ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ  
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ  
УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»**

Институт компьютерных наук и кибербезопасности

Высшая школа программной инженерии

**Курсовая работа**по дисциплине  
«Объектно-ориентированное программирование»

Выполнил:

студент группы в5130904/20322 Девотченко А.В.

Проверил:

преподаватель Маслаков А.П.

Санкт-Петербург

2024

**Задание**

Разработать приложение с графическим интерфейсом для заданий 1–4. Для этого приложения должна быть реализована возможность выбора из списка любого приложения, ввод входных данных и его выполнение. Модифицировать задания 1–4 так, чтобы весь вывод происходил в текстовых областях, защищённых от редактирования. Предусмотреть для заданий:

• 3 - выбор файлов словаря и текста для перевода, возможность ручного ввода текста;

• 4 - ввод входных данных для методов.

Блок практических заданий 1.1-1.4 призван сформировать у студента понимание особенностей хранения, умение настраивать и поддерживать данные.

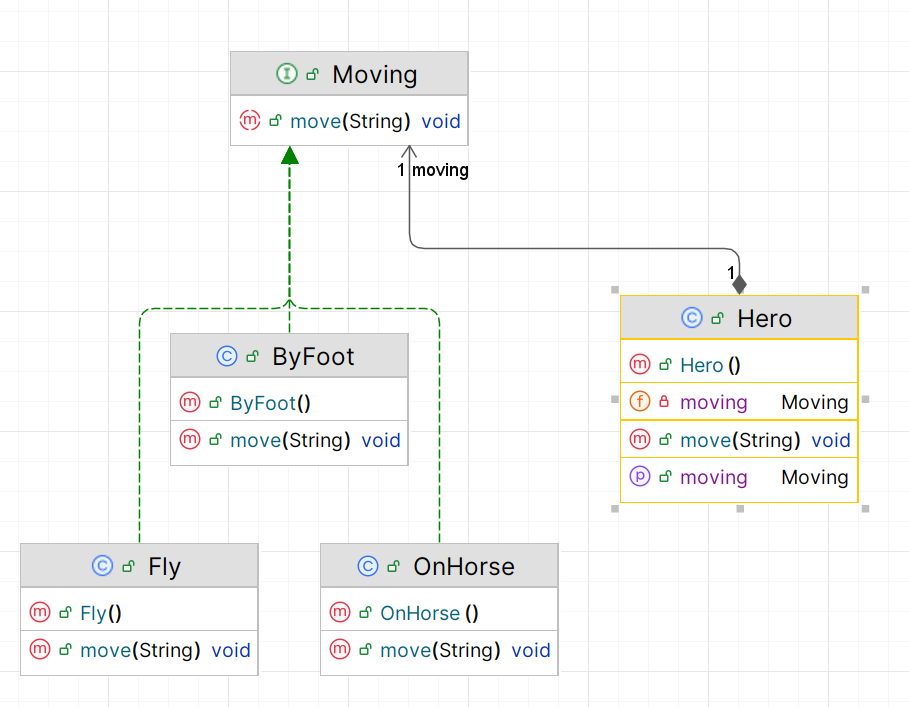
**Ход работы**

**Выбор технологий**

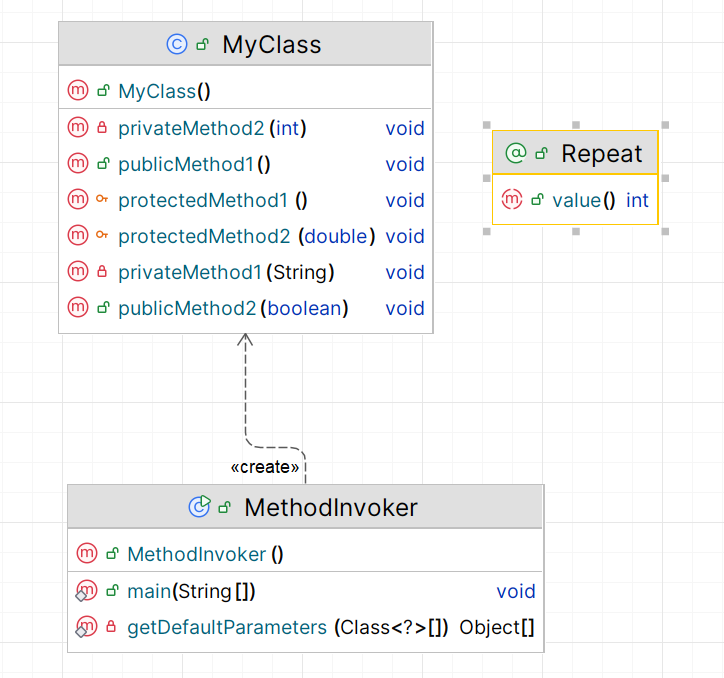
В качестве технологии для создания графического приложения курсовой работы, был выбран изученный в ходе лекций фреймворк JavaFX. Разработка велась в IntelliJ IDEA с использованием Scene Builder для удобства настройки FXML-разметки.

