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 «09» сентября 2024 г. по «09» ноября 2024 г.

Место практики ГБУ КО ПОО «КИТиС»

|  |  |
| --- | --- |
| Выполнил: | студент 4 курса,  группы ИСп 21-2к  Суменков Максим Николаевич  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Проверила: | Большакова-Стрекалова Анна Викторовна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Калининград, 2024

Содержание

[Введение 3](#_Toc182479736)

[Разработка программных модулей 4](#_Toc182479737)

[1.1 Задание 1 4](#_Toc182479738)

[1.2 Задание 2 7](#_Toc182479739)

[1.3 Задание 3 11](#_Toc182479740)

[1.4 Задание 4 13](#_Toc182479741)

[1.5 Задание 5 16](#_Toc182479742)

[1.7 Задание 7 18](#_Toc182479743)

[1.8 Задание 8 20](#_Toc182479744)

[1.9 Задание 9 22](#_Toc182479745)

[Разработка базы данных 29](#_Toc182479746)

[ER-Диаграмма 35](#_Toc182479747)

[Разработка мобильного приложения 36](#_Toc182479748)

[Разработка сайта 38](#_Toc182479749)

# Введение

Этот отчет по УП ПМ01 охватывает дисциплины: разработка программных модулей, мобильных приложений, 1С-программирование, поддержка и тестирование программных модулей, системное программирование.

Цели практики:

- Углубление теоретических знаний: Применение полученных теоретических знаний для лучшего освоения изучаемых дисциплин.

- Развитие практических навыков: Создание реальных программных продуктов, таких как АИС, мобильные приложения и сайты.

- Проверка и улучшение решений: Тестирование и доработка созданных программных модулей.

- Создание комплексных решений: Разработка и документирование проектов, включающих программный код и руководство пользователя.

Результаты практики:

- Укрепление и расширение знаний в соответствующих областях.

- Получение практического опыта в разработке программного обеспечения.

- Развитие навыков анализа, проектирования и реализации программных решений.

- Овладение созданием качественных и удобных для пользователей продуктов.

# Разработка программных модулей

## 1.1 Задание 1



import java.util.Scanner;

public class LocalMaximumFinder {

// Метод для проверки, является ли элемент локальным максимумом

public static boolean isLocalMaximum(int[][] matrix, int row, int col) {

int current = matrix[row][col];

// Проверяем соседей вокруг элемента

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

for (int j = -1; j <= 1; j++) {

if (i == 0 && j == 0) continue; // Пропускаем сам элемент

int newRow = row + i;

int newCol = col + j;

// Проверяем, что сосед находится в пределах матрицы

if (newRow >= 0 && newRow < matrix.length && newCol >= 0 && newCol < matrix[0].length) {

if (matrix[newRow][newCol] >= current) {

return false; // Если хотя бы один сосед больше или равен, то это не локальный максимум

}

}

}

}

return true; // Все соседи меньше, значит это локальный максимум

}

// Метод для поиска наименьшего среди локальных максимумов

public static int findSmallestLocalMaximum(int[][] matrix) {

int minLocalMax = Integer.MAX\_VALUE;

// Проходим по всем элементам матрицы

for (int row = 0; row < matrix.length; row++) {

for (int col = 0; col < matrix[row].length; col++) {

if (isLocalMaximum(matrix, row, col)) {

// Обновляем минимальный локальный максимум

minLocalMax = Math.min(minLocalMax, matrix[row][col]);

}

}

}

// Если локальных максимумов не найдено, возвращаем специальное значение

if (minLocalMax == Integer.MAX\_VALUE) {

System.out.println("Локальные максимумы не найдены.");

return -1;

}

return minLocalMax;

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Ввод размера матрицы

System.out.print("Введите количество строк: ");

int rows = scanner.nextInt();

System.out.print("Введите количество столбцов: ");

int cols = scanner.nextInt();

int[][] matrix = new int[rows][cols];

// Ввод элементов матрицы

System.out.println("Введите элементы матрицы:");

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

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

matrix[i][j] = scanner.nextInt();

}

}

// Поиск наименьшего среди локальных максимумов

int result = findSmallestLocalMaximum(matrix);

if (result != -1) {

System.out.println("Наименьший среди локальных максимумов: " + result);

}

}

}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| matrix | int[][] | Двумерный массив, представляющий матрицу, в которой ведется поиск локальных максимумов. |
| rows | int | Количество строк в матрице, вводимое пользователем. |
| cols | int | Количество столбцов в матрице, вводимое пользователем. |
| row | int | Индекс текущей строки, используемый при переборе элементов матрицы. |
| col | int | Индекс текущего столбца, используемый при переборе элементов матрицы. |
| current | int | Текущий элемент матрицы, который проверяется на локальный максимум. |
| newRow, newCol | int | Координаты соседнего элемента, используемые для проверки, находится ли сосед в пределах матрицы. |
| minLocalMax | int | Наименьший найденный локальный максимум. Изначально принимает значение Integer.MAX\_VALUE, чтобы любой реальный элемент матрицы оказался меньше. |
| result | int | Результат поиска наименьшего локального максимума. Если локальные максимумы не найдены, результат равен -1. |

Таблица 1 – задание 1

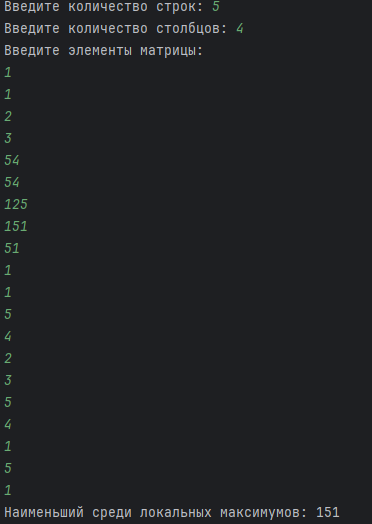
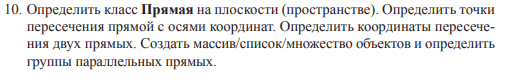


Рисунок 1 – задание 1

## 1.2 Задание 2



import java.util.ArrayList;  
import java.util.List;  
  
class Line {  
 private double A, B, C; // коэффициенты уравнения прямой Ax + By + C = 0  
  
 // Конструктор для создания прямой по коэффициентам A, B и C  
 public Line(double A, double B, double C) {  
 this.A = A;  
 this.B = B;  
 this.C = C;  
 }  
  
