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  Вишняускис Антон Альвидасович  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись) |
| Проверила: | Большакова-Стрекалова Анна Викторовна  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (оценка)  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  (подпись, дата) |

Калининград, 2024

Содержание

[Министерство образования Калининградской области 1](#_Toc182906140)

[1. Разработка программных модулей 4](#_Toc182906141)

[Задача 1 4](#_Toc182906142)

[Задание 2 6](#_Toc182906143)

[Задание 3 12](#_Toc182906144)

[Задание 4 19](#_Toc182906145)

[Задание 5 24](#_Toc182906146)

[Задание 6 26](#_Toc182906147)

[Задание 7 26](#_Toc182906148)

[Задание 8 31](#_Toc182906149)

[Задание 9 35](#_Toc182906150)

[2. Разработка АИС 41](#_Toc182906151)

[1. Описание и цели системы 41](#_Toc182906152)

[2. Функциональные требования 41](#_Toc182906153)

[3. Нефункциональные требования 42](#_Toc182906154)

[4. Структура системы и описание классов 42](#_Toc182906155)

[5. Алгоритмы и описание действий пользователя 43](#_Toc182906156)

[6. Тестирование системы 43](#_Toc182906157)

[7. Техническое задание (ТЗ) 44](#_Toc182906158)

[8. Руководство пользователя 45](#_Toc182906159)

[9. Протокол тестирования 45](#_Toc182906160)

[2. Разработка базы данных 47](#_Toc182906161)

[3. Разработка мобильного приложения 59](#_Toc182906162)

[4. Разработка 1С 60](#_Toc182906163)

Введение

Данный отчёт служит для представления результатов прохождения учебной практики по программированию. Она состоит из пяти дисциплин разработка программных модулей, разработка мобильных приложений, поддержка и тестирование программных модулей, программирование 1С, системное программирование. Так же практика включает создание собственного WEB-сайта.

Цели:

Расширить свои познания и улучшить практические навыки по дисциплинам, создать собственный сайт и решить поставленные задачи для проверки усвоения материала дисциплин.

# Разработка программных модулей

## Задача 1

Найти число, в котором число различных цифр минимально. Если таких чисел несколько, найти первое из них

|  |  |  |
| --- | --- | --- |
| month | int | Введенное пользователем число, которое должно быть в диапазоне от 1 до 12. |
| months | String[] | Массив строк, содержащий названия месяцев в году. Индексы массива соответствуют номерам месяцев. |
| scanner | Scanner | Объект класса Scanner, используется для ввода данных с консоли. |
| input | String | Строка для временного хранения введенных данных до их конвертации в целое число. |
| month | int | Введенное пользователем число, которое должно быть в диапазоне от 1 до 12. |
| months | String[] | Массив строк, содержащий названия месяцев в году. Индексы массива соответствуют номерам месяцев. |
| scanner | Scanner | Объект класса Scanner, используется для ввода данных с консоли. |
| input | String | Строка для временного хранения введенных данных до их конвертации в целое число. |
| month | int | Введенное пользователем число, которое должно быть в диапазоне от 1 до 12. |

Код:   
import java.util.Scanner;  
  
public class MonthFinder {  
  
 public static void main(String[] args) {  
 // Создаем массив с названиями месяцев  
 String[] months = {"", "Январь", "Февраль", "Март", "Апрель", "Май", "Июнь",  
 "Июль", "Август", "Сентябрь", "Октябрь", "Ноябрь", "Декабрь"};  
  
 Scanner scanner = new Scanner(System.*in*); // Объект для ввода данных  
 int month = -1; // Переменная для хранения введенного числа  
  
 while (true) {  
 System.*out*.print("Введите число от 1 до 12: ");  
 String input = scanner.nextLine(); // Ввод с консоли  
  
 try {  
 month = Integer.*parseInt*(input); // Преобразуем строку в целое число  
  
 // Проверяем корректность диапазона  
 if (month >= 1 && month <= 12) {  
 break; // Если число корректно, выходим из цикла  
 } else {  
 System.*out*.println("Число должно быть в диапазоне от 1 до 12.");  
 }  
 } catch (NumberFormatException e) {  
 System.*out*.println("Ошибка: Введите допустимое число.");  
 }  
 }  
  
 // Выводим соответствующий месяц  
 System.*out*.println("Месяц: " + months[month]);  
 }  
}

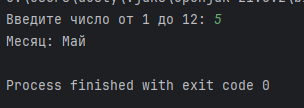


Рисунок - 1

## Задание 2 Abiturient: id, Фамилия, Имя, Отчество, Адрес, Телефон, Оценки. Создать массив объектов. Вывести:

а) список абитуриентов, имеющих неудовлетворительные оценки;

b) список абитуриентов, у которых сумма баллов выше заданной;

с) выбрать заданное число и абитуриентов, имеющих самую высокую сумму баллов (вывести также полный список абитуриентов, имеющих полупроходную сумму).

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| id | int | Уникальный идентификатор абитуриента. |
| lastName | String | Фамилия абитуриента. |
| firstName | String | Имя абитуриента. |
| middleName | String | Отчество абитуриента. |
| address | String | Адрес абитуриента. |
| phone | String | Номер телефона абитуриента. |
| grades | int[] | Массив оценок абитуриента. |
| abiturients | Abiturient[] | Массив объектов типа Abiturient. |
| n | int | Количество абитуриентов с наивысшей суммой баллов, которое нужно выбрать. |
| threshold | int | Пороговая сумма баллов для вывода списка абитуриентов, имеющих сумму баллов выше этой границы. |

Код:  
import java.util.Arrays;

import java.util.Comparator;

import java.util.Scanner;

class Abiturient {

private int id;

private String lastName;

private String firstName;

private String middleName;

private String address;

private String phone;

private int[] grades;

public Abiturient(int id, String lastName, String firstName, String middleName, String address, String phone, int[] grades) {

this.id = id;

this.lastName = lastName;

this.firstName = firstName;

this.middleName = middleName;

this.address = address;

this.phone = phone;

this.grades = grades;

}

public int getSumGrades() {

return Arrays.stream(grades).sum();

}

public boolean hasUnsatisfactoryGrade() {

for (int grade : grades) {

if (grade < 3) { // Оценки ниже 3 считаются неудовлетворительными

return true;

}

}

return false;

}

public int[] getGrades() {

return grades;

}

public String toString() {

return String.format("ID: %d, ФИО: %s %s %s, Адрес: %s, Телефон: %s, Оценки: %s",

id, lastName, firstName, middleName, address, phone, Arrays.toString(grades));

}

}

public class AbiturientApp {

public static void main(String[] args) {

// Пример массива абитуриентов

Abiturient[] abiturients = {

new Abiturient(1, "Иванов", "Иван", "Иванович", "ул. Ленина, 1", "1234567890", new int[]{5, 4, 3}),

new Abiturient(2, "Петров", "Петр", "Петрович", "ул. Садовая, 10", "2345678901", new int[]{2, 3, 4}),

new Abiturient(3, "Сидоров", "Сидор", "Сидорович", "ул. Московская, 15", "3456789012", new int[]{5, 5, 5}),

new Abiturient(4, "Кузнецов", "Андрей", "Андреевич", "ул. Лермонтова, 4", "4567890123", new int[]{3, 2, 3}),

new Abiturient(5, "Григорьев", "Григорий", "Григорьевич", "ул. Пушкина, 9", "5678901234", new int[]{4, 4, 4})

};

Scanner scanner = new Scanner(System.in);

// a) Список абитуриентов с неудовлетворительными оценками

System.out.println("Абитуриенты с неудовлетворительными оценками:");

for (Abiturient abiturient : abiturients) {

if (abiturient.hasUnsatisfactoryGrade()) {

System.out.println(abiturient);

}

}

// b) Список абитуриентов, у которых сумма баллов выше заданной

System.out.println("\nВведите пороговую сумму баллов: ");

int threshold = scanner.nextInt();

System.out.println("Абитуриенты с суммой баллов выше " + threshold + ":");

for (Abiturient abiturient : abiturients) {

if (abiturient.getSumGrades() > threshold) {

System.out.println(abiturient);

}

}

// c) Выбор заданного числа n абитуриентов с самой высокой суммой баллов

System.out.println("\nВведите количество абитуриентов для выбора по самой высокой сумме баллов:");

int n = scanner.nextInt();