**Диаграммы классов**

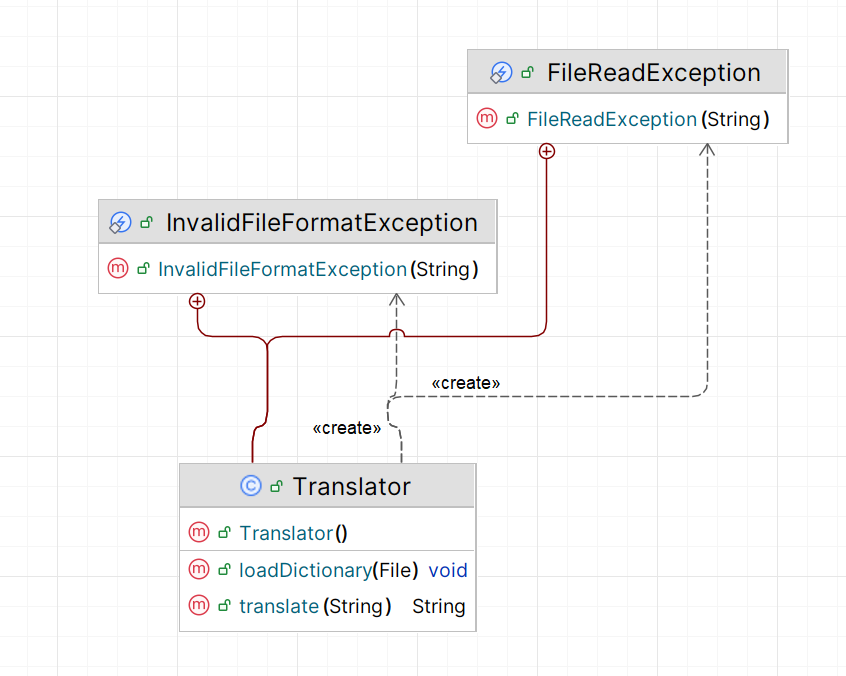
**Лабораторная работа 1**



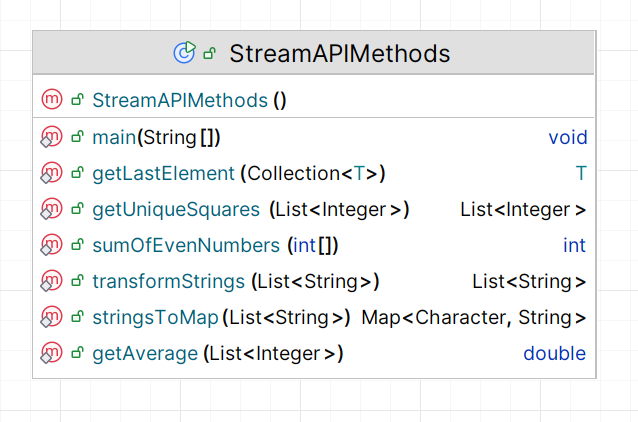
**Лабораторная работа 2**



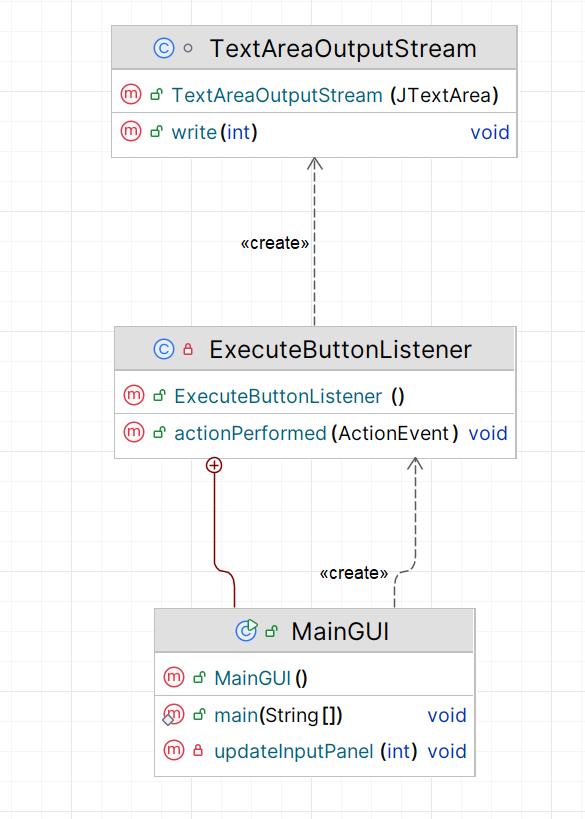
**Лабораторная работа 3**



**Лабораторная работа 4**

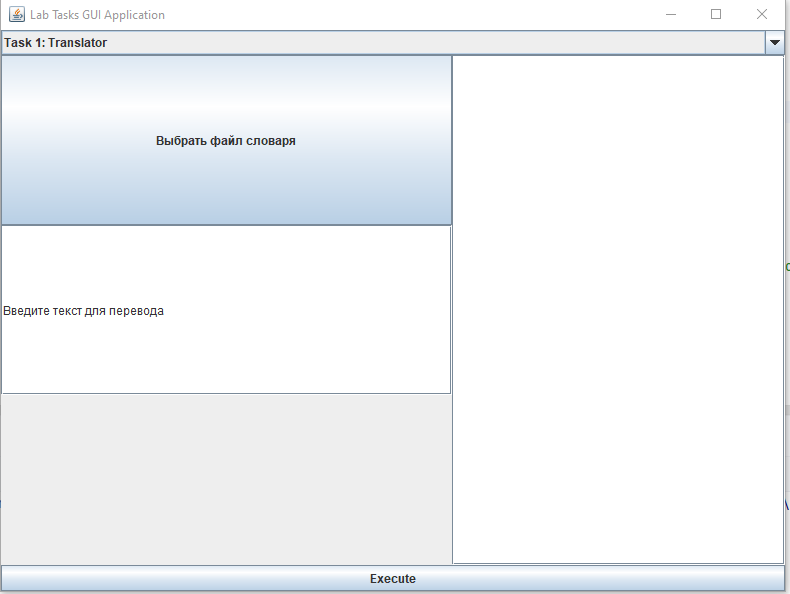


**Курсовая работа**



**Разработка приложения.**

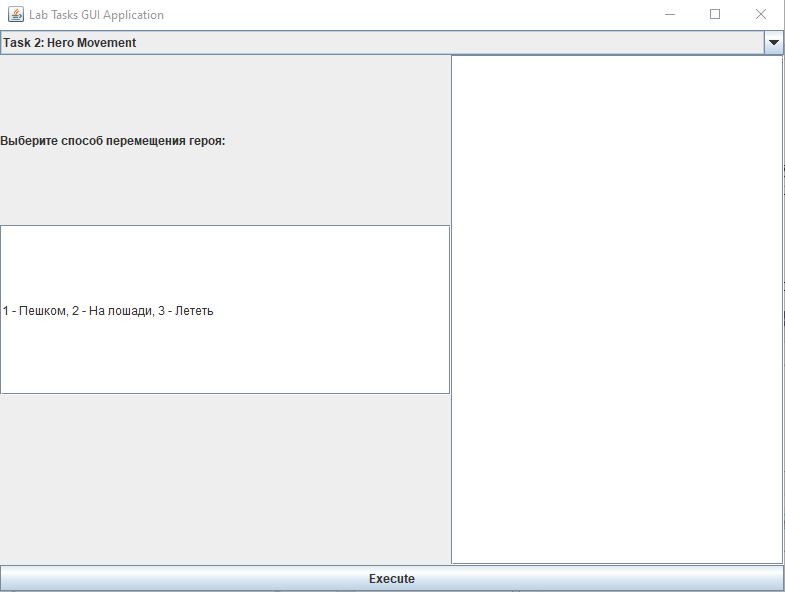
Для каждой из лабораторных работ отведена отдельная вкладка внутри приложения, кнопка для запуска выполнения программы, и поле вывода информации о выполнении программы



