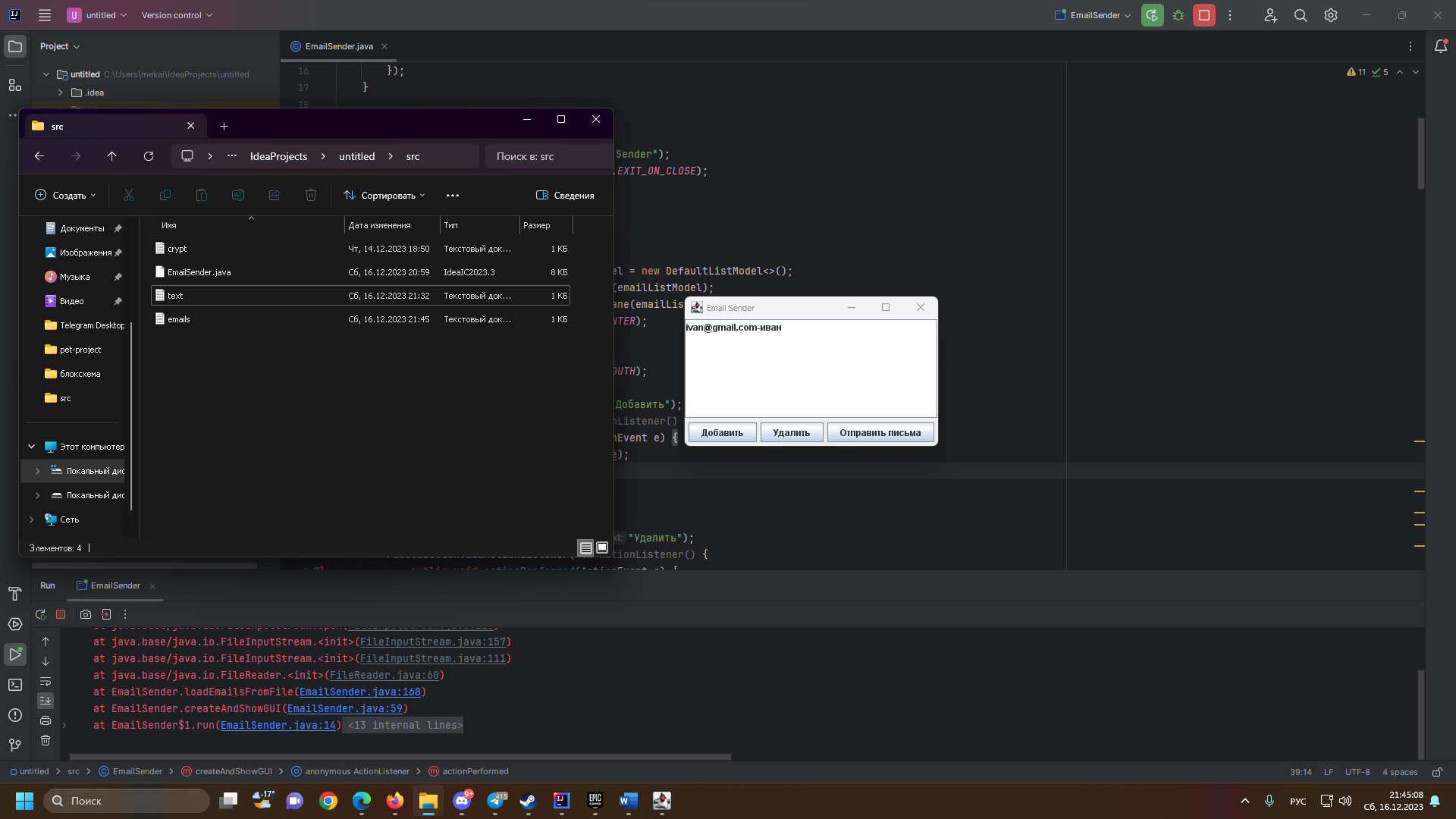


Рис.9 Создание файла “email.txt”

Метод sendEmails():

В этом методе обрабатываются исключения типа MessagingException и IOException, которые могут возникнуть при работе с JavaMail API или при чтении/записи файлов.

private static void sendEmails(DefaultListModel<String> emailListModel) {  
 try {

//наш код  
   
 } catch (MessagingException e) {  
 e.printStackTrace();  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Методы loadEmailsFromFile() и saveEmailsToFile():

Обе эти функции также могут вызвать исключение IOException при чтении или записи файлов.

private static void loadEmailsFromFile(String filePath, DefaultListModel<String> emailListModel) {  
 try {

//наш код  
   
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}  
private static void saveEmailsToFile(String filePath, DefaultListModel<String> emailListModel) {  
 try {

//наш код  
   
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
}

Метод decrypt():

Здесь происходит обработка исключений в случае возникновения проблем при работе с символами в методе шифрования.

private static String decrypt(String encryptedText, int shift) {  
 StringBuilder decrypted = new StringBuilder();  
  
 try {  
 // наш код  
   
 } catch (Exception e) {  
 e.printStackTrace(); // или другая обработка ошибки  
 }  
  
 return decrypted.toString();  
}

# **Заключение**

Разработка приложения для информационной рассылки сервисных почтовых сообщений является важным шагом в обеспечении своевременного и эффективного обмена информацией. В процессе создания данного приложения были учтены основные требования к безопасности, надежности и удобству использования.

Использование JavaMail API, совмещенное с библиотеками Swing, AWT и Java I/O, позволило создать функциональное приложение, способное осуществлять отправку сообщений множеству адресатов с минимальными усилиями со стороны пользователя.

Изучение материалов и руководств по JavaMail API на различных онлайн-платформах, таких как JavaRush, Metanit, официальная документация Oracle.

Разработанное приложение представляет собой удобный инструмент для автоматизации отправки почтовых сообщений, что способствует повышению эффективности и оперативности ведения деловой переписки.

В целом, процесс разработки данного приложения позволил расширить знания в области Java-разработки и применения JavaMail API.

# **Список литературы**

1. Шилдт, Герберт. Ш57 Java. Полное руководство, 12-е изд. : Пер. с англ. - СПб. "Диалектика•; 2023. - 1344 с.: ил. - Парал. тит. англ.

2. Блох, Джошуа Б70 Java: эффективное программирование, 3-е изд. : Пер. с англ. — СПб. : ООО “Диалектика”, 2019. — 464 с.: ил. — Парал. тит. англ.

3. Дубаков А.А. Введение в объектно-ориентированное программирование на Java: учебное пособие – СПб: Университет ИТМО, 2016. – 250 с.

4. Кэти Сьерра, Берт Бейтс **"Изучаем Java"** – 717 c.