Министерство образования Калининградской области

государственное бюджетное учреждение Калининградской области

профессиональная образовательная организация

«Колледж информационных технологий и строительства»

(ГБУ КО ПОО «КИТиС»)

**Отчет по учебной практике**

УП.01 Разработка модулей программного обеспечения для компьютерных систем

по ПМ.01 Разработка модулей программного обеспечения для компьютерных систем

Специальность 09.02.07 «Информационные системы и программирование»

Сроки прохождения практики:

с «20» ноября 2023 г. по «09» декабря 2023 г.

Место практики ГБУ КО ПОО «КИТиС»

|  |  |
| --- | --- |
| Выполнил: | студент 4 курса,  группы ИСп 20-1  Лаптева Ирина Константиновна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Проверила: | Большакова-Стрекалова Анна Викторовна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Калининград, 2023

Содержание

[Введение 4](#_Toc152821503)

[1 Решение задач 5](#_Toc152821504)

[1.1 Задача 1 5](#_Toc152821505)

[1.2 Задача 2 12](#_Toc152821506)

[1.3 Задача 3 18](#_Toc152821507)

[1.4 Задача 4 22](#_Toc152821508)

[1.5 Задача 5 26](#_Toc152821509)

[1.6 Задача 6 32](#_Toc152821510)

[1.7 Задача 7 38](#_Toc152821511)

[1.8 Задача 8 43](#_Toc152821512)

[1.9 Задача 9 50](#_Toc152821513)

[2 Разработка АИС «Автовокзал» 63](#_Toc152821514)

[2.1 ER-диаграмма, диаграмма классов. 63](#_Toc152821515)

[2.2 Техническое задание 64](#_Toc152821516)

[2.3 Краткая спецификация системы 66](#_Toc152821517)

[2.4 Функциональная схема 69](#_Toc152821518)

[2.5 Интерфейс 70](#_Toc152821519)

[2.6 Тестирование 73](#_Toc152821520)

[2.7 Оценка программного продукта 74](#_Toc152821521)

[3 Разработка мобильного приложения 77](#_Toc152821522)

[3.1 Дизайн мобильного приложения 79](#_Toc152821523)

[4 Разработка сайта 83](#_Toc152821524)

[4.1 Выбор инструментов 83](#_Toc152821525)

[4.2 Интерфейс сайта 84](#_Toc152821526)

[5 Руководство пользователя 87](#_Toc152821527)

[Заключение 89](#_Toc152821528)

[Список использованных источников 90](#_Toc152821529)

[Приложение 91](#_Toc152821530)

# Введение

Цель данной учебной практики заключается в углублении и закреплении имеющихся знаний, а также в эффективном их применении на практике для разработки программных модулей.

Задачи учебной практики:

* изучение ключевых этапов жизненного цикла программного обеспечения;
* разработка эффективных алгоритмов;
* анализ требований при создании модулей;
* тестирование и отладка.

Учебная практика проходила в Колледже информационных технологий и строительства.

# 1 Решение задач

## 1.1 Задача 1

Описание задания:

Вариант 11, по учебнику – 7. Повернуть матрицу на 90 (180, 270) градусов против часовой стрелки.

Таблица идентификаторов:

Таблица 1 – таблица идентификаторов к задаче 1

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| n | int | Целочисленная переменная, хранящая размерность матрицы. Входная. |
| matrix | int[][] | Двумерный массив целых чисел, представляющий матрицу.  Выходная. |
| rotated90Matrix | int[][] | Двумерный массив целых чисел, представляющий матрицу после поворота на 90 градусов против часовой стрелки. Выходная. |
| Rotated180Matrix | int[][] | Двумерный массив целых чисел, представляющий матрицу после поворота на 180 градусов против часовой стрелки. Выходная. |
| Rotated270Matrix | int[][] | Двумерный массив целых чисел, представляющий матрицу после поворота на 270 градусов против часовой стрелки. Выходная. |

Блок-схема алгоритма:

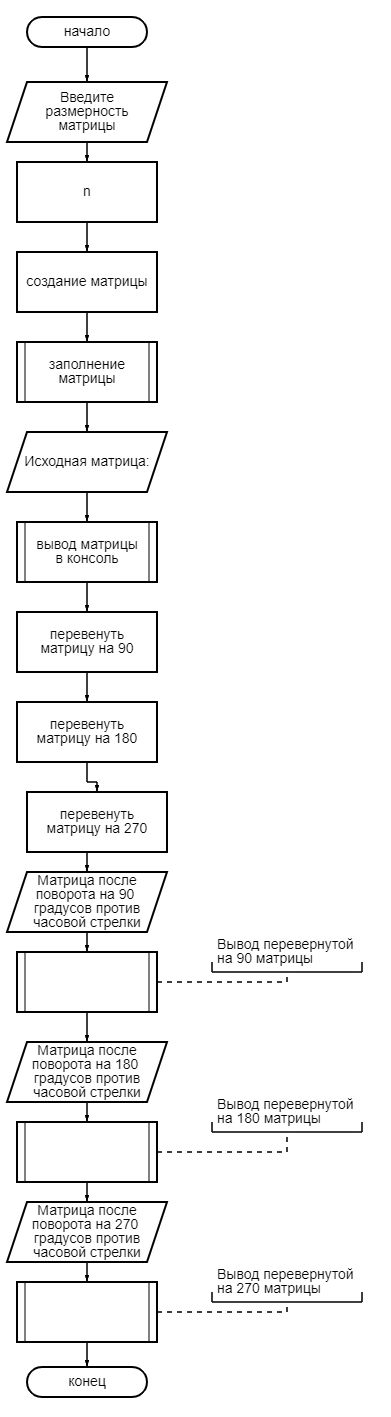


Рисунок 1 – блок схема к задаче 1

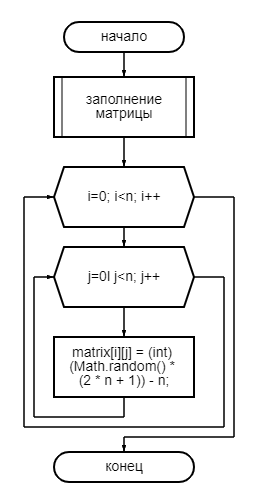


Рисунок 2 – блок схема подпрограммы для заполнения матрицы

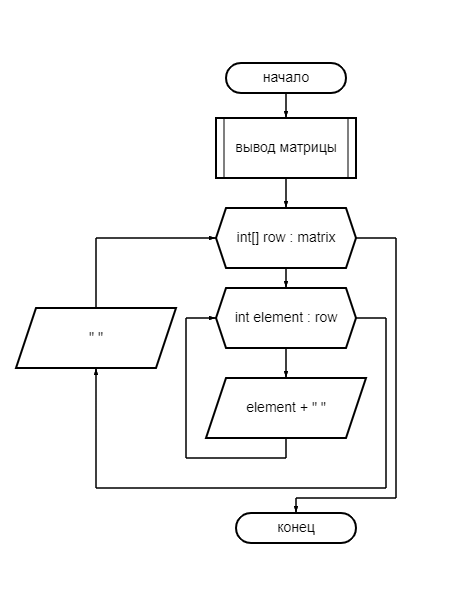


Рисунок 3 – блок схема подпрограммы для вывода матрицы

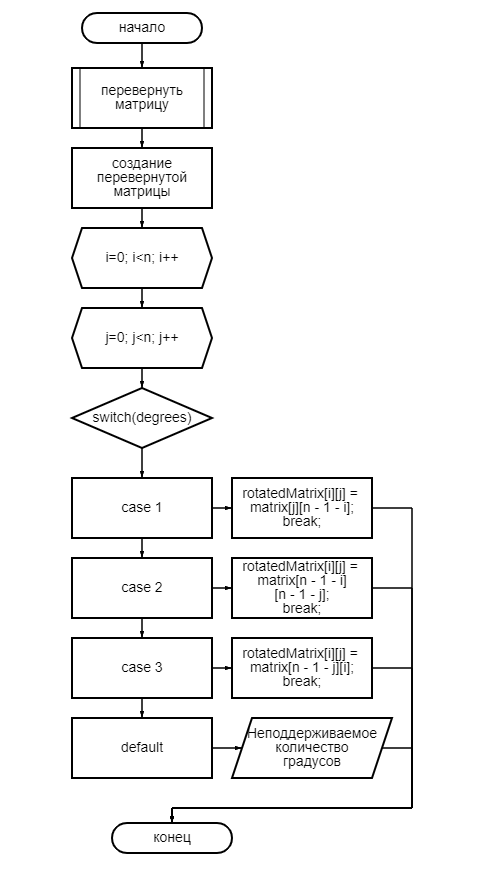


Рисунок 4 – блок схема подпрограммы переворота матрицы

Код программы:

public class zad1 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Введите размерность матрицы (n): ");

int n = scanner.nextInt();

int[][] matrix = new int[n][n];

fillMatrix(matrix, n);

System.out.println("\nИсходная матрица:");

printMatrix(matrix);

int[][] rotated90Matrix = rotateMatrix(matrix, n, 1);

int[][] rotated180Matrix = rotateMatrix(matrix, n, 2);

int[][] rotated270Matrix = rotateMatrix(matrix, n, 3);

System.out.println("\nМатрица после поворота на 90 градусов против часовой стрелки:");

printMatrix(rotated90Matrix);

System.out.println("\nМатрица после поворота на 180 градусов против часовой стрелки:");

printMatrix(rotated180Matrix);

System.out.println("\nМатрица после поворота на 270 градусов против часовой стрелки:");

printMatrix(rotated270Matrix);

}

// Для заполнения матрицы случайными числами в интервале от -n до n

private static void fillMatrix(int[][] matrix, int n) {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

matrix[i][j] = (int) (Math.random() \* (2 \* n + 1)) - n;

}

}

}

// Для вывода матрицы в консоль

private static void printMatrix(int[][] matrix) {

for (int[] row : matrix) {

for (int element : row) {

System.out.print(element + " ");

}

System.out.println();

}

}

// Для поворота матрицы на указанное количество градусов против часовой стрелки

private static int[][] rotateMatrix(int[][] matrix, int n, int degrees) {

int[][] rotatedMatrix = new int[n][n];

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

switch (degrees) {

case 1: // 90 градусов

rotatedMatrix[i][j] = matrix[j][n - 1 - i];

break;

case 2: // 180 градусов

rotatedMatrix[i][j] = matrix[n - 1 - i][n - 1 - j];

break;

case 3: // 270 градусов

rotatedMatrix[i][j] = matrix[n - 1 - j][i];

break;

default:

throw new IllegalArgumentException("Неподдерживаемое количество градусов");

}

}

}

return rotatedMatrix;

}

}

Результат выполнения программы:

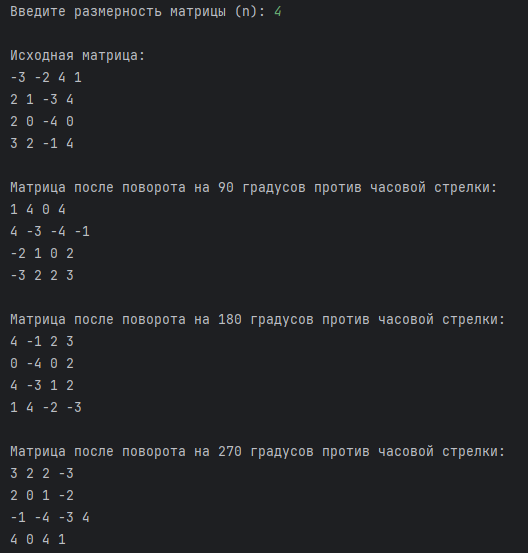


Рисунок 5 – результат первой задачи

## 1.2 Задача 2

Описание задания:

Вариант 11. Bus: Фамилия и инициалы водителя, Номер автобуса, Номер маршрута, Марка, Год начала эксплуатации, Пробег.

Создать массив объектов. Вывести:

a) список автобусов для заданного номера маршрута;

b) список автобусов, которые эксплуатируются больше заданного срока;

c) список автобусов, пробег у которых больше заданного расстояния

Таблица идентификаторов:

Таблица 2 – таблица идентификаторов к задаче 2

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| driverName | String | Строковая переменная, содержащая имя водителя автобуса. Выходная. |
| busNumber | String | Строковая переменная, содержащая номер автобуса. Выходная. |
| routeNumber | int | Целочисленная переменная, хранящая номер маршрута автобуса. Выходная. |
| brand | String | Строковая переменная, содержащая марку автобуса. Выходная. |
| yearOfOperation | int | Целочисленная переменная, хранящая год начала эксплуатации автобуса. Выходная. |
| mileage | double | Переменная с плавающей точкой, представляющая пробег автобуса в километрах. Выходная. |
| buses | Bus[] | Массив объектов класса Bus, представляющий собой группу автобусов. Вспомогательная |
| targetRouteNumber | int | Целочисленная переменная, содержащая номер маршрута для поиска. Выходная. |
| targetYearsOfOperation | int | Целочисленная переменная, представляющая срок эксплуатации для поиска. Вспомогательная |
| targetMileage | double | Переменная с плавающей точкой, представляющая пробег для поиска. Вспомогательная. |
| currentYear | int | Целочисленная переменная, представляющая текущий год. Вспомогательная |

Блок-схема алгоритма:

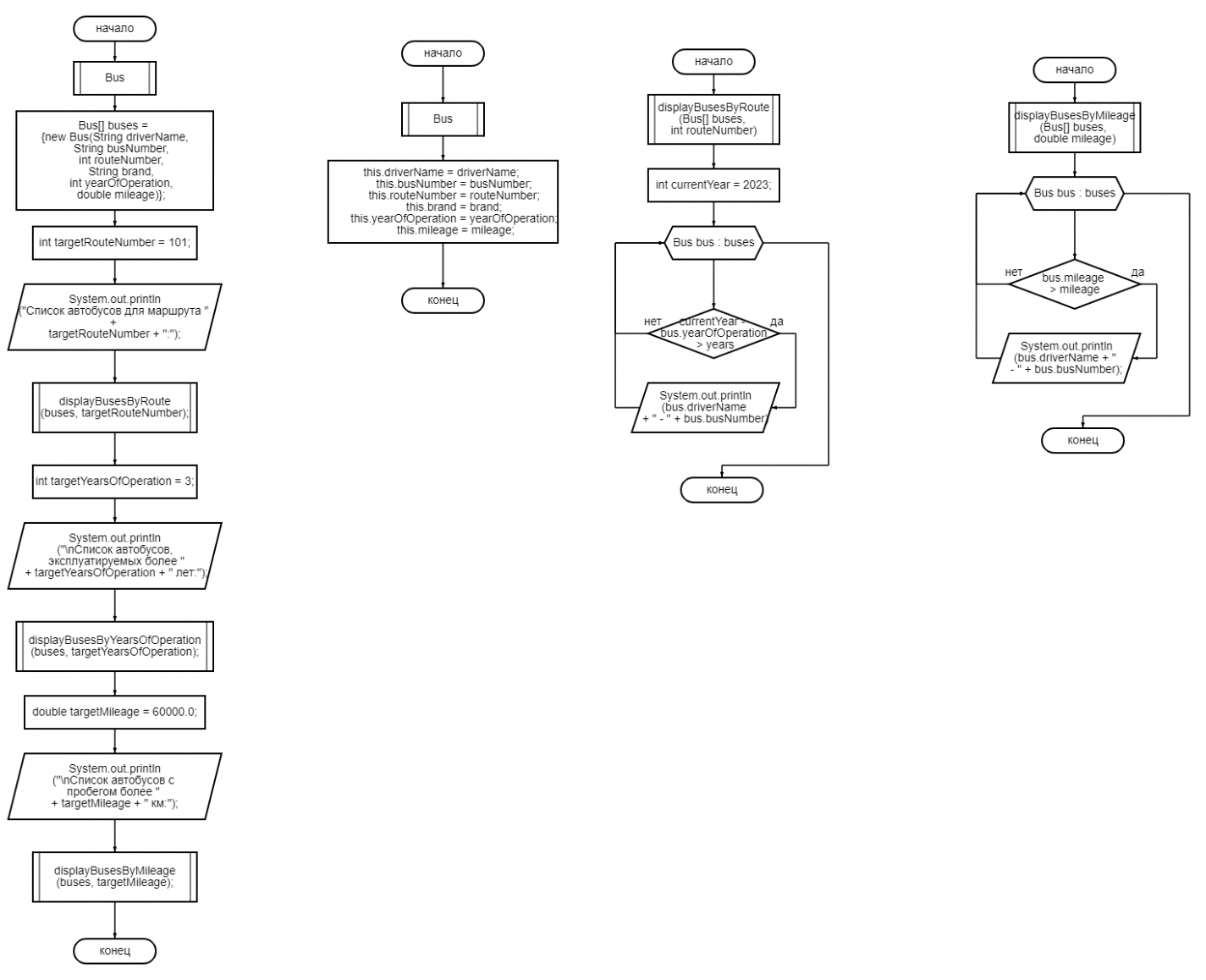


Рисунок 6 – блок схема к задаче 2

Код программы:

class Bus {

String driverName;