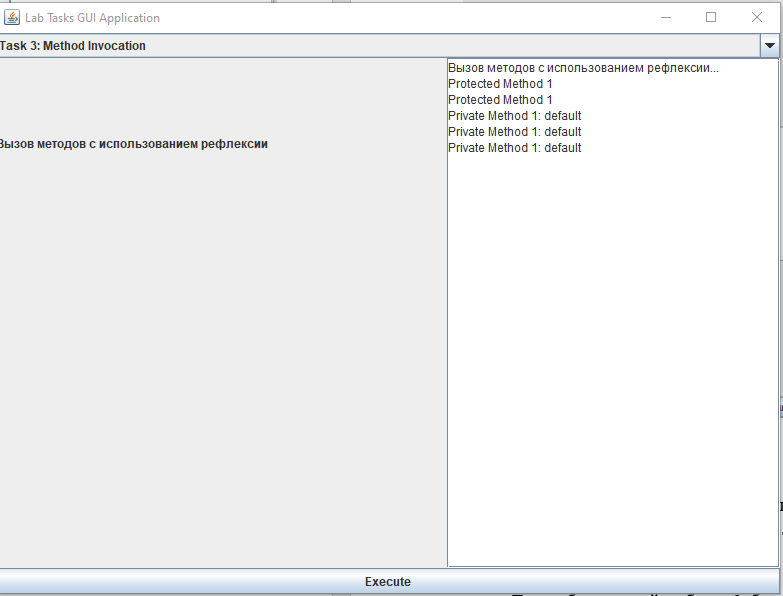
Вызов лабораторных работ происходит посредством вызова их скомпилированного файла.

После вызова лабораторной работы, её стандартный вывод и вывод ошибок перехватываются и выводятся в соответствующее поле вывода.

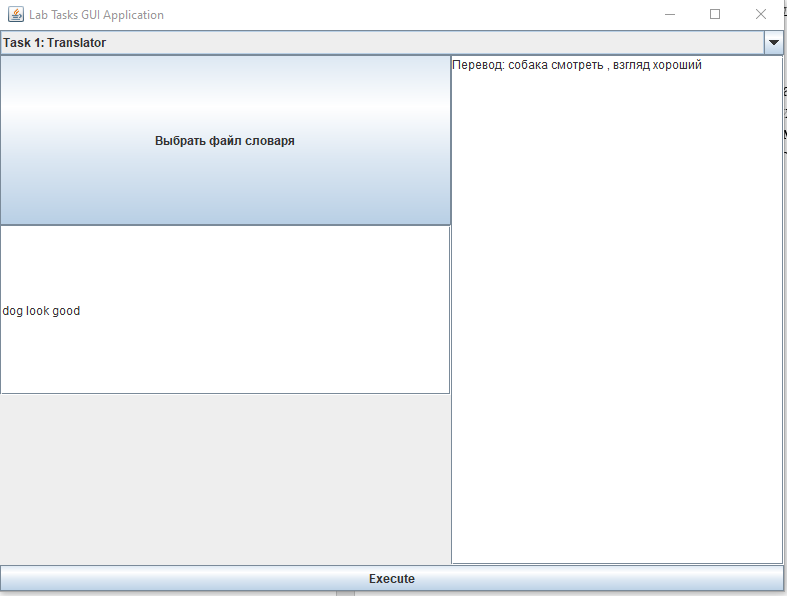
Для первой лабораторной работы было также добавлено поле для ввода действий



Так как в лабораторной работе 2 количество вызовов для методов указываются в аннотации на этапе компиляции, дополнительных полей ввода тут добавлено не было. Пример вывода для лабораторной работы 2.

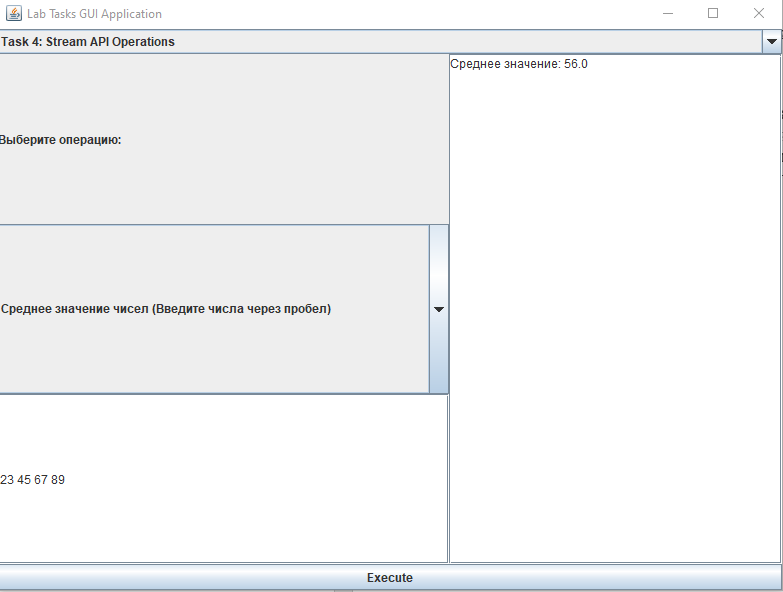


Для лабораторной работы 3 было добавлено поле для ввода путей к файлам словарей, а также поле для ввода текста для перевода. Пример вывода для лабораторной работы 3.



Для последней лабораторной работы были добавлены поля для ввода соответствующих списков с данными.

Пример вывода для лабораторной работы 4.



Для завершения графического приложения достаточно закрыть его окно.

**Вывод**

В ходе курсовой работы было разработано приложение с графическим интерфейсом для лабораторных раб 1-4, которое позволяет вызывать каждую из них с указанными параметрами, которые можно указать через поля ввода. Также, в графическом интерфейсе отображается вывод вызываемых программ, защищённый от изменений.

Сами работы были модифицированы для получения входных данных через параметры главного метода.

Для лабораторной работы 3 предусмотрен выбор файлов словаря и текста для перевода, возможность ручного ввода текста.

Для лабораторной работы 4 предусмотрен ввод входных данных для методов.

По итогу выполнения курсовой работы было сформировано понимание особенностей хранения, умение настраивать и поддерживать данные, а также были закреплены знания по изученному фреймворку JavaFX для создания графических приложений на языке программирования Java.

**Приложение****Исходный код лабораторной работы 1**

package lab\_1;  
  
public class ByFoot implements Moving {  
 @Override  
 public void move(String action) {  
 System.*out*.println("Moving by foot: " + action);  
 }  
}

package lab\_1;  
  
public class Fly implements Moving {  
 @Override  
 public void move(String move) {  
 System.*out*.printf("%s герой летит%n", move); *//test* }  
}

package lab\_1;  
  
import lab\_1.Moving;  
  
public class Hero {  
 private Moving moving;  
  
 *// Метод для установки стратегии перемещения* public void setMoving(Moving moving) {  
 this.moving = moving;  
 }  
  
 *// Метод для выполнения перемещения* public void move(String move) {  
 if (moving != null) {  
 moving.move(move);  
 } else {  
 System.*out*.println("Moving strategy is not set!"); *//test* }  
 }  
}

package lab\_1;  
  
public class OnHorse implements Moving {  
 @Override  
 public void move(String move) {  
 System.*out*.printf("%s герой едет на лошади%n", move); *//test* }  
}

package lab\_1;  
  
public interface Moving {  
 void move(String action);  
}

**Исходный код лабораторной работы 2**

package lab\_2;  
  
import java.lang.reflect.Method;  
  
public class MethodInvoker {  
  
 public static void main(String[] args) {  
 MyClass myObject = new MyClass();  
 Class<?> clazz = myObject.getClass();  
  