 // Метод для нахождения точки пересечения прямой с осью X (когда y = 0)  
 public double[] intersectXAxis() {  
 if (B == 0) { // Если прямая параллельна оси X  
 return null;  
 }  
 double x = -C / A; // при y = 0 уравнение становится Ax + C = 0  
 return new double[] { x, 0 };  
 }  
  
 // Метод для нахождения точки пересечения прямой с осью Y (когда x = 0)  
 public double[] intersectYAxis() {  
 if (A == 0) { // Если прямая параллельна оси Y  
 return null;  
 }  
 double y = -C / B; // при x = 0 уравнение становится By + C = 0  
 return new double[] { 0, y };  
 }  
  
 // Метод для нахождения точки пересечения двух прямых  
 public static double[] intersectLines(Line l1, Line l2) {  
 double determinant = l1.A \* l2.B - l2.A \* l1.B;  
 if (determinant == 0) { // Прямые параллельны или совпадают  
 return null;  
 } else {  
 double x = (l2.C \* l1.B - l1.C \* l2.B) / determinant;  
 double y = (l1.C \* l2.A - l2.C \* l1.A) / determinant;  
 return new double[] { x, y };  
 }  
 }  
  
 // Метод для проверки параллельности двух прямых  
 public static boolean areParallel(Line l1, Line l2) {  
 return (l1.A \* l2.B == l2.A \* l1.B);  
 }  
  
 // Переопределение метода toString для удобного отображения прямой  
 @Override  
 public String toString() {  
 return String.*format*("%.2fx + %.2fy + %.2f = 0", A, B, C);  
 }  
}  
  
public class LineTest {  
 public static void main(String[] args) {  
 // Создаем массив прямых  
 List<Line> lines = new ArrayList<>();  
 lines.add(new Line(2, -1, 3)); // 2x - y + 3 = 0  
 lines.add(new Line(-1, 1, -2)); // -x + y - 2 = 0  
 lines.add(new Line(1, -2, 1)); // x - 2y + 1 = 0  
 lines.add(new Line(2, -1, 5)); // 2x - y + 5 = 0  
  
 // Выводим информацию о пересечении с осями координат для каждой прямой  
 for (Line line : lines) {  
 double[] xIntersect = line.intersectXAxis();  
 double[] yIntersect = line.intersectYAxis();  
  
 System.*out*.println("Прямая: " + line);  
 if (xIntersect != null) {  
 System.*out*.printf("Точка пересечения с осью X: (%.2f, %.2f)%n", xIntersect[0], xIntersect[1]);  
 } else {  
 System.*out*.println("Прямая параллельна оси X.");  
 }  
  
 if (yIntersect != null) {  
 System.*out*.printf("Точка пересечения с осью Y: (%.2f, %.2f)%n", yIntersect[0], yIntersect[1]);  
 } else {  
 System.*out*.println("Прямая параллельна оси Y.");  
 }  
 }  
  
 // Проверка пересечения двух прямых  
 double[] intersection = Line.*intersectLines*(lines.get(0), lines.get(1));  
 if (intersection != null) {  
 System.*out*.printf("Прямые пересекаются в точке: (%.2f, %.2f)%n", intersection[0], intersection[1]);  
 } else {  
 System.*out*.println("Прямые параллельны или совпадают.");  
 }  
  
 // Проверка на параллельность  
 for (int i = 0; i < lines.size(); i++) {  
 for (int j = i + 1; j < lines.size(); j++) {  
 if (Line.*areParallel*(lines.get(i), lines.get(j))) {  
 System.*out*.println("Прямые " + lines.get(i) + " и " + lines.get(j) + " параллельны.");  
 }  
 }  
 }  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| A, B, C | double | Коэффициенты уравнения прямой Ax+By+C=0Ax + By + C = 0Ax+By+C=0. |
| xIntersect | double[] | Координаты точки пересечения прямой с осью Х. |
| yIntersect | double[] | Координаты точки пересечения прямой с осью Y. |
| determinant | double | Определитель для нахождения пересечения двух прямых. |
| intersection | double[] | Координаты точки пересечения двух прямых. Если прямые параллельны или совпадают, переменная принимает значение null. |
| lines | List<Line> | Список объектов класса Line, представляющих несколько прямых. |
| i, j | |  | | --- | |  |  |  | | --- | | int | | Индексы для перебора всех пар прямых в списке. |

Таблица 2 – Задание 2

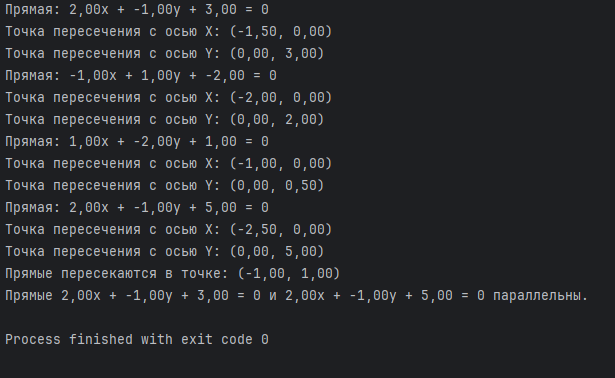


Рисунок 2 – Задание 2

## 1.3 Задание 3



// Класс Крылья  
class Wings {  
 private String type;  
  
 public Wings(String type) {  
 this.type = type;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 // Метод для махания крыльями  
 public void flap() {  
 System.*out*.println("Крылья машут.");  
 }  
}  
  
// Класс Клюв  
class Beak {  
 private String size;  
  
 public Beak(String size) {  
 this.size = size;  
 }  
  
 public String getSize() {  
 return size;  
 }  
  
 // Метод для клевания  
 public void peck() {  
 System.*out*.println("Клюв клюет пищу.");  
 }  
}  
  
// Класс Птица  
class Bird {  
 private Wings wings;  
 private Beak beak;  
  
 public Bird(Wings wings, Beak beak) {  
 this.wings = wings;  
 this.beak = beak;  
 }  
  
 // Метод для полета  
 public void fly() {  
 System.*out*.println("Птица летит с крыльями типа " + wings.getType() + ".");  
 wings.flap();  
 }  
  
 // Метод для посадки  
 public void land() {  
 System.*out*.println("Птица садится.");  
 }  
  