String busNumber;

int routeNumber;

String brand;

int yearOfOperation;

double mileage;

public Bus(String driverName, String busNumber, int routeNumber, String brand, int yearOfOperation, double mileage) {

this.driverName = driverName;

this.busNumber = busNumber;

this.routeNumber = routeNumber;

this.brand = brand;

this.yearOfOperation = yearOfOperation;

this.mileage = mileage;

}

}

public class zad2 {

public static void main(String[] args) {

//массив объектов Bus

Bus[] buses = {

new Bus("Паванов С.П.", "A123BC", 101, "Mercedes", 2018, 50000.0),

new Bus("Петров П.П.", "B456CD", 102, "Volvo", 2019, 60000.0),

new Bus("Сидоров С.С.", "C789DE", 101, "MAN", 2017, 70000.0),

};

//список автобусов для заданного номера маршрута

int targetRouteNumber = 101;

System.out.println("Список автобусов для маршрута " + targetRouteNumber + ":");

displayBusesByRoute(buses, targetRouteNumber);

//список автобусов, которые эксплуатируются больше заданного срока

int targetYearsOfOperation = 3; // например, более 3 лет

System.out.println("\nСписок автобусов, эксплуатируемых более " + targetYearsOfOperation + " лет:");

displayBusesByYearsOfOperation(buses, targetYearsOfOperation);

//список автобусов с пробегом больше заданного расстояния

double targetMileage = 60000.0; // например, более 60000 км

System.out.println("\nСписок автобусов с пробегом более " + targetMileage + " км:");

displayBusesByMileage(buses, targetMileage);

}

//для вывода списка автобусов для заданного номера маршрута

private static void displayBusesByRoute(Bus[] buses, int routeNumber) {

for (Bus bus : buses) {

if (bus.routeNumber == routeNumber) {

System.out.println(bus.driverName + " - " + bus.busNumber);

}

}

}

//для вывода списка автобусов, которые эксплуатируются больше заданного срока

private static void displayBusesByYearsOfOperation(Bus[] buses, int years) {

int currentYear = 2023; // Текущий год (можно заменить на получение текущего года из системы)

for (Bus bus : buses) {

if (currentYear - bus.yearOfOperation > years) {

System.out.println(bus.driverName + " - " + bus.busNumber);

}

}

}

//для вывода списка автобусов с пробегом больше заданного расстояния

private static void displayBusesByMileage(Bus[] buses, double mileage) {

for (Bus bus : buses) {

if (bus.mileage > mileage) {

System.out.println(bus.driverName + " - " + bus.busNumber);

}

}

}

}

Результат выполнения программы:

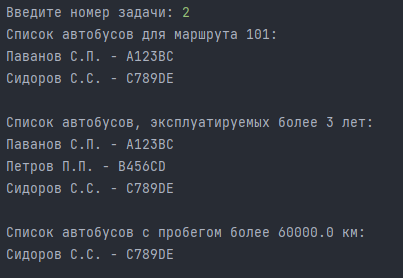


Рисунок 7 – результат второй задачи

## 1.3 Задача 3

Описание задания:

Вариант 20. Создать объект класса Наседка, используя классы Птица, Кукушка. Методы: летать, петь, нести яйца, высиживать птенцов

Таблица идентификаторов:

Таблица 3 – таблица идентификаторов к задаче 3

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Bird1 | Класс | Базовый класс, представляющий птицу. |
| tofly() | Метод | Метод класса Bird1, выводящий сообщение о том, что птица летит. |
| tosing() | Метод | Метод класса Bird1, выводящий сообщение о том, что птица поет. |
| Сuckoo1 | Класс | Класс, наследующий от Bird1, представляющий кукушку. |
| laysegg() | Метод | Метод класса Сuckoo1, выводящий сообщение о том, что кукушка несет яйца. |
| hatchchicks() | Метод | Метод класса Nasedka1, выводящий сообщение о том, что кукушка высиживает птенцов. |
| Nasedka1 | Класс | Класс, наследующий от Сuckoo1, представляющий наседку. |
| zad3 | Класс | Главный класс программы. |
| main() | Метод | Главный метод программы, выполняющий основную логику. |
| nasedka1 | Nasedka1 | Объект класса Nasedka1, представляющий экземпляр наседки. |

Блок-схема алгоритма:

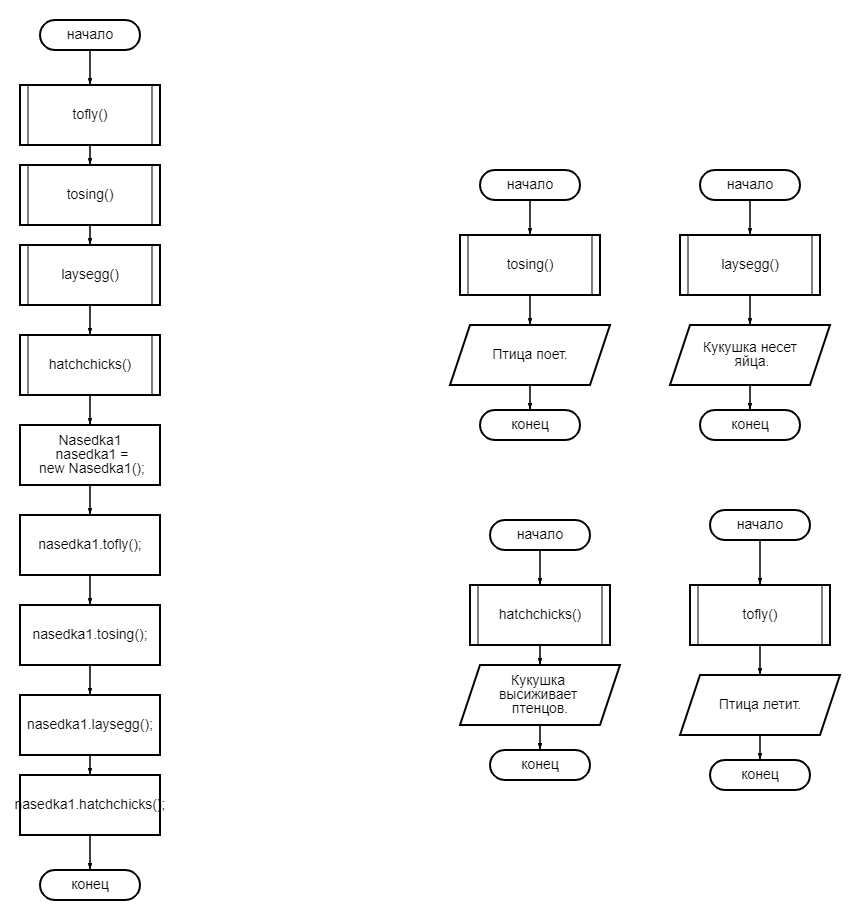


Рисунок 8 – блок схема к задаче 3

Код программы:

class Bird1 {

void tofly() {

System.out.println("Птица летит.");

}

void tosing() {

System.out.println("Птица поет.");

}

}

class Сuckoo1 extends Bird1 {

void laysegg() {

System.out.println("Кукушка несет яйца.");

}

}

class Nasedka1 extends Сuckoo1 {

void hatchchicks() {

System.out.println("Наседка высиживает птенцов.");

}

}

public class zad3 {

public static void main(String[] args) {

Nasedka1 nasedka1 = new Nasedka1();

nasedka1.tofly();

nasedka1.tosing();

nasedka1.laysegg();

nasedka1.hatchchicks();

}

}

Результат выполнения программы:

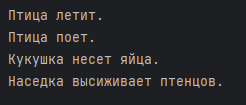


Рисунок 9 – результат третьей задачи

## 1.4 Задача 4

Описание задания:

Вариант 20. Создать класс Художественная Выставка с внутренним классом, с помощью объектов которого можно хранить информацию о картинах, авторах и времени проведения выставок.

Таблица 4 – таблица идентификаторов к задаче 4

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| exhibitionName | String | Строковая переменная, содержащая название выставки. Выходная. |
| artworks | List<Artwork> | Список объектов класса Artwork, представляющий собой коллекцию произведений искусства на выставке. Выходной. |
| title | String | Строковая переменная, содержащая название произведения искусства. Выходная. |
| author | String | Строковая переменная, содержащая имя автора произведения искусства. Выходная. |
| time | String | Строковая переменная, содержащая время представления произведения искусства. Выходная. |
| artwork | Artwork | Объект класса Artwork, представляющий отдельное произведение искусства. |
| exhibition | ArtExhibition | Объект класса ArtExhibition, представляющий выставку и содержащий информацию о произведениях искусства. |

Блок-схема алгоритма:

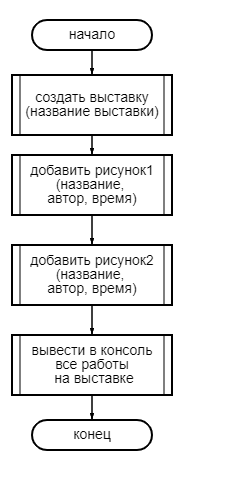


Рисунок 10 – блок схема к задаче 4

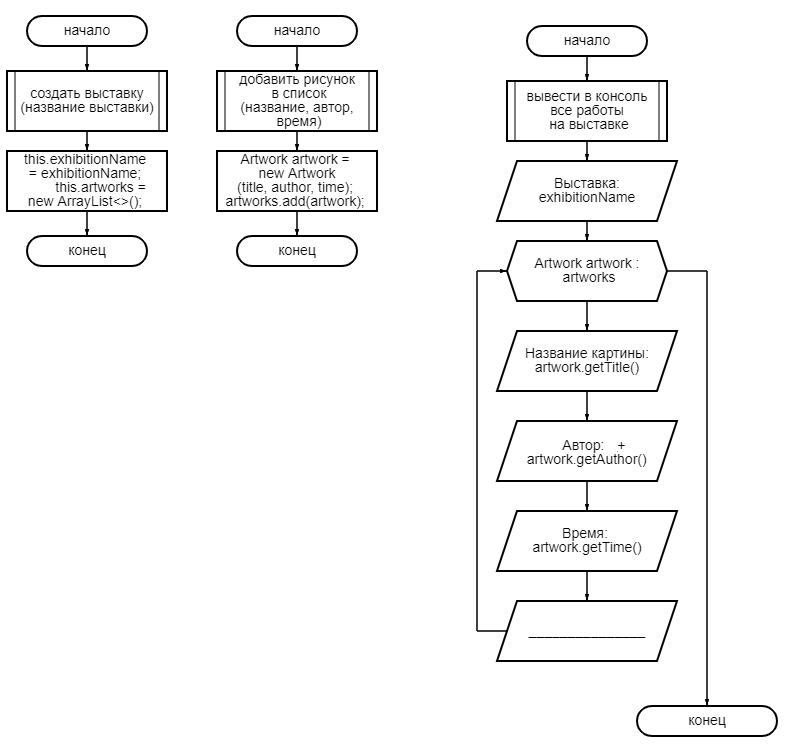


Рисунок 11 – блок схемы подпрограмм

Код программы:

class ArtExhibition {

private String exhibitionName;

private List<Artwork> artworks;

public ArtExhibition(String exhibitionName) {

this.exhibitionName = exhibitionName;

this.artworks = new ArrayList<>();

}

public void addArtwork(String title, String author, String time) {

Artwork artwork = new Artwork(title, author, time);

artworks.add(artwork);

}

public void displayArtworks() {

System.out.println("Выставка: " + exhibitionName);

for (Artwork artwork : artworks) {

System.out.println("Название картины: " + artwork.getTitle());

System.out.println("Автор: " + artwork.getAuthor());

System.out.println("Время: " + artwork.getTime());

System.out.println("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_");

}

}

//для хранения информации о картинах

private class Artwork {

private String title;

private String author;

private String time;

public Artwork(String title, String author, String time) {

this.title = title;

this.author = author;

this.time = time;

}

public String getTitle() {

return title;

}

public String getAuthor() {

return author;

}

public String getTime() {

return time;

}

}

}

public class zad4 {

public static void main(String[] args) {

ArtExhibition exhibition = new ArtExhibition("Выставка рисунков");

exhibition.addArtwork("Пятиминутная разминка", "Хайнц Фуфелшмертц", "10:00 AM");

exhibition.addArtwork("Великая Королева", "Утконос Пэрри", "02:00 PM");

exhibition.displayArtworks();

}

}

Результат выполнения программы:

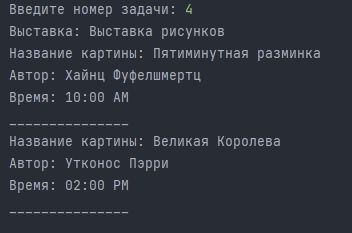


Рисунок 12 – результат четвертой задачи

## 1.5 Задача 5

Описание задания:

Вариант – 20. Найти, каких букв, гласных или согласных, больше в каждом предложении текста.

Блок схема алгоритма:

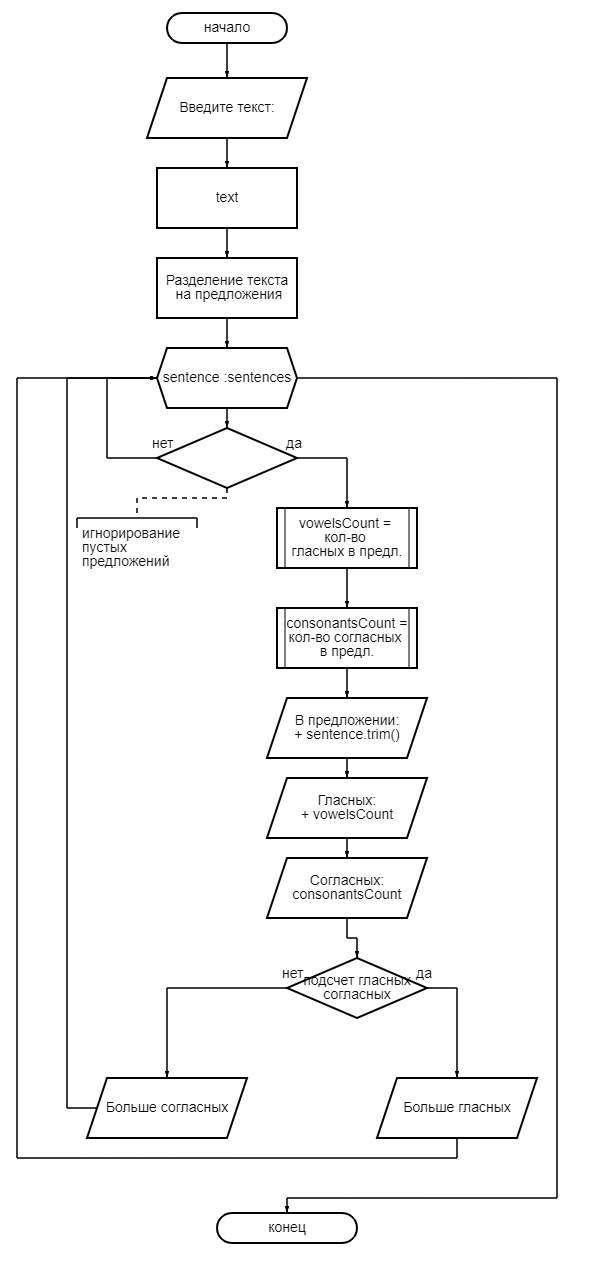


Рисунок 13 – блок схема к задаче 5

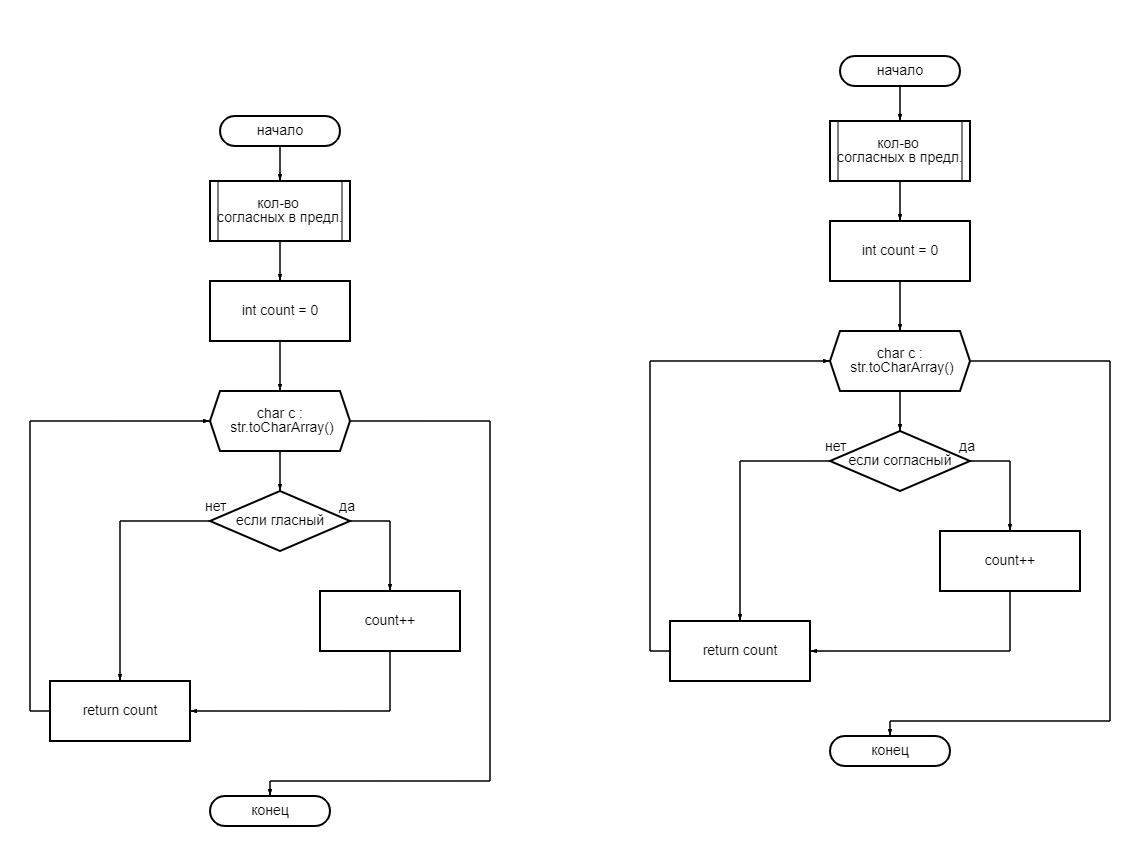


Рисунок 14 – блок схемы к подпрограммам задачи 5

Таблица идентификаторов:

Таблица 5 – таблица идентификаторов к задаче 5

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| text | String | Строковая переменная, содержащая введенный текст. Входная. |
| sentences | String[] | Массив строк, содержащий предложения из введенного текста. |
| sentence | String | Строковая переменная, представляющая отдельное предложение из массива. Выходная. |
| vowelsCount | int | Целочисленная переменная, хранящая количество гласных в предложении. Выходная. |
| consonantsCount | int | Целочисленная переменная, хранящая количество согласных в предложении. Выходная. |
| c | char | Символьная переменная, используемая для итерации по символам предложения. Вспомогательная |
| str | String | Строковая переменная, используемая в методах countVowels и countConsonants. Вспомогательная |
| isVowel | boolean | Метод, возвращающий true, если символ является гласной буквой, иначе false. |
| isConsonant | boolean | Метод, возвращающий true, если символ является согласной буквой, иначе false. |

Код программы:

public class zad5 {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

System.out.print("Введите текст: ");

String text = scanner.nextLine();

// Разделение текста на предложения

String[] sentences = text.split("[.!?]");

for (String sentence : sentences) {

//игнорирование пустых предложений

if (!sentence.trim().isEmpty()) {

int vowelsCount = countVowels(sentence);

int consonantsCount = countConsonants(sentence);

System.out.println("В предложении: \"" + sentence.trim() + "\"");

System.out.println("Гласных: " + vowelsCount);

System.out.println("Согласных: " + consonantsCount);

System.out.println("Больше " + (vowelsCount > consonantsCount ? "гласных" : "согласных"));

System.out.println();

}

}

}

// Метод для подсчета количества гласных и согласных букв в строке

private static int countVowels(String str) {

int count = 0;

for (char c : str.toCharArray()) {

if (isVowel(c)) {

count++;

}

}

return count;

}

private static int countConsonants(String str) {

int count = 0;

for (char c : str.toCharArray()) {

if (isConsonant(c)) {

count++;

}

}

return count;

}

private static boolean isVowel(char c) {

return "AEIOUaeiouАЕОУЫЭЮЯаеиоуыэюя".indexOf(c) != -1;

}

private static boolean isConsonant(char c) {

return "BCDFGHJKLMNPQRSTVWXYZbcdfghjklmnpqrstvwxyzБВГДЖЗЙКЛМНПРСТФХЦЧШЩбвгджзйклмнпрстфхцчшщ".indexOf(c) != -1;

}

}

Результат выполнения программы:

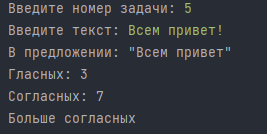


Рисунок 15 – результат пятой задачи, больше согласных

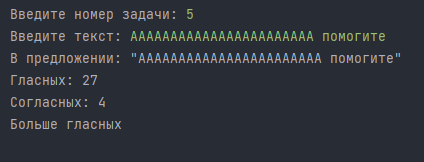


Рисунок 16 – результат пятой задачи, больше гласных

## 1.6 Задача 6

Описание задания:

Выполнить задания на основе задачи № 3, контролируя состояние потоков ввода/вывода. При возникновении ошибок, связанных с корректностью выполнения математических операций, генерировать и обрабатывать исключительные ситуации. Предусмотреть обработку исключений, возникающих при нехватке памяти, отсутствии требуемой записи (объекта) в файле, недопустимом значении поля и т. д.

Выполнить задания из задачи № 3, реализуя собственные обработчики исключений и исключения ввода/вывода.

Таблица идентификаторов:

Таблица 6 – таблица идентификаторов к задаче 6

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| Bird | Класс | Класс, представляющий общие характеристики птиц. |
| tofly | Метод | Метод класса Bird, представляющий действие "лететь". |
| tosing | Метод | Метод класса Bird, представляющий действие "петь". |
| Сuckoo | Класс | Подкласс класса Bird, представляющий кукушку. |
| laysegg | Метод | Метод класса Сuckoo, представляющий действие "нести яйца" и выбрасывающий IOException. |
| hatchchicks | Метод | Метод класса Nasedka, представляющий действие "высиживать птенцов". |
| Nasedka | Класс | Подкласс класса Сuckoo, не добавляет новых методов или полей. |
| zad6 | Класс | Основной класс с методом main для запуска программы. |
| nasedka1 | Nasedka | Объект класса Nasedka, представляющий экземпляр птицы. |
| try-catch блок | Обработка исключений | Блок кода, где происходит вызов методов объекта nasedka1, и обрабатываются возможные исключения IOException и Exception. |

Блок-схема алгоритма:

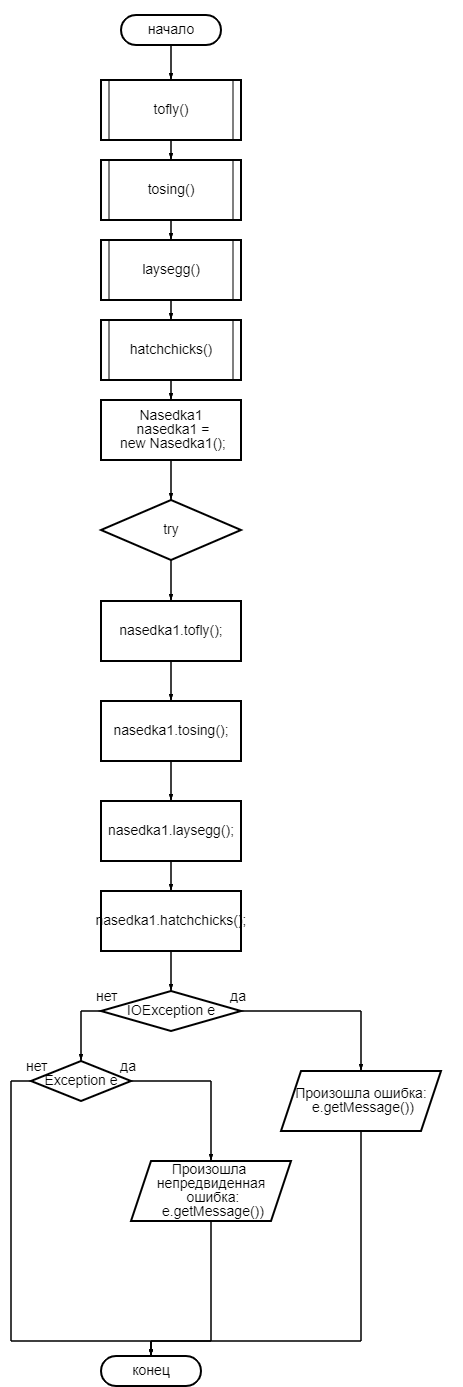


Рисунок 17 – блок схема к задаче 6

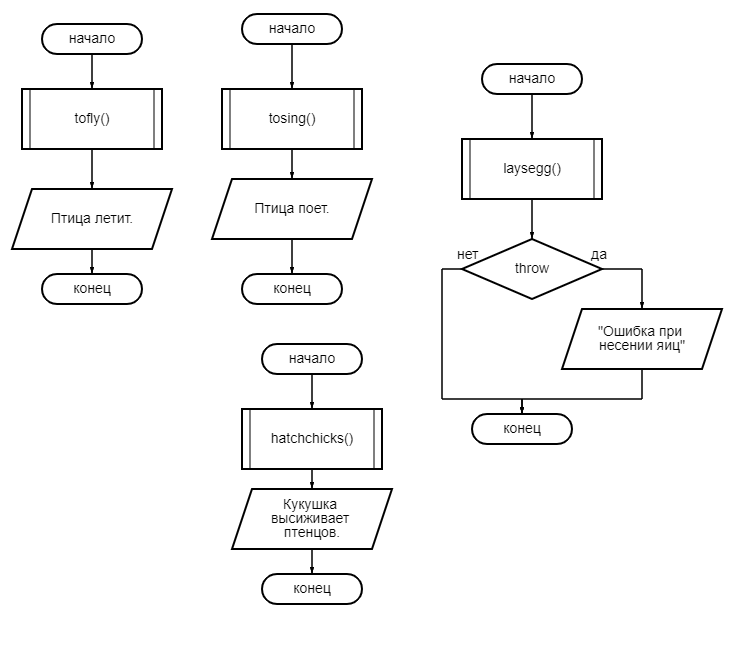


Рисунок 18 – блок схемы к подпрограммам задачи 6

Код программы:

class Bird {

void tofly() {

System.out.println("Птица летит.");

}

void tosing() {

System.out.println("Птица поет.");

}

}

class Сuckoo extends Bird {

void laysegg() throws IOException {

throw new IOException("Ошибка при несении яиц");

}

}

class Nasedka extends Сuckoo {

void hatchchicks() {

System.out.println("Наседка высиживает птенцов.");

}

}

public class zad6 {

public static void main(String[] args) {

Nasedka nasedka1 = new Nasedka();

try {

nasedka1.tofly();

nasedka1.tosing();

nasedka1.laysegg();

nasedka1.hatchchicks();

} catch (IOException e) {

System.out.println("Произошла ошибка: " + e.getMessage());

} catch (Exception e) {

System.out.println("Произошла непредвиденная ошибка: " + e.getMessage());

}

}

}

Результат выполнения программы:

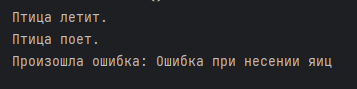


Рисунок 19 – результат шестой задачи

## 1.7 Задача 7

Описание задания:

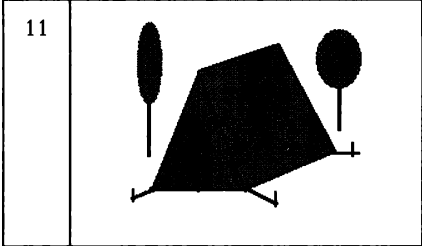


Рисунок 20 – задача семь, вариант 11

Таблица идентификаторов:

Таблица 7 – таблица идентификаторов к задаче семь

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| width, height | int | Целочисленные переменные, представляющие ширину и высоту окна. |
| xPointsTriangle, yPointsTriangle | int[] | Массивы целочисленных переменных, представляющих координаты вершин треугольника. |
| xPointsSquare, yPointsSquare | int[] | Массивы целочисленных переменных, представляющих координаты вершин квадрата. |
| squareSize | int | Целочисленная переменная, представляющая размер стены палатки. |
| leftStickX, rightStickX | int | Целочисленные переменные, представляющие координаты верхних концов палок слева и справа. |
| g2d | Graphics2D | Объект Graphics2D, расширяющий функциональность рисования. |
| triangleHeight | int | Целочисленная переменная, представляющая высоту треугольника. |
| horizontalStickY, horizontalStickLength | int | Целочисленные переменные, представляющие высоту и длину горизонтальной палки. |
| smallStickHeight | int | Целочисленная переменная, представляющая высоту маленьких палок. |
| verticalStickX, verticalStickHeight | int | Целочисленные переменные, представляющие X-координату верхнего конца и высоту вертикальной палки. |

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| zad7 | Класс | Основной класс, наследующий JFrame, представляющий окно с рисунком. |
| DrawingPanel | Класс | Подкласс класса JPanel, представляющий панель для рисования. |
| paintComponent | Метод | Метод класса DrawingPanel, отвечающий за отрисовку компонентов на панели. |
| g | Graphics | Объект Graphics, используемый для рисования на панели. |

Код программы:

public class zad7 extends JFrame {

public zad7() {

setTitle("Силуэт палатки");

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

setSize(400, 400);

setLocationRelativeTo(null);

setContentPane(new DrawingPanel());

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> {

zad7 drawing = new zad7();

drawing.setVisible(true);

});

}

}

class DrawingPanel extends JPanel {

@Override

protected void paintComponent(Graphics g) {

super.paintComponent(g);

int width = getWidth();

int height = getHeight();

// палатка

int[] xPointsTriangle = {width / 4 + 20, width / 2, 3 \* width / 4 - 20};

int[] yPointsTriangle = {height / 4, height / 8- 40, height / 4};

g.setColor(Color.black);

g.fillPolygon(xPointsTriangle, yPointsTriangle, 3);

int squareSize = height / 4; // стена палатки

int[] xPointsSquare = {width / 2, width / 2 + squareSize, width / 2 + squareSize + 40, 3 \* width / 4 - 50};

int[] yPointsSquare = {height / 8-40, 90,60,25};

g.setColor(Color.darkGray);

g.fillPolygon(xPointsSquare, yPointsSquare, 4);

g.setColor(Color.black);

Graphics2D g2d = (Graphics2D) g;

g2d.setStroke(new BasicStroke(5));

int leftStickX = width / 4 - 15;

g.drawLine(leftStickX, height / 4, leftStickX, height / 8);

// Вертикальная палка справа от палатки

int rightStickX = 3 \* width / 4 + 60;

g.drawLine(rightStickX, height / 4 -10, rightStickX, height / 8);

g.fillOval(width / 4 - 25, height / 20, 20, 50);

g.fillOval(3 \* width / 4 + 40, height / 25, 40, 40);

g2d.setStroke(new BasicStroke(3));

int triangleHeight = height / 16; // Уменьшена высота

g.drawLine(xPointsTriangle[0], yPointsTriangle[0], xPointsTriangle[0] - 10, yPointsTriangle[0] + triangleHeight);

g.drawLine(xPointsTriangle[2], yPointsTriangle[2], xPointsTriangle[2] + 10, yPointsTriangle[2] + triangleHeight);

// Горизонтальная палка

int horizontalStickY = 60; // Высота, на которой начинается горизонтальная палка

int horizontalStickLength = height / 25; // Длина горизонтальной палки

g.drawLine(width / 2 + squareSize + 40, horizontalStickY, width / 2 + squareSize + 40 + horizontalStickLength, horizontalStickY);

int smallStickHeight = 10; // Высота маленьких палок

g.drawLine(xPointsTriangle[0] - 10, yPointsTriangle[0] + triangleHeight, xPointsTriangle[0] - 10, yPointsTriangle[0] + triangleHeight - smallStickHeight);

g.drawLine(xPointsTriangle[2] + 10, yPointsTriangle[2] + triangleHeight, xPointsTriangle[2] + 10, yPointsTriangle[2] + triangleHeight - smallStickHeight);

int verticalStickX = width / 2 + squareSize + 40 + horizontalStickLength; // X-координата верхнего конца

int verticalStickHeight = 10; // Высота вертикальной палки

g.drawLine(verticalStickX, horizontalStickY, verticalStickX, horizontalStickY - verticalStickHeight);

}

}

Результат выполнения программы:

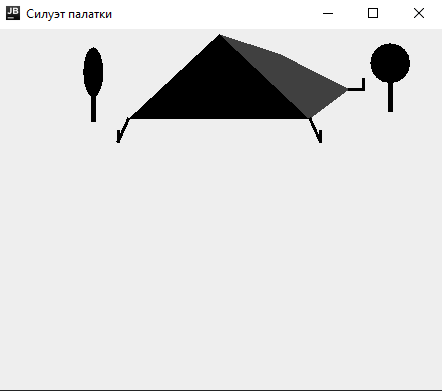


Рисунок 21 – результат седьмой задачи

## 1.8 Задача 8

Описание задания:

Вариант 20. Составить описание класса для объектов-векторов, задаваемых координатами их концов в трехмерном пространстве. Обеспечить при этом выполнение операций сложения и вычитания векторов с получением нового вектора (суммы или разности), вычисление скалярного произведения.