 *// Получаем все методы класса* Method[] methods = clazz.getDeclaredMethods();  
  
 for (Method method : methods) {  
 *// Проверяем, что метод защищённый или приватный* int modifiers = method.getModifiers();  
 if (!(java.lang.reflect.Modifier.*isPrivate*(modifiers) || java.lang.reflect.Modifier.*isProtected*(modifiers))) {  
 continue; *// Пропускаем, если метод не приватный и не защищённый* }  
  
 *// Проверяем, есть ли аннотация @Repeat* if (method.isAnnotationPresent(Repeat.class)) {  
 Repeat repeat = method.getAnnotation(Repeat.class);  
 int times = repeat.value();  
  
 method.setAccessible(true); *// Делаем метод доступным* for (int i = 0; i < times; i++) {  
 try {  
 *// Генерируем массив параметров с нужной длиной и значениями по умолчанию* Object[] params = *getDefaultParameters*(method.getParameterTypes());  
  
 *// Вызываем метод с параметрами* method.invoke(myObject, params);  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
 }  
 }  
 }  
  
 *// Метод для генерации параметров по умолчанию* private static Object[] getDefaultParameters(Class<?>[] parameterTypes) {  
 Object[] params = new Object[parameterTypes.length];  
  
 for (int i = 0; i < parameterTypes.length; i++) {  
 Class<?> paramType = parameterTypes[i];  
  
 if (paramType == int.class) {  
 params[i] = 0;  
 } else if (paramType == double.class) {  
 params[i] = 0.0;  
 } else if (paramType == boolean.class) {  
 params[i] = false;  
 } else if (paramType == String.class) {  
 params[i] = "default";  
 } else {  
 params[i] = null; *// Для остальных типов* }  
 }  
  
 return params;  
 }  
}

package lab\_2;  
  
public class MyClass {  
  
 @Repeat(3)  
 private void privateMethod1(String message) {  
 System.*out*.println("Private Method 1: " + message);  
 }  
  
 private void privateMethod2(int number) {  
 System.*out*.println("Private Method 2: " + number);  
 }  
  
 @Repeat(2)  
 protected void protectedMethod1() {  
 System.*out*.println("Protected Method 1");  
 }  
  
 protected void protectedMethod2(double value) {  
 System.*out*.println("Protected Method 2: " + value);  
 }  
  
 public void publicMethod1() {  
 System.*out*.println("Public Method 1");  
 }  
  
 @Repeat(1)  
 public void publicMethod2(boolean flag) {  
 System.*out*.println("Public Method 2: " + flag);  
 }  
}

package lab\_2;  
  
import java.lang.annotation.Retention;  
import java.lang.annotation.RetentionPolicy;  
  
@Retention(RetentionPolicy.*RUNTIME*)  
public @interface Repeat {  
 int value();  
}

**Исходный код лабораторной работы 3**

package lab\_3;  
  
import java.io.BufferedReader;  
import java.io.File;  
import java.io.FileReader;  
import java.io.IOException;  
import java.util.HashMap;  
import java.util.Map;  
  
public class Translator {  
  
 private Map<String, String> dictionary = new HashMap<>();  
 private int maxPhraseLength = 1;  
  
 *// Метод для чтения словаря из файла* public void loadDictionary(File dictionaryFile) throws InvalidFileFormatException, FileReadException {  
 try (BufferedReader reader = new BufferedReader(new FileReader(dictionaryFile))) {  
 String line;  
 while ((line = reader.readLine()) != null) {  
 String[] parts = line.split("\\|");  
 if (parts.length == 2) {  
 String key = parts[0].trim().toLowerCase();  
 dictionary.put(key, parts[1].trim());  
 *// Обновление максимальной длины фразы* int wordCount = key.split(" ").length;  
 if (wordCount > maxPhraseLength) {  
 maxPhraseLength = wordCount;  
 }  
 } else {  
 throw new InvalidFileFormatException("Неверный формат строки в словаре: " + line);  
 }  
 }  
 } catch (IOException e) {  
 throw new FileReadException("Ошибка при чтении файла словаря: " + e.getMessage());  
 }  
 }  
  
 *// Метод для перевода текста* public String translate(String input) {  
 String[] words = input.split(" ");  
 StringBuilder translatedText = new StringBuilder();  
  
 int i = 0;  
 while (i < words.length) {  
 boolean found = false;  
 int maxMatchLength = Math.*min*(maxPhraseLength, words.length - i);  
 for (int len = maxMatchLength; len > 0; len--) {  
 StringBuilder phraseBuilder = new StringBuilder();  
 for (int j = 0; j < len; j++) {  
 if (j > 0) {  
 phraseBuilder.append(" ");  
 }  
 phraseBuilder.append(words[i + j].toLowerCase());  
 }  
 String phrase = phraseBuilder.toString();  
 if (dictionary.containsKey(phrase)) {  
 translatedText.append(dictionary.get(phrase)).append(" ");  
 i += len;  
 found = true;  
 break;  
 }  
 }  
 if (!found) {  
 translatedText.append(words[i]).append(" ");  
 i++;  
 }  
 }  
  
 return translatedText.toString().trim();  
 }  
  
 *// Исключения* public static class InvalidFileFormatException extends Exception {  
 public InvalidFileFormatException(String message) {  
 super(message);  
 }  
 }  
  
 public static class FileReadException extends Exception {  
 public FileReadException(String message) {  
 super(message);  
 }  
 }  
}

**Исходный код лабораторной работы 4**

package lab\_4;  
  
import java.util.\*;  
import java.util.stream.\*;  
  
public class StreamAPIMethods {  
  
 *// Метод, возвращающий среднее значение списка целых чисел* public static double getAverage(List<Integer> numbers) {  
 return numbers.stream()  
 .mapToInt(Integer::intValue)  
 .average()  
 .orElseThrow(() -> new IllegalArgumentException("Список пуст"));  
 }  
  
 *// Метод, приводящий все строки в списке в верхний регистр и добавляющий к ним префикс "\_new\_"* public static List<String> transformStrings(List<String> strings) {  
 return strings.stream()  
 .map(s -> "\_new\_" + s.toUpperCase())  
 .collect(Collectors.*toList*());  
 }  
  
