Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ **ИНФОРМАТИКА, ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ И СИСТЕМЫ УПРАВЛЕНИЯ**

КАФЕДРА **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

О Т Ч Е Т

по лабораторной работе №6

Название: Коллекции

Дисциплина: Языки программирования для работы с большими данными

Студент	ИУ6-22М		А.М. Панфилкин
	(Группа)	(Подпись, дата)	(И.О. Фамилия)
Преподаватель			П.В. Степанов
		(Подпись, дата)	(И.О. Фамилия)

Москва, 2022 г.

Весь приведенный ниже код также доступен в следующем репозитории:

https://github.com/SilkSlime/iu6plfbd

Задание 1: Умножить два многочлена заданной степени, если коэффициенты многочленов хранятся в различных списках.

Листинг 1 – Задание 1

```
package l6;

import java.util.List;
import java.util.ArrayList;
import java.util.Random;
import java.util.Arrays;

public class e1 {

    /**
     * Вариант 1. Задача 5.
     * Умножить два многочлена заданной степени, если коэффициенты многочленов хранятся в
различных списках.
     */
    public static void main(String[] args) {
        // // Создаем два списка со случайными числами - коэффициентами многочленов
        // List<Integer> list1 = new ArrayList<>();
        // List<Integer> list2 = new ArrayList<>();
        // Random random = new Random();
        // for (int i = 0; i < 5; i++) {
        //     list1.add(random.nextInt(10));
        //     list2.add(random.nextInt(10));
        // }

        // Список со значениями 2x^4 + 7x^3 + 2x^2 + 2x + 1
        List<Integer> list1 = new ArrayList<>(Arrays.asList(2, 7, 2, 2, 1));
        // Список со значениями 2x^4 + 2x^3 + 6x^2 + 6x + 7
        List<Integer> list2 = new ArrayList<>(Arrays.asList(2, 2, 6, 6, 7));



        // Выводим многочлены
        System.out.println("Многочлен 1: " + list1);
        System.out.println("Многочлен 2: " + list2);

        // Умножаем многочлены
        List<Integer> result = new ArrayList<>();
        for (int i = 0; i < list1.size(); i++) {
            for (int j = 0; j < list2.size(); j++) {
                if (i + j >= result.size()) {
                    result.add(list1.get(i) * list2.get(j));
                } else {
                    result.set(i + j, result.get(i + j) + list1.get(i) * list2.get(j));
                }
            }
```

```
        }

        // Выводим результат
        System.out.println("Результат: " + result);

        // Пример (2x^4 + 7x^3 + 2x^2 + 2x + 1)*(2x^4 + 2x^3 + 6x^2 + 6x + 7) = 4x^8 + 18x^7 +
30x^6 + 62x^5 + 74x^4 + 75x^3 + 32x^2 + 20x + 7

    }
}
```

Задание 2: Не используя вспомогательных объектов, переставить отрицательные элементы данного списка в конец, а положительные – в начало этого списка.

Листинг 2 – Задание 2

```
package l6;

import java.util.List;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Random;

public class e2 {

    /**
     * Вариант 1. Задача 6.
     * Не используя вспомогательных объектов, переставить отрицательные элементы данного списка
в конец, а положительные – в начало этого списка.
     */
    public static void main(String[] args) {
        // Создаем список со случайными числами
        List<Integer> list = new ArrayList<>();
        Random random = new Random();
        for (int i = 0; i < 10; i++) {
            list.add(random.nextInt(20) - 10);
        }
        System.out.println("Исходный список: " + list);

        // Переставляем отрицательные элементы в конец, а положительные – в начало списка
        // Для этого можно просто отсортировать список
        Collections.sort(list);
        System.out.println("Итоговый список: " + list);
    }
 }
```

Задание 3: Во входном файле расположены два набора положительных чисел; между наборами стоит отрицательное число. Построить два списка С1 и С2, элементы которых содержат соответственно числа 1-го и 2-го набора таким образом, чтобы внутри одного списка числа были упорядочены по возрастанию. Затем объединить списки С1 и С2 в один упорядоченный список, изменяя только значения полей ссылочного типа.

Листинг 3 – Задание 3

```
package l6;

import java.io.File;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.Scanner;
import java.io.IOException;


public class e3 {

    /**
     * Вариант 2. Задача 5.
     * Во входном файле расположены два набора положительных чисел;
     * между наборами стоит отрицательное число. Построить два
     * списка C1 и C2, элементы которых содержат соответственно
     * числа 1-го и 2-го набора таким образом, чтобы внутри одного
     * списка числа были упорядочены по возрастанию. Затем
     * объединить списки C1 и C2 в один упорядоченный список,
     * изменяя только значения полей ссылочного типа.
     */
     public static void main(String[] args) throws IOException {
        Scanner scanner = new Scanner(new File("l6/e3.txt"));

        List<Integer> c1 = new ArrayList<Integer>();
        List<Integer> c2 = new ArrayList<Integer>();

        // Разделяем числа на две группы
        while (scanner.hasNext()) {
            int num = scanner.nextInt();
            if (num < 0) {
                break;
            }
            c1.add(num);
        }
        while (scanner.hasNext()) {
            int num = scanner.nextInt();
            if (num < 0) {
                break;
            }
            c2.add(num);
        }

        // Сортируем списки по возрастанию
        Collections.sort(c1);
        Collections.sort(c2);

        // Объединяем списки
        List<Integer> combinedList = new ArrayList<>(c1);
        combinedList.addAll(c2);

        // Сортируем объединенный список
```

```
        Collections.sort(combinedList);

        // Выводим результат
        System.out.println("C1: " + c1);
        System.out.println("C2: " + c2);
        System.out.println("Combined: " + combinedList);
    }
 }
```

Задание 4: На плоскости задано N точек. Вывести в файл описания всех прямых, которые проходят более чем через одну точку из заданных. Для каждой прямой указать, через сколько точек она проходит. Использовать класс HashMap.