Таблица идентификаторов:

Таблица 8 – таблица идентификаторов к задаче восемь

|  |  |  |
| --- | --- | --- |
| Название | Тип | Описание |
| x, y, z | double | Переменные класса Vector3D, представляющие компоненты вектора. |
| x1Field, y1Field, z1Field | JTextField | Текстовые поля для ввода компонент первого вектора. |
| x2Field, y2Field, z2Field | JTextField | Текстовые поля для ввода компонент второго вектора. |
| resultArea | JTextArea | Область текста для вывода результатов вычислений. |
| calculateButton | JButton | Кнопка для запуска вычислений. |

Таблица 9 – таблица к задаче восемь

|  |  |  |
| --- | --- | --- |
| Vector3D | Класс | Класс, представляющий трехмерный вектор и обеспечивающий основные операции с векторами. |
| add | Метод | Метод класса Vector3D, выполняющий сложение векторов. |
| subtract | Метод | Метод класса Vector3D, выполняющий вычитание векторов. |
| dotProduct | Метод | Метод класса Vector3D, выполняющий скалярное произведение векторов. |
| length | Метод | Метод класса Vector3D, вычисляющий длину вектора. |
| cosineAngle | Метод | Метод класса Vector3D, вычисляющий косинус угла между векторами. |
| toString | Метод | Переопределенный метод toString для класса Vector3D, возвращающий строковое представление вектора. |
| zad8 | Класс | Основной класс, наследующий JFrame, представляющий окно приложения. |
| calculateVectors | Метод | Метод класса zad8, выполняющий вычисления на основе введенных пользователем данных. |
| main | Метод | Основной метод приложения, запускающий графический интерфейс. |
| run | Runnable | Анонимный класс, реализующий интерфейс Runnable и предоставляющий метод run для запуска приложения. |

Код программы:

class Vector3D {

private double x;

private double y;

private double z;

public Vector3D(double x, double y, double z) {

this.x = x;

this.y = y;

this.z = z;

}

public Vector3D add(Vector3D other) {

return new Vector3D(this.x + other.x, this.y + other.y, this.z + other.z);

}

public Vector3D subtract(Vector3D other) {

return new Vector3D(this.x - other.x, this.y - other.y, this.z - other.z);

}

public double dotProduct(Vector3D other) {

return this.x \* other.x + this.y \* other.y + this.z \* other.z;

}

public double length() {

return Math.sqrt(this.x \* this.x + this.y \* this.y + this.z \* this.z);

}

public double cosineAngle(Vector3D other) {

double dotProduct = this.dotProduct(other);

double lengthProduct = this.length() \* other.length();

if (lengthProduct == 0) {

return 0;

}

return dotProduct / lengthProduct;

}

@Override

public String toString() {

return "(" + x + ", " + y + ", " + z + ")";

}

}

public class zad8 extends JFrame {

private JTextField x1Field, y1Field, z1Field, x2Field, y2Field, z2Field;

private JTextArea resultArea;

public zad8() {

setTitle("Калькулятор");

setSize(400, 300);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

// Создание компонентов

JLabel vector1Label = new JLabel("Вектор 1:");

x1Field = new JTextField(5);

y1Field = new JTextField(5);

z1Field = new JTextField(5);

JLabel vector2Label = new JLabel("Вектор 2:");

x2Field = new JTextField(5);

y2Field = new JTextField(5);

z2Field = new JTextField(5);

JButton calculateButton = new JButton("Посчитать");

resultArea = new JTextArea(10, 30);

resultArea.setEditable(false);

// Добавление обработчика событий для кнопки

calculateButton.addActionListener(e -> calculateVectors());

// Размещение компонентов на панели

JPanel panel = new JPanel();

panel.setLayout(new GridLayout(4, 2));

panel.add(vector1Label);

panel.add(new JPanel()); // Пустая ячейка для выравнивания

panel.add(new JLabel("x:"));

panel.add(x1Field);

panel.add(new JLabel("y:"));

panel.add(y1Field);

panel.add(new JLabel("z:"));

panel.add(z1Field);

panel.add(vector2Label);

panel.add(new JPanel()); // Пустая ячейка для выравнивания

panel.add(new JLabel("x:"));

panel.add(x2Field);

panel.add(new JLabel("y:"));

panel.add(y2Field);

panel.add(new JLabel("z:"));

panel.add(z2Field);

// Добавление компонентов на основное окно

add(panel, BorderLayout.NORTH);

add(calculateButton, BorderLayout.CENTER);

add(new JScrollPane(resultArea), BorderLayout.SOUTH);

}

private void calculateVectors() {

try {

double x1 = Double.parseDouble(x1Field.getText());

double y1 = Double.parseDouble(y1Field.getText());

double z1 = Double.parseDouble(z1Field.getText());

double x2 = Double.parseDouble(x2Field.getText());

double y2 = Double.parseDouble(y2Field.getText());

double z2 = Double.parseDouble(z2Field.getText());

Vector3D vector1 = new Vector3D(x1, y1, z1);

Vector3D vector2 = new Vector3D(x2, y2, z2);

Vector3D sum = vector1.add(vector2);

Vector3D difference = vector1.subtract(vector2);

double dotProduct = vector1.dotProduct(vector2);

double length = vector1.length();

double cosineAngle = vector1.cosineAngle(vector2);

// Вывод результатов

resultArea.setText("Сумма векторов: " + sum.toString() + "\n"

+ "Разность векторов: " + difference.toString() + "\n"

+ "Скалярное произведение : " + dotProduct + "\n"

+ "Длина вектора: " + length + "\n"

+ "Косинус угла между векторами: " + cosineAngle);

} catch (NumberFormatException ex) {

resultArea.setText("Ошибка.");

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new zad8().setVisible(true));

}

}

Результат выполнения программы:

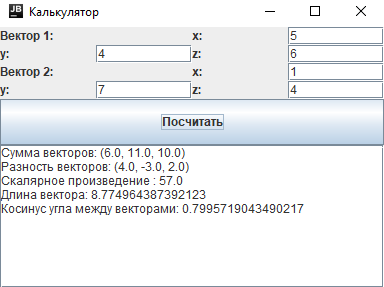


Рисунок 22 – результат восьмой задачи

## 1.9 Задача 9

Описание задания:

Имеется квадратное поле из 25 клеток и набор из 52 карточек, на которых записаны числа от 1 до 13, причем карточки с каждым из этих чисел встречаются по четыре раза.

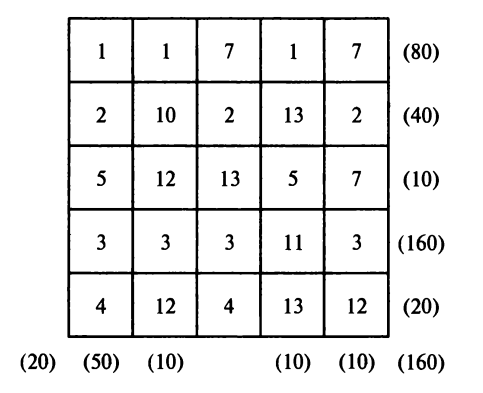


Рисунок 23 – пример игрового поля

Разработать программу, которая позволит имитировать игру человека с компьютером: случайным образом извлекается какая-либо из имеющихся карточек и выдается записанное на ней число. Каждый игрок заносит это число в одну из клеток квадрата, и так продолжается до тех пор, пока не будут заполнены все клетки квадрата.

По окончании игры заполнение клеток поля оценивается определенным количеством очков. Цель игры — разместить числа в клетках таким образом, чтобы набрать наибольшее количество очков в соответствии с рисунком 24.

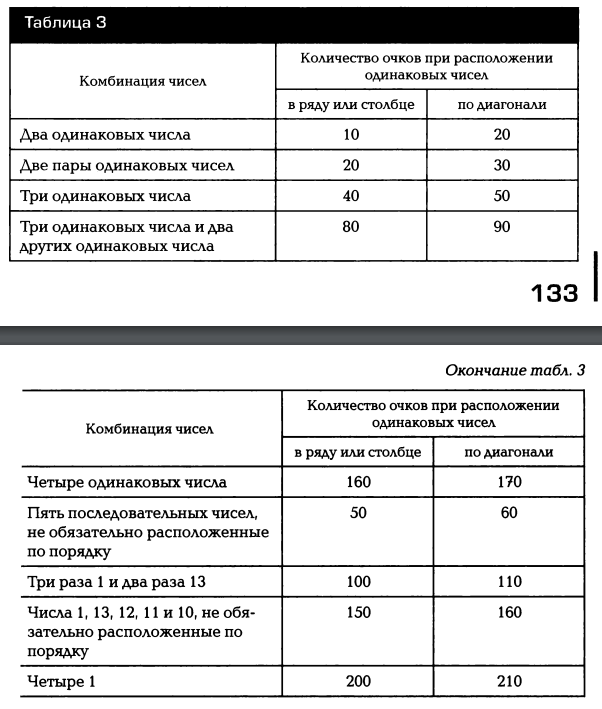


Рисунок 24 – таблица, в соответствии с которой считаются очки

Таблица идентификаторов:

Таблица 10 - таблица идентификаторов к задаче 9

|  |  |  |  |
| --- | --- | --- | --- |
| Название | Тип | Тип переменной | Описание |
| BOARD\_SIZE | int | Вспомогательная | Константа, определяющая размер доски (5x5). |
| numberButtons | JButton[][] | Вспомогательная | Двумерный массив кнопок, представляющих ячейки на доске. |
| numberCards | ArrayList<Integer> | Вспомогательная | Список, содержащий карты (числа от 1 до 13) для игры. |
| gameBoard | int[][] | Вспомогательная | Двумерный массив, представляющий текущее состояние доски. |
| isPlayerTurn | boolean | Вспомогательная | Флаг, указывающий на ход игрока или компьютера. |
| playerPoints | int | Выход | Переменная для хранения очков игрока. |
| computerPoints | int | Выход | Переменная для хранения очков компьютера. |
| scoreLabel | JLabel | Вспомогательная | Метка для отображения текущих очков игрока и компьютера. |
| x1Field, y1Field, z1Field, x2Field, y2Field, z2Field | JTextField | Вход | Поля для ввода координат векторов в интерфейсе. |
| resultArea | JTextArea | Выход | Область для отображения результатов вычислений векторов. |
| selectedCard | int | Вспомогательная | Выбранная карта для установки в ячейку доски. |
| emptyCell | int[] | Вспомогательная | Массив, представляющий пустую ячейку на доске. |
| i, j | int | Вспомогательная | Переменные для итерации по доске и массивам. |

Таблица компонентов

Таблица 11 - таблица компонентов для задачи 9

|  |  |  |
| --- | --- | --- |
| Элемент кода | Тип | Описание |
| zad9 | Класс | Основной класс приложения, расширяющий JFrame. |
| main | Метод | Точка входа в программу, инициализирующая zad9. |
| initializeNumberCards | Метод | Инициализация списка numberCards с картами. |
| initializeGameBoard | Метод | Инициализация массива gameBoard. |
| initializeUserInterface | Метод | Создание пользовательского интерфейса. |
| startGame | Метод | Начало новой игры. |
| updateScoreLabel | Метод | Обновление метки с текущими очками. |
| handleButtonClick | Метод | Обработка нажатия кнопки игроком. |
| playComputerTurn | Метод | Ход компьютера. |
| findEmptyCellForComputer | Метод | Поиск пустой ячейки для компьютера. |
| checkPoints | Метод | Подсчёт очков после хода. |
| calculatePoints | Метод | Подсчёт очков для заданного направления. |
| calculateScore | Метод | Подсчёт очков для конкретного набора чисел. |
| isDiagonal | Метод | Проверка, является ли направление диагональным. |
| isConsecutive | Метод | Проверка, является ли набор чисел последовательным. |
| containsNumbers | Метод | Проверка наличия конкретных чисел в наборе. |
| containsOtherNumbers | Метод | Проверка наличия других чисел в наборе. |
| endGame | Метод | Завершение игры и вывод диалогового окна. |
| showEndGameDialog | Метод | Отображение диалогового окна с результатами игры. |

Блок-схема алгоритма:

Код программы:

public class zad9 extends JFrame {

private static final int BOARD\_SIZE = 5;

private final JButton[][] numberButtons;

private final ArrayList<Integer> numberCards;

private final int[][] gameBoard = new int[BOARD\_SIZE][BOARD\_SIZE];

private boolean isPlayerTurn;

private int playerPoints;

private int computerPoints;

private JLabel scoreLabel;

public zad9() {

setTitle("Итальянская игра 'мАтЕмАтИкО'");

setSize(400, 450);

setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

numberButtons = new JButton[BOARD\_SIZE][BOARD\_SIZE];

numberCards = new ArrayList<>();

isPlayerTurn = true;

initializeNumberCards();

initializeGameBoard();

initializeUserInterface();

}

private void initializeNumberCards() {

for (int i = 1; i <= 13; i++) {

for (int j = 0; j < 4; j++) {

numberCards.add(i);

}

}

Collections.shuffle(numberCards);

}

private void initializeGameBoard() {

for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) {

gameBoard[i][j] = 0;

}

}

}

private void initializeUserInterface() {

setLayout(new BorderLayout());

JPanel gamePanel = new JPanel(new GridLayout(BOARD\_SIZE, BOARD\_SIZE));

for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) {

numberButtons[i][j] = new JButton();

numberButtons[i][j].setFont(new Font("Arial", Font.PLAIN, 20));

numberButtons[i][j].setEnabled(false);

final int row = i;

final int col = j;

numberButtons[i][j].setBackground(Color.BLUE);

numberButtons[i][j].addActionListener(e -> handleButtonClick(row, col));

gamePanel.add(numberButtons[i][j]);

}

}

add(gamePanel, BorderLayout.CENTER);

// Создаем новую панель для отображения счета

JPanel scorePanel = new JPanel(new FlowLayout(FlowLayout.CENTER));

scoreLabel = new JLabel("Очки игрока: 0 Очки компьютера: 0");

scorePanel.add(scoreLabel);

add(scorePanel, BorderLayout.SOUTH);

startGame();

}

private void startGame() {

isPlayerTurn = true;

playerPoints = 0;

computerPoints = 0;

for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) {

numberButtons[i][j].setEnabled(true);

}

}

playComputerTurn();

}

private void updateScoreLabel() {

scoreLabel.setText("Очки игрока: " + playerPoints + " Очки компьютера: " + computerPoints);

}

private void handleButtonClick(int row, int col) {

if (isPlayerTurn && gameBoard[row][col] == 0) {

int selectedCard = numberCards.remove(0);

gameBoard[row][col] = selectedCard;

numberButtons[row][col].setText(Integer.toString(selectedCard));

numberButtons[row][col].setEnabled(false);

numberButtons[row][col].setForeground(Color.BLACK);

isPlayerTurn = false;

checkPoints();

playComputerTurn();

updateScoreLabel();

}

}

private void playComputerTurn() {

if (!isPlayerTurn) {

int selectedCard = numberCards.remove(0);

int[] emptyCell = findEmptyCellForComputer();

if (emptyCell != null) {

gameBoard[emptyCell[0]][emptyCell[1]] = selectedCard;

numberButtons[emptyCell[0]][emptyCell[1]].setText(Integer.toString(selectedCard));

numberButtons[emptyCell[0]][emptyCell[1]].setEnabled(false);

}

isPlayerTurn = true;

checkPoints();

updateScoreLabel();

}

}

private int[] findEmptyCellForComputer() {

ArrayList<int[]> emptyCells = new ArrayList<>();

for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) {

if (gameBoard[i][j] == 0) {

emptyCells.add(new int[]{i, j});

}

}

}

if (!emptyCells.isEmpty()) {

return emptyCells.get((int) (Math.random() \* emptyCells.size()));

} else {

return null;

}

}

private void checkPoints() {

if (isPlayerTurn) {

playerPoints = calculatePoints(true);

} else {

computerPoints = calculatePoints(false);

}

updateScoreLabel();

if (numberCards.isEmpty()) {

endGame();

}

}

private int calculatePoints(boolean isPlayer) {

int points = 0;

for (int i = 0; i < BOARD\_SIZE; i++) {

points += calculateScore(gameBoard[i], isPlayer);

}

for (int j = 0; j < BOARD\_SIZE; j++) {

int[] column = new int[BOARD\_SIZE];

for (int i = 0; i < BOARD\_SIZE; i++) {

column[i] = gameBoard[i][j];

}

points += calculateScore(column, isPlayer);

}

int[] diagonal1 = new int[BOARD\_SIZE];

int[] diagonal2 = new int[BOARD\_SIZE];

for (int i = 0; i < BOARD\_SIZE; i++) {

diagonal1[i] = gameBoard[i][i];

diagonal2[i] = gameBoard[i][BOARD\_SIZE - 1 - i];

}

points += calculateScore(diagonal1, isPlayer);

points += calculateScore(diagonal2, isPlayer);

return points;

}

private int calculateScore(int[] numbers, boolean isPlayer) {

int points = 0;

int[] counts = new int[14];

for (int num : numbers) {

counts[num]++;

}

for (int i = 1; i <= 13; i++) {

int count = counts[i];

if (count > 0) {

switch (count) {

case 2:

points += (isDiagonal(numbers) ? 20 : 10);

break;

case 4:

points += (isDiagonal(numbers) ? 30 : 20);

break;

case 3:

if (containsOtherNumbers(counts, i)) {

points += (isDiagonal(numbers) ? 50 : 40);