 *// Метод, возвращающий список квадратов всех встречающихся только один раз элементов списка* public static List<Integer> getUniqueSquares(List<Integer> numbers) {  
 return numbers.stream()  
 .collect(Collectors.*groupingBy*(n -> n, Collectors.*counting*()))  
 .entrySet().stream()  
 .filter(e -> e.getValue() == 1)  
 .map(e -> e.getKey() \* e.getKey())  
 .collect(Collectors.*toList*());  
 }  
  
 *// Метод, принимающий на вход коллекцию и возвращающий ее последний элемент или кидающий исключение, если коллекция пуста* public static <T> T getLastElement(Collection<T> collection) {  
 return collection.stream()  
 .reduce((first, second) -> second)  
 .orElseThrow(() -> new NoSuchElementException("Коллекция пуста"));  
 }  
  
 *// Метод, принимающий на вход массив целых чисел, возвращающий сумму чётных чисел или 0, если чётных чисел нет* public static int sumOfEvenNumbers(int[] numbers) {  
 return Arrays.*stream*(numbers)  
 .filter(n -> n % 2 == 0)  
 .sum();  
 }  
  
 *// Метод, преобразовывающий все строки в списке в Map, где первый символ – ключ, оставшиеся – значение* public static Map<Character, String> stringsToMap(List<String> strings) {  
 return strings.stream()  
 .filter(s -> !s.isEmpty())  
 .collect(Collectors.*toMap*(  
 s -> s.charAt(0),  
 s -> s.substring(1),  
 (existing, replacement) -> existing *// при наличии дубликатов ключей* ));  
 }  
  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 while(true) {  
 System.*out*.println("Выберите метод для выполнения:");  
 System.*out*.println("1. Найти среднее значение списка целых чисел");  
 System.*out*.println("2. Преобразовать строки в верхний регистр с префиксом \"\_new\_\"");  
 System.*out*.println("3. Получить список квадратов уникальных элементов списка");  
 System.*out*.println("4. Получить последний элемент коллекции");  
 System.*out*.println("5. Найти сумму четных чисел в массиве");  
 System.*out*.println("6. Преобразовать строки в Map (первый символ - ключ, остальные - значение)");  
 System.*out*.println("0. Выход");  
  
 int choice = Integer.*parseInt*(scanner.nextLine());  
  
 switch(choice) {  
 case 1:  
 System.*out*.println("Введите числа через пробел:");  
 String[] nums1 = scanner.nextLine().split("\\s+");  
 List<Integer> list1 = Arrays.*stream*(nums1)  
 .map(Integer::*parseInt*)  
 .collect(Collectors.*toList*());  
 double average = *getAverage*(list1);  
 System.*out*.println("Среднее значение: " + average);  
 break;  
 case 2:  
 System.*out*.println("Введите строки через запятую:");  
 String[] strs2 = scanner.nextLine().split(",");  
 List<String> list2 = Arrays.*stream*(strs2)  
 .map(String::trim)  
 .collect(Collectors.*toList*());  
 List<String> transformedStrings = *transformStrings*(list2);  
 System.*out*.println("Преобразованные строки: " + transformedStrings);  
 break;  
 case 3:  
 System.out.println("Введите числа через пробел:");  
 String[] nums3 = scanner.nextLine().split("\\s+");  
 List<Integer> list3 = Arrays.stream(nums3)  
 .map(Integer::parseInt)  
 .collect(Collectors.toList());  
 List<Integer> uniqueSquares = getUniqueSquares(list3);  
 System.out.println("Квадраты уникальных элементов: " + uniqueSquares);  
 break;  
 case 4:  
 System.out.println("Введите элементы коллекции через запятую:");  
 String[] elems = scanner.nextLine().split(",");  
 List<String> collection = Arrays.stream(elems)  
 .map(String::trim)  
 .collect(Collectors.toList());  
 try {  
 String lastElement = getLastElement(collection);  
 System.out.println("Последний элемент: " + lastElement);  
 } catch (NoSuchElementException e) {  
 System.out.println(e.getMessage());  
 }  
 break;  
 case 5:  
 System.out.println("Введите числа через пробел:");  
 String[] nums5 = scanner.nextLine().split("\\s+");  
 int[] array = Arrays.stream(nums5)  
 .mapToInt(Integer::parseInt)  
 .toArray();  
 int sumEven = sumOfEvenNumbers(array);  
 System.out.println("Сумма четных чисел: " + sumEven);  
 break;  
 case 6:  
 System.out.println("Введите строки через запятую:");  
 String[] strs6 = scanner.nextLine().split(",");  
 List<String> list6 = Arrays.stream(strs6)  
 .map(String::trim)  
 .collect(Collectors.toList());  
 Map<Character, String> map = stringsToMap(list6);  
 System.out.println("Результирующая Map: " + map);  
 break;  
 case 0:  
 System.out.println("Выход из программы.");  
 scanner.close();  
 return;  
 default:  
 System.out.println("Неверный выбор. Попробуйте снова.");  
 }  
 System.out.println();  
 }  
 }  
}

**Исходный код курсовой работы**

package Lab\_5;  
  
import lab\_1.Hero;  
import lab\_1.Fly;  
import lab\_1.ByFoot;  
import lab\_1.OnHorse;  
  
import lab\_2.MethodInvoker;  
  
import lab\_3.Translator;  
  
import lab\_4.StreamAPIMethods;  
  
import java.util.List;  
import java.util.Arrays;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.File;  
import java.io.PrintStream;  
import java.util.Map;  
import java.util.NoSuchElementException;  
import java.util.stream.Collectors;  
  
public class MainGUI extends JFrame {  
 private JTextArea outputArea;  
 private JComboBox<String> taskSelector;  
 private JPanel inputPanel;  
 private File dictionaryFile; *// Выбранный файл словаря* public MainGUI() {  
 setTitle("Lab Tasks GUI Application");  
 setSize(800, 600);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 *// Components* taskSelector = new JComboBox<>(new String[]{  
 "Task 1: Translator",  
 "Task 2: Hero Movement",  
 "Task 3: Method Invocation",  
 "Task 4: Stream API Operations"  
 });  
 JButton executeButton = new JButton("Execute");  
 outputArea = new JTextArea(20, 30);  
 outputArea.setEditable(false);  
 JScrollPane scrollPane = new JScrollPane(outputArea);  
 inputPanel = new JPanel(new GridLayout(3, 1));  
  
 *// Add components to frame* add(taskSelector, BorderLayout.*NORTH*);  
 add(inputPanel, BorderLayout.*CENTER*);  
 add(executeButton, BorderLayout.*SOUTH*);  
 add(scrollPane, BorderLayout.*EAST*);  
  
 *// Action listener for the execute button* executeButton.addActionListener(new ExecuteButtonListener());  
  
 *// Action listener for task selection to update input panel* taskSelector.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 updateInputPanel(taskSelector.getSelectedIndex());  
 }  
 });  
  