// Сортировка абитуриентов по сумме баллов

Arrays.sort(abiturients, Comparator.comparingInt(Abiturient::getSumGrades).reversed());

System.out.println("Топ " + n + " абитуриентов с наивысшей суммой баллов:");

for (int i = 0; i < n && i < abiturients.length; i++) {

System.out.println(abiturients[i]);

}

// d) Абитуриенты с полупроходной суммой

int semiPassingThreshold = 10; // Это примерная полупроходная сумма баллов

System.out.println("\nАбитуриенты с полупроходной суммой баллов (более " + semiPassingThreshold + "):");

for (Abiturient abiturient : abiturients) {

if (abiturient.getSumGrades() > semiPassingThreshold) {

System.out.println(abiturient);

}

}

scanner.close();

}

}

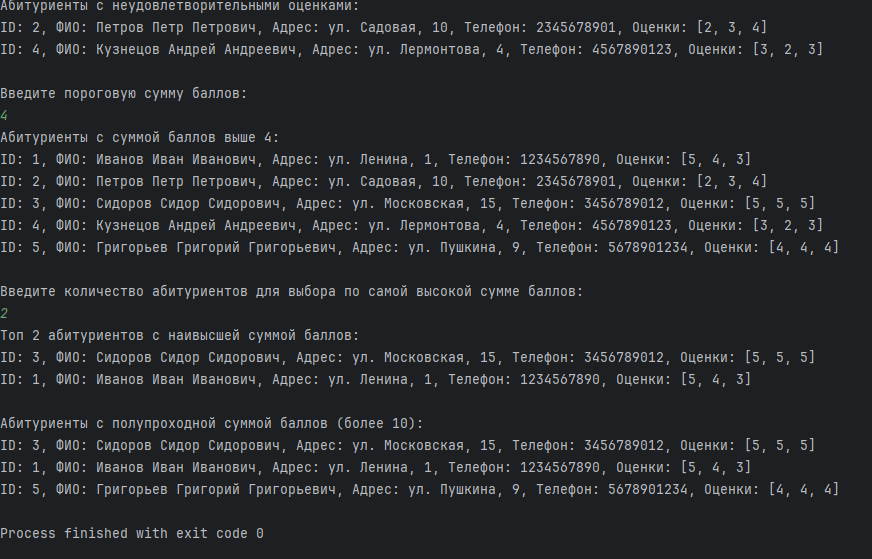
}

Рисунок – 2

## Задание 3

Создать объект класса Государство, используя классы Область, Район, Город. Методы: вывести на консоль столицу, количество областей, площадь, областные центры.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| name | String | Название области, района, города или государства. |
| capital | City | Столица государства. |
| regions | Region[] | Массив объектов типа Region, представляющий области государства. |
| area | double | Площадь области, района, города или государства. |
| centerCity | City | Областной центр (город) для каждого объекта Region. |
| districts | District[] | Массив объектов типа District, представляющий районы области. |
| cities | City[] | Массив объектов типа City, представляющий города района или области. |
| population | int | Численность населения города, района или области. |
| stateName | String | Название государства. |
| regionsCount | int | Количество областей в государстве. |
| totalArea | double | Общая площадь государства, рассчитанная на основе всех областей. |
| capitalName | String | Название столицы государства. |
| regionCenters | List<City> | Список объектов типа City, представляющий областные центры. |

Код:   
import java.util.ArrayList;

import java.util.List;

// Класс Город

class City {

private String name;

private int population;

private double area;

public City(String name, int population, double area) {

this.name = name;

this.population = population;

this.area = area;

}

public String getName() {

return name;

}

public int getPopulation() {

return population;

}

public double getArea() {

return area;

}

public String toString() {

return String.format("%s (население: %d, площадь: %.2f кв.км)", name, population, area);

}

}

// Класс Район

class District {

private String name;

private City[] cities;

public District(String name, City[] cities) {

this.name = name;

this.cities = cities;

}

public String getName() {

return name;

}

public City[] getCities() {

return cities;

}

}

// Класс Область

class Region {

private String name;

private City centerCity;

private District[] districts;

private double area;

public Region(String name, City centerCity, District[] districts, double area) {

this.name = name;

this.centerCity = centerCity;

this.districts = districts;

this.area = area;

}

public String getName() {

return name;

}

public City getCenterCity() {

return centerCity;

}

public District[] getDistricts() {

return districts;

}

public double getArea() {

return area;

}

}

// Класс Государство

class State {

private String name;

private City capital;

private Region[] regions;

public State(String name, City capital, Region[] regions) {

this.name = name;

this.capital = capital;

this.regions = regions;

}

public String getCapital() {

return capital.getName();

}

public int getRegionsCount() {

return regions.length;

}

public double getTotalArea() {

double totalArea = 0;

for (Region region : regions) {

totalArea += region.getArea();

}

return totalArea;

}

public List<City> getRegionCenters() {

List<City> regionCenters = new ArrayList<>();

for (Region region : regions) {

regionCenters.add(region.getCenterCity());

}

return regionCenters;

}

public void printStateInfo() {

System.out.println("Государство: " + name);

System.out.println("Столица: " + getCapital());

System.out.println("Количество областей: " + getRegionsCount());

System.out.println("Общая площадь: " + getTotalArea() + " кв.км");

System.out.println("Областные центры:");

for (City city : getRegionCenters()) {

System.out.println("- " + city);

}

}

}

public class StateApp {

public static void main(String[] args) {

// Создаем города

City minsk = new City("Минск", 2000000, 348.84);

City gomel = new City("Гомель", 500000, 121.58);

City vitebsk = new City("Витебск", 370000, 124.54);

City mogilev = new City("Могилев", 380000, 118.48);

City brest = new City("Брест", 340000, 144.80);

// Создаем районы и добавляем города

District[] minskDistricts = { new District("Центральный район", new City[]{minsk}) };

District[] gomelDistricts = { new District("Новобелицкий район", new City[]{gomel}) };

District[] vitebskDistricts = { new District("Октябрьский район", new City[]{vitebsk}) };

District[] mogilevDistricts = { new District("Ленинский район", new City[]{mogilev}) };

District[] brestDistricts = { new District("Московский район", new City[]{brest}) };

// Создаем области

Region minskRegion = new Region("Минская область", minsk, minskDistricts, 39000);

Region gomelRegion = new Region("Гомельская область", gomel, gomelDistricts, 40372);

Region vitebskRegion = new Region("Витебская область", vitebsk, vitebskDistricts, 40051);

Region mogilevRegion = new Region("Могилевская область", mogilev, mogilevDistricts, 29068);

Region brestRegion = new Region("Брестская область", brest, brestDistricts, 32786);

// Создаем государство

State belarus = new State("Беларусь", minsk, new Region[]{minskRegion, gomelRegion, vitebskRegion, mogilevRegion, brestRegion});

// Выводим информацию о государстве

belarus.printStateInfo();

}

}

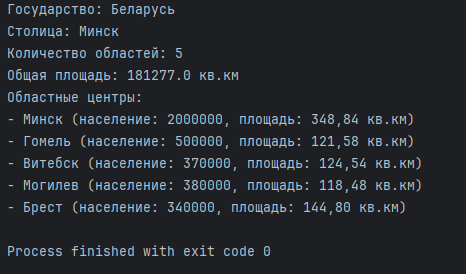


Рисунок - 3

## Задание 4

Создать класс Зачетная Книжка с внутренним классом, с помощью объектов которого можно хранить информацию о сессиях, зачетах, экзаменах.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| studentName | String | Имя студента. |
| studentId | String | Уникальный номер зачетной книжки (например, номер студенческого билета). |
| sessions | List<Session> | Список сессий, каждая из которых содержит информацию о зачетах и экзаменах. |
| semester | int | Номер семестра, для которого хранится информация о сессии. |
| subjects | List<Subject> | Список предметов, в которых сданы зачеты или экзамены. |
| subjectName | String | Название предмета. |
| grade | int | Оценка за экзамен (если это экзамен) или "зачет/незачет" для зачетов. |

Код:  
import java.util.ArrayList;

import java.util.List;

// Внешний класс Зачетная Книжка

class GradeBook {

private String studentName;

private String studentId;

private List<Session> sessions;

public GradeBook(String studentName, String studentId) {

this.studentName = studentName;

this.studentId = studentId;

this.sessions = new ArrayList<>();

