МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Кафедра 319 «Системы интеллектуального мониторинга»

ЛАБОРАТОРНАЯ РАБОТА №3

по дисциплине «Программирование»

Тема: динамические структуры данных

Студент _____ Андрианова Е.А.

Группа М3О-135Б-20

Проверил _____ Шилов В.В.

Оценка _____ Дата защиты «__» 2021 г.

Москва 2021

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(национальный исследовательский университет)»

Кафедра 319 «Системы интеллектуального мониторинга»

З А Д А Н И Е

на лабораторную работу №3 по дисциплине

«Программирование»

Студент МЗО-135Б-20, Андрианова Елена Андреевна
(№ группы, Ф. И. О.)

Тема динамические структуры данных

Перечень вопросов, подлежащих разработке в лабораторной работе

Изучение принципов работы динамических структур данных
Реализация предложенных интерфейсов
Разработка пояснительной записки

Рекомендуемая литература

1. Герберт Шилдт. Java. Полное руководство, 10-е изд. : Пер. с англ. – СПб. ООО "Альфакнига', 2018. – 1488 с. ISBN 978-5-6040043-6-4
2. Бертран Мейер. Объектно-ориентированное конструирование программных систем: Русская Редакция; 2015. – 768 с. ISBN 5-7502-0255-0
3. Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computers Science. 2-е изд. – СПб.:Питер, 2013. – 704 с. ISBN 978-5-496-007405

Задание выдано «17» апрель 2021 г.
Проверил *Шилов В.В*
(Ф. И. О., подпись)
Студент
(подпись)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Данная лабораторная работа направлена на разработку динамических структур данных – Array, Linked, Map. Динамические структуры данных – это структуры данных, память под которые выделяется и освобождается по мере необходимости. Задачи, обрабатывающие данные, которые по своей природе являются динамическими, удобно решать с помощью динамических структур. Для решения проблемы адресации динамических структур данных используется метод, называемый динамическим распределением памяти, то есть память под отдельные элементы выделяется в момент, когда они "начинают существовать" в процессе выполнения программы, а не во время компиляции. Компилятор в этом случае только выделяет фиксированный объем памяти для хранения адреса динамически размещаемого элемента, а не самого элемента.

1. ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

Array – динамический массив. Основан на статическом массиве, который при необходимости расширяется. При добавлении нового элемента проверяется наличие свободной ячейки в статическом массиве. Если ячейка имеется, то происходит вставку. Если ячейки нет происходит выделение свободной памяти под новый статический массив, который превосходит старый по размерности, далее происходит копирование элементов из старого массива в новый, далее удаление старого массива и вставка нового элемента в новый массив.

Преимущество: быстрая вставка в конец массива, быстрый поиск по индексу.

Недостатки**:** копирование из старого массива в новый, дорогая операция вставки не в конец массива, дорогая операция удаления не из конца списка.

Linked – односвязный список. Каждый элемент имеет ссылку на следующий элемент. При добавлении нового элемента значение помешается в специальный объект, который может хранить не только значение, но и ссылку на следующий элемент.

Преимущество: быстрая вставка в конец массива и в начало массива

Недостатки**:** дорогая операция поиска по индексу.

Map – структура ключ-значение. По ключу (объект) возможно однозначно найти и определить значение (объект).

2 Процесс разработки

2.1 Ключевые части кода

2.1.1 Реализация Array

Основные функции в Array добавления и удаления представлены в листингах 1, 2, 3, 4.

Листинг 1 – Добавляет указанный элемент в конец этого списка

```
public boolean add(String o) {
        if (freeIndex != amountOfElements) {
            array[freeIndex] = o;
        } else {
            String[] newArray = new String[freeIndex +
STANDART_VALUE];
            amountOfElements += STANDART_VALUE;
            System.arraycopy(array, 0, newArray, 0, freeIndex);
            newArray[freeIndex] = o;
            array = newArray;
        }
        ++freeIndex;
        return true;
    }
```

Листинг 2 – Вставляет указанный элемент в указанную позицию в этом списке

```
public boolean addAll(int index, Array c) {

        checkIndexToSize(index);
        checkArray(c);
        if (c.isEmpty()) {
            return false;
        }

        if ((freeIndex + c.size()) >= amountOfElements) {
            if ((c.size() + freeIndex) % amountOfElements == 0) {
                int count = c.size() / amountOfElements;
                amountOfElements += count * STANDART_VALUE;
            } else {
                int count = c.size() / amountOfElements + 1;
                amountOfElements += count * STANDART_VALUE;
            }
            String[] newArray = new String[amountOfElements];
            System.arraycopy(array, 0, newArray, 0, freeIndex);
            array = newArray;
        }

        for (int i = index; i < freeIndex; ++i) {
```

```
            this.array[i + c.size()] = this.array[i];
        }

        for (int i = 0; i < c.size(); ++i) {
            this.array[i + index] = c.get(i);
        }

        freeIndex += c.size();
        return true;
    }
```

Листинг 3 – Удаляет элемент в указанной позиции в этом списке

```
public String remove(int index) {
        checkIndexLess(index);
        String line = array[index];
        System.arraycopy(array, index + 1, array, index, freeIndex
- index - 1);
        --freeIndex;
        return line;
    }
```

Листинг 4 – Удаляет первое вхождение указанного элемента из этого списка, если он присутствует

```
public boolean remove(Object o) {
        int size = freeIndex;
        for (int i = 0; i < size; ++i) {
            if (Objects.equals(array[i], o)) {
                System.arraycopy(array, i + 1, array, i, freeIndex
- i - 1);
                --freeIndex;
                return true;
            }
        }
        return false;
    }
```

2.1.2 Реализация Linked:

Основные функции в Linked добавления и удаления представлены в листингах 5, 6, 7, 8.

Листинг 5 – Добавляет указанный элемент в конец этого списка

```
public boolean add(String o) {
        Node newNode = new Node(o);
        if (firstElement == null) {
            firstElement = newNode;
            ++freeIndex;
        } else {
```

```
            Node last = firstElement;
            while (last.next != null) {
                last = last.next;
            }
            ++freeIndex;
            last.next = newNode;
        }
        return true;
    }
```

Листинг 6 – Заменяет элемент в указанной позиции в этом списке на указанный элемент

```
public boolean addAll(int index, Linked c) {

        checkIndexToSize(index);
        checkLinked(c);
        if (c.isEmpty()) {
            return false;
        }

        for (int i = index; i < c.size() + index; ++i) {
            add(i, c.get(i - index));
        }
        return true;
    }
```

Листинг 7 – Удаляет элемент в указанной позиции в этом списке

```
public String remove(int index) {

        checkIndexLess(index);

        String line = get(index);
        if (index == 0) {
            firstElement = firstElement.next;
            --freeIndex;
        } else {
            Node buff = firstElement;
            Node newBuff = firstElement;
            for (int i = 0; i < index; i++) {
                newBuff = buff;
                buff = newBuff.next;
            }
            newBuff.next = buff.next;
            --freeIndex;
        }
        return line;
    }
```

Листинг 8 – Удаляет первое вхождение указанного элемента из этого списка, если он присутствует

```
public boolean remove(Object o) {
        for (int i = 0; i < freeIndex; ++i) {
            if (Objects.equals(o, get(i))) {
                remove(i);
                return true;
            }
        }
        return false;
    }
```

2.1.3 Реализация Map:

Основные функции Map: добавления и удаления, представлены в листингах 9, 10.