 *// Initialize the input panel for the first selected task* updateInputPanel(taskSelector.getSelectedIndex());  
  
 setVisible(true);  
 }  
  
 private void updateInputPanel(int selectedTask) {  
 inputPanel.removeAll();  
 switch (selectedTask) {  
 case 0:  
 *// Task 1: Translator* JButton chooseFileButton = new JButton("Выбрать файл словаря");  
 chooseFileButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser fileChooser = new JFileChooser();  
 int result = fileChooser.showOpenDialog(null);  
 if (result == JFileChooser.*APPROVE\_OPTION*) {  
 dictionaryFile = fileChooser.getSelectedFile();  
 outputArea.setText("Выбран файл словаря: " + dictionaryFile.getAbsolutePath());  
 }  
 }  
 });  
 JTextField textInput = new JTextField("Введите текст для перевода");  
 textInput.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (textInput.getText().equals("Введите текст для перевода")) {  
 textInput.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (textInput.getText().isEmpty()) {  
 textInput.setText("Введите текст для перевода");  
 }  
 }  
 });  
 inputPanel.add(chooseFileButton);  
 inputPanel.add(textInput);  
 break;  
 case 1:  
 *// Task 2: Hero Movement* JTextField heroMovementInput = new JTextField("1 - Пешком, 2 - На лошади, 3 - Лететь");  
 heroMovementInput.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (heroMovementInput.getText().equals("1 - Пешком, 2 - На лошади, 3 - Лететь")) {  
 heroMovementInput.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (heroMovementInput.getText().isEmpty()) {  
 heroMovementInput.setText("1 - Пешком, 2 - На лошади, 3 - Лететь");  
 }  
 }  
 });  
 inputPanel.add(new JLabel("Выберите способ перемещения героя:"));  
 inputPanel.add(heroMovementInput);  
 break;  
 case 2:  
 *// Task 3: Method Invocation* inputPanel.add(new JLabel("Вызов методов с использованием рефлексии"));  
 break;  
 case 3:  
 *// Task 4: Stream API Operations* JComboBox<String> streamMethodSelector = new JComboBox<>(new String[]{  
 "Среднее значение чисел (Введите числа через пробел)",  
 "Строки в верхнем регистре с префиксом '\_new\_' (Введите строки через запятую)",  
 "Квадраты уникальных элементов (Введите числа через пробел)",  
 "Последний элемент коллекции (Введите элементы через запятую)",  
 "Сумма четных чисел (Введите числа через пробел)",  
 "Преобразовать строки в Map (первый символ - ключ) (Введите строки через запятую)"  
 });  
 JTextField streamInput = new JTextField("Введите значения здесь");  
 streamInput.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (streamInput.getText().equals("Введите значения здесь")) {  
 streamInput.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (streamInput.getText().isEmpty()) {  
 streamInput.setText("Введите значения здесь");  
 }  
 }  
 });  
 inputPanel.add(new JLabel("Выберите операцию:"));  
 inputPanel.add(streamMethodSelector);  
 inputPanel.add(streamInput);  
 break;  
 }  
 inputPanel.revalidate();  
 inputPanel.repaint();  
 }  
  
 private class ExecuteButtonListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedTask = taskSelector.getSelectedIndex();  
 Component[] components = inputPanel.getComponents();  
  
 try {  
 switch (selectedTask) {  
 case 0:  
 *// Translator Task* if (components.length > 1 && components[1] instanceof JTextField) {  
 JTextField textInput = (JTextField) components[1];  
 String textToTranslate = textInput.getText();  
 if (dictionaryFile != null) {  
 Translator translator = new Translator();  
 translator.loadDictionary(dictionaryFile);  
 String translatedText = translator.translate(textToTranslate);  
 outputArea.setText("Перевод: " + translatedText);  
 } else {  
 outputArea.setText("Пожалуйста, сначала выберите файл словаря.");  
 }  
 } else {  
 outputArea.setText("Ошибка: Поля ввода не инициализированы должным образом.");  
 }  
 break;  
 case 1:  
 *// Hero Movement Task* if (components.length > 1 && components[1] instanceof JTextField) {  
 JTextField commandInput = (JTextField) components[1];  
 String command = commandInput.getText();  
 Hero hero = new Hero();  
 switch (command) {  
 case "1":  
 hero.setMoving(new ByFoot());  
 hero.move("пешком");  
 outputArea.setText("Герой переместился пешком.");  
 break;  
 case "2":  
 hero.setMoving(new OnHorse());  
 hero.move("на лошади");  
 outputArea.setText("Герой переместился на лошади.");  
 break;  
 case "3":  
 hero.setMoving(new Fly());  
 hero.move("лететь");  
 outputArea.setText("Герой полетел.");  
 break;  
 default:  
 outputArea.setText("Неверная команда.");  
 }  
 } else {  
 outputArea.setText("Ошибка: Поля ввода не инициализированы должным образом.");  
 }  
 break;  
 case 2:  
 *// Method Invocation Task* outputArea.setText("Вызов методов с использованием рефлексии...\n");  
 PrintStream consoleStream = System.*out*;  
 System.*setOut*(new PrintStream(new TextAreaOutputStream(outputArea)));  
 MethodInvoker.*main*(new String[]{});  
 System.*setOut*(consoleStream);  
 break;  
 case 3:  
 *// Stream API Operations* if (components.length > 2 && components[1] instanceof JComboBox && components[2] instanceof JTextField) {  
 JComboBox<String> streamMethodSelector = (JComboBox<String>) components[1];  
 String selectedMethod = (String) streamMethodSelector.getSelectedItem();  
 JTextField listInput = (JTextField) components[2];  
 String inputData = listInput.getText();  
  