} else {

points += (isDiagonal(numbers) ? 90 : 80);

}

break;

case 5:

if (isConsecutive(numbers)) {

points += (isDiagonal(numbers) ? 60 : 50);

} else {

points += (isDiagonal(numbers) ? 170 : 60);

}

break;

}

}

}

if (counts[1] == 3 && counts[13] == 2) {

points += (isDiagonal(numbers) ? 110 : 100);

}

if (containsNumbers(numbers, 1, 13, 12, 11, 10)) {

points += (isDiagonal(numbers) ? 160 : 150);

}

if (counts[1] == 4) {

points += (isDiagonal(numbers) ? 210 : 200);

}

return points;

}

private boolean isDiagonal(int[] numbers) {

return numbers.length == BOARD\_SIZE;

}

private boolean isConsecutive(int[] numbers) {

int firstNum = numbers[0];

for (int i = 1; i < numbers.length; i++) {

if (numbers[i] != firstNum + i) {

return false;

}

}

return true;

}

private boolean containsNumbers(int[] numbers, int... targetNumbers) {

for (int target : targetNumbers) {

boolean contains = false;

for (int num : numbers) {

if (num == target) {

contains = true;

break;

}

}

if (!contains) {

return false;

}

}

return true;

}

private boolean containsOtherNumbers(int[] counts, int excludeNumber) {

for (int i = 1; i <= 13; i++) {

if (i != excludeNumber && counts[i] > 0) {

return true;

}

}

return false;

}

private void endGame() {

updateScoreLabel();

for (int i = 0; i < BOARD\_SIZE; i++) {

for (int j = 0; j < BOARD\_SIZE; j++) {

numberButtons[i][j].setEnabled(false);

}

}

// Добавим вызов метода showEndGameDialog для отображения окна "Конец!"

showEndGameDialog();

}

// Добавим новый метод showEndGameDialog

private void showEndGameDialog() {

// Сформируем текст для диалогового окна

String result = "Игра окончена!\n";

result += "Очки игрока: " + playerPoints + "\n";

result += "Очки компьютера: " + computerPoints;

// Выведем диалоговое окно

JOptionPane.showMessageDialog(this, result, "Конец!", JOptionPane.INFORMATION\_MESSAGE);

// Закроем приложение после окончания игры

System.exit(0);

}

public static void main(String[] args) {

SwingUtilities.invokeLater(() -> new zad9().setVisible(true));

}

}

Результат выполнения программы:

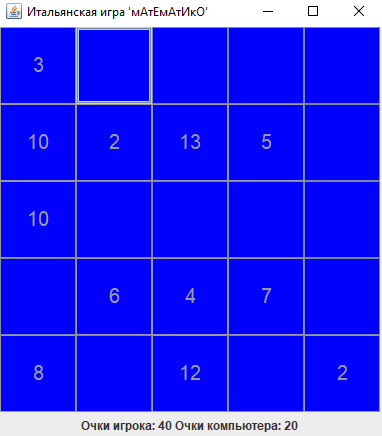


Рисунок 25 – результат задачи 9

# 2 Разработка АИС «Автовокзал»

## 2.1 ER-диаграмма, диаграмма классов.

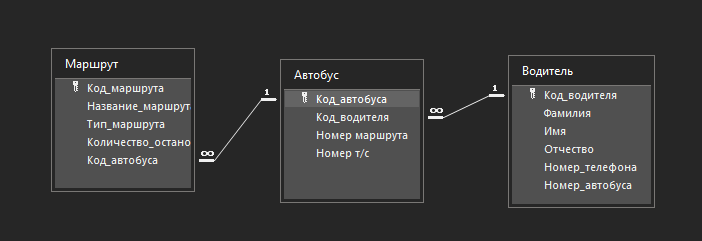


Рисунок 26 – ER-диаграмма базы данных "Автовокзал"

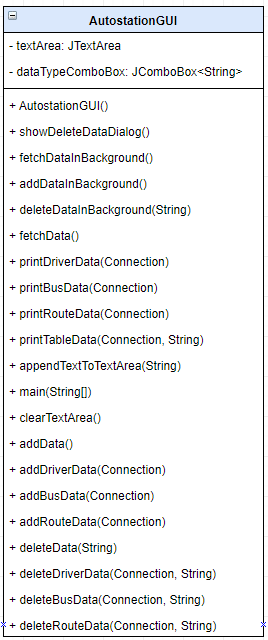


Рисунок 27 – диаграмма классов

## 2.2 Техническое задание

Наименование проекта:

Автоматизированная информационная система для управления автовокзалом.

Цель проекта:

Разработать систему, которая позволит эффективно управлять операционными процессами автовокзала, просматривать данные автобусов, водителей и маршрутов, удалять и добавлять данные.

Задачи проекта:

1. Учет водителей:
   * создать базу данных, включающую информацию о водителях;
   * разработать интерфейс для добавления и удаления новых водителей.
2. Учет автопарка:
   * создать базу данных, включающую информацию об автобусах;
   * разработать интерфейс для добавления и удаления новых автобусов.
3. Управление маршрутами:
   * возможность добавления и удаления маршрутов.
4. Интерфейс пользователя:
   * разработать удобный интерфейс для работы с системой.

Технологии:

* + backend: Java;
  + frontend: Java;
  + база данных: PostgreSQL;
  + среда разработки: IntelliJ IDEA, Android Studio.

Сроки реализации:

* + проект должен быть завершен и готов к внедрению не позднее 9 декабря 2023 года.

Критерии приемки проекта:

* + все функциональные и технические требования должны быть реализованы;
  + система должна быть стабильной, безопасной и масштабируемой;
  + должны быть предоставлены документация по использованию системы и ее техническое описание.

## 2.3 Краткая спецификация системы

Окно и компоненты GUI:

1. AutostationGUI:
   * Класс AutostationGUI наследуется от JFrame и представляет собой основное окно GUI.
   * В конструкторе устанавливаются параметры окна, такие как заголовок, размер и операция при закрытии.
2. Компоненты GUI:
   * textArea: JTextArea для отображения текстовой информации (результатов запросов к базе данных);
   * dataTypeComboBox: JComboBox для выбора типа данных (водители, автобусы, маршруты);
   * кнопки (fetchDataButton, addDataButton, deleteDataButton) для выполнения операций.

Методы для работы с базой данных:

1. Методы для фоновых операций:
   * fetchDataInBackground, addDataInBackground, deleteDataInBackground - используют SwingWorker для выполнения длительных операций в фоновом режиме.
2. Методы для работы с базой данных:
   * fetchData: подключается к базе данных PostgreSQL и в зависимости от выбранного типа данных выводит соответствующую информацию в textArea.
   * addData: добавляет данные в базу данных в зависимости от выбранного типа данных.
   * deleteData: удаляет данные из базы данных в зависимости от выбранного типа данных.
3. Методы для вывода данных в textArea:
   * printDriverData, printBusData, printRouteData - выводят данные о водителях, автобусах и маршрутах соответственно.
   * printTableData - общий метод для вывода данных из таблицы.

Методы для обновления GUI:

Методы для обновления textArea:

* + appendTextToTextArea: добавляет текст в textArea в потокобезопасном режиме.
  + clearTextArea: очищает textArea.

Методы для добавления данных в базу данных:

* + addDriverData, addBusData, addRouteData - предоставляют диалоговые окна для ввода данных и добавляют их в соответствующую таблицу базы данных.

Методы для удаления данных из базы данных:

* + deleteDriverData, deleteBusData, deleteRouteData - предоставляют диалоговое окно для ввода ID данных, которые нужно удалить, и затем удаляют их из соответствующей таблицы.

Главный метод:

main:

* + Запускает приложение, создавая объект AutostationGUI и устанавливая его видимость.

Работа с базой данных:

Подключение к базе данных:

* + Используется JDBC для подключения к базе данных PostgreSQL.

Обработка исключений:

* + Используется обработка исключений для отлавливания ошибок при работе с базой данных и отображения соответствующих сообщений об ошибке.

## 2.4 Функциональная схема

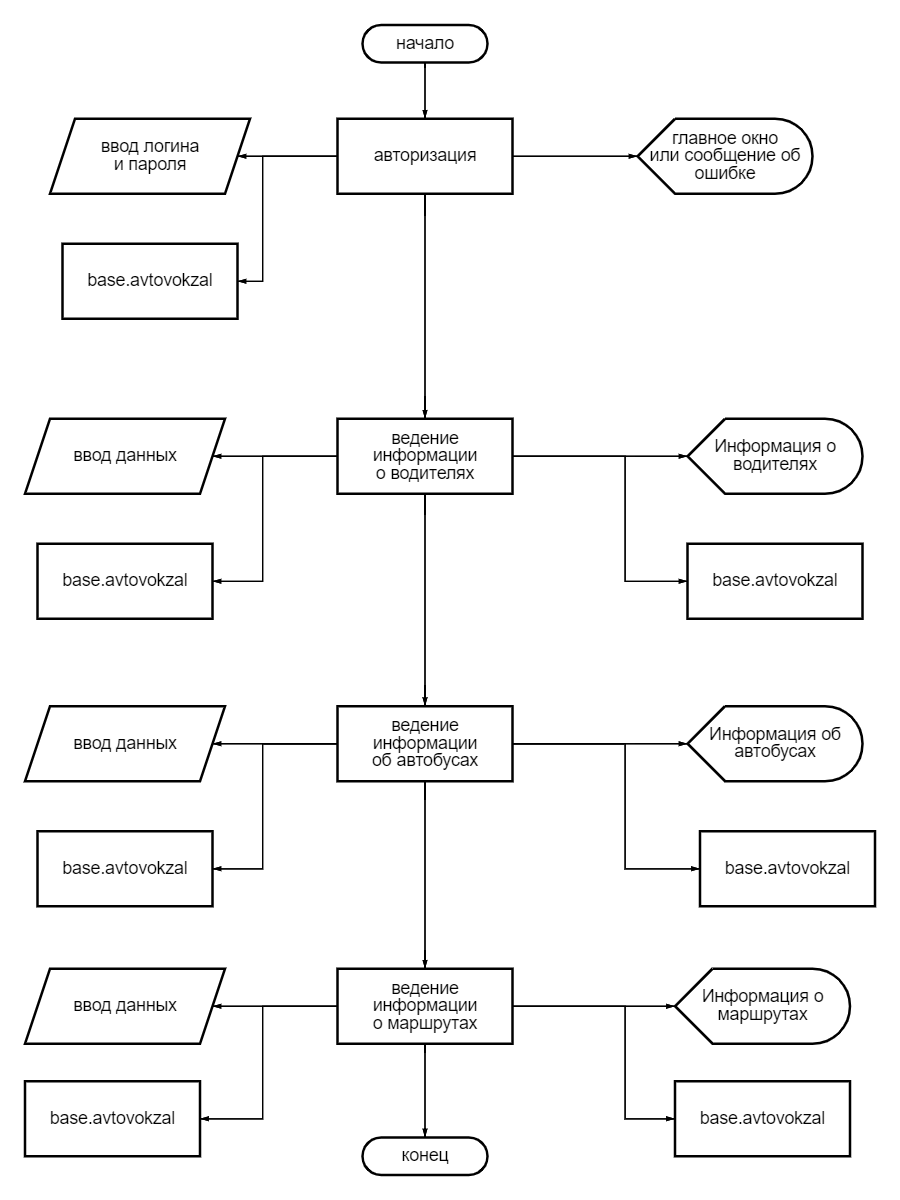


Рисунок 28 – функциональная схема АИС "Автовокзал"

## 2.5 Интерфейс

При создании АИС был выбран наименее примитивный и интуитивно-понятный интерфейс, чтобы даже пользователь с минимальными навыками пользования компьютера мог разобраться и эффективно использовать.

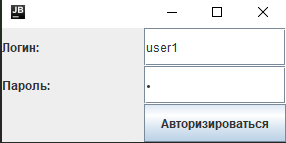


Рисунок 29 – окно авторизации

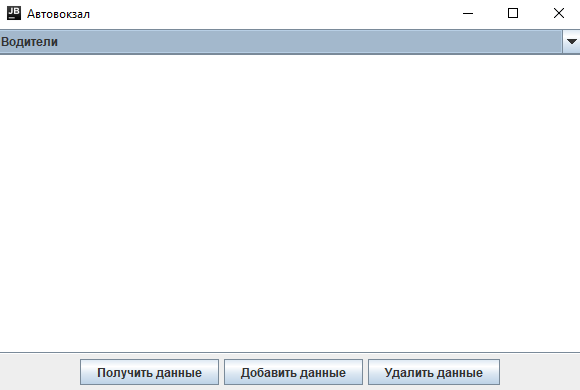


Рисунок 30 – главное меню

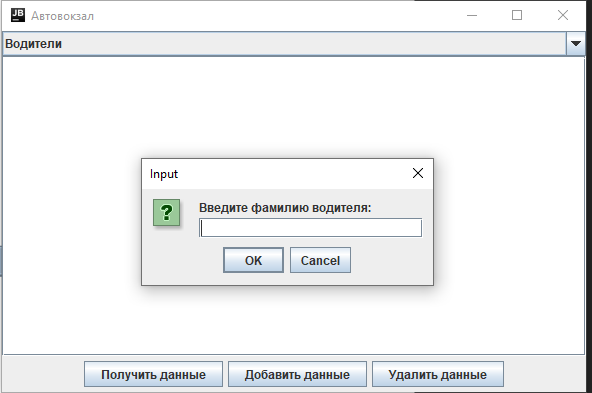


Рисунок 31 – интерфейс добавления данных в таблицу "водители"

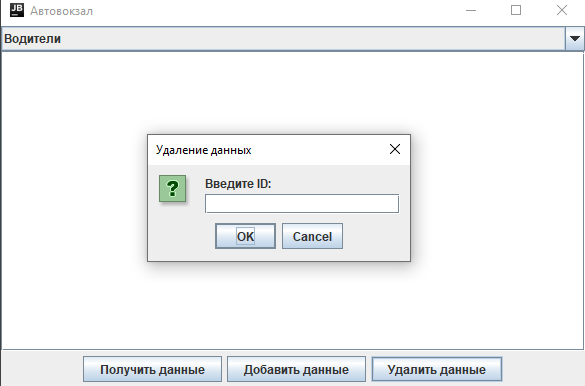


Рисунок 32 – интерфейс удаления данных из таблицы "водители"

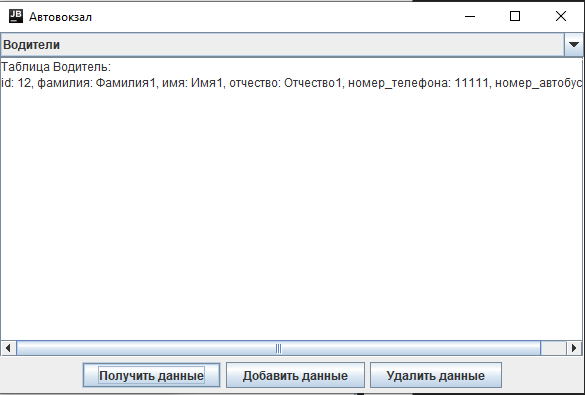


Рисунок 33 – вывод данных после нажатия кнопки "Получить данные"

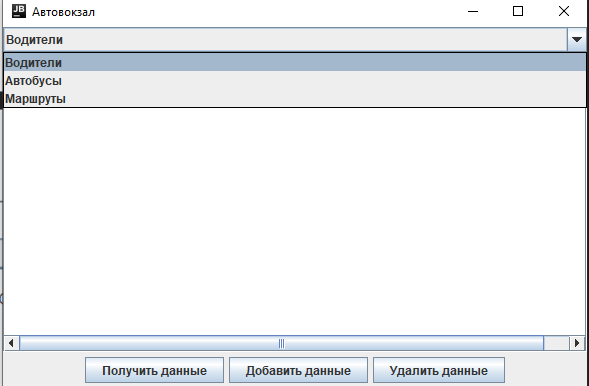


Рисунок 34 – интерфейс выбора нужной таблицы

Начало формы

## 2.6 Тестирование

Попытка ввода несуществующего водителя при создании нового автобуса:

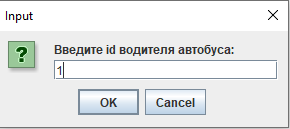


Рисунок 35 – введен несуществующий ID водителя

После окончания ввода всех данных появляется окно, сообщающее об ошибке

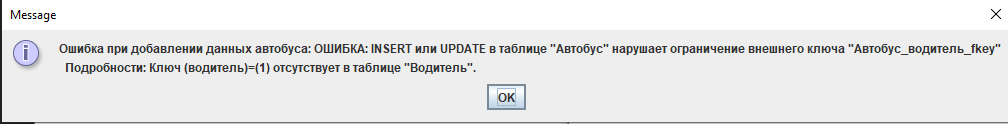


Рисунок 36 – окно ошибки при добавлении данных

Проверка добавился ли “неправильный” объект в таблицу:

После получения сообщения об ошибке при добавлении данных, получаем данные таблицы «Автобусы»

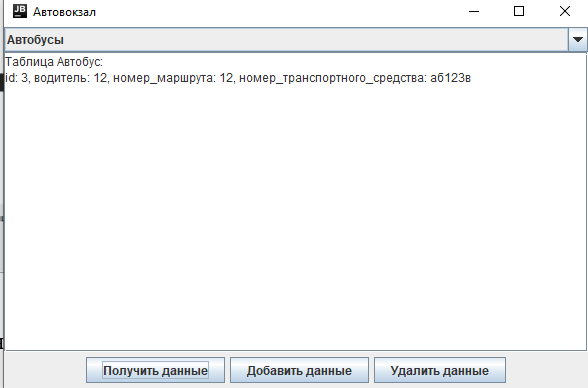


Рисунок 37 – данные, находящиеся внутри таблицы "Автобусы"

При анализе полученных данных было выяснено, что ошибочные данные не заносятся в таблицу.