Листинг 9 – Связывает указанное значение с указанным ключом на этой карте.

```
public Integer put(String key, Integer value) {
        int index = hash(key);

        if (basket[index] == null) {
            basket[index] = new Basket(key, value);
            ++countElements;
            this.resize();
            return null;
        } else {
            Node find = basket[index].findKey(key);
            if (find == null) {
                basket[index].addNode(key, value);
                ++countElements;
                this.resize();
                return null;
            } else {
                int prevValue = find.value;
                find.value = value;
                return prevValue;
            }
        }
    }
```

Листинг 10 – Удаляет сопоставление для ключа с этой карты, если оно присутствует.

```
public Integer remove(String key) {

        int index = hash(key);
        Basket bask = basket[index];
        if (bask == null) {
            return null;
```

```
        }
        Node node = bask.first;
        Node remover = node;
        Node helper = remover;

        while (remover != null) {
            if (Objects.equals(remover.key, key)) {
                int prevValue = remover.value;
                if (bask.last == bask.first) {
                    bask.first = null;
                    basket[index] = null;
                    --countElements;
                    return prevValue;
                }
                helper.next = remover.next;
                basket[index].first = node;
                --countElements;
                return prevValue;
            }
            helper = remover;
            remover = remover.next;
        }
        return null;
    }
```

2.2 Диаграмма классов

Диаграмма классов программы представлена на рисунке 1.

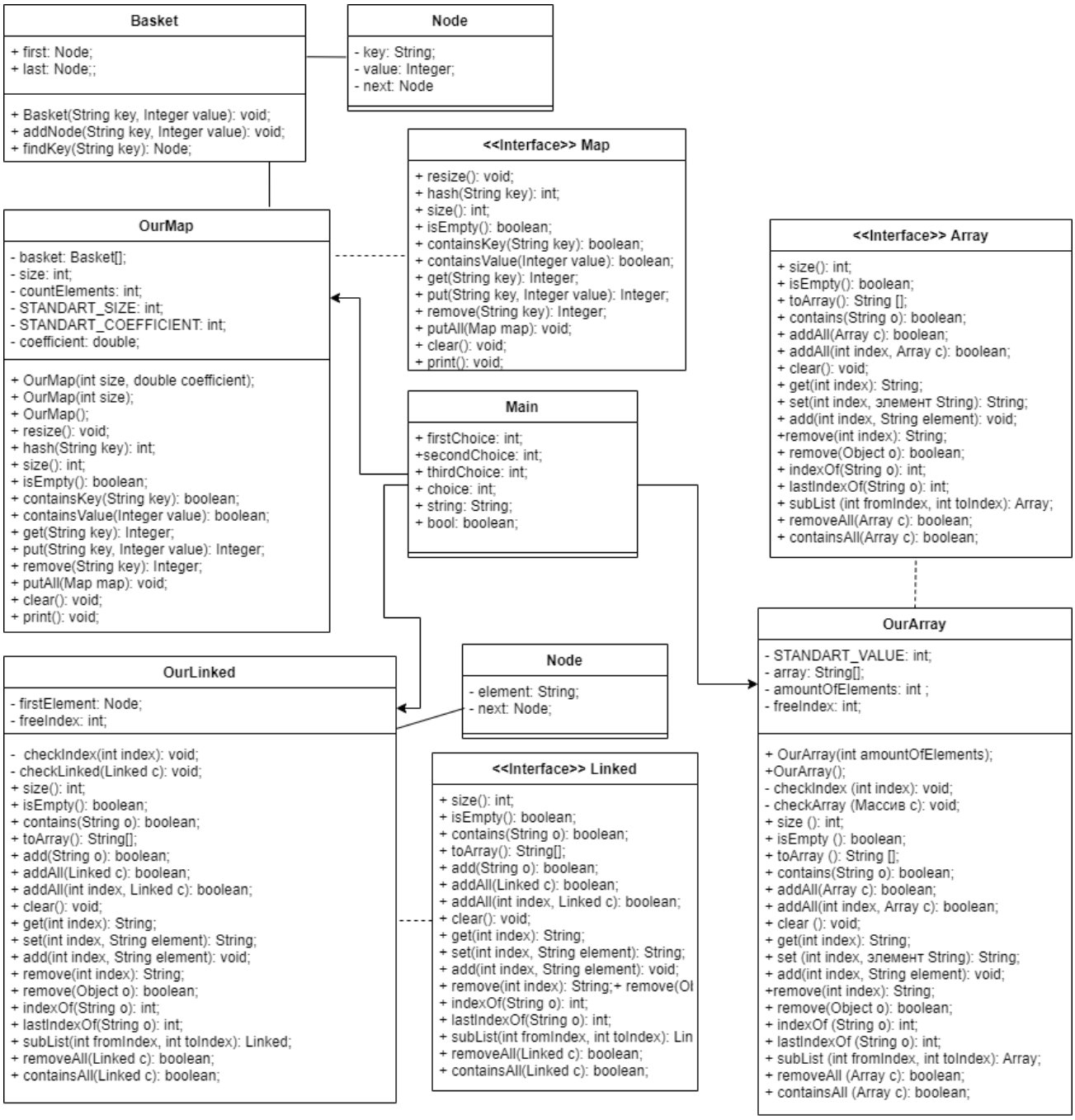

Рис 1 – URL диаграмма программы

3. ПРОЦЕСС ТЕСТИРОВАНИЯ

Процесс тестирования программы представлен в таблицах 1, 2, 3, 4.

Таблица 1 – Тестирование черным ящиком перед выбором коллекции.

№	Назначение теста	Входные данные	Реакция программы	Эталонный результат
1	Проверка выбора структур	ARRAY	Переход к добавлению элементов в Array	Переход к добавлению элементов в Array
2	Проверка выбора структур	no	Программа просит ввести название еще раз	Программа просит ввести название еще раз

Таблица 2 – Тестирование черным ящиком коллекции Array

№	Назначение теста	Входные данные	Реакция программы	Эталонный результат
1	Введение элементов №1	Line	Программа продолжает добавлять элементы	Программа продолжает добавлять элементы
2	Введение элементов №2	END	Программа перестает добавлять элементы и переходит к следующему шагу	Программа перестает добавлять элементы и переходит к следующему шагу
3	Проверка выводов всех элементов в консоль №1	да	Программа просит ввести выбор снова	Программа просит ввести выбор снова
4	Проверка выводов всех элементов в консоль №2	YES	Программа выводит элементы	Программа выводит элементы
5	Проверка вывода элемента по индексу №1	YES	Программа просит ввести индекс элемента для вывода.	Программа просит ввести индекс элемента для вывода.
6	Проверка вывода элемента по индексу №2	да	Программа просит ввести выбор снова	Программа просит ввести выбор снова
7	Проверка вывода по конкретному элементу	2	Программа выводит элемент по индексу и затраченное время	Программа выводит элемент по индексу и затраченное время