 switch (selectedMethod) {  
 case "Среднее значение чисел (Введите числа через пробел)":  
 if (!inputData.isEmpty()) {  
 List<Integer> numbers = Arrays.*stream*(inputData.split(" "))  
 .map(Integer::*parseInt*)  
 .collect(Collectors.*toList*());  
 double average = StreamAPIMethods.*getAverage*(numbers);  
 outputArea.setText("Среднее значение: " + average);  
 }  
 break;  
 case "Строки в верхнем регистре с префиксом '\_new\_' (Введите строки через запятую)":  
 if (!inputData.isEmpty()) {  
 List<String> strings = Arrays.*asList*(inputData.split(","));  
 List<String> transformedStrings = StreamAPIMethods.*transformStrings*(strings);  
 outputArea.setText("Преобразованные строки: " + transformedStrings);  
 }  
 break;  
 case "Квадраты уникальных элементов (Введите числа через пробел)":  
 if (!inputData.isEmpty()) {  
 List<Integer> numbers = Arrays.*stream*(inputData.split(" "))  
 .map(Integer::*parseInt*)  
 .collect(Collectors.*toList*());  
 List<Integer> uniqueSquares = StreamAPIMethods.*getUniqueSquares*(numbers);  
 outputArea.setText("Квадраты уникальных элементов: " + uniqueSquares);  
 }  
 break;  
 case "Последний элемент коллекции (Введите элементы через запятую)":  
 if (!inputData.isEmpty()) {  
 List<String> collection = Arrays.*asList*(inputData.split(","));  
 try {  
 String lastElement = StreamAPIMethods.*getLastElement*(collection);  
 outputArea.setText("Последний элемент: " + lastElement);  
 } catch (NoSuchElementException ex) {  
 outputArea.setText("Ошибка: Коллекция пуста.");  
 }  
 }  
 break;  
 case "Сумма четных чисел (Введите числа через пробел)":  
 if (!inputData.isEmpty()) {  
 int[] numbers = Arrays.*stream*(inputData.split(" "))  
 .mapToInt(Integer::*parseInt*)  
 .toArray();  
 int sum = StreamAPIMethods.*sumOfEvenNumbers*(numbers);  
 outputArea.setText("Сумма четных чисел: " + sum);  
 }  
 break;  
 case "Преобразовать строки в Map (первый символ - ключ) (Введите строки через запятую)":  
 if (!inputData.isEmpty()) {  
 List<String> strings = Arrays.*asList*(inputData.split(","));  
 Map<Character, String> map = StreamAPIMethods.*stringsToMap*(strings);  
 outputArea.setText("Результирующая Map: " + map);  
 }  
 break;  
 default:  
 outputArea.setText("Выберите корректную операцию.");  
 }  
 } else {  
 outputArea.setText("Ошибка: Поля ввода не инициализированы должным образом.");  
 }  
 break;  
 }  
 } catch (Exception ex) {  
 outputArea.setText("Ошибка: " + ex.getMessage());  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(MainGUI::new);  
 }  
}  
  
class TextAreaOutputStream extends java.io.OutputStream {  
 private JTextArea textArea;  
  
 public TextAreaOutputStream(JTextArea textArea) {  
 this.textArea = textArea;  
 }  
  
 @Override  
 public void write(int b) throws java.io.IOException {  
 textArea.append(String.*valueOf*((char) b));  
 textArea.setCaretPosition(textArea.getDocument().getLength());  
 }  
}

package Lab\_5;  
  
import lab\_1.Hero;  
import lab\_1.Fly;  
import lab\_1.ByFoot;  
import lab\_1.OnHorse;  
  
import lab\_2.MethodInvoker;  
  
import lab\_3.Translator;  
  
import lab\_4.StreamAPIMethods;  
  
import java.util.List;  
import java.util.Arrays;  
import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.\*;  
import java.io.File;  
import java.io.PrintStream;  
import java.util.Map;  
import java.util.NoSuchElementException;  
import java.util.stream.Collectors;  
  
public class MainGUI extends JFrame {  
 private JTextArea outputArea;  
 private JComboBox<String> taskSelector;  
 private JPanel inputPanel;  
 private File dictionaryFile; *// Выбранный файл словаря* public MainGUI() {  
 setTitle("Lab Tasks GUI Application");  
 setSize(800, 600);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLayout(new BorderLayout());  
  
 *// Components* taskSelector = new JComboBox<>(new String[]{  
 "Task 1: Translator",  
 "Task 2: Hero Movement",  
 "Task 3: Method Invocation",  
 "Task 4: Stream API Operations"  
 });  
 JButton executeButton = new JButton("Execute");  
 outputArea = new JTextArea(20, 30);  
 outputArea.setEditable(false);  
 JScrollPane scrollPane = new JScrollPane(outputArea);  
 inputPanel = new JPanel(new GridLayout(3, 1));  
  
 *// Add components to frame* add(taskSelector, BorderLayout.*NORTH*);  
 add(inputPanel, BorderLayout.*CENTER*);  
 add(executeButton, BorderLayout.*SOUTH*);  
 add(scrollPane, BorderLayout.*EAST*);  
  
 *// Action listener for the execute button* executeButton.addActionListener(new ExecuteButtonListener());  
  
 *// Action listener for task selection to update input panel* taskSelector.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 updateInputPanel(taskSelector.getSelectedIndex());  
 }  
 });  
  
 *// Initialize the input panel for the first selected task* updateInputPanel(taskSelector.getSelectedIndex());  
  
 setVisible(true);  
 }  
  
 private void updateInputPanel(int selectedTask) {  
 inputPanel.removeAll();  
 switch (selectedTask) {  
 case 0:  
 *// Task 1: Translator* JButton chooseFileButton = new JButton("Выбрать файл словаря");  
 chooseFileButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 JFileChooser fileChooser = new JFileChooser();  
 int result = fileChooser.showOpenDialog(null);  
 if (result == JFileChooser.*APPROVE\_OPTION*) {  
 dictionaryFile = fileChooser.getSelectedFile();  
 outputArea.setText("Выбран файл словаря: " + dictionaryFile.getAbsolutePath());  
 }  
 }  
 });  
 JTextField textInput = new JTextField("Введите текст для перевода");  
 textInput.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (textInput.getText().equals("Введите текст для перевода")) {  
 textInput.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (textInput.getText().isEmpty()) {  
 textInput.setText("Введите текст для перевода");  
 }  
 }  
 });  
 inputPanel.add(chooseFileButton);  
 inputPanel.add(textInput);  
 break;  
 case 1:  
 *// Task 2: Hero Movement* JTextField heroMovementInput = new JTextField("1 - Пешком, 2 - На лошади, 3 - Лететь");  
 heroMovementInput.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (heroMovementInput.getText().equals("1 - Пешком, 2 - На лошади, 3 - Лететь")) {  
 heroMovementInput.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (heroMovementInput.getText().isEmpty()) {  
 heroMovementInput.setText("1 - Пешком, 2 - На лошади, 3 - Лететь");  
 }  
 }  
 });  
 inputPanel.add(new JLabel("Выберите способ перемещения героя:"));  
 inputPanel.add(heroMovementInput);  
 break;  
 case 2:  
 *// Task 3: Method Invocation* inputPanel.add(new JLabel("Вызов методов с использованием рефлексии"));  
 break;  
 case 3:  
 *// Task 4: Stream API Operations* JComboBox<String> streamMethodSelector = new JComboBox<>(new String[]{  
 "Среднее значение чисел (Введите числа через пробел)",  
 "Строки в верхнем регистре с префиксом '\_new\_' (Введите строки через запятую)",  
 "Квадраты уникальных элементов (Введите числа через пробел)",  
 "Последний элемент коллекции (Введите элементы через запятую)",  
 "Сумма четных чисел (Введите числа через пробел)",  
 "Преобразовать строки в Map (первый символ - ключ) (Введите строки через запятую)"  
 });  
 JTextField streamInput = new JTextField("Введите значения здесь");  
 streamInput.addFocusListener(new FocusAdapter() {  
 @Override  
 public void focusGained(FocusEvent e) {  
 if (streamInput.getText().equals("Введите значения здесь")) {  
 streamInput.setText("");  
 }  
 }  
  