 // Метод для питания  
 public void eat() {  
 System.*out*.println("Птица питается, используя клюв размера " + beak.getSize() + ".");  
 beak.peck();  
 }  
  
 // Метод для атаки  
 public void attack() {  
 System.*out*.println("Птица атакует.");  
 beak.peck();  
 }  
}  
  
public class BirdTest {  
 public static void main(String[] args) {  
 // Создание объекта Крылья и Клюв  
 Wings wings = new Wings("большие");  
 Beak beak = new Beak("маленький");  
  
 // Создание объекта Птица с заданными крыльями и клювом  
 Bird bird = new Bird(wings, beak);  
  
 // Демонстрация действий птицы  
 bird.fly(); // Птица летит  
 bird.land(); // Птица садится  
 bird.eat(); // Птица питается  
 bird.attack(); // Птица атакует  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| type | String | Тип крыльев (например, "большие", "маленькие"). |
| size | String | Размер клюва (например, "большой", "маленький"). |
| wings | Wings | Объект, представляющий крылья птицы. |
| beak | Beak | Объект, представляющий клюв птицы. |
| bird | Bird | Объект класса Птица, содержащий крылья и клюв. |

Таблица 3 – Задание 3

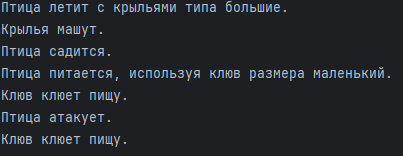


Рисунок 3 – Задание 3

## 1.4 Задание 4



import java.util.ArrayList;  
import java.util.List;  
  
// Основной класс Account  
class Account {  
 private double balance; // Баланс счета  
 private List<Transaction> transactions; // Список всех операций  
  
 // Конструктор для создания аккаунта с начальным балансом  
 public Account(double initialBalance) {  
 this.balance = initialBalance;  
 this.transactions = new ArrayList<>();  
 }  
  
 // Метод для пополнения счета  
 public void deposit(double amount) {  
 balance += amount;  
 transactions.add(new Transaction("Поступление", amount, balance));  
 System.*out*.println("Пополнение счета на " + amount + ". Текущий баланс: " + balance);  
 }  
  
 // Метод для снятия средств  
 public void withdraw(double amount) {  
 if (amount > balance) {  
 System.*out*.println("Недостаточно средств для снятия.");  
 } else {  
 balance -= amount;  
 transactions.add(new Transaction("Снятие", amount, balance));  
 System.*out*.println("Снятие средств на " + amount + ". Текущий баланс: " + balance);  
 }  
 }  
  
 // Метод для осуществления платежа  
 public void payment(double amount) {  
 if (amount > balance) {  
 System.*out*.println("Недостаточно средств для оплаты.");  
 } else {  
 balance -= amount;  
 transactions.add(new Transaction("Платеж", amount, balance));  
 System.*out*.println("Платеж на " + amount + ". Текущий баланс: " + balance);  
 }  
 }  
  
 // Метод для вывода всех операций  
 public void printTransactions() {  
 System.*out*.println("Все операции по счету:");  
 for (Transaction transaction : transactions) {  
 System.*out*.println(transaction);  
 }  
 }  
  
 // Внутренний класс Transaction для хранения данных об операциях  
 private class Transaction {  
 private String type; // Тип операции (Снятие, Платеж, Поступление)  
 private double amount; // Сумма операции  
 private double balanceAfter; // Баланс после операции  
  
 public Transaction(String type, double amount, double balanceAfter) {  
 this.type = type;  
 this.amount = amount;  
 this.balanceAfter = balanceAfter;  
 }  
  
 // Переопределение метода toString для удобного вывода  
 @Override  
 public String toString() {  
 return String.*format*("Операция: %s, Сумма: %.2f, Баланс после операции: %.2f", type, amount, balanceAfter);  
 }  
 }  
}  
  
public class AccountTest {  
 public static void main(String[] args) {  
 // Создаем аккаунт с начальным балансом 1000  
 Account account = new Account(1000);  
  
 // Выполняем различные операции  
 account.deposit(500); // Пополнение счета  
 account.withdraw(200); // Снятие средств  
 account.payment(150); // Платеж  
 account.withdraw(1200); // Попытка снять больше средств, чем есть на счете  
  
 // Вывод всех операций  
 account.printTransactions();  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| balance | double | Текущий баланс счета. |
| transactions | List<Transaction> | Список всех операций, выполненных со счетом. |
| type | String | Тип операции: "Поступление", "Снятие", "Платеж". |
| amount | double | Сумма операции. |
| balanceAfter | double | Баланс после проведения операции. |

Таблица 4 – Задание 4

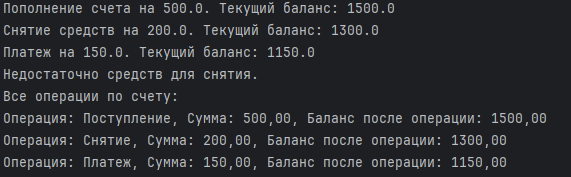
****

Рисунок 4 – Задание 4

## 1.5 Задание 5



import java.io.\*;  
import java.util.regex.\*;  
  
public class RemoveComments {  
 public static void main(String[] args) {  
 // Имя файла с кодом Java  
 String filePath = "PathToFile/YourJavaFile.java"; // Укажите путь к вашему Java файлу  
  
 try {  
 // Чтение файла  
 BufferedReader reader = new BufferedReader(new FileReader(filePath));  
 StringBuilder code = new StringBuilder();  
 String line;  
  
 while ((line = reader.readLine()) != null) {  
 code.append(line).append("\n");  
 }  
 reader.close();  
  
 // Регулярное выражение для удаления комментариев  
 String noComments = *removeComments*(code.toString());  
  
 // Запись обновленного кода обратно в файл  
 BufferedWriter writer = new BufferedWriter(new FileWriter("CleanedFile.java"));  
 writer.write(noComments);  
 writer.close();  
  
 System.*out*.println("Комментарии успешно удалены.");  
 } catch (IOException e) {  
 e.printStackTrace();  
 }  
 }  
  
 // Метод для удаления комментариев  
 public static String removeComments(String code) {  
 // Регулярное выражение для однострочных и многострочных комментариев  
 String regex = "(?s)/\\\*.\*?\\\*/|//.\*";  
  