## 2.7 Оценка программного продукта

Оценить эффективность использования ресурсов программного продукта можно на нескольких уровнях: от языка программирования и использования библиотек до организации кода. В данном случае, рассмотрим общий взгляд на несколько аспектов:

1. Язык программирования и Фреймворк:
   * Программа написана на Java, что в целом является хорошим выбором для кросс-платформенных приложений и систем среднего и большого размера.
   * Использование библиотеки Swing для создания графического интерфейса является стандартным подходом, но существуют более современные альтернативы, такие как JavaFX.
2. Эффективность Работы с Базой Данных:
   * Запросы к базе данных могут быть оптимизированы. Например, при выводе данных из таблицы можно использовать индексы и выбирать только необходимые столбцы.
   * Переиспользование соединения с базой данных может улучшить эффективность, нежели создание нового соединения для каждого запроса.
3. Паттерн Проектирования:
   * В коде представлены элементы паттерна проектирования "Наблюдатель" (Observer), что позволяет асинхронно обновлять интерфейс при получении новых данных.
   * Однако, в коде не используются многие другие популярные паттерны, такие как "Фабрика", "Строитель" или "Стратегия", которые могли бы улучшить структуру приложения.
4. Оптимизация Интерфейса:
   * Графический интерфейс работает в основном на событиях, что обеспечивает отзывчивость приложения. Однако, для более сложных и крупных приложений могут потребоваться асинхронные механизмы.
5. Управление Ресурсами:
   * Использование try-with-resources для управления ресурсами, такими как соединения с базой данных, является хорошей практикой.
6. Структура Кода:
   * Код может быть организован более структурированно. Разделение логики на классы и методы может улучшить читаемость и поддерживаемость кода.
7. Тестирование:
   * В коде присутствует некоторый функционал фоновых операций. Обеспечьте, чтобы они были поддерживаемыми и тестировались для предотвращения возможных проблем.
8. Локализация и Интернационализация:
   * В коде отсутствуют механизмы локализации и интернационализации текстовых сообщений, что может быть важным для глобальных приложений.
9. Безопасность:
   * Необходимо обеспечить безопасность запросов к базе данных, например, использовать параметризованные запросы для предотвращения SQL-инъекций.
10. Оптимизация Запросов:
    * Оптимизация запросов к базе данных, особенно при работе с большим объемом данных, может значительно повысить производительность.

Код функционален, есть возможности для улучшения эффективности и структурирования. Использование дополнительных паттернов проектирования, оптимизация запросов и структурирование кода могут сделать приложение более эффективным и поддерживаемым.

# 3 Разработка мобильного приложения

Выбор Android Studio для разработки мобильного приложения на платформе Android может быть обоснован несколькими важными причинами:

1. Официальная Интегрированная Среда Разработки (IDE):
   * Android Studio является официальной IDE для разработки приложений под Android. Это означает, что она поддерживается и поддерживается напрямую командой Google, ответственной за операционную систему Android. Таким образом, разработчики могут рассчитывать на актуальные инструменты и ресурсы для эффективной работы.
2. Богатый Набор Инструментов:
   * Android Studio предоставляет широкий набор инструментов для разработки, от отладки и профилирования приложения до создания пользовательского интерфейса. Интегрированный визуальный редактор, эмуляторы устройств и другие инструменты значительно облегчают процесс разработки.
3. Поддержка Языков Программирования:
   * Android Studio поддерживает несколько языков программирования, включая Java и Kotlin. Это предоставляет разработчикам гибкость в выборе языка в зависимости от их предпочтений и требований проекта.
4. Эффективная Интеграция с Android SDK:
   * Android Studio предоставляет прямую интеграцию с Android Software Development Kit (SDK), что обеспечивает доступ к необходимым библиотекам, инструментам и API для разработки приложений на Android.
5. Поддержка Kotlin как Официального Языка:
   * Android Studio официально поддерживает Kotlin как альтернативу Java для разработки Android-приложений. Kotlin предлагает более современный синтаксис, безопасность типов и другие преимущества, что может сделать код более чистым и эффективным.
6. Активное Сообщество и Обширная Документация:
   * Android Studio имеет активное сообщество разработчиков, а также обширную документацию и учебные ресурсы. Это облегчает получение помощи, решение проблем и обучение новичков.
7. Интеграция с Системами Контроля Версий:
   * Android Studio хорошо интегрируется с системами контроля версий, такими как Git, что облегчает совместную работу над проектами и отслеживание изменений.
8. Непрерывные Обновления и Улучшения:
   * Android Studio регулярно обновляется, что гарантирует получение новых функций, исправлений ошибок и поддержки последних технологических трендов в области мобильной разработки.

Android Studio предоставляет разработчикам всё необходимое для эффективной и комфортной разработки Android-приложений, и это является причиной его выбора для разработки мобильного приложения.

## 3.1 Дизайн мобильного приложения



Рисунок 38 – главное меню

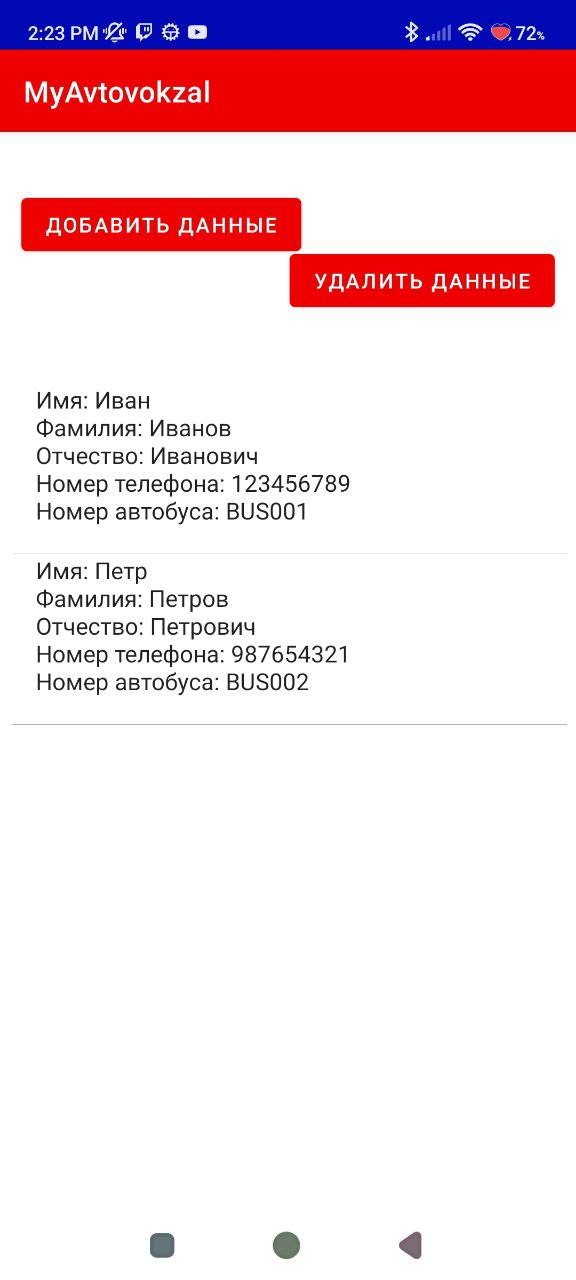


Рисунок 39 – экран с данными водителей

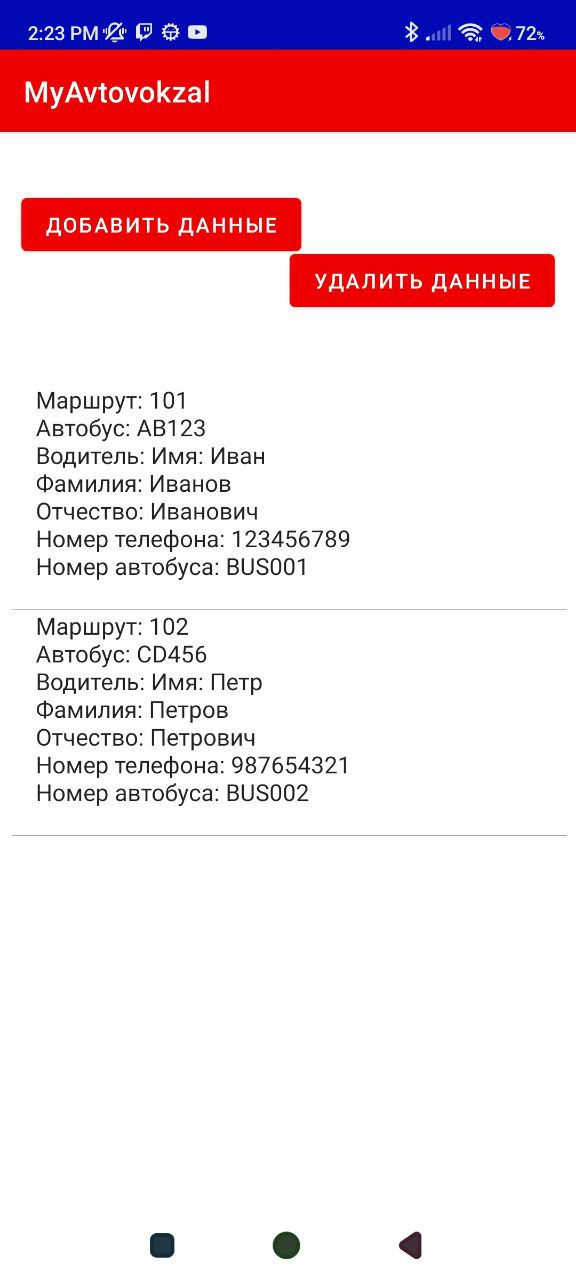


Рисунок 40 – экран с данными автобусов

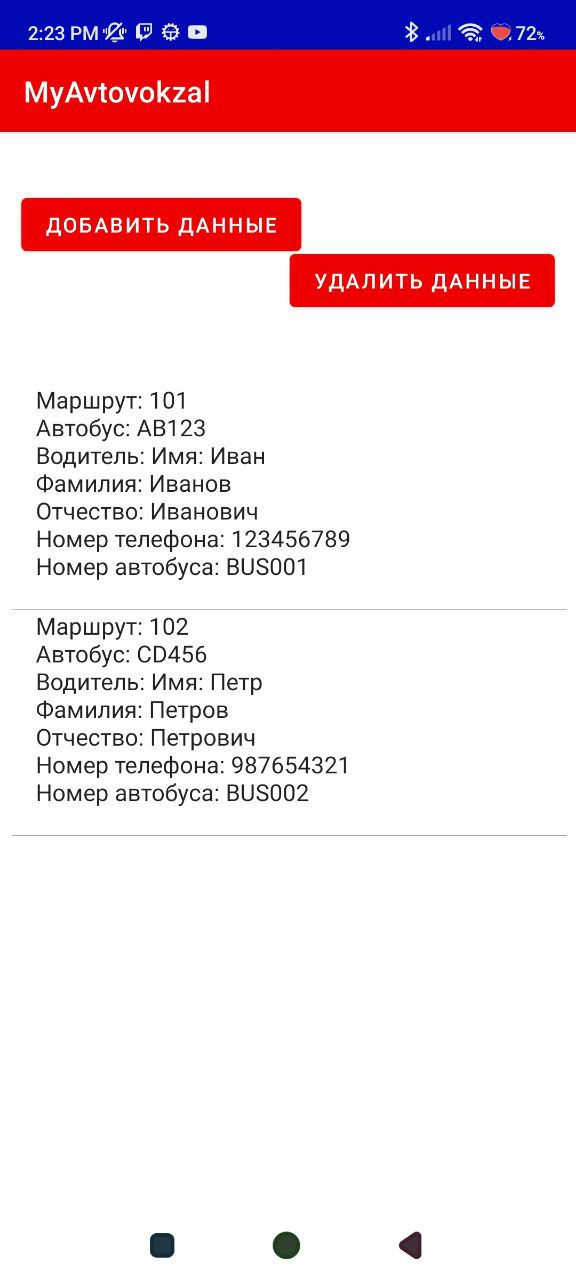


Рисунок 41 – экран с данными маршрутов

# 4 Разработка сайта

Ссылка на сайт - <https://sites.google.com/view/praktika231/главная-страница>

## 4.1 Выбор инструментов

Для разработки сайта был выбран сервис Google Sites. Причины почему был выбран Google Sites для разработки сайта:

1. Простота использования:
   * Google Sites предоставляет простой и интуитивно понятный интерфейс, что делает его доступным для широкого круга пользователей, включая тех, кто не является профессиональным веб-разработчиком.
2. Быстрое развертывание:
   * Google Sites позволяет быстро создавать и развертывать веб-сайты без необходимости в сложной конфигурации серверов или кодирования.
3. Интеграция с другими сервисами Google:
   * При использовании экосистемы Google, такой как Google Workspace (ранее G Suite), Google Sites может легко интегрироваться с другими сервисами, такими как Google Drive, Google Calendar и другими.
4. Бесплатность или низкая стоимость:
   * В некоторых случаях, особенно для небольших проектов или личных веб-страниц, использование Google Sites может быть бесплатным или стоить намного меньше по сравнению с альтернативными решениями.
5. Хостинг на серверах Google:
   * Сайты, созданные с помощью Google Sites, хостятся на серверах Google, что может обеспечивать надежность и быстрый доступ.

Учитывая эти факторы, использование Google Sites может быть оправданным в случае, если простота использования, быстрое развертывание и интеграция с другими сервисами Google важны для конкретного проекта или задачи.