Таблица 3 – Тестирование черным ящиком коллекции Linked

1	Проверка выводов всех элементов в консоль	No	Выводится надпись «Вы не захотели получать элементы.»	Программа пропускает этот шаг
2	Проверка вывода элемента по индексу №1	yes	Программа выводит «Введите индекс элемента, который хотите получить:»	Программа выводит «Введите индекс элемента, который хотите получить:»
3	Проверка вывода элемента по индексу №2	what	Программа просит ввести выбор снова	Программа просит ввести выбор снова
4	Проверка вывода по конкретному элементу	a	Вывод надписи: «Введите число. »	Вывод надписи: «Введите число. »

Таблица 4 – Тестирование черным ящиком коллекции Map

1	Введение элементов на нечетное место	End	Программа перестает добавлять элементы	Программа перестает добавлять элементы
2	Введение элементов на четное место	3	Программа добавляет элемент	Программа добавляет элемент
3	Проверка выводов всех элементов в консоль	no	Выводится надпись «Попробуйте снова»	Выводится надпись «Попробуйте снова»
4	Проверка вывода элемента по ключу №1	3(существующий ключ)	Выводится элемент по ключу.	Выводится элемент по ключу.
5	Проверка вывода элемента по ключу №2	3(несуществующий ключ)	Выводится null и количество затраченного времени.	Выводится null и количество затраченного времени.

ЗАКЛЮЧЕНИЕ

В данной лабораторной работе были реализованы коллекции. В отличие же от массивов, Array и Linked содержат только ссылочные типы данных, то есть сохранить в них можно только объекты. Но в Java существуют механизмы автоупаковки и автораспаковки, которые позволяют хранить в динамических массивах примитивные типы. Динамический массив в Java используется для обработки наборов однородных данных, размер которых неизвестен на момент написания программы.

Map – это структура данных, которая содержит набор пар “ключ-значение”. По своей структуре данных напоминает словарь. Самая используемая её реализация – Hashmap. В нашей лабораторной работе мы как раз её и реализовывали. Главным превосходством этой коллекции является константное время поиска элемента по ключу (O(1)).

СПИСОК ЛИТЕРАТУРЫ

1. Класс ArrayList и интерфейс List // URL: https://metanit.com/java/tutorial/5.2.php (дата обращения 25.05.2021)
2. Класс LinkedList // URL: https://metanit.com/java/tutorial/5.3.php (дата обращения 25.05.2021)

ПРИЛОЖЕНИЕ А

Код Main.java

```
package com.company;

import java.util.Arrays;
import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        final int firstChoice = 1;
        final int secondChoice = 2;
        final int thirdChoice = 3;
        int choice = 0;

        System.out.println("какую структуру следует создать?
(ARRAY/LINKED/MAP)");
        String string = in.nextLine();

        if (string.equalsIgnoreCase("ARRAY")) {
            choice = firstChoice;
        } else if (string.equalsIgnoreCase("LINKED")) {
            choice = secondChoice;
        } else if (string.equalsIgnoreCase("MAP")) {
            choice = thirdChoice;
        } else {
            System.out.println("error.");
            System.exit(0);
        }

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

        switch (choice) {
            case firstChoice:
                OurArray arr = new OurArray();
                string = in.nextLine();
                while (!string.equalsIgnoreCase("END")) {
                    arr.add(string);
                    string = in.nextLine();
                }
                System.out.println("сейчас в структуре " +
arr.size() + " элементов.");
                System.out.println("хотите ли вы вывести все
элементы в консоль? (YES/NO)");
                string = in.nextLine();
                if (string.equalsIgnoreCase("YES")) {
                    StringBuffer buff = new StringBuffer();
                    String[] mass = arr.toArray();
                    for (int i = 0; i < arr.size(); ++i) {
                        buff.append(mass[i]);
```

```
                    buff.append(" ");
                }
                System.out.println(buff);
            } else if (!string.equalsIgnoreCase("NO")) {
                System.out.println("ошибка ввода.");
                break;
            }

            System.out.println("хотите ли вы получить
конкретный элемент по индексу?");
            string = in.nextLine();
            if (string.equalsIgnoreCase("YES")) {
                System.out.println("введите индекс элемента,
который хотите получить: ");
                int ind = in.nextInt();
                long start =
java.lang.System.currentTimeMillis();
                System.out.println(arr.get(ind));
                long fin =
java.lang.System.currentTimeMillis();
                long res = fin - start;
                System.out.println("затраченное время: " +
res);
            } else if (!string.equalsIgnoreCase("NO")) {
                System.out.println("ошибка ввода.");
                break;
            }
            break;

        case secondChoice:
            OurLinked lin = new OurLinked();
            string = in.nextLine();
            while (!string.equalsIgnoreCase("END")) {
                lin.add(string);
                string = in.nextLine();
            }

            System.out.println("сейчас в структуре " +
lin.size() + " элементов.");
            System.out.println("хотите ли вы вывести все
элементы в консоль? (YES/NO)");
            string = in.nextLine();
            if (string.equalsIgnoreCase("YES")) {

System.out.println(Arrays.toString(lin.toArray()));
            } else if (!string.equalsIgnoreCase("NO")) {
                System.out.println("ошибка ввода.");
                break;
            }

            System.out.println("хотите ли вы получить
конкретный элемент по индексу?");
            string = in.nextLine();
```

```
                if (string.equalsIgnoreCase("YES")) {
                    System.out.println("введите индекс элемента,
который хотите получить: ");
                    int ind = in.nextInt();
                    long start =
java.lang.System.currentTimeMillis();
                    System.out.println(lin.get(ind));
                    long fin =
java.lang.System.currentTimeMillis();
                    long res = fin - start;
                    System.out.println("затраченное время: " +
res);
                } else if (!string.equalsIgnoreCase("NO")) {
                    System.out.println("ошибка ввода.");
                    break;
                }
                break;

            case thirdChoice:
                OurMap map = new OurMap();
                do {
                    int value;
                    string = in.nextLine();
                    if (!string.equalsIgnoreCase("END")) {
                        value = in.nextInt();
                        in.nextLine();
                        map.put(string, value);
                    }
                } while (!string.equalsIgnoreCase("END"));

                System.out.println("сейчас в структуре " +
map.size() + " элементов.");
                System.out.println("хотите ли вы вывести все
элементы в консоль? (YES/NO)");
                string = in.nextLine();
                if (string.equalsIgnoreCase("YES")) {
                    map.print();
                } else if (!string.equalsIgnoreCase("NO")) {
                    System.out.println("ошибка ввода.");
                    break;
                }

                System.out.println("хотите ли вы получить
конкретный элемент по индексу?");
                string = in.nextLine();
                if (string.equalsIgnoreCase("YES")) {
                    System.out.println("введите ключ элемента,
который хотите получить: ");
                    String str = in.nextLine();
                    long start =
java.lang.System.currentTimeMillis();
                    System.out.println(map.get(str));
```

```
                    long fin =
java.lang.System.currentTimeMillis();
                    long res = fin - start;
                    System.out.println("затраченное время: " +
res);
                } else if (!string.equalsIgnoreCase("NO")) {
                    System.out.println("ошибка ввода.");
                    break;
                }
                break;
            default:
                break;
        }
    }
}
```