 // Замена всех комментариев на пустую строку  
 return code.replaceAll(regex, "").trim();  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| filePath | String | Путь к исходному файлу с кодом Java. |
| reader | BufferedReader | Объект для чтения строк из файла. |
| code | StringBuilder | Строка, в которой хранится весь код программы для обработки. |
| regex | String | Регулярное выражение для поиска комментариев. |
| noComments | String | Строка, в которой содержится код программы после удаления комментариев. |
| writer | BufferedWriter | Объект для записи очищенного кода в файл. |

Таблица 5 – Задание 5

## 1.7 Задание 7



public class CylinderSectionCoordinates {  
  
 // Класс для хранения 2D координат  
 static class Point {  
 double x, y;  
  
 Point(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
  
 @Override  
 public String toString() {  
 return "(" + x + ", " + y + ")";  
 }  
 }  
  
 // Метод для вычисления и отображения осевого сечения  
 public static void calculateSection(double height, double radius) {  
 // Координаты начала и конца осевого сечения  
 Point topIntersection = new Point(0, height); // Верхняя точка на оси цилиндра  
 Point bottomIntersection = new Point(0, 0); // Нижняя точка на оси цилиндра  
  
 // Радиус и высота цилиндра  
 System.*out*.println("Радиус цилиндра: " + radius);  
 System.*out*.println("Высота цилиндра: " + height);  
  
 // Точки начала и конца осевого сечения  
 System.*out*.println("Осевое сечение проходит через точки: ");  
 System.*out*.println("Начальная точка (в верхней грани цилиндра): " + topIntersection);  
 System.*out*.println("Конечная точка (в нижней грани цилиндра): " + bottomIntersection);  
  
 // Точки пересечения с верхней и нижней гранями цилиндра  
 System.*out*.println("Точка пересечения с верхней гранью: " + topIntersection);  
 System.*out*.println("Точка пересечения с нижней гранью: " + bottomIntersection);  
 }  
  
 public static void main(String[] args) {  
 double height = 10.0; // Высота цилиндра  
 double radius = 5.0; // Радиус цилиндра  
  
 // Вызываем метод для вычисления осевого сечения  
 *calculateSection*(height, radius);  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| height | double | Высота цилиндра, которая задается в главной программе (расстояние между верхней и нижней гранью цилиндра). |
| radius | double | Радиус цилиндра, это расстояние от оси цилиндра до его края. |
| topIntersection | Point | Координаты точки пересечения осевого сечения с верхней гранью цилиндра. |
| bottomIntersection | Point | Координаты точки пересечения осевого сечения с нижней гранью цилиндра. |
| x | double | Координата по оси X для точек осевого сечения (в данном случае равна 0, так как осевое сечение проходит по оси симметрии цилиндра). |
| y | double | Координата по оси Y для точек осевого сечения (высота от 0 до значения высоты цилиндра). |

Таблица 7 – Задание 7

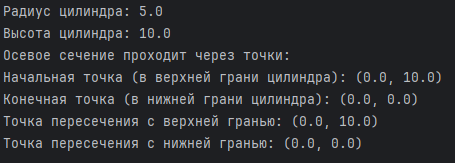
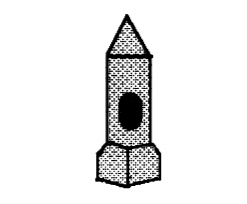


Рисунок 7 – Задание 7

## 1.8 Задание 8



import javax.swing.\*;  
import java.awt.\*;  
  
public class RocketDrawingGraphics extends JPanel {  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
  
 // Преобразование в Graphics2D для более качественной отрисовки  
 Graphics2D g2d = (Graphics2D) g;  
  
 // Устанавливаем цвет для рисования  
 g2d.setColor(Color.*BLACK*);  
  
 // Рисуем верхушку ракеты (треугольник)  
 int[] xPoints = {150, 200, 100}; // Координаты x для треугольника  
 int[] yPoints = {50, 150, 150}; // Координаты y для треугольника  
 g2d.fillPolygon(xPoints, yPoints, 3);  
  
 // Рисуем корпус ракеты (прямоугольник)  
 g2d.fillRect(100, 150, 100, 200);  
  
 // Рисуем окно на корпусе ракеты (овал)  
 g2d.setColor(Color.*WHITE*);  
 g2d.fillOval(125, 200, 50, 50);  
 g2d.setColor(Color.*BLACK*);  
 g2d.drawOval(125, 200, 50, 50);  
  
 // Рисуем нижние опоры ракеты (треугольники)  
 int[] xLeftWing = {100, 80, 100}; // Левая опора  
 int[] yLeftWing = {350, 400, 400};  
 g2d.fillPolygon(xLeftWing, yLeftWing, 3);  
  
 int[] xRightWing = {200, 220, 200}; // Правая опора  
 int[] yRightWing = {350, 400, 400};  
 g2d.fillPolygon(xRightWing, yRightWing, 3);  
  
   
 }  
  
 public static void main(String[] args) {  
 // Создаем окно  
 JFrame frame = new JFrame("Rocket Drawing");  
 RocketDrawingGraphics panel = new RocketDrawingGraphics();  
  
 frame.add(panel);  
 frame.setSize(400, 500); // Размер окна  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*); // Закрытие программы  
 frame.setVisible(true); // Отображаем окно  
 }  
}