## 4.2 Интерфейс сайта

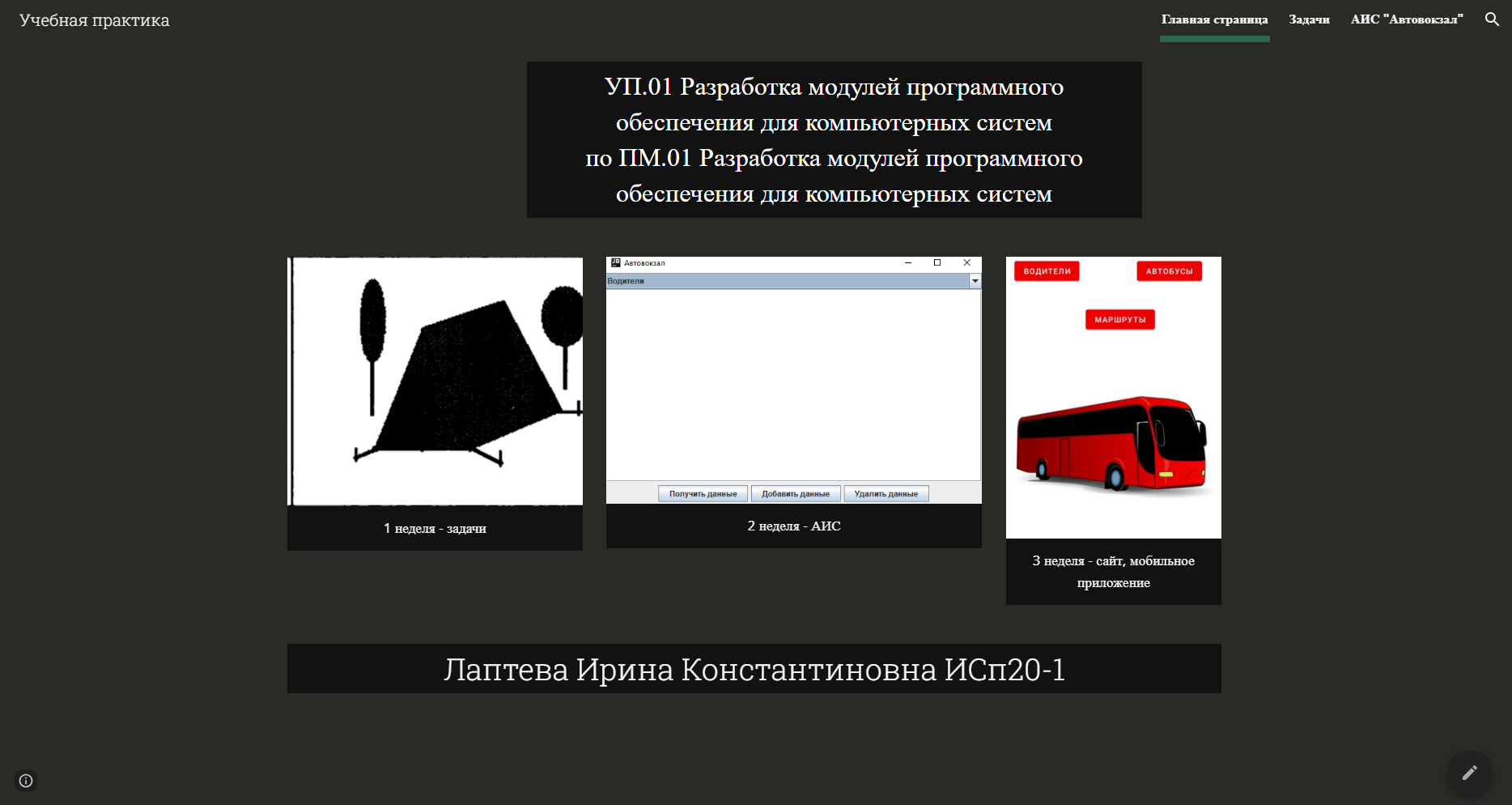


Рисунок 42 – главная страница сайта

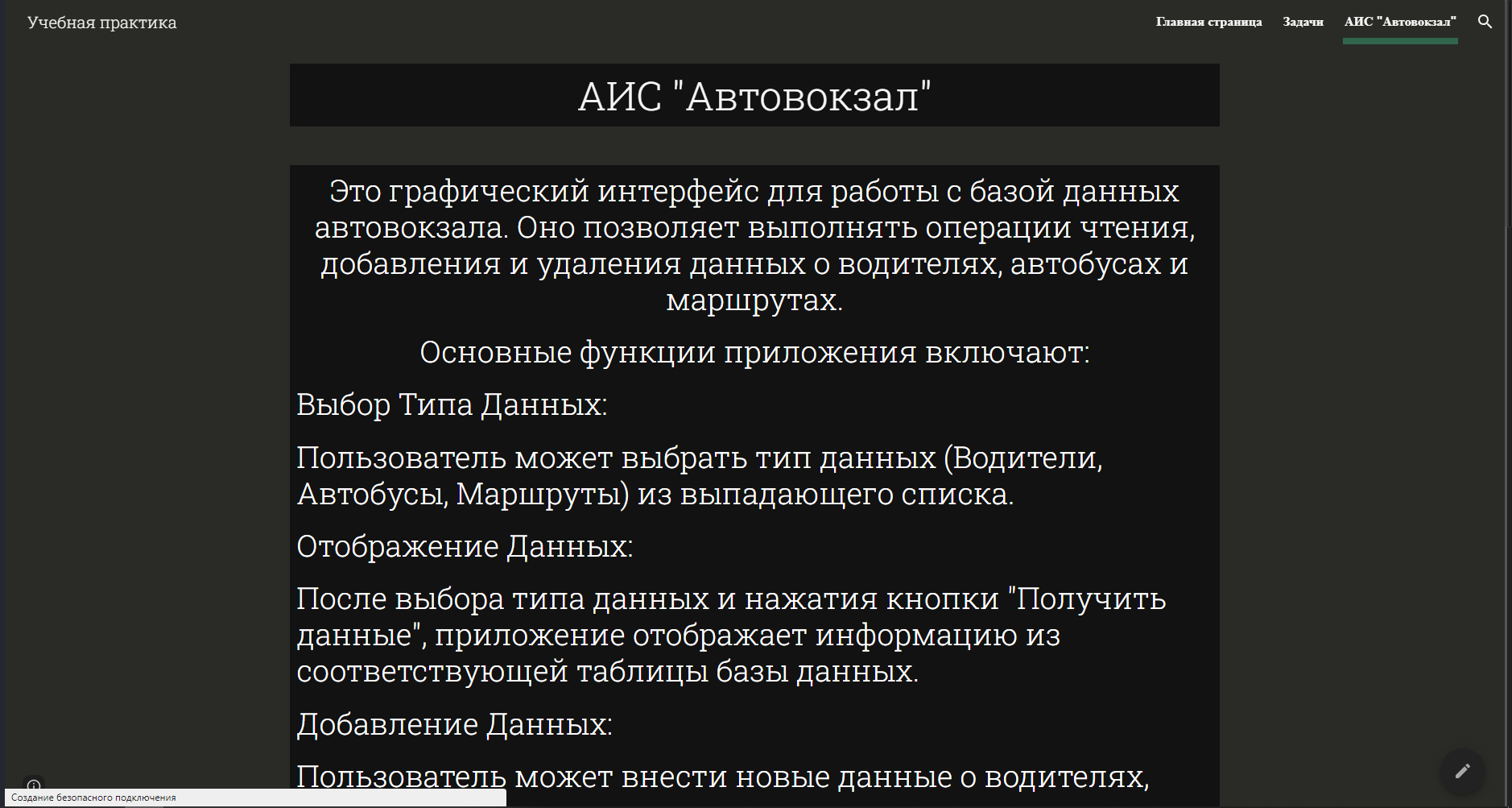


Рисунок 43 – страница АИС «Автовокзал» 1

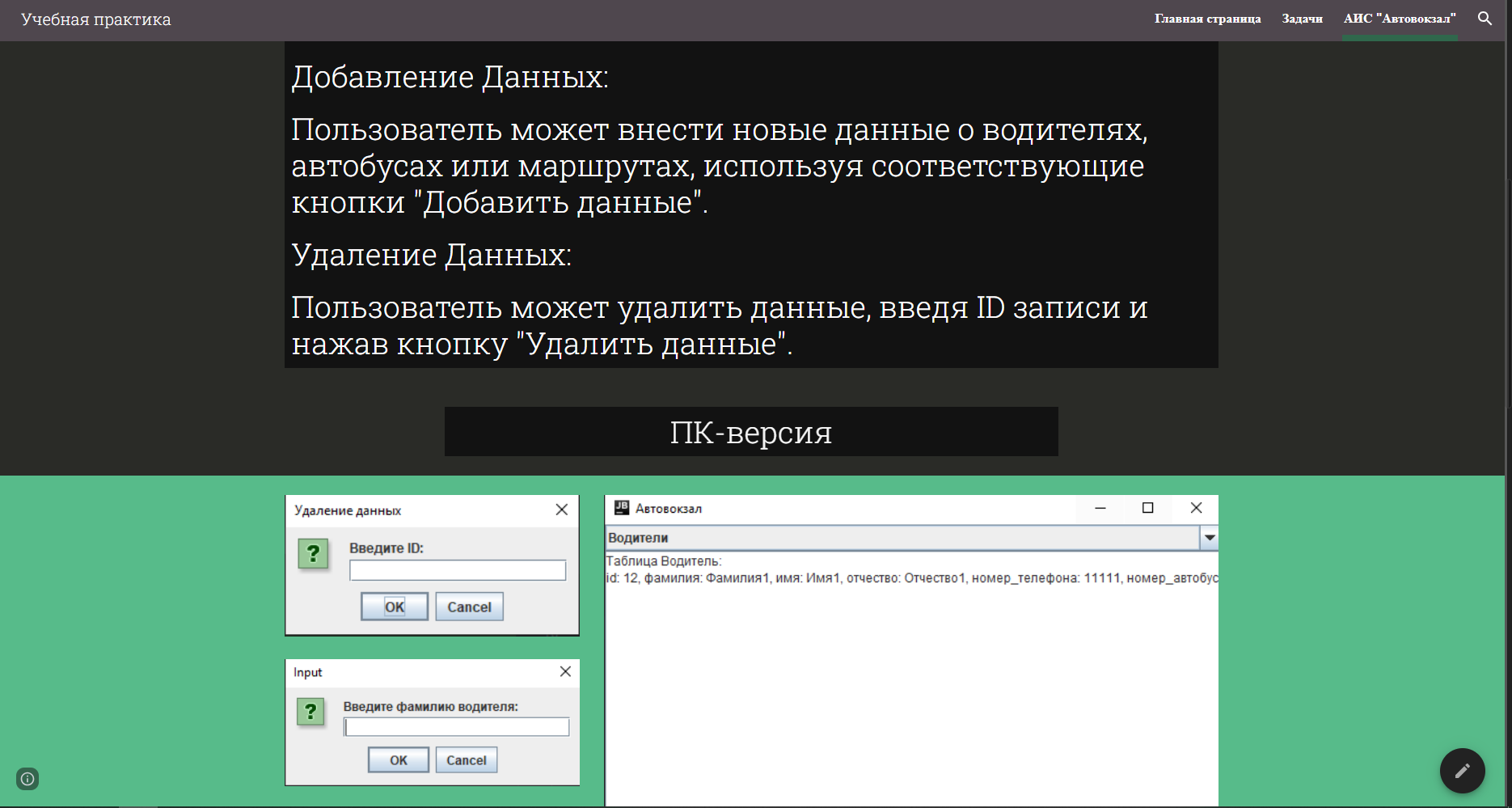


Рисунок 44 – страница АИС «Автовокзал» 2

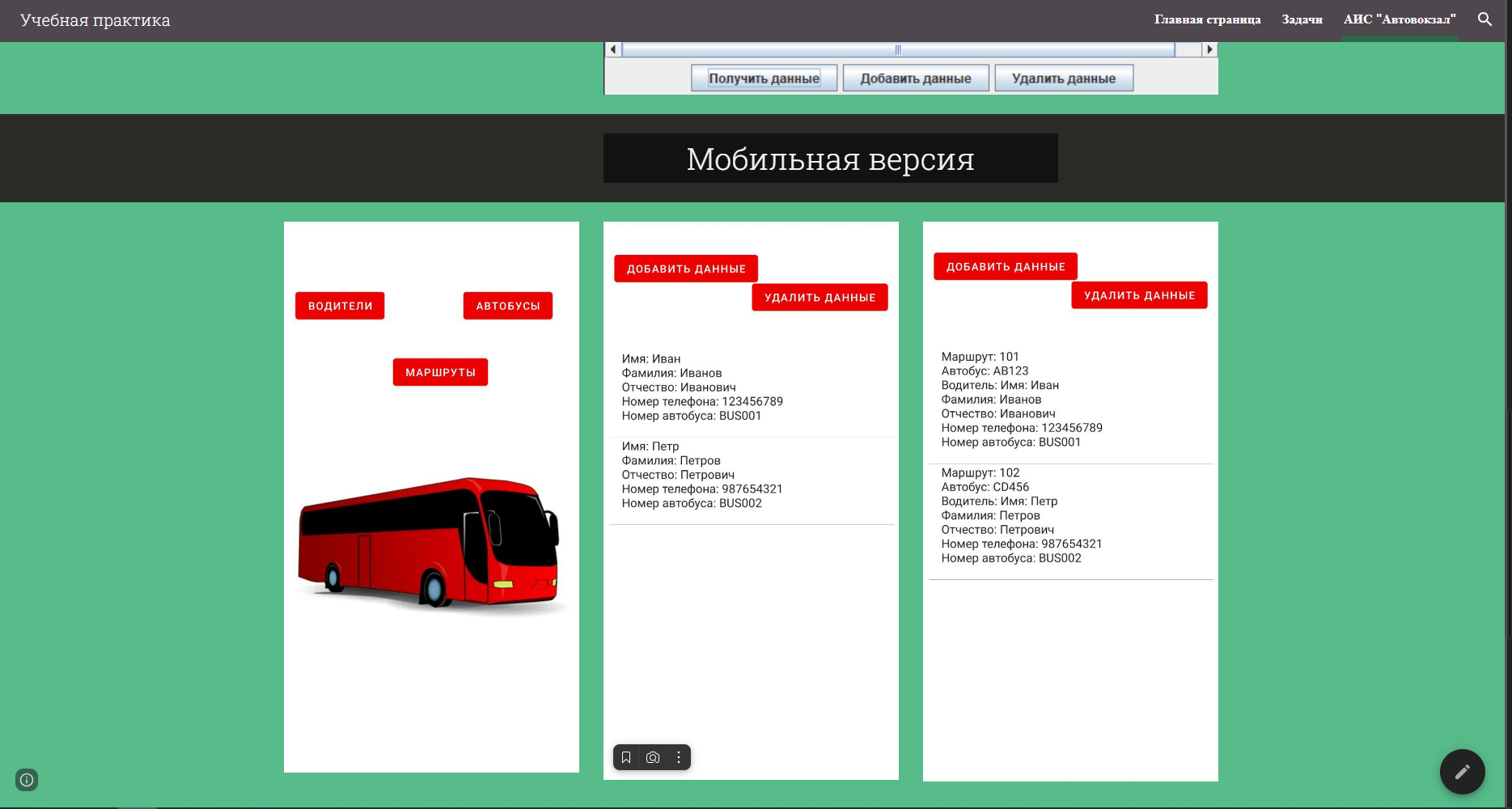


Рисунок 45 – страница АИС «Автовокзал» 3

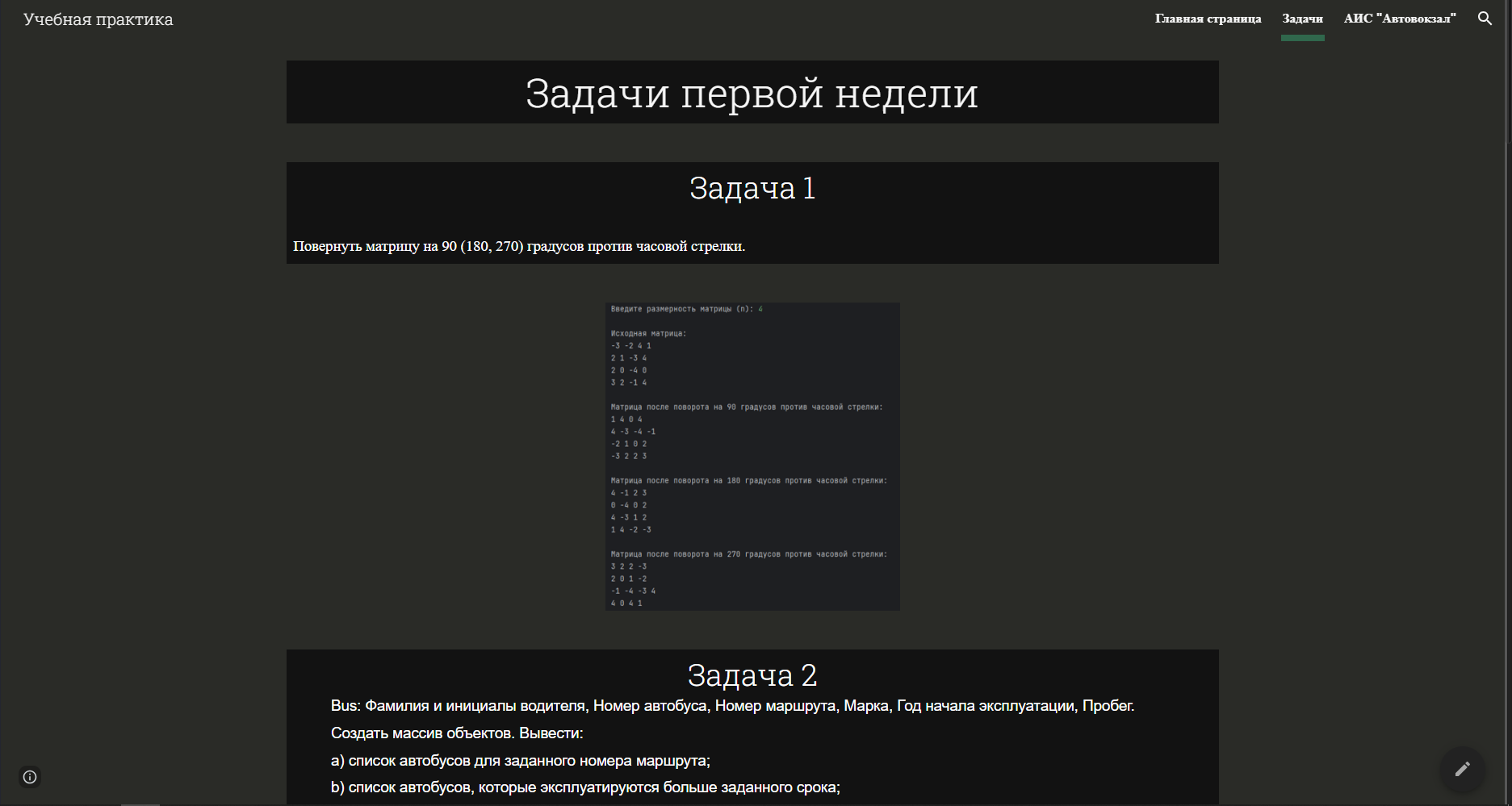


Рисунок - страница Задачи

# 5 Руководство пользователя

Руководство пользователя для АИС «Автовокзал»:

1. Запуск приложения:
   * для запуска приложения выполните метод main в классе AutostationGUI.
   * после запуска появится окно с авторизацией.
   * после авторизации появится окно с названием "Автовокзал".
2. Выбор данных:
   * выберите тип данных (Водители, Автобусы, Маршруты) с помощью выпадающего списка.
3. Получение данных:
   * чтобы получить данные, нажмите кнопку "Получить данные". Полученные данные отобразятся в текстовой области.
4. Добавление данных:
   * нажмите кнопку "Добавить данные".
   * в появившемся диалоговом окне введите необходимые данные в соответствии с выбранным типом данных:
     + - для "Водителей": фамилию, имя, отчество, номер телефона и номер автобуса.
       - для "Автобусов": ID водителя, номер маршрута и номер транспортного средства (автобуса).
       - для "Маршрутов": название маршрута, тип маршрута, количество остановок и ID автобуса.
5. Удаление данных:
   * нажмите кнопку "Удалить данные".
   * в появившемся диалоговом окне введите ID записи, которую вы хотите удалить, и нажмите "ОК".
6. Фоновые операции:
   * все операции выполняются в фоновом режиме, что позволяет избежать блокировки пользовательского интерфейса при выполнении длительных операций.
7. Сообщения об успешных/неуспешных операциях:
   * после завершения операций (добавление, удаление) появятся сообщения об успешном выполнении или ошибке.
8. Выход из приложения:
   * закройте окно приложения для завершения работы.