ПРИЛОЖЕНИЕ Б

Файл Array.java

```
package com.company;

interface Array {
    /**
     * Возвращает количество элементов в этом списке.
     * Если этот список содержит больше элементов, чем
Integer.MAX_VALUE, то возвращает Integer.MAX_VALUE.
     *
     * @return количество элементов в этом списке
     */
    int size();
    /**
     * Возвращает true, если этот список не содержит элементов.
     *
     * @return true, если этот список не содержит элементов, иначе
false
     */
    boolean isEmpty();
    /**
     * Возвращает true, если этот список содержит указанный
элемент.
     * Более формально, возвращает true тогда и только тогда,
когда этот список содержит
     * хотя бы один элемент e, такой что Objects.equals (o, e).
     *
     * @param o - элемент, наличие которого в этом списке
необходимо проверить
     * @return true, если этот список содержит указанный элемент
     */
    boolean contains(String o);
    /**
     * Возвращает массив, содержащий все элементы в этом списке в
правильной последовательности
     * (от первого до последнего элемента).
     * Возвращенный массив будет «безопасным» в том смысле, что в
этом списке не будет никаких ссылок на него.
     * (Другими словами, этот метод должен выделить новый массив,
даже если этот список поддерживается массивом).
     * Таким образом, вызывающий может изменить возвращаемый
массив.
     * Этот метод действует как мост между API на основе массива и
на основе коллекции.
     *
     * @return массив, содержащий все элементы в этом списке в
правильной последовательности
     */
    String[] toArray();
    /**
```

```
     * Добавляет указанный элемент в конец этого списка.
     *
     * @param o - элемент, который будет добавлен к этому списку
     * @return истина (как указано в Collection.add)
     */
    boolean add(String o);
    /**
     * Добавляет все элементы в указанной коллекции в конец этого
списка в том порядке.
     *
     * @param c коллекция, содержащая элементы, которые будут
добавлены в этот список
     * @return истина, если этот список изменился в результате
вызова
     */
    boolean addAll(Array c);
    /**
     * Вставляет все элементы указанной коллекции в этот список в
указанную позицию.
     * Сдвигает элемент, который в данный момент находится в этой
позиции (если есть), и любые последующие
     * элементы вправо (увеличивает их индексы). Новые элементы
появятся в этом списке в том порядке,
     * в котором они возвращаются итератором указанной коллекции.
     *
     * @param index - индекс, в который нужно вставить первый
элемент из указанной коллекции
     * @param c     коллекция, содержащая элементы, которые будут
добавлены в этот список
     * @return истина, если этот список изменился в результате
вызова
     */
    boolean addAll(int index, Array c);
    /**
     * Удаляет все элементы из этого списка.
     * Список будет пустым после того, как этот вызов вернется.
     */
    void clear();
    /**
     * Возвращает элемент в указанной позиции в этом списке.
     *
     * @param index - индекс возвращаемого элемента
     * @return элемент в указанной позиции в этом списке
     */
    String get(int index);
    /**
     * Заменяет элемент в указанной позиции в этом списке на
указанный элемент.
     *
     * @param index   - индекс заменяемого элемента
     * @param element - элемент, который будет сохранен в
указанной позиции
     * @return элемент ранее в указанной позиции
```

```
     */
    String set(int index, String element);
    /**
     * Вставляет указанный элемент в указанную позицию в этом
списке.
     * Сдвигает элемент, который в данный момент находится в этой
позиции (если есть),
     * и любые последующие элементы вправо (добавляет единицу к их
индексам).
     *
     * @param index   - индекс, по которому должен быть вставлен
указанный элемент
     * @param element - элемент для вставки
     */
    void add(int index, String element);
    /**
     * Удаляет элемент в указанной позиции в этом списке .
     * Сдвигает любые последующие элементы влево (вычитает единицу
из их индексов).
     * Возвращает элемент, который был удален из списка.
     *
     * @param index - индекс удаляемого элемента
     * @return элемент ранее в указанной позиции
     */
    String remove(int index);
    /**
     * Удаляет первое вхождение указанного элемента из этого
списка, если он присутствует.
     * Если в этом списке нет элемента, он остается неизменным.
     * Более формально удаляет элемент с наименьшим индексом i,
так что Objects.equals (o, get (i))
     * (если такой элемент существует). Возвращает истину, если
этот список содержит указанный элемент
     * (или, что эквивалентно, если этот список изменился в
результате вызова).
     *
     * @param o - элемент, который нужно удалить из этого списка,
если есть
     * @return истина, если этот список содержал указанный элемент
     */
    boolean remove(Object o);
    /**
     * Возвращает индекс первого вхождения указанного элемента в
этом списке или -1,
     * если этот список не содержит элемент.
     * Более формально, возвращает наименьший индекс i, такой что
Objects.equals (o, get (i)), или -1,
     * если такого индекса нет.
     *
     * @param o - элемент для поиска
     * @return индекс первого вхождения указанного элемента в этом
списке или -1, если этот список не содержит элемент
     */
```

```
    int indexOf(String o);
    /**
     * Возвращает индекс последнего вхождения указанного элемента
в этом списке или -1,
     * если этот список не содержит элемент.
     * Более формально возвращает наивысший индекс i,
     * такой что Objects.equals (o, get (i)), или -1, если такого
индекса нет.
     *
     * @param o - элемент для поиска
     * @return индекс последнего вхождения указанного элемента в
этом списке или -1,
     * если этот список не содержит элемент
     */
    int lastIndexOf(String o);
    /**
     * Возвращает представление части этого списка между указанным
fromIndex, включительно, и toIndex, исключая.
     * (Если fromIndex и toIndex равны, возвращаемый список пуст.)
     *
     * @param fromIndex нижняя конечная точка (включительно)
подсписка
     * @param toIndex   высокая конечная точка (исключая)
подсписка
     * @return вид указанного диапазона в этом списке
     */
    Array subList(int fromIndex, int toIndex);
    /**
     * Удаляет из этого списка все его элементы, содержащиеся в
указанной коллекции.
     *
     * @param c - коллекция, содержащая элементы, которые нужно
удалить из этого списка
     * @return истина, если этот список изменился в результате
вызова
     */
    boolean removeAll(Array c);
    /**
     * Возвращает true, если этот список содержит все элементы
указанной коллекции.
     *
     * @param c - коллекция, подлежащая проверке на наличие в этом
списке
     * @return истина, если этот список содержит все элементы
указанной коллекции
     */
    boolean containsAll(Array c);
}
```