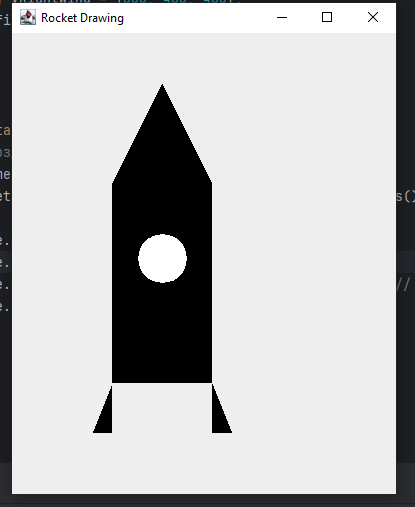


Рисунок 8 – Задание 8

|  |  |  |
| --- | --- | --- |
| **Переменная / Метод** | **Тип данных** | **Описание** |
| g | Graphics | Объект для рисования графики на панели. |
| g2d | Graphics2D | Объект для более качественной отрисовки, который позволяет использовать дополнительные функции, такие как антиалиасинг. |
| xPoints | int[] | Массив координат по оси X для рисования треугольника (верхушки ракеты). |
| yPoints | int[] | Массив координат по оси Y для рисования треугольника (верхушки ракеты). |
| xLeftWing | int[] | Массив координат по оси X для рисования левой опоры ракеты. |
| yLeftWing | int[] | Массив координат по оси Y для рисования левой опоры ракеты. |
| xRightWing | int[] | Массив координат по оси X для рисования правой опоры ракеты. |
| yRightWing | int[] | Массив координат по оси Y для рисования правой опоры ракеты. |
| xBottomWing | int[] | Массив координат по оси X для рисования центральной нижней части ракеты. |
| yBottomWing | int[] | Массив координат по оси Y для рисования центральной нижней части ракеты. |
| paintComponent() | void | Метод для рисования графики на панели. Внутри него происходит отрисовка всех элементов ракеты. |

Таблица 8 – Задание 8

## 1.9 Задание 9

import java.util.Arrays;  
import java.util.Random;  
import java.util.Scanner;  
  
public class MathGame {  
 static final int *SIZE* = 5; // Размер игрового поля (5x5)  
 static final int[] *NUMBERS* = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13}; // Числа от 0 до 13  
 static int[][] *field* = new int[*SIZE*][*SIZE*]; // Поле игры  
 static boolean[][] *filled* = new boolean[*SIZE*][*SIZE*]; // Заполненные клетки  
  
 // Таблица 3: Количество чисел в каждой группе  
 static final int[] *counts* = {4, 4, 4, 4, 4, 3, 3, 3, 3, 2, 2, 2, 2, 1};  
  
 static int[] *remainingCounts* = Arrays.*copyOf*(*counts*, *counts*.length); // Оставшиеся количества для чисел  
 static Scanner *scanner* = new Scanner(System.*in*);  
  
 // Инициализация пустого поля  
 public static void initializeField() {  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j < *SIZE*; j++) {  
 *field*[i][j] = -1; // Изначально все клетки пустые (обозначаем -1)  
 }  
 }  
 }  
  
 // Печать игрового поля  
 public static void printField() {  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j < *SIZE*; j++) {  
 if (*field*[i][j] == -1) {  
 System.*out*.print(".\t"); // Пустая клетка  
 } else {  
 System.*out*.print(*field*[i][j] + "\t"); // Заполненная клетка  
 }  
 }  
 System.*out*.println();  
 }  
 }  
  
 // Проверка, можно ли добавить число (остались ли свободные ячейки для числа)  
 public static boolean canAddNumber(int number) {  
 int index = Arrays.*binarySearch*(*NUMBERS*, number);  
 return *remainingCounts*[index] > 0;  
 }  
  
 // Получение случайного доступного числа  
 public static int getRandomNumber() {  
 Random random = new Random();  
 int index;  
 do {  
 index = random.nextInt(*NUMBERS*.length);  
 } while (*remainingCounts*[index] == 0); // Ищем число, которое можно использовать  
 *remainingCounts*[index]--;  
 return *NUMBERS*[index];  
 }  
  
 // Ход игрока (человека)  
 public static void playerMove() {  
 System.*out*.println("Ваш ход. Введите координаты (строка и столбец):");  
 int row, col;  
 do {  
 System.*out*.print("Строка (0-4): ");  
 row = *scanner*.nextInt();  
 System.*out*.print("Столбец (0-4): ");  
 col = *scanner*.nextInt();  
 } while (row < 0 || row >= *SIZE* || col < 0 || col >= *SIZE* || *filled*[row][col]); // Проверка на корректность хода  
  
 int number;  
 do {  
 System.*out*.print("Введите число от 0 до 13: ");  
 number = *scanner*.nextInt();  
 } while (number < 0 || number > 13 || !*canAddNumber*(number)); // Проверка, можно ли использовать это число  
  
 *field*[row][col] = number;  
 *filled*[row][col] = true; // Отмечаем клетку как заполненную  
 }  
  
 // Ход компьютера  
 public static void computerMove() {  
 System.*out*.println("Ход компьютера:");  
 Random random = new Random();  
 int row, col;  
 do {  
 row = random.nextInt(*SIZE*);  
 col = random.nextInt(*SIZE*);  
 } while (*filled*[row][col]); // Ищем свободную клетку  
  
 int number = *getRandomNumber*(); // Получаем случайное доступное число  
 *field*[row][col] = number;  
 *filled*[row][col] = true; // Отмечаем клетку как заполненную  
 System.*out*.println("Компьютер поставил " + number + " в клетку (" + row + ", " + col + ")");  
 }  
  
 // Подсчет очков за комбинации  
 public static int calculatePoints() {  
 int points = 0;  
  
 // Проверка на четыре одинаковых числа в строке  
 for (int i = 0; i < *SIZE*; i++) {  
 for (int j = 0; j <= *SIZE* - 4; j++) {  
 if (*field*[i][j] == *field*[i][j + 1] && *field*[i][j] == *field*[i][j + 2] && *field*[i][j] == *field*[i][j + 3]) {  
 points += 160; // Награда за 4 одинаковых числа в ряду  
 }  
 }  
 }  
  
 // Можно добавить дополнительные проверки (столбцы, диагонали и т.д.)  
 return points;  
 }  
  
 // Основной игровой цикл  
 public static void playGame() {  
 *initializeField*(); // Инициализация пустого поля  
 boolean isPlayerTurn = true; // Ход игрока  
  
 // Игра продолжается, пока есть пустые клетки  
 while (true) {  
 *printField*(); // Печать текущего состояния поля  
  
 if (isPlayerTurn) {  
 *playerMove*(); // Ход игрока  
 } else {  
 *computerMove*(); // Ход компьютера  
 }  
  
 // Проверяем, остались ли свободные клетки  
 boolean hasEmptyCells = false;  
 for (boolean[] row : *filled*) {  
 for (boolean cell : row) {  
 if (!cell) {  
 hasEmptyCells = true;  
 break;  
 }  
 }  
 }  
 if (!hasEmptyCells) break; // Если нет свободных клеток, игра заканчивается  
  