}

// Добавление новой сессии

public void addSession(int semester) {

sessions.add(new Session(semester));

}

// Добавление предмета в последнюю сессию

public void addSubjectToLastSession(String subjectName, boolean isExam, int grade) {

if (sessions.isEmpty()) {

System.out.println("Сначала добавьте сессию.");

} else {

Session lastSession = sessions.get(sessions.size() - 1);

lastSession.addSubject(subjectName, isExam, grade);

}

}

// Вывод всей информации по зачетной книжке

public void printGradeBook() {

System.out.println("Зачетная книжка студента: " + studentName + " (ID: " + studentId + ")");

for (Session session : sessions) {

session.printSessionInfo();

}

}

// Внутренний класс Сессия

class Session {

private int semester;

private List<Subject> subjects;

public Session(int semester) {

this.semester = semester;

this.subjects = new ArrayList<>();

}

// Добавление предмета

public void addSubject(String subjectName, boolean isExam, int grade) {

subjects.add(new Subject(subjectName, isExam, grade));

}

// Вывод информации о сессии

public void printSessionInfo() {

System.out.println("Сессия " + semester + "-го семестра:");

for (Subject subject : subjects) {

subject.printSubjectInfo();

}

}

// Внутренний класс Предмет (зачет или экзамен)

class Subject {

private String subjectName;

private boolean isExam;

private int grade;

public Subject(String subjectName, boolean isExam, int grade) {

this.subjectName = subjectName;

this.isExam = isExam;

this.grade = grade;

}

// Вывод информации о предмете

public void printSubjectInfo() {

String result = isExam ? "Экзамен" : "Зачет";

String gradeInfo = isExam ? "Оценка: " + grade : (grade == 1 ? "Незачет" : "Зачет");

System.out.println(subjectName + " (" + result + ") - " + gradeInfo);

}

}

}

}

// Основной класс для демонстрации работы

public class GradeBookApp {

public static void main(String[] args) {

// Создание зачетной книжки для студента

GradeBook gradeBook = new GradeBook("Иван Иванов", "12345");

// Добавляем первую сессию и предметы

gradeBook.addSession(1);

gradeBook.addSubjectToLastSession("Математика", true, 5); // Экзамен с оценкой 5

gradeBook.addSubjectToLastSession("История", true, 4); // Экзамен с оценкой 4

gradeBook.addSubjectToLastSession("Физкультура", false, 2); // Зачет, незачет

// Добавляем вторую сессию и предметы

gradeBook.addSession(2);

gradeBook.addSubjectToLastSession("Физика", true, 3); // Экзамен с оценкой 3

gradeBook.addSubjectToLastSession("Английский", false, 1); // Зачет, незачет

gradeBook.addSubjectToLastSession("Химия", true, 4); // Экзамен с оценкой 4

// Выводим всю информацию о зачетной книжке

gradeBook.printGradeBook();

}

}

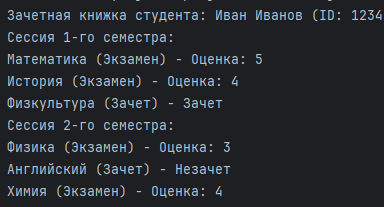


Рисунок - 4

## Задание 5

В тексте после какого символа вставить заданную подстроку.

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| text | String | Исходная строка, в которую нужно вставить подстроку. |
| substring | String | Заданная подстрока, которую нужно вставить в текст. |
| k | int | Позиция в исходной строке, после которой будет вставлена подстрока. |
| result | String | Результирующая строка после вставки подстроки в исходную строку. |

Код:  
import java.util.Scanner;

public class TextManipulation {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

// Ввод исходного текста

System.out.println("Введите текст: ");

String text = scanner.nextLine();

// Ввод подстроки

System.out.println("Введите подстроку для вставки: ");

String substring = scanner.nextLine();

// Ввод позиции k

System.out.println("Введите номер позиции k: ");

int k = scanner.nextInt();

// Проверяем корректность ввода позиции

if (k < 0 || k >= text.length()) {

System.out.println("Ошибка: значение k должно быть в диапазоне от 0 до " + (text.length() - 1));

} else {

// Вставляем подстроку после k-го символа

String result = insertSubstring(text, substring, k);

// Выводим результат

System.out.println("Результирующая строка: " + result);

}

scanner.close();

}

// Метод вставки подстроки в текст после k-го символа

public static String insertSubstring(String text, String substring, int k) {

// Разбиваем строку на две части и вставляем подстроку между ними

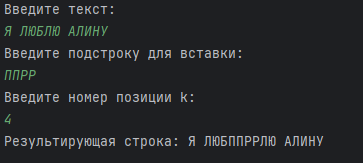
String before = text.substring(0, k + 1); // Первая часть до k-го символа включительно

String after = text.substring(k + 1); // Вторая часть после k-го символа

// Склеиваем части с подстрокой

return before + substring + after;

}

}  
  
Рисунок - 5

## Задание 6

## Задание 7

В треугольной пирамиде построить сечение, проходящее через среднюю линию боковой грани и противоположную вершину

основания.

Код:

// Класс для представления точки в 3D-пространстве  
class Point {  
 private double x, y, z;  
  
 public Point(double x, double y, double z) {  
 this.x = x;  
 this.y = y;  
 this.z = z;  
 }  
  
 public double getX() {  
 return x;  
 }  
  
 public double getY() {  
 return y;  
 }  
  
 public double getZ() {  
 return z;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("Point(%.2f, %.2f, %.2f)", x, y, z);  
 }  
}  
  
// Класс для представления треугольника  
class Triangle {  
 Point p1;  
 private Point p2;  
 private Point p3;  
  
 public Triangle(Point p1, Point p2, Point p3) {  
 this.p1 = p1;  
 this.p2 = p2;  
 this.p3 = p3;  
 }  
  
 public Point getMidpoint(Point a, Point b) {  
 double midX = (a.getX() + b.getX()) / 2;  
 double midY = (a.getY() + b.getY()) / 2;  
 double midZ = (a.getZ() + b.getZ()) / 2;  
 return new Point(midX, midY, midZ);  
 }  
  
 public Point getMiddleLinePoint() {  
 return getMidpoint(p2, p3); // Середина стороны p2p3  
 }  
}  
  
// Класс для представления треугольной пирамиды  
class Pyramid {  
 private Point apex; // Вершина пирамиды  
 private Triangle base; // Основание пирамиды  
  
 public Pyramid(Point apex, Triangle base) {  
 this.apex = apex;  
 this.base = base;  
 }  
  
 public Point getApex() {  
 return apex;  
 }  
  
 public Triangle getBase() {  
 return base;  
 }  
  
 // Метод для построения сечения  
 public void createSection() {  
 Point middleLinePoint = base.getMiddleLinePoint();  
 Point oppositeVertex = base.p1; // Противоположная вершина основания  
  
 System.*out*.println("Средняя точка боковой грани: " + middleLinePoint);  
 System.*out*.println("Противоположная вершина: " + oppositeVertex);  
 System.*out*.println("Сечение проходит через: " + middleLinePoint + " и " + oppositeVertex);  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 // Создаем вершину пирамиды  
 Point apex = new Point(0, 1, 0);  
  
 // Создаем точки основания (треугольника)  
 Point p1 = new Point(-1, 0, 1);  
 Point p2 = new Point(1, 0, 1);  
 Point p3 = new Point(0, 0, -1);  
  
 // Создаем основание пирамиды  
 Triangle base = new Triangle(p1, p2, p3);  
  
 // Создаем пирамиду  
 Pyramid pyramid = new Pyramid(apex, base);  
  
 // Создаем сечение  
 pyramid.createSection();  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| x | double | Координата X точки в 3D-пространстве. |
| y | double | Координата Y точки в 3D-пространстве. |
| z | double | Координата Z точки в 3D-пространстве. |
| apex | Point | Вершина пирамиды, представляющая собой точку в 3D-пространстве. |
| p1, p2, p3 | Point | Вершины основания (треугольника), представляющие собой точки в 3D-пространстве. |
| base | Triangle | Основание пирамиды, представляющее собой треугольник, составленный из трех точек. |
| middleLinePoint | Point | Средняя точка боковой грани пирамиды, находящаяся между двумя вершинами боковой грани. |
| oppositeVertex | Point | Противоположная вершина основания пирамиды, которая не принадлежит выбранной боковой грани. |