ПРИЛОЖЕНИЕ В

Код OurArray.java

```
package com.company;

import java.util.Objects;

public class OurArray implements com.company.Array {

    private static final int STANDART_VALUE = 10;
    private String[] array;
    private int amountOfElements;
    private int freeIndex = 0;

    public OurArray(int amountOfElements) throws
IndexOutOfBoundsException {
        this.amountOfElements = amountOfElements;
        if (amountOfElements < 1) {
            throw new IndexOutOfBoundsException("Число меньше 1");
        }
        array = new String[amountOfElements];
    }

    public OurArray() {
        this.array = new String[STANDART_VALUE];
        amountOfElements = STANDART_VALUE;
    }

    private void checkIndex(int index) {
        if (index > freeIndex - 1 || index < 0) {
            throw new IndexOutOfBoundsException("Индекс " + index
+
                    " выходит за пределы массива размером " +
freeIndex);
        }
    }

    private void checkArray(Array c) {
        if (c == null) {
            throw new NullPointerException("Введенная вами
коллекция null");
        }
    }

    @Override
    public int size() {
        return freeIndex;
    }

    @Override
    public boolean isEmpty() {
```

```
        return freeIndex == 0;
    }

    @Override
    public boolean contains(String o) {
        for (int k = 0; k < freeIndex; ++k) {
            if (Objects.equals(array[k], o)) {
                return true;
            }
        }
        return false;
    }

    @Override
    public String[] toArray() {
        String[] newArray = new String[freeIndex];
        if (freeIndex >= 0) {
            System.arraycopy(array, 0, newArray, 0, freeIndex);
        }
        return newArray;
    }

    @Override
    public boolean add(String o) {
        if (freeIndex != amountOfElements) {
            array[freeIndex] = o;
        } else {
            String[] newArray = new String[freeIndex +
STANDART_VALUE];
            amountOfElements += STANDART_VALUE;
            System.arraycopy(array, 0, newArray, 0, freeIndex);
            newArray[freeIndex] = o;
            array = newArray;
        }
        ++freeIndex;
        return true;
    }

    @Override
    public boolean addAll(Array c) {

        checkArray(c);
        if (c.isEmpty()) {
            return false;
        }

        if ((freeIndex + c.size()) >= amountOfElements) {
            if ((c.size() + freeIndex) % amountOfElements == 0) {
                int count = c.size() / amountOfElements;
                amountOfElements += count * STANDART_VALUE;
            } else {
                int count = c.size() / amountOfElements + 1;
                amountOfElements += count * STANDART_VALUE;
```

```
            }
            String[] newArray = new String[amountOfElements];
            System.arraycopy(array, 0, newArray, 0, freeIndex);
            array = newArray;
        }

        for (int i = 0; i < c.size(); ++i) {
            this.add(c.get(i));
        }
        return true;
    }

    @Override
    public boolean addAll(int index, Array c) {

        checkIndex(index);
        checkArray(c);
        if (c.isEmpty()) {
            return false;
        }

        if ((freeIndex + c.size()) >= amountOfElements) {
            if ((c.size() + freeIndex) % amountOfElements == 0) {
                int count = c.size() / amountOfElements;
                amountOfElements += count * STANDART_VALUE;
            } else {
                int count = c.size() / amountOfElements + 1;
                amountOfElements += count * STANDART_VALUE;
            }
            String[] newArray = new String[amountOfElements];
            System.arraycopy(array, 0, newArray, 0, freeIndex);
            array = newArray;
        }

        for (int i = index; i < freeIndex; ++i) {
            this.array[i + c.size()] = this.array[i];
        }

        for (int i = 0; i < c.size(); ++i) {
            this.array[i + index] = c.get(i);
        }

        freeIndex += c.size();
        return true;
    }

    @Override
    public void clear() {
        array = new String[STANDART_VALUE];
        amountOfElements = STANDART_VALUE;
        freeIndex = 0;
    }
```

```
    @Override
    public String get(int index) {
        checkIndex(index);
        return array[index];
    }

    @Override
    public String set(int index, String element) {
        checkIndex(index);
        String line = array[index];
        array[index] = element;
        return line;
    }

    @Override
    public void add(int index, String element) {

        checkIndex(index);

        if (freeIndex != amountOfElements) {
            for (int i = freeIndex; i > index; i--) {
                array[i] = array[i - 1];
            }
            array[index] = element;
            ++freeIndex;
        } else if (freeIndex == amountOfElements) {
            ++freeIndex;
            String[] ourArray = new String[amountOfElements +
STANDART_VALUE];
            amountOfElements += STANDART_VALUE;
            System.arraycopy(array, 0, ourArray, 0, freeIndex -
1);
            array = ourArray;
            if (freeIndex != amountOfElements) {
                if (freeIndex - index >= 0) {
                    System.arraycopy(array, index, array, index +
1, freeIndex - index);
                }
                array[index] = element;
            }
        }
    }

    @Override
    public String remove(int index) {
        checkIndex(index);
        String line = array[index];
        System.arraycopy(array, index + 1, array, index, freeIndex
- index - 1);
        --freeIndex;
        return line;
    }
```

```
    @Override
    public boolean remove(Object o) {
        int size = freeIndex;
        for (int i = 0; i < size; ++i) {
            if (Objects.equals(array[i], o)) {
                System.arraycopy(array, i + 1, array, i, freeIndex
- i - 1);
                --freeIndex;
                return true;
            }
        }
        return false;
    }


    @Override
    public int indexOf(String o) {
        for (int i = 0; i < size(); i++) {
            if (Objects.equals(array[i], o)) {
                return i;
            }
        }
        return -1;
    }

    @Override
    public int lastIndexOf(String o) {
        int index = -1;
        int size = freeIndex;
        for (int i = 0; i < size; i++) {
            if (Objects.equals(array[i], o)) {
                index = i;
            }
        }
        return index;
    }

    @Override
    public Array subList(int fromIndex, int toIndex) {
        checkIndex(fromIndex);
        checkIndex(toIndex);
        Array newArray = new OurArray();
        int size = freeIndex;
        for (int i = size; i < toIndex; ++i) {
            newArray.add(array[i]);
        }
        return newArray;
    }

    @Override
    public boolean removeAll(Array c) {

        checkArray(c);
```

```
        if (c.isEmpty()) {
            return false;
        }

        int count = 0;
        int size = freeIndex;
        for (int j = size - 1; j >= 0; --j) {
            for (int i = 0; i < c.size(); ++i) {
                if (Objects.equals(array[j], c.get(i))) {
                    remove(j);
                    ++count;
                }
            }
        }
        return count != 0;
    }

    @Override
    public boolean containsAll(Array c) {

        checkArray(c);
        if (c.isEmpty()) {
            return false;
        }

        int i = 0;
        for (int j = 0; j < c.size(); ++j) {
            for (int k = 0; k < freeIndex; ++k) {
                if (Objects.equals(array[k], c.get(j))) {
                    ++i;
                }
            }
        }
        return i != c.size();
    }
}
```