 @Override  
 public void focusLost(FocusEvent e) {  
 if (streamInput.getText().isEmpty()) {  
 streamInput.setText("Введите значения здесь");  
 }  
 }  
 });  
 inputPanel.add(new JLabel("Выберите операцию:"));  
 inputPanel.add(streamMethodSelector);  
 inputPanel.add(streamInput);  
 break;  
 }  
 inputPanel.revalidate();  
 inputPanel.repaint();  
 }  
  
 private class ExecuteButtonListener implements ActionListener {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 int selectedTask = taskSelector.getSelectedIndex();  
 Component[] components = inputPanel.getComponents();  
  
 try {  
 switch (selectedTask) {  
 case 0:  
 *// Translator Task* if (components.length > 1 && components[1] instanceof JTextField) {  
 JTextField textInput = (JTextField) components[1];  
 String textToTranslate = textInput.getText();  
 if (dictionaryFile != null) {  
 Translator translator = new Translator();  
 translator.loadDictionary(dictionaryFile);  
 String translatedText = translator.translate(textToTranslate);  
 outputArea.setText("Перевод: " + translatedText);  
 } else {  
 outputArea.setText("Пожалуйста, сначала выберите файл словаря.");  
 }  
 } else {  
 outputArea.setText("Ошибка: Поля ввода не инициализированы должным образом.");  
 }  
 break;  
 case 1:  
 *// Hero Movement Task* if (components.length > 1 && components[1] instanceof JTextField) {  
 JTextField commandInput = (JTextField) components[1];  
 String command = commandInput.getText();  
 Hero hero = new Hero();  
 switch (command) {  
 case "1":  
 hero.setMoving(new ByFoot());  
 hero.move("пешком");  
 outputArea.setText("Герой переместился пешком.");  
 break;  
 case "2":  
 hero.setMoving(new OnHorse());  
 hero.move("на лошади");  
 outputArea.setText("Герой переместился на лошади.");  
 break;  
 case "3":  
 hero.setMoving(new Fly());  
 hero.move("лететь");  
 outputArea.setText("Герой полетел.");  
 break;  
 default:  
 outputArea.setText("Неверная команда.");  
 }  
 } else {  
 outputArea.setText("Ошибка: Поля ввода не инициализированы должным образом.");  
 }  
 break;  
 case 2:  
 *// Method Invocation Task* outputArea.setText("Вызов методов с использованием рефлексии...\n");  
 PrintStream consoleStream = System.*out*;  
 System.*setOut*(new PrintStream(new TextAreaOutputStream(outputArea)));  
 MethodInvoker.*main*(new String[]{});  
 System.*setOut*(consoleStream);  
 break;  
 case 3:  
 *// Stream API Operations* if (components.length > 2 && components[1] instanceof JComboBox && components[2] instanceof JTextField) {  
 JComboBox<String> streamMethodSelector = (JComboBox<String>) components[1];  
 String selectedMethod = (String) streamMethodSelector.getSelectedItem();  
 JTextField listInput = (JTextField) components[2];  
 String inputData = listInput.getText();  
  
 switch (selectedMethod) {  
 case "Среднее значение чисел (Введите числа через пробел)":  
 if (!inputData.isEmpty()) {  
 List<Integer> numbers = Arrays.*stream*(inputData.split(" "))  
 .map(Integer::*parseInt*)  
 .collect(Collectors.*toList*());  
 double average = StreamAPIMethods.*getAverage*(numbers);  
 outputArea.setText("Среднее значение: " + average);  
 }  
 break;  
 case "Строки в верхнем регистре с префиксом '\_new\_' (Введите строки через запятую)":  
 if (!inputData.isEmpty()) {  
 List<String> strings = Arrays.*asList*(inputData.split(","));  
 List<String> transformedStrings = StreamAPIMethods.*transformStrings*(strings);  
 outputArea.setText("Преобразованные строки: " + transformedStrings);  
 }  
 break;  
 case "Квадраты уникальных элементов (Введите числа через пробел)":  
 if (!inputData.isEmpty()) {  
 List<Integer> numbers = Arrays.*stream*(inputData.split(" "))  
 .map(Integer::*parseInt*)  
 .collect(Collectors.*toList*());  
 List<Integer> uniqueSquares = StreamAPIMethods.*getUniqueSquares*(numbers);  
 outputArea.setText("Квадраты уникальных элементов: " + uniqueSquares);  
 }  
 break;  
 case "Последний элемент коллекции (Введите элементы через запятую)":  
 if (!inputData.isEmpty()) {  
 List<String> collection = Arrays.*asList*(inputData.split(","));  
 try {  
 String lastElement = StreamAPIMethods.*getLastElement*(collection);  
 outputArea.setText("Последний элемент: " + lastElement);  
 } catch (NoSuchElementException ex) {  
 outputArea.setText("Ошибка: Коллекция пуста.");  
 }  
 }  
 break;  
 case "Сумма четных чисел (Введите числа через пробел)":  
 if (!inputData.isEmpty()) {  
 int[] numbers = Arrays.*stream*(inputData.split(" "))  
 .mapToInt(Integer::*parseInt*)  
 .toArray();  
 int sum = StreamAPIMethods.*sumOfEvenNumbers*(numbers);  
 outputArea.setText("Сумма четных чисел: " + sum);  
 }  
 break;  
 case "Преобразовать строки в Map (первый символ - ключ) (Введите строки через запятую)":  
 if (!inputData.isEmpty()) {  
 List<String> strings = Arrays.*asList*(inputData.split(","));  
 Map<Character, String> map = StreamAPIMethods.*stringsToMap*(strings);  
 outputArea.setText("Результирующая Map: " + map);  
 }  
 break;  
 default:  
 outputArea.setText("Выберите корректную операцию.");  
 }  
 } else {  
 outputArea.setText("Ошибка: Поля ввода не инициализированы должным образом.");  
 }  
 break;  
 }  
 } catch (Exception ex) {  
 outputArea.setText("Ошибка: " + ex.getMessage());  
 }  
 }  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(MainGUI::new);  
 }  
}  
  
class TextAreaOutputStream extends java.io.OutputStream {  
 private JTextArea textArea;  
  
 public TextAreaOutputStream(JTextArea textArea) {  
 this.textArea = textArea;  
 }  
  
 @Override  
 public void write(int b) throws java.io.IOException {  
 textArea.append(String.*valueOf*((char) b));  
 textArea.setCaretPosition(textArea.getDocument().getLength());  
 }  
}