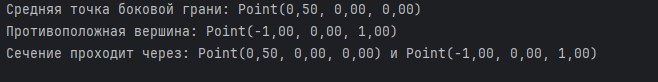
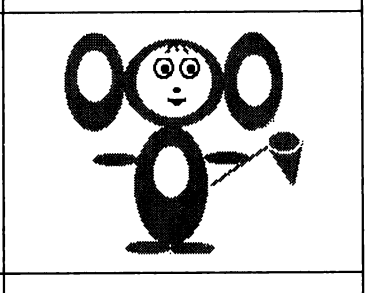


Рисунок - 6

## Задание 8



Код:

import javax.swing.\*;  
import java.awt.\*;  
  
public class DrawingExample extends JPanel {  
  
 @Override  
 protected void paintComponent(Graphics g) {  
 super.paintComponent(g);  
 drawCharacter(g);  
 }  
  
 private void drawCharacter(Graphics g) {  
 // Устанавливаем цвет и рисуем тело  
 g.setColor(Color.*BLACK*);  
 g.fillOval(150, 100, 100, 100); // голова  
  
 // Рисуем белую часть головы  
 g.setColor(Color.*WHITE*);  
 g.fillOval(175, 135, 50, 50); // лицо  
  
 // Рисуем глаза  
 g.setColor(Color.*BLACK*);  
 g.fillOval(185, 145, 10, 10); // левый глаз  
 g.fillOval(205, 145, 10, 10); // правый глаз  
  
 // Рисуем рот  
 g.drawArc(185, 155, 30, 20, 0, -180); // рот  
  
 // Рисуем уши  
 g.setColor(Color.*BLACK*);  
 g.fillOval(100, 130, 50, 50); // левое ухо  
 g.fillOval(250, 130, 50, 50); // правое ухо  
  
 // Рисуем тело  
 g.setColor(Color.*BLACK*);  
 g.fillOval(175, 200, 50, 80); // тело  
  
 // Рисуем руки  
 g.fillRect(120, 200, 50, 10); // левая рука  
 g.fillRect(230, 200, 50, 10); // правая рука  
  
 // Рисуем ноги  
 g.fillRect(175, 280, 20, 50); // левая нога  
 g.fillRect(200, 280, 20, 50); // правая нога  
  
 // Рисуем руку с конусом  
 g.setColor(Color.*BLACK*);  
 g.drawLine(150, 200, 100, 160); // рука  
 g.setColor(Color.*RED*);  
 g.fillPolygon(new int[]{100, 90, 110}, new int[]{160, 170, 170}, 3); // конус  
 }  
  
 public static void main(String[] args) {  
 JFrame frame = new JFrame("Drawing Example");  
 DrawingExample drawing = new DrawingExample();  
 frame.add(drawing);  
 frame.setSize(400, 400);  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.setVisible(true);  
 }  
}

|  |  |  |
| --- | --- | --- |
| Переменная | Тип данных | Описание |
| g | Graphics | Объект, используемый для рисования на панели. |
| xPoints | int[] | Массив для хранения координат X вершин конуса. |
| yPoints | int[] | Массив для хранения координат Y вершин конуса. |
| x | int | Координата X для рисования (используется для различных фигур). |
| y | int | Координата Y для рисования (используется для различных фигур). |
| width | int | Ширина фигур (например, для рисования головы, тела, рук и ног). |
| height | int | Высота фигур (например, для рисования головы, тела, рук и ног). |

Результат:

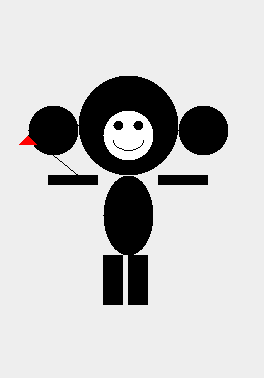
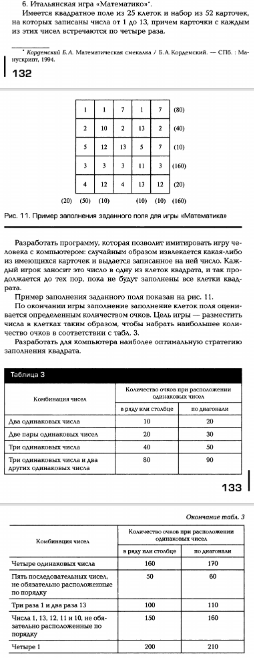
****

Рисунок - 7

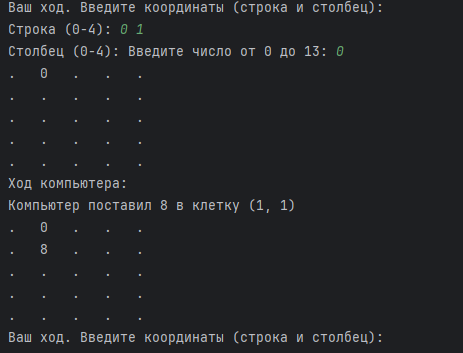
## Задание 9



Код:

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). |

  
Рисунок – 8

# 2. Разработка АИС

### 1. Описание и цели системы

Название: Автоматизированная Информационная Система (АИС) "Аналитический отдел фирмы"  
Цель:  
Система предназначена для автоматизации учета информации о проектах, сотрудниках и задачах аналитического отдела фирмы. АИС позволяет добавлять, редактировать и удалять данные, а также управлять связями между задачами, проектами и сотрудниками.

### 2. Функциональные требования

* Учет проектов:  
  Возможность добавления, редактирования и удаления данных о проектах. Каждый проект имеет уникальный идентификатор, название и описание.
* Учет сотрудников:  
  Управление информацией о сотрудниках, включая добавление, редактирование и удаление записей. Каждый сотрудник имеет уникальный ID, имя и должность.
* Учет задач:  
  Учет информации о задачах, включая название задачи, идентификатор проекта и идентификатор сотрудника, который выполняет задачу.
* Связи между сущностями:  
  Система должна обеспечивать корректное связывание задач с проектами и сотрудниками. Например, каждая задача связана с одним проектом и одним сотрудником, но проект может быть связан с несколькими задачами.

### 3. Нефункциональные требования

* Надежность:  
  Система должна обеспечивать корректное хранение данных и целостность связей между сущностями.
* Безопасность:  
  Реализовать разграничение прав доступа, чтобы предотвратить несанкционированное изменение данных.
* Удобство использования:  
  Простота и интуитивность интерфейса.
* Масштабируемость:  
  Возможность расширения системы при увеличении объема данных.
* Инструментальная платформа:  
  Приложение реализовано на Java с использованием Swing для графического интерфейса и JUnit для тестирования.

### 4. Структура системы и описание классов

Классы:

1. Project:  
   Содержит поля id, name, description для хранения информации о проекте.
2. Employee:  
   Включает id, name, position для хранения данных о сотрудниках.
3. Task:  
   Хранит id, title, projectId, employeeId для описания задачи и связи её с проектом и сотрудником.
4. AnalyticalDepartmentDatabase:  
   Основной класс для управления данными. Включает методы для добавления, редактирования и удаления записей в таблицах "Проекты", "Сотрудники" и "Задачи".

Основные методы:

* addProject(), editProject(), deleteProject(): Управление данными проектов.
* addEmployee(), editEmployee(), deleteEmployee(): Управление данными сотрудников.
* addTask(), editTask(), deleteTask(): Управление данными задач и их связями с проектами и сотрудниками.

### 5. Алгоритмы и описание действий пользователя

Алгоритм добавления задачи

1. Пользователь вводит название задачи, выбирает проект и сотрудника.
2. Система проверяет, существуют ли указанные проект и сотрудник.
3. Если данные корректны, задача добавляется в базу данных с уникальным идентификатором.

Алгоритм удаления задачи

1. Пользователь выбирает задачу для удаления.
2. Система подтверждает, что задача связана только с одним проектом и сотрудником.
3. После подтверждения система удаляет запись.

### 6. Тестирование системы

План тестирования

1. Функциональные тесты:  
   Проверка добавления, редактирования и удаления данных для каждой сущности (проекты, сотрудники, задачи).
2. Тесты на целостность данных:  
   Проверка корректности связи между задачами, проектами и сотрудниками при редактировании и удалении данных.
3. Проверка безопасности:  
   Тестирование ограничений для предотвращения некорректного ввода.