Листинг 4 – Задание 4

```
package l6;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;


public class e4 {

    /**
     * Вариант 2. Задача 6.
     * На плоскости задано N точек. Вывести в файл описания
     * всех прямых, которые проходят более чем через одну
     * точку из заданных. Для каждой прямой указать, через
     * сколько точек она проходит. Использовать класс HashMap.
     */
    public static void main(String[] args) {
        // Создаем список точек
        List<Point> points = new ArrayList<Point>();
        points.add(new Point(1, 2));
        points.add(new Point(3, 4));
        points.add(new Point(5, 6));
        points.add(new Point(7, 8));

        points.add(new Point(1, 3));
        points.add(new Point(2, 4));

        points.add(new Point(1, 4));
        points.add(new Point(2, 3));
        points.add(new Point(3, 2));

        // Создаем хэш-таблицу для хранения прямых и количества точек, через которые они
проходят
        Map<Line, ArrayList<Point>> linePoints = new HashMap<Line, ArrayList<Point>>();

        // Добавляем все возможные прямые в хэш-таблицу
        for (int i = 0; i < points.size(); i++) {
```

```
            for (int j = i + 1; j < points.size(); j++) {
                // Проверка на совпадение точек
                if (points.get(i).equals(points.get(j))) {
                    continue;
                }
                // Создаем прямую
                Line line = new Line(points.get(i), points.get(j));
                // Проверяем, есть ли прямая в хэш-таблице
                if (!linePoints.containsKey(line)) {
                    // Если нет, добавляем прямую в хэш-таблицу
                    linePoints.put(line, new ArrayList<Point>());
                    // Добавляем в список точек
                    linePoints.get(line).add(points.get(i));
                    linePoints.get(line).add(points.get(j));
                } else {
                    // Если есть, проверяем, есть ли в списке точек текущая точка
                    if (!linePoints.get(line).contains(points.get(i))) {
                        linePoints.get(line).add(points.get(i));
                    }
                    if (!linePoints.get(line).contains(points.get(j))) {
                        linePoints.get(line).add(points.get(j));
                    }
                }
            }
        }

        // Выводим результат в файл e4.txt в папке l6
        try {
            java.io.FileWriter fw = new java.io.FileWriter("l6/e4.txt");
            // Перебираем все прямые
            for (Map.Entry<Line, ArrayList<Point>> entry : linePoints.entrySet()) {
                // Если точек больше 2, выводим прямую и количество точек, через которые она
проходит
                if (entry.getValue().size() > 2) {
                    fw.write(entry.getKey().toString() + " passes through " +
entry.getValue().size() + " points.\n");
                }
            }
            // Закрываем файл
            fw.close();
        } catch (Exception e) {
            // Выводим сообщение об ошибке если что-то пошло не так, хах :)
            System.out.println("Error writing to file: " + e.getMessage());
        }
    }
 }

// Класс Point для хранения координат точки
class Point {
    private double x;
    private double y;

    public Point(double x, double y) {
```

```
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    public String toString() {
        return "(" + x + ", " + y + ")";
    }
}

// Класс Line для хранения прямой
class Line {
    // Прямая задается уравнением y = slope * x + yIntercept
    // где slope - коэффициент наклона, yIntercept - y-сдвиг
    private double slope;
    private double yIntercept;

    public Line(Point p1, Point p2) {
        if (p1.getX() == p2.getX()) {
            // Вертикальная прямая: бесконечный коэффициент наклона, используем x-сдвиг как y-
сдвиг
            slope = Double.POSITIVE_INFINITY;
            yIntercept = p1.getX();
        } else {
            // Не вертикальная прямая: вычисляем коэффициент наклона и y-сдвиг
            slope = (p2.getY() - p1.getY()) / (p2.getX() - p1.getX());
            yIntercept = p1.getY() - slope * p1.getX();
        }
    }

    public double getSlope() {
        return slope;
    }

    public double getYIntercept() {
        return yIntercept;
    }

    public String toString() {
        if (slope == Double.POSITIVE_INFINITY) {
            return "x = " + yIntercept;
        } else {
            return "y = " + slope + "x + " + yIntercept;
        }
    }
```

```
    // Переопределяем методы equals и hashCode для корректной работы хэш-таблицы
    public boolean equals(Object other) {
        if (other instanceof Line) {
            Line otherLine = (Line) other;
            return slope == otherLine.getSlope() && yIntercept == otherLine.getYIntercept();
        } else {
            return false;
        }
    }

    public int hashCode() {
        return Double.hashCode(slope) ^ Double.hashCode(yIntercept);
    }
}
```

Вывод: В ходе лабораторной работы мы изучили работу с коллекциями в Java и реализовали решение четырех задач. В первой задаче мы умножили два многочлена заданной степени, используя различные списки для коэффициентов многочленов. Во второй задаче мы переставили элементы списка так, чтобы отрицательные элементы находились в конце, а положительные - в начале списка. В третьей задаче мы построили два упорядоченных списка из положительных чисел, разделенных отрицательным числом, и объединили их в один упорядоченный список, изменяя только значения полей ссылочного типа. Наконец, в четвертой задаче мы использовали класс HashMap для поиска всех прямых, проходящих более чем через одну заданную точку на плоскости, и вывода описания каждой прямой и количества точек, через которые она проходит. В результате работы мы получили опыт работы с различными типами коллекций в Java и узнали о применении класса HashMap для решения задач связанных с ассоциативными массивами.