Примечание:

* для корректной работы приложения необходимо наличие PostgreSQL-сервера, а также базы данных "Avtovokzal1", пользователя "user1" с паролем "1".
* в случае возникновения ошибок при выполнении операций, проверьте консольное окно наличия сообщений об ошибках.

Обратите внимание:

Приложение предоставляет базовый текстовый интерфейс для взаимодействия с базой данных и может потребовать доработки для улучшения пользовательского опыта.

# Заключение

В ходе выполнения учебной практики мной была разработана и успешно реализована автоматизированная информационная система (АИС) под названием "Автовокзал". Эта система представляет собой комплексное программное решение, доступное как на персональных компьютерах, так и на мобильных устройствах.

Основными этапами выполнения проекта были проведение необходимых тестов, разработка технического задания и создание руководства пользователя. Для более наглядного представления работы программы, были разработаны все необходимые схемы и диаграммы, отражающие реальные данные и содержащие актуальную информацию о функциональности программы.

# Список использованных источников

1. Java. Swing. Основы: <https://pro-prof.com/forums/topic/java-swing-основы>
2. Java AWT Tutorial: <https://www.javatpoint.com/java-awt>
3. Package java.awt: <https://docs.oracle.com/javase/7/docs/api/java/awt/package-summary.html>
4. Что такое блок-схема и как ее создать?: <https://www.lucidchart.com/pages/ru/блок-схема>
5. Что такое ER-диаграмма и как ее создать?: <https://www.lucidchart.com/pages/ru/erd-диаграмма>
6. Простое руководство по использованию классовых диаграмм UML | Учебное пособие по использованию классовых диаграмм : <https://creately.com/blog/ru/uncategorized-ru/учебное-пособие-по-диаграмме-классов/>
7. Функциональная часть АИС: <https://geratekaterina.jimdofree.com/дисциплины/уфис/состав-и-структура-ис-функциональная-часть/>
8. 10 лучших инструментов для веб-разработки в 2023 году: <https://vc.ru/dev/694289-10-luchshih-instrumentov-dlya-veb-razrabotki-v-2023-godu>
9. Чек-лист: как составить ТЗ, чтобы разработчики вас поняли: <https://vc.ru/u/1006729-kozhindev/450890-chek-list-kak-sostavit-tz-chtoby-razrabotchiki-vas-ponyali>
10. Как написать руководство пользователя программы или сайта - инструкции, советы, помощь, программное обеспечение: <https://www.drexplain.ru/articles/kak_napisat_rukovodstvo_polzovatelya_k_programmnomu_obespecheniyu_ili_web_servisu/>

# Приложение

Листнинг к АИС «Автовокзал»

public class AutostationGUI extends JFrame {  
 private JTextArea textArea;  
 private JComboBox<String> dataTypeComboBox;  
  
 public AutostationGUI() {  
 setTitle("Автовокзал");  
 setSize(600, 400);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
  
 textArea = new JTextArea();  
 textArea.setEditable(false);  
  
 dataTypeComboBox = new JComboBox<>(new String[]{"Водители", "Автобусы", "Маршруты"});  
 dataTypeComboBox.setSelectedIndex(0); // По умолчанию выбраны водители  
  
 JScrollPane scrollPane = new JScrollPane(textArea);  
  
 JPanel buttonPanel = new JPanel(new FlowLayout());  
  
 // Кнопка для запуска операции в фоновом режиме - Получить данные  
 JButton fetchDataButton = new JButton("Получить данные");  
 fetchDataButton.addActionListener(e -> fetchDataInBackground());  
 buttonPanel.add(fetchDataButton);  
  
 // Кнопка для запуска операции в фоновом режиме - Добавить данные  
 JButton addDataButton = new JButton("Добавить данные");  
 addDataButton.addActionListener(e -> addDataInBackground());  
 buttonPanel.add(addDataButton);  
 // Кнопка для запуска операции в фоновом режиме - Удалить данные  
 JButton deleteDataButton = new JButton("Удалить данные");  
 deleteDataButton.addActionListener(e -> showDeleteDataDialog());  
 buttonPanel.add(deleteDataButton);  
 add(dataTypeComboBox, BorderLayout.*NORTH*);  
 add(scrollPane, BorderLayout.*CENTER*);  
 add(buttonPanel, BorderLayout.*SOUTH*);  
 }  
 private void showDeleteDataDialog() {  
 JTextField idTextField = new JTextField();  
 Object[] message = {"Введите ID:", idTextField};  
  
 int option = JOptionPane.*showConfirmDialog*(null, message, "Удаление данных", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 deleteDataInBackground(idTextField.getText());  
 }  
 }  
 private void fetchDataInBackground() {  
 SwingWorker<Void, String> worker = new SwingWorker<Void, String>() {  
 @Override  
 protected Void doInBackground() {  
 fetchData();  
 return null;  
 }  
  
 @Override  
 protected void done() {  
 // Вызывается после завершения фоновой операции  
  
 }  
 };  
  
 worker.execute();  
 }  
 private void addDataInBackground() {  
 SwingWorker<Void, String> worker = new SwingWorker<Void, String>() {  
 @Override  
 protected Void doInBackground() {  
 addData();  
 return null;  
 }  
  
 @Override  
 protected void done() {  
 // Вызывается после завершения фоновой операции  
  
 }  
 };  
  
 worker.execute();  
 }  
 private void deleteDataInBackground(String idToDelete) {  
 SwingWorker<Void, String> worker = new SwingWorker<Void, String>() {  
 @Override  
 protected Void doInBackground() {  
 deleteData(idToDelete);  
 return null;  
 }  
  
 @Override  
 protected void done() {  
 // Вызывается после завершения фоновой операции  
 }  
 };  
  
 worker.execute();  
 }  
 private void fetchData() {  
 String url = "jdbc:postgresql://localhost:5432/Avtovokzal1";  
 String user = "user1";  
 String password = "1";  
  
 try (Connection connection = DriverManager.*getConnection*(url, user, password)) {  
 if (connection != null) {  
 System.*out*.println("Соединение с базой данных установлено.");  
  
 // Вывод данных в зависимости от выбранного типа данных  
 String selectedDataType = (String) dataTypeComboBox.getSelectedItem();  
 switch (selectedDataType) {  
 case "Водители":  
 clearTextArea();  
 printDriverData(connection);  
 break;  
 case "Автобусы":  
 clearTextArea();  
 printBusData(connection);  
 break;  
 case "Маршруты":  
 clearTextArea();  
 printRouteData(connection);  
 break;  
 default:  
 break;  
 }  
  
 System.*out*.println("Данные успешно получены.");  
 } else {  
 System.*out*.println("Не удалось установить соединение с базой данных.");  
 }  
 } catch (SQLException e) {  
 System.*out*.println("Ошибка при работе с базой данных: " + e.getMessage());  
 }  
 }  
  
 private void printDriverData(Connection connection) throws SQLException {  
 appendTextToTextArea("Таблица Водитель:");  
 printTableData(connection, "Водитель");  
 }  
  
 private void printBusData(Connection connection) throws SQLException {  
 appendTextToTextArea("Таблица Автобус:");  
 printTableData(connection, "Автобус");  
 }  
  
 private void printRouteData(Connection connection) throws SQLException {  
 appendTextToTextArea("Таблица Маршрут:");  
 printTableData(connection, "Маршрут");  
 }  
  
 private void printTableData(Connection connection, String tableName) throws SQLException {  
 String query = "SELECT \* FROM " + tableName;  
  
 try (PreparedStatement preparedStatement = connection.prepareStatement(query);  
 ResultSet resultSet = preparedStatement.executeQuery()) {  
  
 ResultSetMetaData metaData = resultSet.getMetaData();  
 int columnCount = metaData.getColumnCount();  
  
 clearTextArea();  
 appendTextToTextArea("Таблица " + tableName + ":");  
  
 while (resultSet.next()) {  
 StringBuilder rowData = new StringBuilder();  
 for (int i = 1; i <= columnCount; i++) {  
 String columnName = metaData.getColumnName(i);  
 String columnValue = resultSet.getString(i);  
 rowData.append(columnName).append(": ").append(columnValue).append(", ");  
 }  
  
 rowData.delete(rowData.length() - 2, rowData.length());  
  
 appendTextToTextArea(rowData.toString());  
 }  
 }  
 }  
  
  
 private void appendTextToTextArea(String text) {  
 SwingUtilities.*invokeLater*(() -> {  
 textArea.append(text + "\n");  
 textArea.setCaretPosition(textArea.getDocument().getLength());  
 });  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 AutostationGUI autostationGUI = new AutostationGUI();  
 autostationGUI.setVisible(true);  
 });  
 }  
 private void clearTextArea() {  
 SwingUtilities.*invokeLater*(() -> {  
 textArea.setText("");  
 });  
 }  
 private void addData() {  
 String url = "jdbc:postgresql://localhost:5432/Avtovokzal1";  
 String user = "user1";  
 String password = "1";  
  
 try (Connection connection = DriverManager.*getConnection*(url, user, password)) {  
 if (connection != null) {  
 System.*out*.println("Соединение с базой данных установлено.");  
  
 // Вывод данных в зависимости от выбранного типа данных  
 String selectedDataType = (String) dataTypeComboBox.getSelectedItem();  
  
 switch (selectedDataType) {  
 case "Водители":  
 addDriverData(connection);  
 break;  
 case "Автобусы":  
 addBusData(connection);  
 break;  
 case "Маршруты":  
 addRouteData(connection);  
 break;  
 default:  
 break;  
 }  
  
 System.*out*.println("Данные успешно добавлены.");  
 } else {  
 System.*out*.println("Не удалось установить соединение с базой данных.");  
 }  
 } catch (SQLException e) {  
 System.*out*.println("Ошибка при работе с базой данных: " + e.getMessage());  
 }  
 }  
  
 private void addDriverData(Connection connection) {  
 try {  
 String lastName = JOptionPane.*showInputDialog*("Введите фамилию водителя:");  
 String firstName = JOptionPane.*showInputDialog*("Введите имя водителя:");  
 String middleName = JOptionPane.*showInputDialog*("Введите отчество водителя:");  
 String phoneNumber = JOptionPane.*showInputDialog*("Введите номер телефона водителя:");  
 String busNumber = JOptionPane.*showInputDialog*("Введите номер автобуса:");  
  
 String insertQuery = "INSERT INTO Водитель (фамилия, имя, отчество, номер\_телефона, номер\_автобуса) " +  
 "VALUES (?, ?, ?, ?, ?)";  
  
 try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {  
 preparedStatement.setString(1, lastName);  
 preparedStatement.setString(2, firstName);  
 preparedStatement.setString(3, middleName);  
 preparedStatement.setString(4, phoneNumber);  
 preparedStatement.setString(5, busNumber);  
  
 preparedStatement.executeUpdate();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Данные водителя успешно добавлены.");  
 } catch (SQLException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка при добавлении данных водителя: " + e.getMessage());  
 }  
 }  
  
 private void addBusData(Connection connection) {  
 try {  
 int driver = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите id водителя автобуса:"));  
 String routeNumber = JOptionPane.*showInputDialog*("Введите номер маршрута:");  
 String vehicleNumber = JOptionPane.*showInputDialog*("Введите номер транспортного средства (автобуса):");  
  
 String insertQuery = "INSERT INTO Автобус (водитель, номер\_маршрута, номер\_транспортного\_средства) " +  
 "VALUES (?, ?, ?)";  
  
 try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {  
 preparedStatement.setInt(1, driver);  
 preparedStatement.setString(2, routeNumber);  
 preparedStatement.setString(3, vehicleNumber);  
  
 preparedStatement.executeUpdate();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Данные автобуса успешно добавлены.");  
 } catch (SQLException | NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка при добавлении данных автобуса: " + e.getMessage());  
 }  
 }  
  
  
  
 private void addRouteData(Connection connection) {  
 try {  
 String routeName = JOptionPane.*showInputDialog*("Введите название маршрута:");  
 String routeType = JOptionPane.*showInputDialog*("Введите тип маршрута (городской/междугородний):");  
 int stopsCount = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите количество остановок:"));  
 int busId = Integer.*parseInt*(JOptionPane.*showInputDialog*("Введите ID автобуса:"));  
  
 String insertQuery = "INSERT INTO Маршрут (название\_маршрута, тип\_маршрута, количество\_остановок, автобус) " +  
 "VALUES (?, ?, ?, ?)";  
  
 try (PreparedStatement preparedStatement = connection.prepareStatement(insertQuery)) {  
 preparedStatement.setString(1, routeName);  
 preparedStatement.setString(2, routeType);  
 preparedStatement.setInt(3, stopsCount);  
 preparedStatement.setInt(4, busId);  
  
 preparedStatement.executeUpdate();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Данные маршрута успешно добавлены.");  
 } catch (SQLException | NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка при добавлении данных маршрута: " + e.getMessage());  
 }  
 }  
  
  
 private void deleteData(String idToDelete) {  
 String url = "jdbc:postgresql://localhost:5432/Avtovokzal1";  
 String user = "user1";  
 String password = "1";  
  
 try (Connection connection = DriverManager.*getConnection*(url, user, password)) {  
 if (connection != null) {  
 System.*out*.println("Соединение с базой данных установлено.");  
  
 // Удаление данных в зависимости от выбранного типа данных  
 String selectedDataType = (String) dataTypeComboBox.getSelectedItem();  
 switch (selectedDataType) {  
 case "Водители":  
 deleteDriverData(connection, idToDelete);  
 break;  
 case "Автобусы":  
 deleteBusData(connection, idToDelete);  
 break;  
 case "Маршруты":  
 deleteRouteData(connection, idToDelete);  
 break;  
 default:  
 // Неизвестный тип данных  
 break;  
 }  
  
 System.*out*.println("Данные успешно удалены.");  
 } else {  
 System.*out*.println("Не удалось установить соединение с базой данных.");  
 }  
 } catch (SQLException e) {  
 System.*out*.println("Ошибка при работе с базой данных: " + e.getMessage());  
 }  
 }  
  
 // Метод удаления данных по ID из таблицы Водитель  
 private void deleteDriverData(Connection connection, String idToDelete) {  
 try {  
 int id = Integer.*parseInt*(idToDelete);  
  
 String deleteQuery = "DELETE FROM Водитель WHERE id = ?";  
 try (PreparedStatement preparedStatement = connection.prepareStatement(deleteQuery)) {  
 preparedStatement.setInt(1, id);  
 preparedStatement.executeUpdate();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Данные водителя с ID " + id + " успешно удалены.");  
 } catch (NumberFormatException | SQLException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка при удалении данных водителя: " + e.getMessage());  
 }  
 }  
  
 // Метод удаления данных по ID из таблицы Автобус  
 private void deleteBusData(Connection connection, String idToDelete) {  
 try {  
 int id = Integer.*parseInt*(idToDelete);  
  
 String deleteQuery = "DELETE FROM Автобус WHERE id = ?";  
 try (PreparedStatement preparedStatement = connection.prepareStatement(deleteQuery)) {  
 preparedStatement.setInt(1, id);  
 preparedStatement.executeUpdate();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Данные автобуса с ID " + id + " успешно удалены.");  
 } catch (NumberFormatException | SQLException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка при удалении данных автобуса: " + e.getMessage());  
 }  
 }  
 // Метод удаления данных по ID из таблицы Маршрут  
 private void deleteRouteData(Connection connection, String idToDelete) {  
 try {  
 int id = Integer.*parseInt*(idToDelete);  
  
 String deleteQuery = "DELETE FROM Маршрут WHERE id = ?";  
 try (PreparedStatement preparedStatement = connection.prepareStatement(deleteQuery)) {  
 preparedStatement.setInt(1, id);  
 preparedStatement.executeUpdate();  
 }  
  
 JOptionPane.*showMessageDialog*(null, "Данные маршрута с ID " + id + " успешно удалены.");  
 } catch (NumberFormatException | SQLException e) {  
 JOptionPane.*showMessageDialog*(null, "Ошибка при удалении данных маршрута: " + e.getMessage());  
 }  
 }  
  
}