Основные сценарии тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Тест-кейс | Описание | Ожидаемый результат | Статус |
| Добавление проекта | Добавление нового проекта | Проект добавлен и отображается в таблице | Пройден |
| Редактирование сотрудника | Редактирование данных сотрудника | Данные сотрудника обновлены | Пройден |
| Удаление задачи | Удаление задачи из базы данных | Задача удалена без нарушений целостности | Пройден |

### 7. Техническое задание (ТЗ)

Цель:  
Разработка системы для управления данными о проектах, сотрудниках и задачах с графическим интерфейсом и встроенной системой тестирования для обеспечения надежности.

Функциональные требования:

* Управление данными проектов, сотрудников и задач.
* Возможность выполнения CRUD-операций (Create, Read, Update, Delete).
* Поддержка целостности данных и правильности ссылок между сущностями.

Нефункциональные требования:

* Платформа: Java.
* Интерфейс: Swing GUI.
* Тестирование: JUnit.

Разграничение доступа:

* Два уровня пользователей:
  + Администратор – права на редактирование и удаление данных.
  + Пользователь – права только на просмотр данных.

### 8. Руководство пользователя

1. Добавление данных:
   * Для добавления данных в таблицу выберите нужный тип сущности и заполните поля в диалоге добавления.
2. Редактирование данных:
   * Выберите сущность в таблице и нажмите "Редактировать", чтобы открыть диалог редактирования.
3. Удаление данных:
   * Выберите запись в таблице и нажмите "Удалить" для её удаления.

Совет:  
Пользователь может просматривать все записи, переключаясь между таблицами (проекты, сотрудники, задачи).

### 9. Протокол тестирования

|  |  |  |  |
| --- | --- | --- | --- |
| Тест-кейс | Описание | Ожидаемый результат | Статус |
| Добавление проекта | Добавление нового проекта | Проект добавлен и отображается в таблице | Пройден |
| Редактирование сотрудника | Редактирование данных сотрудника | Данные сотрудника обновлены | Пройден |
| Удаление задачи | Удаление задачи из базы данных | Задача удалена без нарушений целостности | Пройден |

## 2. Разработка базы данных

import javax.swing.\*;  
import javax.swing.table.DefaultTableModel;  
import java.awt.\*;  
import java.util.ArrayList;  
import java.util.List;  
  
// Класс для представления проекта  
class Project {  
 int id;  
 String name;  
 String description;  
  
 public Project(int id, String name, String description) {  
 this.id = id;  
 this.name = name;  
 this.description = description;  
 }  
}  
  
// Класс для представления сотрудника  
class Employee {  
 int id;  
 String name;  
 String position;  
  
 public Employee(int id, String name, String position) {  
 this.id = id;  
 this.name = name;  
 this.position = position;  
 }  
}  
  
// Класс для представления задачи  
class Task {  
 int id;  
 String title;  
 int projectId;  
 int employeeId;  
  
 public Task(int id, String title, int projectId, int employeeId) {  
 this.id = id;  
 this.title = title;  
 this.projectId = projectId;  
 this.employeeId = employeeId;  
 }  
}  
  
// Основной класс для GUI  
public class AnalyticalDepartmentGUI extends JFrame {  
 private List<Project> projects = new ArrayList<>();  
 private List<Employee> employees = new ArrayList<>();  
 private List<Task> tasks = new ArrayList<>();  
  
 private DefaultTableModel projectTableModel;  
 private DefaultTableModel employeeTableModel;  
 private DefaultTableModel taskTableModel;  
 private JTable mainTable;  
 private JPanel buttonPanel;  
  
 public AnalyticalDepartmentGUI() {  
 super("Аналитический отдел фирмы");  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setSize(800, 600);  
 setLocationRelativeTo(null);  
  
 // Создание таблиц  
 projectTableModel = new DefaultTableModel(new Object[]{"ID", "Название", "Описание"}, 0);  
 employeeTableModel = new DefaultTableModel(new Object[]{"ID", "Имя", "Должность"}, 0);  
 taskTableModel = new DefaultTableModel(new Object[]{"ID", "Название", "Проект ID", "Сотрудник ID"}, 0);  
  
 mainTable = new JTable(projectTableModel);  
 JScrollPane tableScrollPane = new JScrollPane(mainTable);  
  
 // Панель для переключения таблиц  
 JPanel tableSwitchPanel = new JPanel();  
 JButton projectsButton = new JButton("Проекты");  
 JButton employeesButton = new JButton("Сотрудники");  
 JButton tasksButton = new JButton("Задачи");  
  
 tableSwitchPanel.add(projectsButton);  
 tableSwitchPanel.add(employeesButton);  
 tableSwitchPanel.add(tasksButton);  
  
 // Панель действий  
 buttonPanel = new JPanel();  
 JButton addButton = new JButton("Добавить");  
 JButton editButton = new JButton("Редактировать");  
 JButton deleteButton = new JButton("Удалить");  
  
 buttonPanel.add(addButton);  
 buttonPanel.add(editButton);  
 buttonPanel.add(deleteButton);  
  
 // Добавление компонентов в окно  
 getContentPane().add(tableSwitchPanel, BorderLayout.*NORTH*);  
 getContentPane().add(tableScrollPane, BorderLayout.*CENTER*);  
 getContentPane().add(buttonPanel, BorderLayout.*SOUTH*);  
  
 // Переключение таблиц  
 projectsButton.addActionListener(e -> switchToProjects());  
 employeesButton.addActionListener(e -> switchToEmployees());  
 tasksButton.addActionListener(e -> switchToTasks());  
  
 // Обработка кнопок действий  
 addButton.addActionListener(e -> handleAddAction());  
 editButton.addActionListener(e -> handleEditAction());  
 deleteButton.addActionListener(e -> handleDeleteAction());  
  
 setVisible(true);  
 }  
  
 // Методы переключения таблиц  
 private void switchToProjects() {  
 mainTable.setModel(projectTableModel);  
 }  
  
 private void switchToEmployees() {  
 mainTable.setModel(employeeTableModel);  
 }  
  
 private void switchToTasks() {  
 mainTable.setModel(taskTableModel);  
 }  
  
 // Метод добавления записи  
 private void handleAddAction() {  
 DefaultTableModel currentModel = (DefaultTableModel) mainTable.getModel();  
 if (currentModel == projectTableModel) {  
 addProject();  
 } else if (currentModel == employeeTableModel) {  
 addEmployee();  
 } else if (currentModel == taskTableModel) {  
 addTask();  
 }  
 }  
  
 private void addProject() {  
 JTextField nameField = new JTextField();  
 JTextField descriptionField = new JTextField();  
  
 Object[] fields = {  
 "Название проекта:", nameField,  
 "Описание:", descriptionField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Добавить проект", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 String name = nameField.getText();  
 String description = descriptionField.getText();  
 int id = projects.size() + 1;  
  
 projects.add(new Project(id, name, description));  
 projectTableModel.addRow(new Object[]{id, name, description});  
 }  
 }  
  
 private void addEmployee() {  
 JTextField nameField = new JTextField();  
 JTextField positionField = new JTextField();  
  
 Object[] fields = {  
 "Имя сотрудника:", nameField,  
 "Должность:", positionField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Добавить сотрудника", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 String name = nameField.getText();  
 String position = positionField.getText();  
 int id = employees.size() + 1;  
  
 employees.add(new Employee(id, name, position));  
 employeeTableModel.addRow(new Object[]{id, name, position});  
 }  
 }  
  
 private void addTask() {  
 JTextField titleField = new JTextField();  
 JTextField projectIdField = new JTextField();  
 JTextField employeeIdField = new JTextField();  
  
 Object[] fields = {  
 "Название задачи:", titleField,  
 "ID проекта:", projectIdField,  
 "ID сотрудника:", employeeIdField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Добавить задачу", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 String title = titleField.getText();  
 int projectId = Integer.*parseInt*(projectIdField.getText());  
 int employeeId = Integer.*parseInt*(employeeIdField.getText());  
 int id = tasks.size() + 1;  
  
 tasks.add(new Task(id, title, projectId, employeeId));  
 taskTableModel.addRow(new Object[]{id, title, projectId, employeeId});  
 }  
 }  
  
 private void handleEditAction() {  
 int selectedRow = mainTable.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(this, "Выберите запись для редактирования.");  
 return;  
 }  
  