ПРИЛОЖЕНИЕ Г

Код Linked.java

```
package com.company;

interface Linked {
    /**
     * Возвращает количество элементов в этом списке.
     * Если этот список содержит больше элементов, чем
Integer.MAX_VALUE, то возвращает Integer.MAX_VALUE.
     *
     * @return количество элементов в этом списке
     */
    int size();

    /**
     * Возвращает true, если этот список не содержит элементов.
     *
     * @return true, если этот список не содержит элементов, иначе
false
     */
    boolean isEmpty();

    /**
     * Возвращает true, если этот список содержит указанный
элемент.
     * Более формально, возвращает true тогда и только тогда,
когда этот список содержит
     * хотя бы один элемент e, такой что Objects.equals (o, e).
     *
     * @param o - элемент, наличие которого в этом списке
необходимо проверить
     * @return true, если этот список содержит указанный элемент
     */
    boolean contains(String o);

    /**
     * Возвращает массив, содержащий все элементы в этом списке в
правильной последовательности
     * (от первого до последнего элемента).
     * Возвращенный массив будет «безопасным» в том смысле, что в
этом списке не будет никаких ссылок на него.
     * (Другими словами, этот метод должен выделить новый массив,
даже если этот список поддерживается массивом).
     * Таким образом, вызывающий может изменить возвращаемый
массив.
     * Этот метод действует как мост между API на основе массива и
на основе коллекции.
     *
     * @return массив, содержащий все элементы в этом списке в
правильной последовательности
```

```
     */
    String[] toArray();

    /**
     * Добавляет указанный элемент в конец этого списка.
     *
     * @param o - элемент, который будет добавлен к этому списку
     * @return истина (как указано в Collection.add)
     */
    boolean add(String o);

    /**
     * Добавляет все элементы в указанной коллекции в конец этого
списка в том порядке.
     *
     * @param c коллекция, содержащая элементы, которые будут
добавлены в этот список
     * @return истина, если этот список изменился в результате
вызова
     */
    boolean addAll(Linked c);

    /**
     * Вставляет все элементы указанной коллекции в этот список в
указанную позицию.
     * Сдвигает элемент, который в данный момент находится в этой
позиции (если есть), и любые последующие
     * элементы вправо (увеличивает их индексы). Новые элементы
появятся в этом списке в том порядке,
     * в котором они возвращаются итератором указанной коллекции.
     *
     * @param index - индекс, в который нужно вставить первый
элемент из указанной коллекции
     * @param c     коллекция, содержащая элементы, которые будут
добавлены в этот список
     * @return истина, если этот список изменился в результате
вызова
     */
    boolean addAll(int index, Linked c);

    /**
     * Удаляет все элементы из этого списка.
     * Список будет пустым после того, как этот вызов вернется.
     */
    void clear();

    /**
     * Возвращает элемент в указанной позиции в этом списке.
     *
     * @param index - индекс возвращаемого элемента
     * @return элемент в указанной позиции в этом списке
     */
    String get(int index);
```

```
    /**
     * Заменяет элемент в указанной позиции в этом списке на
указанный элемент.
     *
     * @param index   - индекс заменяемого элемента
     * @param element - элемент, который будет сохранен в
указанной позиции
     * @return элемент ранее в указанной позиции
     */
    String set(int index, String element);

    /**
     * Вставляет указанный элемент в указанную позицию в этом
списке.
     * Сдвигает элемент, который в данный момент находится в этой
позиции (если есть),
     * и любые последующие элементы вправо (добавляет единицу к их
индексам).
     *
     * @param index   - индекс, по которому должен быть вставлен
указанный элемент
     * @param element - элемент для вставки
     */
    void add(int index, String element);

    /**
     * Удаляет элемент в указанной позиции в этом списке .
     * Сдвигает любые последующие элементы влево (вычитает единицу
из их индексов).
     * Возвращает элемент, который был удален из списка.
     *
     * @param index - индекс удаляемого элемента
     * @return элемент ранее в указанной позиции
     */
    String remove(int index);

    /**
     * Удаляет первое вхождение указанного элемента из этого
списка, если он присутствует.
     * Если в этом списке нет элемента, он остается неизменным.
     * Более формально удаляет элемент с наименьшим индексом i,
так что Objects.equals (o, get (i))
     * (если такой элемент существует). Возвращает истину, если
этот список содержит указанный элемент
     * (или, что эквивалентно, если этот список изменился в
результате вызова).
     *
     * @param o - элемент, который нужно удалить из этого списка,
если есть
     * @return истина, если этот список содержал указанный элемент
     */
    boolean remove(Object o);
```

```
    /**
     * Возвращает индекс первого вхождения указанного элемента в
этом списке или -1,
     * если этот список не содержит элемент.
     * Более формально, возвращает наименьший индекс i, такой что
Objects.equals (o, get (i)), или -1,
     * если такого индекса нет.
     *
     * @param o - элемент для поиска
     * @return индекс первого вхождения указанного элемента в этом
списке или -1, если этот список не содержит элемент
     */
    int indexOf(String o);

    /**
     * Возвращает индекс последнего вхождения указанного элемента
в этом списке или -1,
     * если этот список не содержит элемент.
     * Более формально возвращает наивысший индекс i,
     * такой что Objects.equals (o, get (i)), или -1, если такого
индекса нет.
     *
     * @param o - элемент для поиска
     * @return индекс последнего вхождения указанного элемента в
этом списке или -1,
     * если этот список не содержит элемент
     */
    int lastIndexOf(String o);

    /**
     * Возвращает представление части этого списка между указанным
fromIndex, включительно, и toIndex, исключая.
     * (Если fromIndex и toIndex равны, возвращаемый список пуст.)
     *
     * @param fromIndex нижняя конечная точка (включительно)
подсписка
     * @param toIndex   высокая конечная точка (исключая)
подсписка
     * @return вид указанного диапазона в этом списке
     */
    Linked subList(int fromIndex, int toIndex);

    /**
     * Удаляет из этого списка все его элементы, содержащиеся в
указанной коллекции.
     *
     * @param c - коллекция, содержащая элементы, которые нужно
удалить из этого списка
     * @return истина, если этот список изменился в результате
вызова
     */
    boolean removeAll(Linked c);
```

```
    /**
     * Возвращает true, если этот список содержит все элементы
указанной коллекции.
     *
     * @param c - коллекция, подлежащая проверке на наличие в этом
списке
     * @return истина, если этот список содержит все элементы
указанной коллекции
     */
    boolean containsAll(Linked c);
}
```