 // Смена хода  
 isPlayerTurn = !isPlayerTurn;  
 }  
  
 // Подсчет очков после игры  
 int playerPoints = *calculatePoints*();  
 System.*out*.println("Игра завершена.");  
 *printField*(); // Печать финального состояния поля  
 System.*out*.println("Ваши очки: " + playerPoints);  
 }  
  
 public static void main(String[] args) {  
 *playGame*(); // Запуск игры  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| SIZE | final int | Размер игрового поля (5x5). |
| NUMBERS | final int[] | Массив чисел от 0 до 13, которые используются для заполнения поля. |
| field | int[][] | Двумерный массив для хранения чисел, которыми заполняется игровое поле. |
| filled | boolean[][] | Двумерный массив, указывающий, заполнена ли клетка на поле (true - заполнена, false - нет). |
| counts | final int[] | Массив, задающий количество каждого числа от 0 до 13, которое можно использовать в игре. |
| remainingCounts | int[] | Копия массива counts, используемая для отслеживания оставшихся доступных чисел. |
| scanner | Scanner | Объект для ввода данных от игрока (используется для ввода координат и числа). |
| random | Random | Объект для генерации случайных чисел для хода компьютера. |
| row | int | Переменная для хранения координаты строки, вводимой игроком. |
| col | int | Переменная для хранения координаты столбца, вводимой игроком. |
| number | int | Число, которое вводит игрок или выбирает компьютер для размещения на поле. |
| isPlayerTurn | boolean | Флаг, указывающий, чей сейчас ход (true - ход игрока, false - ход компьютера). |
| playerPoints | int | Количество очков игрока, подсчитанных в конце игры. |
| hasEmptyCells | boolean | Флаг для проверки, остались ли на поле пустые клетки (true - есть пустые клетки). |

Таблица 9 – Задание 9

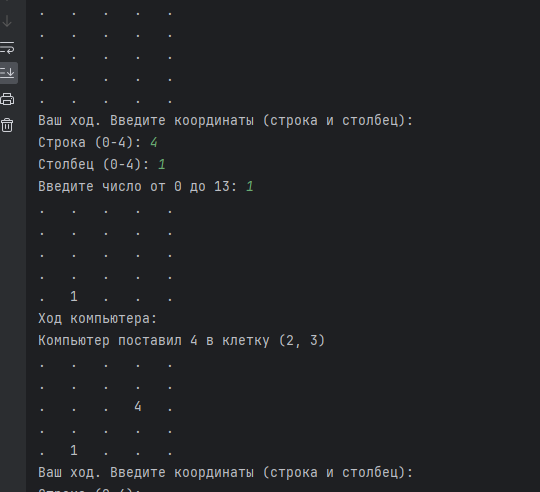


Рисунок 9 – Задание 9

# Разработка базы данных

import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.util.ArrayList;  
import java.util.List;  
  
// Класс, представляющий пассажира  
class Passenger {  
 int id;  
 String name;  
 String surname;  
 String phone;  
  
 public Passenger(int id, String name, String surname, String phone) {  
 this.id = id;  
 this.name = name;  
 this.surname = surname;  
 this.phone = phone;  
 }  
  
 @Override  
 public String toString() {  
 return "ID: " + id + ", Имя: " + name + ", Фамилия: " + surname + ", Телефон: " + phone;  
 }  
}  
  
// Класс для управления "базой данных" пассажиров  
class RailwayDatabase {  
 private List<Passenger> passengers = new ArrayList<>();  
 private int nextId = 1;  
  
 public List<Passenger> getPassengers() {  
 return passengers;  
 }  
  
 public void addPassenger(String name, String surname, String phone) {  
 Passenger passenger = new Passenger(nextId++, name, surname, phone);  
 passengers.add(passenger);  
 }  
  
 public void editPassenger(int id, String newName, String newSurname, String newPhone) {  
 for (Passenger passenger : passengers) {  
 if (passenger.id == id) {  
 passenger.name = newName;  
 passenger.surname = newSurname;  
 passenger.phone = newPhone;  
 return;  
 }  
 }  
 }  
  
 public void deletePassenger(int id) {  
 passengers.removeIf(passenger -> passenger.id == id);  
 }  
}  
  
// Основной класс для графического интерфейса  
public class RailwayStationApp extends JFrame {  
 private RailwayDatabase database = new RailwayDatabase();  
 private DefaultTableModel tableModel;  
 private JTable passengerTable;  
  
 public RailwayStationApp() {  
 setTitle("ЖД Вокзал - База пассажиров");  
 setSize(600, 400);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLocationRelativeTo(null);  
  
 // Настройка таблицы для отображения пассажиров  
 tableModel = new DefaultTableModel(new Object[]{"ID", "Имя", "Фамилия", "Телефон"}, 0);  
 passengerTable = new JTable(tableModel);  
 refreshPassengerTable();  
  
 JScrollPane scrollPane = new JScrollPane(passengerTable);  
 add(scrollPane, BorderLayout.*CENTER*);  
  
 // Панель с кнопками  
 JPanel buttonPanel = new JPanel();  
 JButton addButton = new JButton("Добавить");  
 JButton editButton = new JButton("Редактировать");  
 JButton deleteButton = new JButton("Удалить");  
  
 addButton.addActionListener(this::showAddPassengerDialog);  
 editButton.addActionListener(this::showEditPassengerDialog);  
 deleteButton.addActionListener(this::deleteSelectedPassenger);  
  
 buttonPanel.add(addButton);  
 buttonPanel.add(editButton);  
 buttonPanel.add(deleteButton);  
  
 add(buttonPanel, BorderLayout.*SOUTH*);  
 }  
  
 private void refreshPassengerTable() {  
 tableModel.setRowCount(0);  
 for (Passenger passenger : database.getPassengers()) {  
 tableModel.addRow(new Object[]{passenger.id, passenger.name, passenger.surname, passenger.phone});  
 }  
 }  
  
 private void showAddPassengerDialog(ActionEvent e) {  
 JTextField nameField = new JTextField();  
 JTextField surnameField = new JTextField();  
 JTextField phoneField = new JTextField();  
  