 DefaultTableModel currentModel = (DefaultTableModel) mainTable.getModel();  
 if (currentModel == projectTableModel) {  
 editProject(selectedRow);  
 } else if (currentModel == employeeTableModel) {  
 editEmployee(selectedRow);  
 } else if (currentModel == taskTableModel) {  
 editTask(selectedRow);  
 }  
 }  
  
 private void editProject(int selectedRow) {  
 Project project = projects.get(selectedRow);  
  
 JTextField nameField = new JTextField(project.name);  
 JTextField descriptionField = new JTextField(project.description);  
  
 Object[] fields = {  
 "Название проекта:", nameField,  
 "Описание:", descriptionField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Редактировать проект", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 project.name = nameField.getText();  
 project.description = descriptionField.getText();  
  
 // Обновляем данные в таблице  
 projectTableModel.setValueAt(project.name, selectedRow, 1);  
 projectTableModel.setValueAt(project.description, selectedRow, 2);  
 }  
 }  
  
 private void editEmployee(int selectedRow) {  
 Employee employee = employees.get(selectedRow);  
  
 JTextField nameField = new JTextField(employee.name);  
 JTextField positionField = new JTextField(employee.position);  
  
 Object[] fields = {  
 "Имя сотрудника:", nameField,  
 "Должность:", positionField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Редактировать сотрудника", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 employee.name = nameField.getText();  
 employee.position = positionField.getText();  
  
 // Обновляем данные в таблице  
 employeeTableModel.setValueAt(employee.name, selectedRow, 1);  
 employeeTableModel.setValueAt(employee.position, selectedRow, 2);  
 }  
 }  
  
 private void editTask(int selectedRow) {  
 Task task = tasks.get(selectedRow);  
  
 JTextField titleField = new JTextField(task.title);  
 JTextField projectIdField = new JTextField(String.*valueOf*(task.projectId));  
 JTextField employeeIdField = new JTextField(String.*valueOf*(task.employeeId));  
  
 Object[] fields = {  
 "Название задачи:", titleField,  
 "ID проекта:", projectIdField,  
 "ID сотрудника:", employeeIdField  
 };  
  
 int option = JOptionPane.*showConfirmDialog*(this, fields, "Редактировать задачу", JOptionPane.*OK\_CANCEL\_OPTION*);  
 if (option == JOptionPane.*OK\_OPTION*) {  
 task.title = titleField.getText();  
 task.projectId = Integer.*parseInt*(projectIdField.getText());  
 task.employeeId = Integer.*parseInt*(employeeIdField.getText());  
  
 // Обновляем данные в таблице  
 taskTableModel.setValueAt(task.title, selectedRow, 1);  
 taskTableModel.setValueAt(task.projectId, selectedRow, 2);  
 taskTableModel.setValueAt(task.employeeId, selectedRow, 3);  
 }  
 }  
  
  
 private void handleDeleteAction() {  
 int selectedRow = mainTable.getSelectedRow();  
 if (selectedRow == -1) {  
 JOptionPane.*showMessageDialog*(this, "Выберите запись для удаления.");  
 return;  
 }  
  
 DefaultTableModel currentModel = (DefaultTableModel) mainTable.getModel();  
 if (currentModel == projectTableModel) {  
 projects.remove(selectedRow);  
 } else if (currentModel == employeeTableModel) {  
 employees.remove(selectedRow);  
 } else if (currentModel == taskTableModel) {  
 tasks.remove(selectedRow);  
 }  
 currentModel.removeRow(selectedRow);  
 }  
  
 public static void main(String[] args) {  
 SwingUtilities.*invokeLater*(AnalyticalDepartmentGUI::new);  
 }  
}

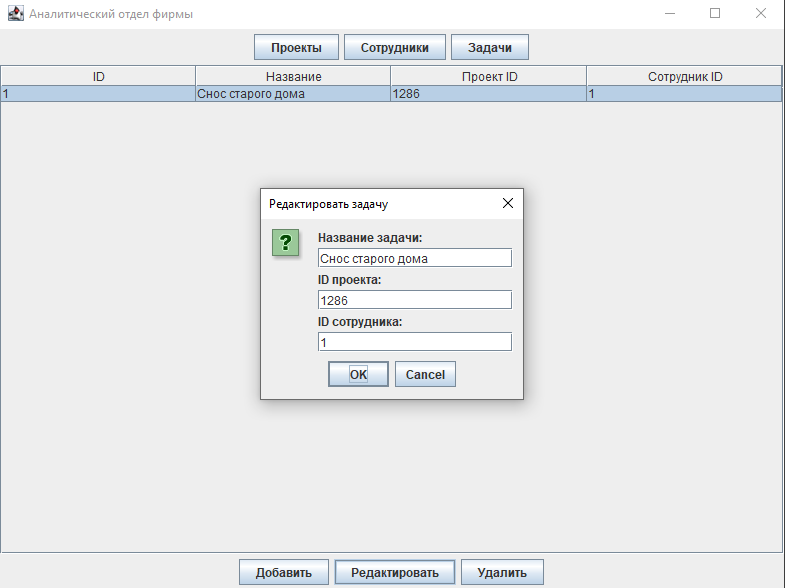


Рисунок 9 – База данных на Java, таблица «Проекты», реализация редактирования.

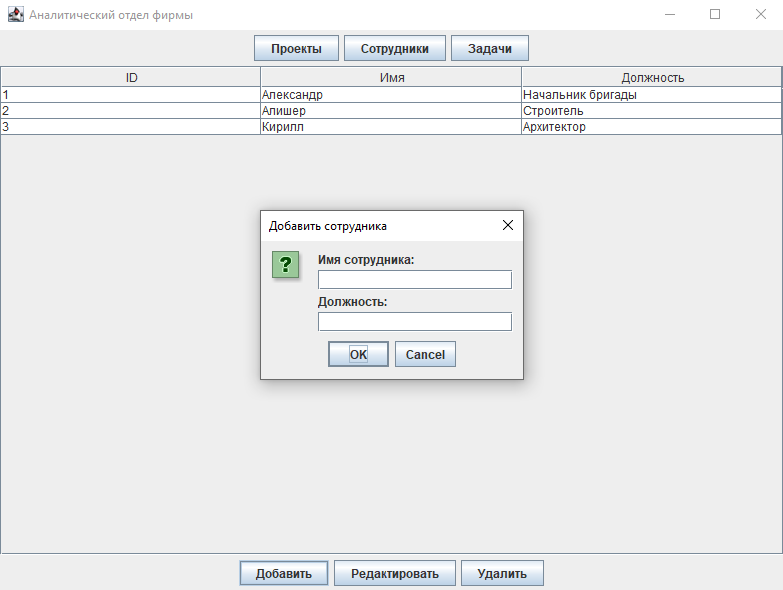


Рисунок 10 – Таблица «Сотрудники», реализация добавления.