ПРИЛОЖЕНИЕ Д

Код OurLinked.java

```
package com.company;

import java.util.Objects;

public class OurLinked implements com.company.Linked {

    private Node firstElement;
    private int freeIndex = 0;

    private static class Node {
        private String element;
        private Node next;

        Node(String element) {
            this.element = element;
            next = null;
        }
    }

    private void checkIndex(int index) {
        if (index > freeIndex - 1 || index < 0) {
            throw new IndexOutOfBoundsException("Индекс " + index
+
                    " выходит за пределы массива размером " +
freeIndex);
        }
    }

    private void checkLinked(Linked c) {
        if (c == null) {
            throw new NullPointerException("Введенная вами
коллекция null");
        }
    }

    @Override
    public int size() {
        return freeIndex;
    }

    @Override
    public boolean isEmpty() {
        return freeIndex == 0;
    }

    @Override
    public boolean contains(String o) {
        for (int i = 0; i < freeIndex; i++) {
```

```
        if (Objects.equals(get(i), o)) {
            return true;
        }
    }
    return false;
}

@Override
public String[] toArray() {
    String[] newArray = new String[freeIndex];
    Node el = firstElement;
    for (int i = 0; i < freeIndex; i++) {
        newArray[i] = el.element;
        el = el.next;
    }
    return newArray;
}

@Override
public boolean add(String o) {
    Node newNode = new Node(o);
    if (firstElement == null) {
        firstElement = newNode;
        ++freeIndex;
    } else {
        Node last = firstElement;
        while (last.next != null) {
            last = last.next;
        }
        ++freeIndex;
        last.next = newNode;
    }
    return true;
}

@Override
public boolean addAll(Linked c) {
    checkLinked(c);
    if (c.isEmpty()) {
        return false;
    }
    int size = c.size();
    for (int i = 0; i < size; ++i) {
        add(c.get(i));
    }
    return true;
}

@Override
public boolean addAll(int index, Linked c) {

    checkIndex(index);
    checkLinked(c);
```

```
    if (c.isEmpty()) {
        return false;
    }

    for (int i = index; i < c.size() + index; ++i) {
        add(i, c.get(i - index));
    }
    return true;
}

@Override
public void clear() {
    firstElement = null;
    freeIndex = 0;
}

@Override
public String get(int index) {

    checkIndex(index);

    Node buff = firstElement;
    for (int i = 0; i < index; i++) {
        buff = buff.next;
    }
    return buff.element;
}

@Override
public String set(int index, String element) {

    checkIndex(index);

    String line = get(index);
    Node newNode = new Node(element);
    if (index == 0) {
        newNode.next = firstElement.next;
        firstElement = newNode;
    } else {
        Node buff = firstElement;
        Node superBuff = firstElement;
        for (int i = 0; i < index; i++) {
            superBuff = buff;
            buff = superBuff.next;
        }
        superBuff.next = newNode;
        newNode.next = buff.next;
    }
    return line;
}

@Override
public void add(int index, String element) {
```

```
    checkIndex(index);

    Node newNode = new Node(element);
    if (index < freeIndex) {
        if (index == 0) {
            newNode.next = firstElement;
            firstElement = newNode;
        } else {
            Node buff = firstElement;
            for (int i = 0; i < index - 1; i++) {
                buff = buff.next;
            }
            Node superBuff = buff.next;
            buff.next = newNode;
            newNode.next = superBuff;
        }
        ++freeIndex;
    }
}

@Override
public String remove(int index) {

    checkIndex(index);

    String line = get(index);
    if (index == 0) {
        firstElement = firstElement.next;
        --freeIndex;
    } else {
        Node buff = firstElement;
        Node newBuff = firstElement;
        for (int i = 0; i < index; i++) {
            newBuff = buff;
            buff = newBuff.next;
        }
        newBuff.next = buff.next;
        --freeIndex;
    }
    return line;
}

@Override
public boolean remove(Object o) {
    for (int i = 0; i < freeIndex; ++i) {
        if (Objects.equals(o, get(i))) {
            remove(i);
            return true;
        }
    }
    return false;
}
```

```
@Override
public int indexOf(String o) {
    for (int i = 0; i < freeIndex; i++) {
        if (Objects.equals(get(i), o)) {
            return i;
        }
    }
    return -1;
}

@Override
public int lastIndexOf(String o) {
    int index = -1;
    for (int i = 0; i < freeIndex; i++) {
        if (Objects.equals(get(i), o)) {
            index = i;
        }
    }
    return index;
}

@Override
public Linked subList(int fromIndex, int toIndex) {

    checkIndex(fromIndex);
    checkIndex(toIndex);

    Linked newArray = new OurLinked();
    for (int i = fromIndex; i < toIndex; ++i) {
        newArray.add(get(i));
    }
    return newArray;
}

@Override
public boolean removeAll(Linked c) {

    checkLinked(c);
    if (c.isEmpty()) {
        return false;
    }

    int oldSize = freeIndex;
    for (int i = 0; i < c.size(); ++i) {
        do {
            remove(c.get(i));
        } while (contains(c.get(i)));
    }
    return oldSize != freeIndex;
}

@Override
```

```
    public boolean containsAll(Linked c) {

        checkLinked(c);
        if (c.isEmpty()) {
            return false;
        }

        int j = 0;
        for (int i = 0; i < c.size(); i++) {
            if (contains(c.get(i))) {
                j++;
            }
        }
        return j == c.size();
    }
}
```

ПРИЛОЖЕНИЕ Е

Код Map.java

```
package com.company;


interface Map {

    /**
     * Возвращает количество сопоставлений "ключ-значение" на этой
карте.
     * Если карта содержит более элементов Integer.MAX_VALUE,
возвращает Integer.MAX_VALUE.
     *
     * @return количество сопоставлений "ключ-значение" на этой
карте
     */
    int size();

    /**
     * Возвращает истину, если эта карта не содержит сопоставлений
"ключ-значение".
     *
     * @return истина, если эта карта не содержит сопоставлений
"ключ-значение"
     */
    boolean isEmpty();

    /**
     * Возвращает истину, если эта карта содержит отображение для
указанного ключа.
     * Более формально, возвращает истину тогда и только тогда,
когда эта карта содержит отображение для ключа k,
     * такое что Objects.equals (key, k). (Может быть не более
одного такого сопоставления.)
     *
     * @param key - ключ, наличие которого в этой карте должно
быть проверено
     * @return истина, если эта карта содержит отображение для
указанного ключа
     */
    boolean containsKey(String key);

    /**
     * Возвращает истину, если эта карта сопоставляет один или
несколько ключей с указанным значением.
     * Более формально, возвращает true тогда и только тогда,
когда эта карта содержит хотя бы одно отображение
     * на значение v, такое что Objects.equals (value, v). Эта
операция, вероятно, потребует линейного времени
     * по размеру карты для большинства реализаций интерфейса Map.
```

```
     *
     * @param value, наличие которого в этой карте должно быть
проверено
     * @return истина, если эта карта сопоставляет один или
несколько ключей с указанным значением
     */
    boolean containsValue(Integer value);

    /**
     * Возвращает значение, которому сопоставлен указанный ключ,
или null, если эта карта не содержит
     * сопоставления для ключа.
     * Более формально, если эта карта содержит отображение ключа
k на значение v,
     * такое что Objects.equals (key, k), то этот метод возвращает
v; в противном случае возвращается null.
     * (Может быть не более одного такого сопоставления.)
     * Если эта карта допускает значения NULL, то возвращаемое
значение NULL не обязательно означает,
     * что карта не содержит сопоставления для ключа; также
возможно, что карта явно отображает ключ в null.
     * Операция containsKey может использоваться для различения
этих двух случаев.
     *
     * @param key - ключ, связанное значение которого должно быть
возвращено
     * @return значение, которому сопоставлен указанный ключ, или
null, если эта карта не содержит сопоставления для ключа
     */
    Integer get(String key);

    /**
     * Связывает указанное значение с указанным ключом на этой
карте.
     * Если карта ранее содержала сопоставление для ключа, старое
значение заменяется указанным значением.
     * (Считается, что карта m содержит отображение для ключа k
тогда и только тогда, когда m.containsKey (k) вернет true.)
     *
     * @param key   ключ, с которым должно быть связано указанное
значение
     * @param value значение, которое будет связано с указанным
ключом
     * @return предыдущее значение, связанное с ключом, или null,
если для ключа не было сопоставления. (Возврат null также может
указывать на то, что карта ранее ассоциировала null с ключом, если
реализация поддерживает нулевые значения.)
     */
    Integer put(String key, Integer value);

    /**
     * Удаляет сопоставление для ключа с этой карты, если оно
присутствует. Более формально,
```

```
     * если эта Map содержит отображение из ключа k в значение v,
такое, что Objects.equals (key, k),
     * это отображение удаляется. (Карта может содержать не более
одного такого сопоставления.)
     * Возвращает значение, с которым эта Map ранее связала key,
или null,
     * если Map не содержала сопоставления для ключа.
     * Если эта Map допускает значения NULL, то возвращаемое
значение NULL не обязательно означает,
     * что Map не содержала сопоставления для ключа; также
возможно,
     * что Map явно сопоставила key со значением null.
     * Карта не будет содержать сопоставление для указанного ключа
после возврата вызова.
     *
     * @param key, отображение которого нужно удалить с карты
     * @return предыдущее значение, связанное с ключом, или null,
если для ключа не было сопоставления.
     */
    Integer remove(String key);

    /**
     * Копирует все сопоставления с указанной карты на эту карту.
     * Эффект этого вызова эквивалентен вызову put (k, v) на этой
карте один раз для каждого отображения ключа
     * k в значение v в указанной карте.
     *
     * @param map - отображения, которые будут храниться в этой
map
     */
    void putAll(Map map);

    /**
     * Удаляет все сопоставления с этой карты.
     * Карта будет пустой после того, как этот вызов вернется.
     */
    void clear();
}
```