 Object[] fields = {  
 "Имя:", nameField,  
 "Фамилия:", surnameField,  
 "Телефон:", phoneField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Добавить пассажира", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 String name = nameField.getText();  
 String surname = surnameField.getText();  
 String phone = phoneField.getText();  
 database.addPassenger(name, surname, phone);  
 refreshPassengerTable();  
 }  
 }  
  
 private void showEditPassengerDialog(ActionEvent e) {  
 int selectedRow = passengerTable.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(this, "Пожалуйста, выберите пассажира для редактирования.");  
 return;  
 }  
  
 int id = (int) tableModel.getValueAt(selectedRow, 0);  
 Passenger passenger = database.getPassengers().stream().filter(p -> p.id == id).findFirst().orElse(null);  
  
 JTextField nameField = new JTextField(passenger.name);  
 JTextField surnameField = new JTextField(passenger.surname);  
 JTextField phoneField = new JTextField(passenger.phone);  
  
 Object[] fields = {  
 "Имя:", nameField,  
 "Фамилия:", surnameField,  
 "Телефон:", phoneField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Редактировать пассажира", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 passenger.name = nameField.getText();  
 passenger.surname = surnameField.getText();  
 passenger.phone = phoneField.getText();  
 database.editPassenger(id, passenger.name, passenger.surname, passenger.phone);  
 refreshPassengerTable();  
 }  
 }  
  
 private void deleteSelectedPassenger(ActionEvent e) {  
 int selectedRow = passengerTable.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(this, "Пожалуйста, выберите пассажира для удаления.");  
 return;  
 }  
  
 int id = (int) tableModel.getValueAt(selectedRow, 0);  
 database.deletePassenger(id);  
 refreshPassengerTable();  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(() -> {  
 RailwayStationApp app = new RailwayStationApp();  
 app.setVisible(true);  
 });  
 }  
}