## 3. Разработка мобильного приложения

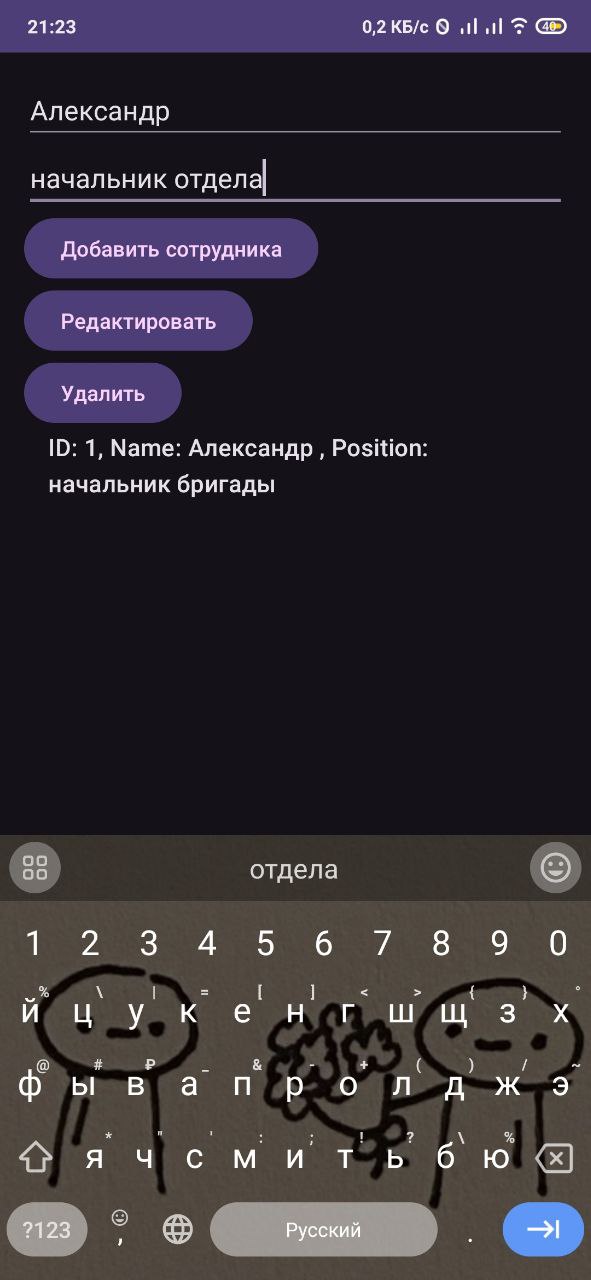


Рисунок 11 – Реализация мобильного приложения.

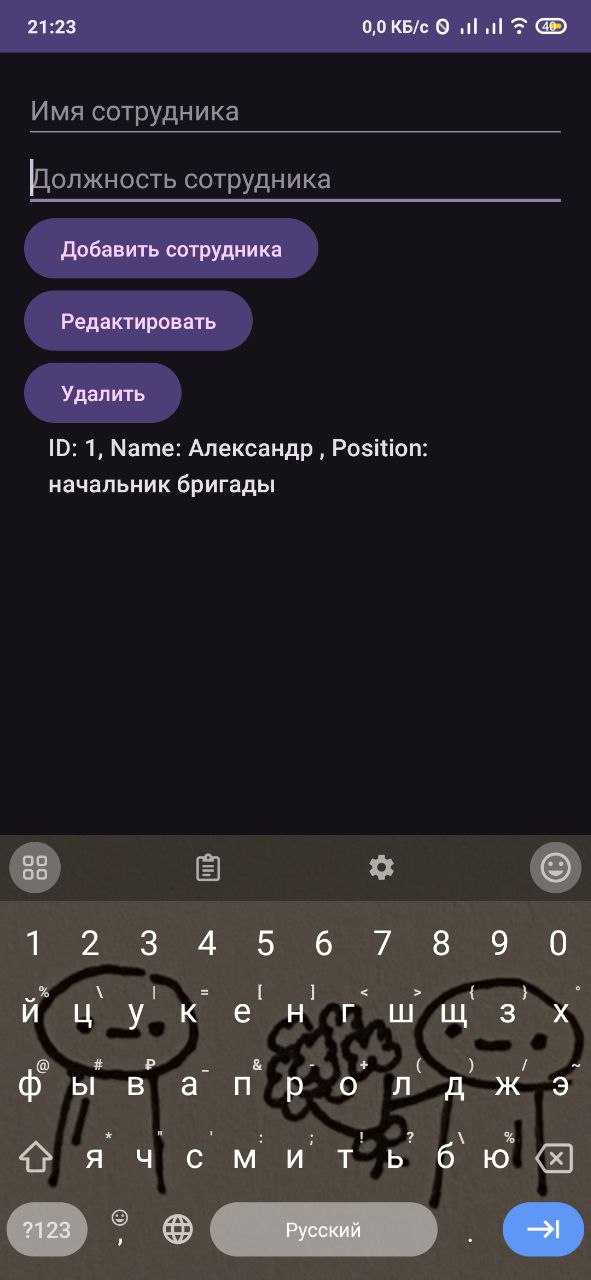


Рисунок 12 – Добавление сотрудника.

Описание мобильного приложения: Добавление сотрудников происходит по заполнению поля и нажатию кнопки «Добавить сотрудника». Редактирование происходит по нажатию на сотрудника, перезаполнению поля и нажатию кнопки «Редактировать». Удаление происходит по выбору сотрудника и нажатию кнопки «Удалить».

## Разработка 1С

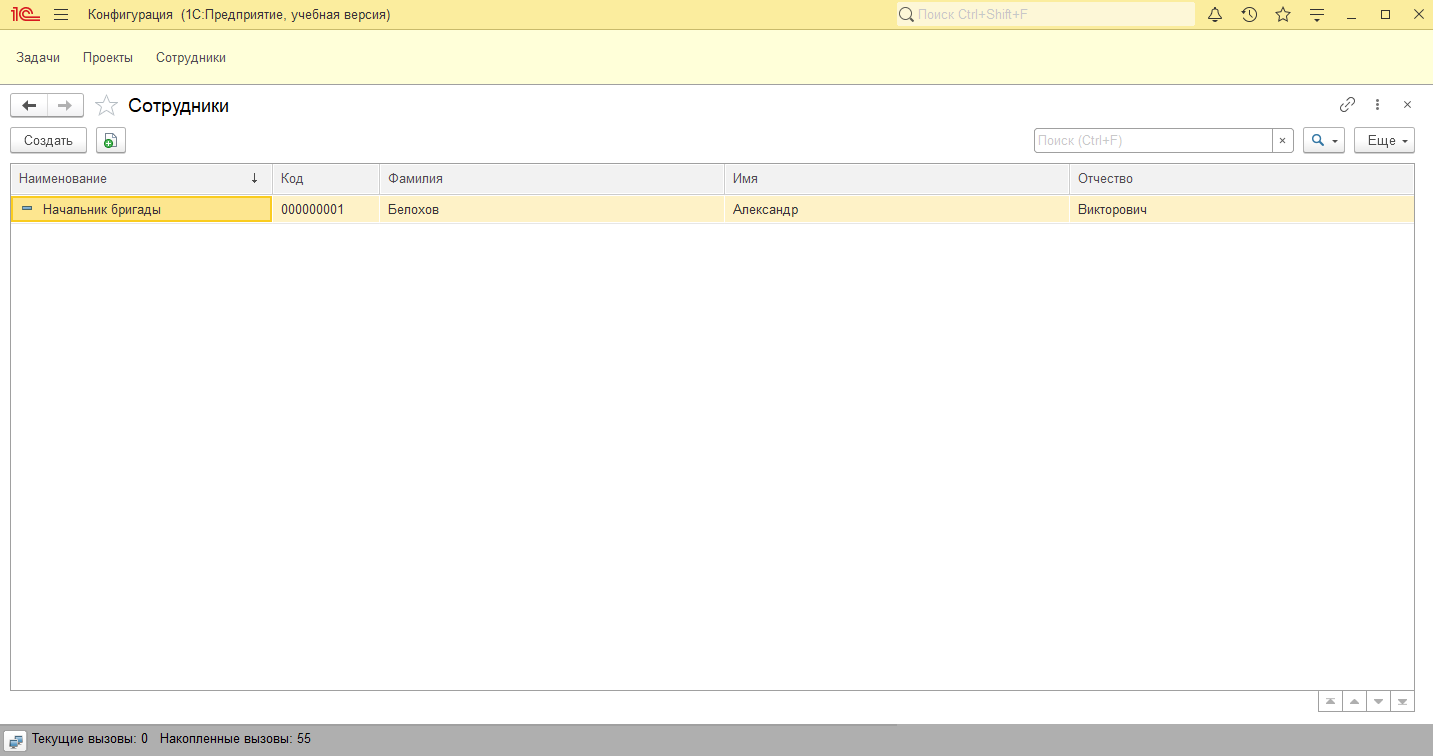


Рисунок 13 – База данных на 1с, таблица сотрудники.