ПРИЛОЖНЕНИЕ Ж

Файл OurMap.java

```
package com.company;

import java.util.Objects;

public class OurMap implements com.company.Map {

    private Basket[] basket;
    private int size;
    private int countElements;
    private double coefficient;
    private static final int STANDART_SIZE = 10;
    private static final double STANDART_COEFFICIENT = 0.75;

    public OurMap(int size, double coefficient) {
        if (size <= 0 || coefficient <= 0) {
            throw new IllegalArgumentException("Поля должны быть
больше 0");
        } else {
            this.coefficient = coefficient;
            this.size = size;
            this.basket = new Basket[size];
            countElements = 0;
        }
    }

    public OurMap(int size) {
        this(size, STANDART_COEFFICIENT);
    }

    public OurMap() {
        this(STANDART_SIZE, STANDART_COEFFICIENT);
    }

    private void resize() {
        if (countElements >= size * coefficient) {
            int outSize = size;
            size = size + countElements;
            countElements = 0;

            Basket[] massive = basket;
            this.basket = new Basket[size];
            for (int i = 0; i < outSize; ++i) {
                Basket basket = massive[i];
                if (basket == null) {
                    continue;
                }
                Node node = basket.first;
                while (node != null) {
```

```
                this.put(node.key, node.value);
                node = node.next;
            }
        }
    }
}

private static class Node {
    private String key;
    private Integer value;
    private Node next;

    Node(String key, Integer value) {
        this.key = key;
        this.value = value;
        next = null;
    }
}

// суммирует все значения по ASCII и делит на введенный размер
private int hash(String key) {

    int sum = 0;

    if (key == null) {
        return 0;
    }

    for (int i = 0; i < key.length(); i++) {
        sum += key.charAt(i);
    }
    return sum % size;
}

private class Basket {

    Node first;
    Node last;

    public Basket(String key, Integer value) {
        first = new Node(key, value);
        last = first;
    }

    public void addNode(String key, Integer value) {
        last.next = new Node(key, value);
        last = last.next;
    }

    public Node findKey(String key) {

        Node seeker = this.first;
```

```
        while (!Objects.equals(key, seeker.key)) {
            seeker = seeker.next;
            if (seeker == null) {
                return null;
            }
        }
        return seeker;
    }
}

@Override
public int size() {
    return countElements;
}

@Override
public boolean isEmpty() {
    return countElements == 0;
}

@Override
public boolean containsKey(String key) {

    int index = hash(key);

    if (basket[index] == null) {
        return false;
    }
    if (basket[index].findKey(key) == null) {
        return false;
    }
    return true;
}

@Override
public boolean containsValue(Integer value) {

    if (this.isEmpty()) {
        return false;
    }

    for (Basket o : basket) {
        if (o == null) {
            continue;
        }
        Node node = o.first;
        while (node != null) {
            if ((Objects.equals(node.value, value))) {
                return true;
            } else {
                node = node.next;
            }
        }
```

```
    }
    return false;
}

@Override
public Integer get(String key) {

    int index = hash(key);

    if (basket[index] == null) {
        return null;
    }

    Node find = basket[index].findKey(key);

    if (find == null) {
        return null;
    } else {
        return find.value;
    }
}

@Override
public Integer put(String key, Integer value) {

    int index = hash(key);

    if (basket[index] == null) {
        basket[index] = new Basket(key, value);
        ++countElements;
        this.resize();
        return null;
    } else {
        Node find = basket[index].findKey(key);
        if (find == null) {
            basket[index].addNode(key, value);
            ++countElements;
            this.resize();
            return null;
        } else {
            int prevValue = find.value;
            find.value = value;
            return prevValue;
        }
    }
}

@Override
public Integer remove(String key) {

    int index = hash(key);
    Basket bask = basket[index];
    if (bask == null) {
```

```
            return null;
        }
        Node node = bask.first;
        Node remover = node;
        Node helper = remover;

        while (remover != null) {
            if (Objects.equals(remover.key, key)) {
                int prevValue = remover.value;
                if (bask.last == bask.first) {
                    bask.first = null;
                    basket[index] = null;
                    --countElements;
                    return prevValue;
                }
                helper.next = remover.next;
                basket[index].first = node;
                --countElements;
                return prevValue;
            }
            helper = remover;
            remover = remover.next;
        }
        return null;
    }

    @Override
    public void putAll(Map map) {

        if (map == null) {
            throw new NullPointerException("Введенная вами
коллекция null");
        }

        OurMap mapa = (OurMap) map;
        Basket[] massive = mapa.basket;

        for (int i = 0; i < mapa.size; ++i) {
            Basket basket = massive[i];
            if (basket == null) {
                continue;
            }
            Node node = basket.first;
            while (node != null) {
                this.put(node.key, node.value);
                node = node.next;
            }
        }
    }

    @Override
    public void clear() {
        for (int i = 0; i < size; i++) {
```

```
            basket[i] = null;
        }
        countElements = 0;
    }

    public void print() {
        for (int i = 0; i < size; i++) {
            if (basket[i] != null) {
                Node printer = basket[i].first;
                while (printer != null) {
                    System.out.println(printer.key + "->" +
printer.value);
                    printer = printer.next;
                }
            }
        }
    }
}
```