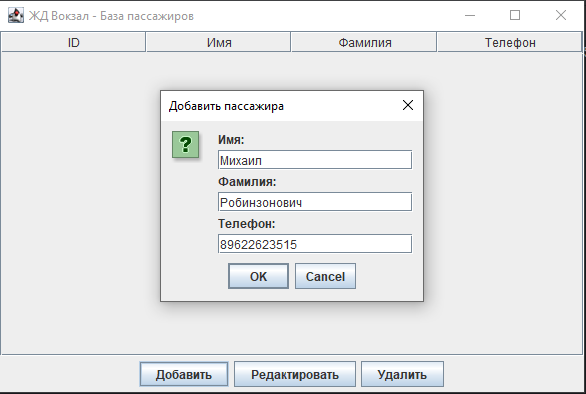


Рисунок 10 – Добавление пассажира в базу данных

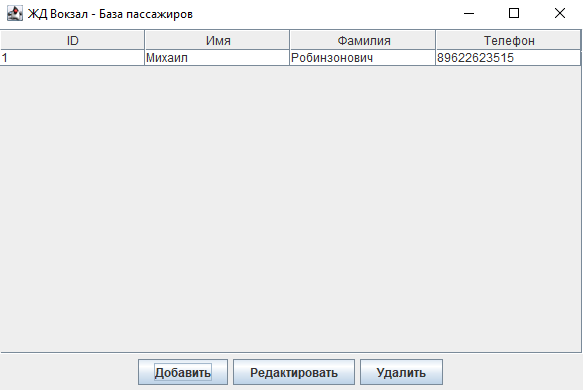


Рисунок 11 – База данных

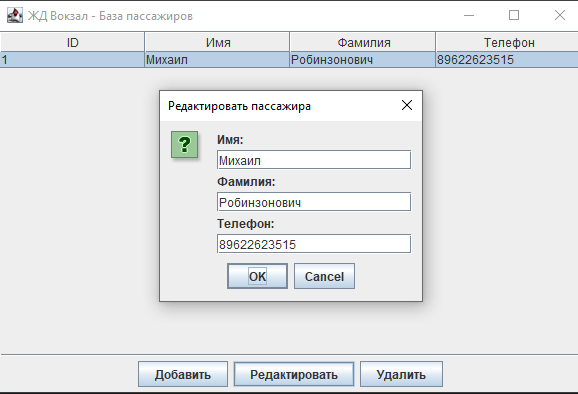


Рисунок 12 – Редактирование пассажира в базе данных

## ER-Диаграмма

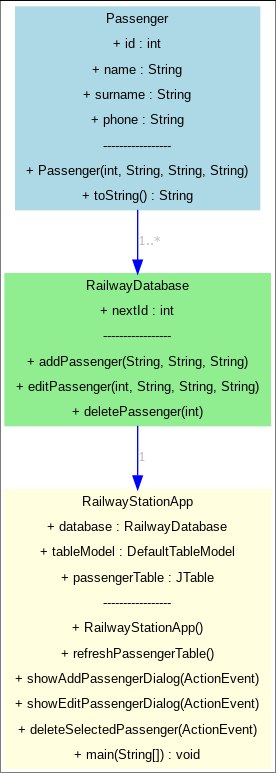


Рисунок 13 – ER-Диаграмма/Диаграмма классов

# Разработка мобильного приложения

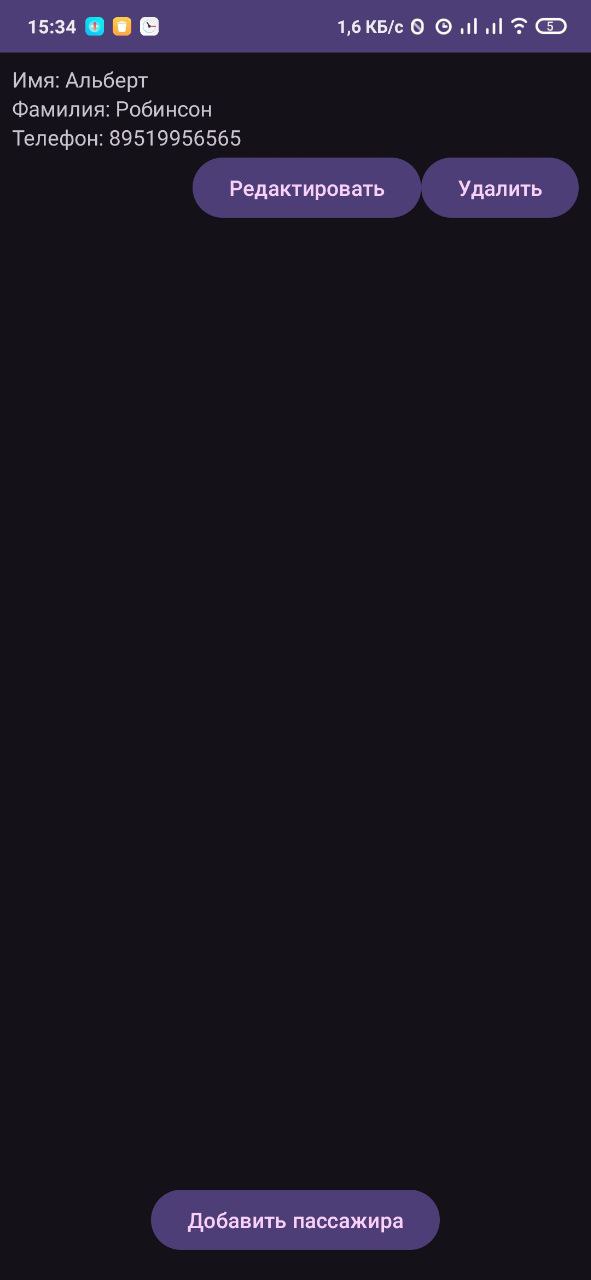


Рисунок 14 – Главное Activity мобильного приложения

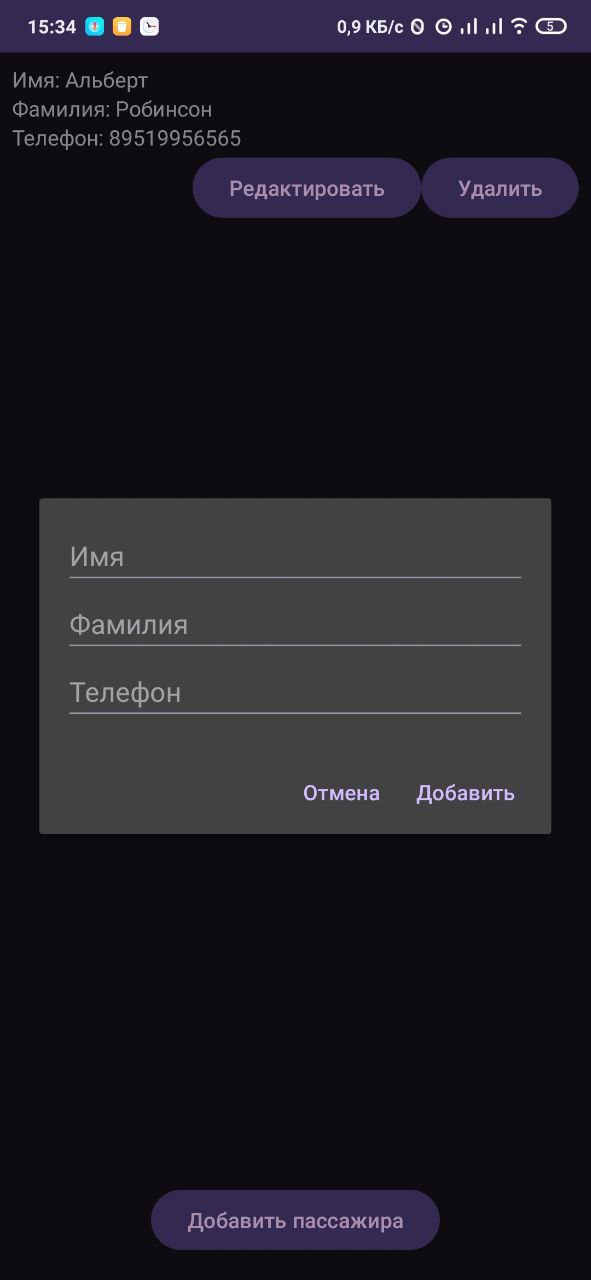


Рисунок 15 – Добавление в мобильное приложение

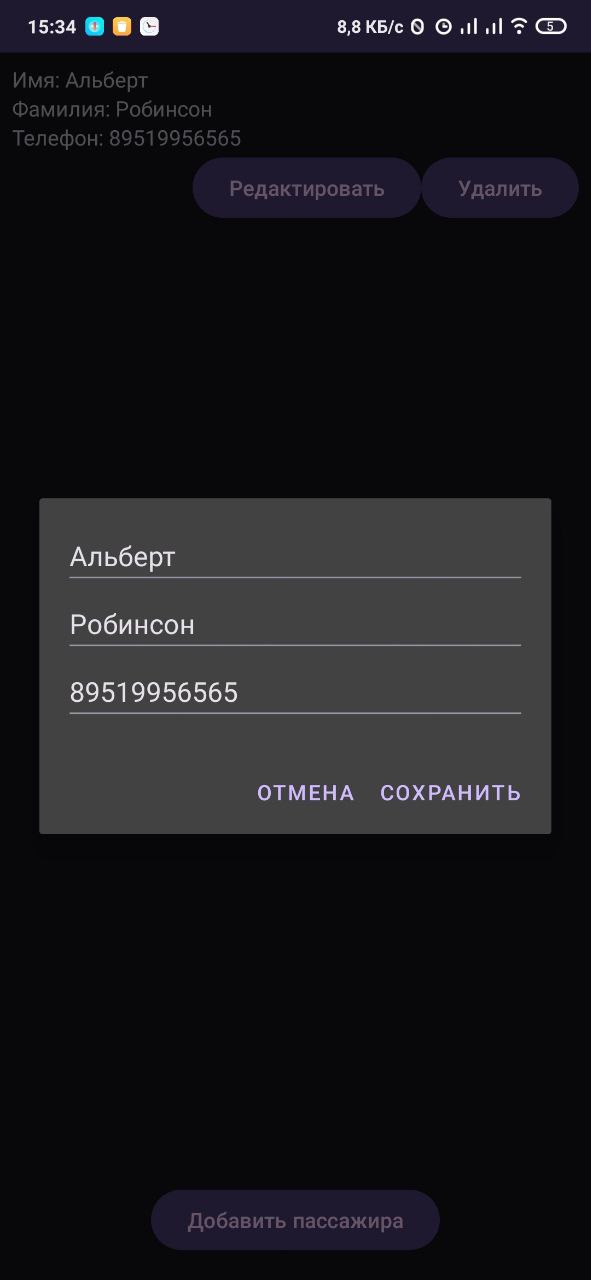


Рисунок 16 – Редактирование в мобильном приложени

## Разработка сайта

https://gv8144.craftum.io/