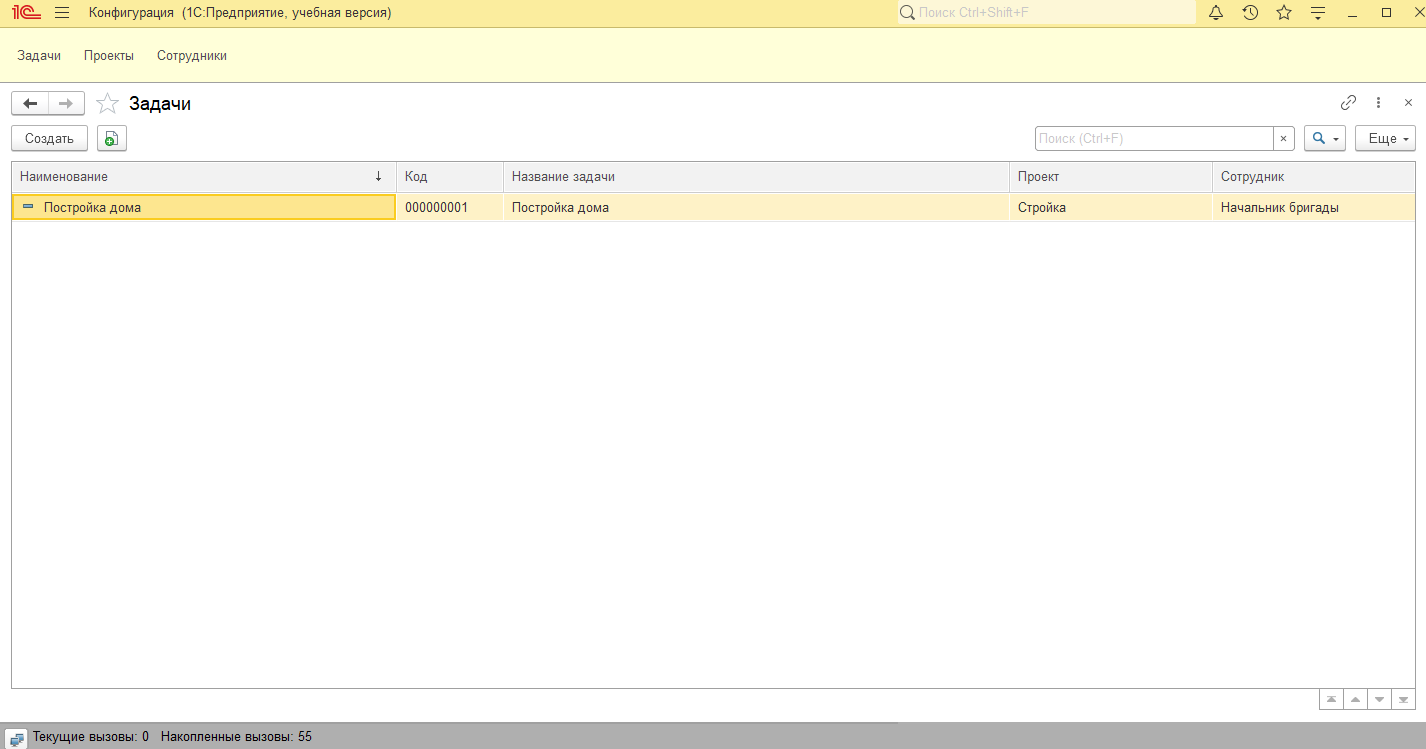


Рисунок 14 – База данных на 1с, таблица задачи.

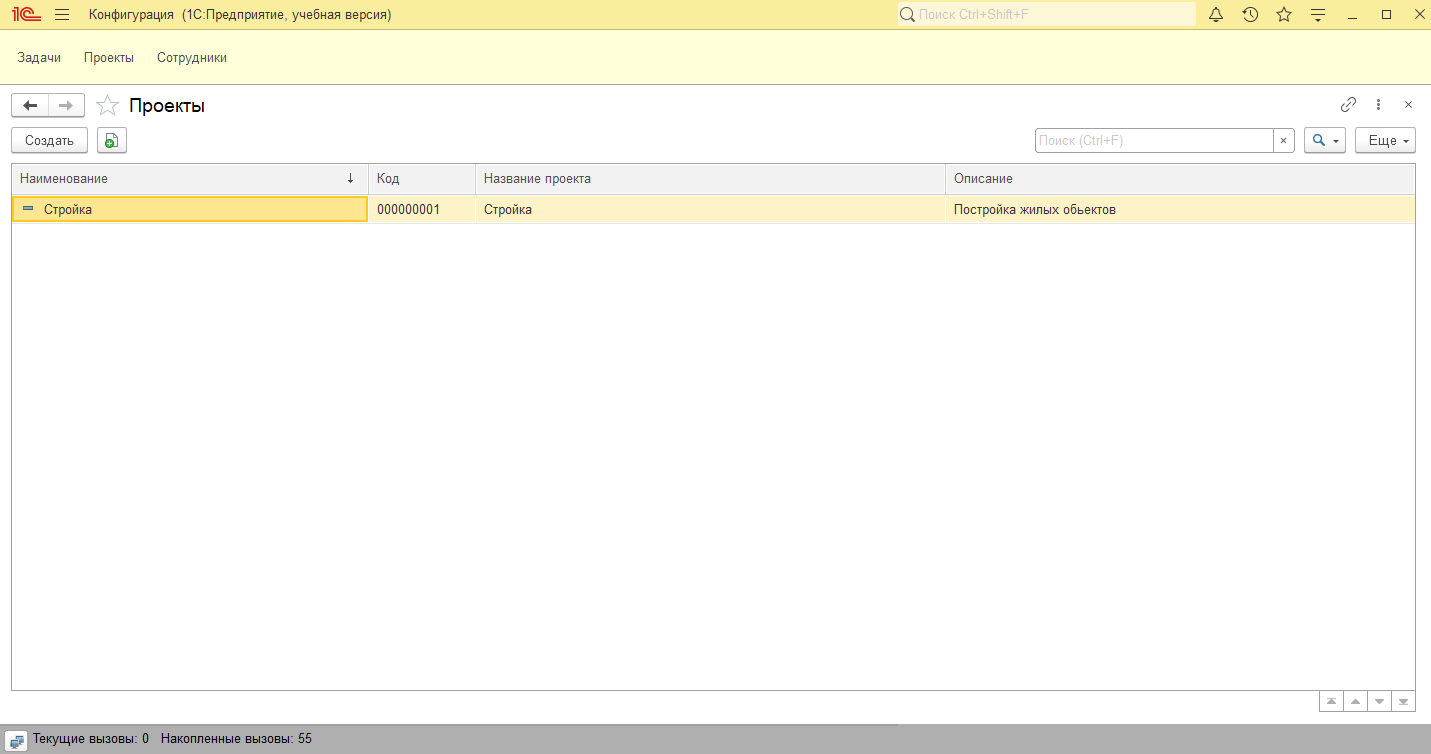


Рисунок 15 – База данных на 1с, таблица проекты.

# Заключение

Практика позволила создать рабочий прототип системы управления данными пассажиров. В процессе работы были отработаны навыки проектирования и тестирования информационных систем, обеспечена базовая функциональность и выполнены первичные меры безопасности. Задания выполнены в соответствии с требованиями, а результаты работы представлены на сайте, созданном для демонстрации работы.

Список литературы

1. Микросервисы с Spring Boot и Netflix / Джош Лонг. – O'Reilly Media, 2019. – 400 с.

2. Java 9 для начинающих / Патрик Нимейер. – O'Reilly Media, 2018. – 336 с.

3. Архитектура корпоративных приложений в Java / Мартин Фаулер. – Addison-Wesley Professional, 2020. – 576 с.

4. Современные подходы к разработке на Java / Энди Хантер, Кэмерон МакКул, Билл Бурке. – Manning Publications, 2021. – 512 с.

5. Spring в действии / Крэйг Уоллс.– Manning Publications, 2023. – 896 с.

6. Эффективные Java-приложения: разработка, тестирование и производительность / Кей С. Хорстманн. – Pearson Education, 2021. – 704 с.

7. Масштабируемая Java для крупных предприятий / Кевин Донохью, Армандо Файя, Алекс Сильва. – O'Reilly Media, 2019. – 464 с.

8. Реактивное программирование на Java / Скотт Майерс. – O'Reilly Media, 2020. – 352 с.

9. Разработка высокопроизводительных приложений на Java / Майкл Моррисон, Патрик Нимейер, Рафаэль Сандерс, Даг Лиди, Джеймс Суэйл. – Manning Publications, 2022. – 448 с.

10. Java Persistence with Hibernate / Крис Вермюлен, Гэвин Кинг, Лиам Хьюитт, Максим Коломыцев, Джонатан Ганц, Питер Дентон. – Packt Publishing, 2022. – 528 с.