МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ

Ордена Трудового Красного Знамени

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Московский технический университет связи и информатики»

Кафедра «Программная инженерия»

Отчёт по Лабораторной работе №6

По дисциплине: «Информационные технологии и программирование»

«Работа с коллекциями»

Выполнила: студентка группы БПИ2401

Алексеева Татьяна Игоревна

Проверил: Харрасов Камиль Раисович

Москва

2025

Цель работы

Изучить и практически освоить работу с коллекциями языка Java, а также приобрести навыки использования стандартных коллекций (Map, ArrayList) и обобщённых классов для хранения, обработки и анализа данных.

Выполнение работы

Задание 1. Написать программу, которая считывает текстовый файл и выводит на экран топ-10 самых часто встречающихся слов в этом файле. Для решения задачи использовать коллекцию Map, где ключом будет слово, а значением — количество его повторений в файле.

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.*;

public class TopWords {
    public static void main(String[] args) {
        String filepath = "text.txt";
        File file = new File(filepath);

        Scanner s = null;
        try {
            s = new Scanner(file);
        } catch (FileNotFoundException e) {
            System.out.println("Файл не найден");
            return;
        }

        Map<String, Integer> wordCounter = new HashMap<>();

        while (s.hasNext()) {
            String word = s.next().toLowerCase();

            word = word.replaceAll("[^a-zA-Za-яА-Я]", "");

            if (word.isEmpty()) {
                continue;
            }

            wordCounter.put(word, wordCounter.getOrDefault(word, 0) + 1);
        }

        s.close();

        List<Map.Entry<String, Integer>> list = new ArrayList<>(wordCounter.entrySet());

        Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
            @Override
            public int compare(Map.Entry<String, Integer> o1,
```

```
                                Map.Entry<String, Integer> o2) {
                return o2.getValue().compareTo(o1.getValue());
            }
        });

        System.out.println("Топ 10 самых частых слов");
        int count = 0;
        for (Map.Entry<String, Integer> entry : list) {
            System.out.println(entry.getKey() + " - " + entry.getValue());
            count++;
            if (count == 10) {
                break;
            }
        }
    }

}
```

В первую очередь импортируем все необходимые библиотеки: File нужна, чтобы работать с файлом на компьютере, FileNotFoundException – ошибка, если файл не найден, java.util.* подключает Scanner, Map, HashMap, List, ArrayList, Collections, Comparator.

Далее указываем путь к файлу, filepath – прописано сразу имя файла (так текстовый файл и код лежат в одной папке), File file – создаём объект, который указывает на файл.

Используем Scanner для чтения файла, если файла не будет, то new Scanner(file) может вызвать ошибку, поэтому обрабатываем её с помощью try-catch.

Создаем Map для подсчёта слов. Map – это таблица из пар: ключ (слово) – значение (число, сколько раз встретилось). String – слово, Integer – количество повторений. Мы считаем частоту слов, поэтому Map отлично подходит.

Читаем файл по словам. hasNext() – пока в файле есть слова, цикл идёт слово за словом. String word = s.nest().toLowerCase() – берём следующее слово и переводим его в нижний регистр. Также убираем ненужные знаки препинания. Если после очистки слово стало пусты, то пропускаем его.

Далее идёт строка wordCounter.put(word, wordCounter.getOrDefault(word, 0) + 1); Здесь getOrDefault(word, 0) проверяет, если уже есть слово, возвращается его количество, если слова нет, возвращается 0. +1 увеличивает счётчик. put(word, …) сохраняем новое значение в Map.

В конце обработки файла закрываем Scanner.

Так как Map (таблицу) нельзя сортировать, то преобразовываем её в List (лист), который уже можно отсортировать. entrySet() превращает Map в набор пар.

Далее идёт сортировка списка. Collections.sort сортирует список, а Comparator говорит, как сортировать. o1.getValue() – количество первого слова, o2.getValue() – количество второго слова, o2 - o1 - сортировка по убыванию.

И наконец, выводим топ 10 самых частых слов. Перебираем отсортированный список, берём слово и сколько раз встречается. Если count становится равен 10 завершаем цикл.

```
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkSix> javac .\TopWords.java
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkSix> java TopWords
Топ 10 самых частых слов
назад - 17
где - 13
в - 9
пора - 9
там - 7
будущее - 5
но - 4
мне - 4
будущего - 3
и - 3
PS C:\Users\taale\OneDrive\Desktop\ITandP\LabWorkSix>
```

Задание 2. Написать обобщенный класс Stack< T> , который реализует стек на основе массива. Класс должен иметь методы push для добавления элемента в стек, pop для удаления элемента из стека и peek для получения верхнего элемента стека без его удаления.

```java
public class Stack<T> {
    private T[] data;
    private int size;

    public Stack(int capacity) {
        data = (T[]) new Object[capacity];
        size = 0;
    }

    public void push(T element) {
        if (size == data.length) {
            throw new RuntimeException("Стек переполнен");
        }
        data[size] = element;
        size++;
    }

    public T pop() {
```

```
        if (size == 0) {
            throw new RuntimeException("Стек пуст");
        }
        T element = data[size - 1];
        data[size - 1] = null;
        size--;
        return element;
    }

    public T peek() {
        if (size == 0) {
            throw new RuntimeException("Стек пуст");
        }
        return data[size - 1];
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Stack<Integer> stack = new Stack<>(10);

        stack.push(1);
        stack.push(2);
        stack.push(3);

        System.out.println(stack.pop());
        System.out.println(stack.peek());

        stack.push(4);
        System.out.println(stack.pop());
    }
}
```

Объявляем класс Stack в качестве типа данных используем джейнерик <T>, как бы говорим, что пока не знаем, какие именно данные будем хранить, но пользователь скажет об этом позже. T – как бы заглушка для типа данных.

Объявляем поля класса: data – массив, в котором хранятся элемента стека, тип T[], потому что стек обобщённый, size показывает, сколько элементов сейчас в стеке, также указывает, где храниться вершина стека.

Далее прописывает конструктор, где capacity – максимальное количество элементов в стеке, размер массива. Создаём массив Object, приводя его к T[]. Сразу после создания стека пустой, в массиве пока нет ни одного элемента, поэтому size = 0.

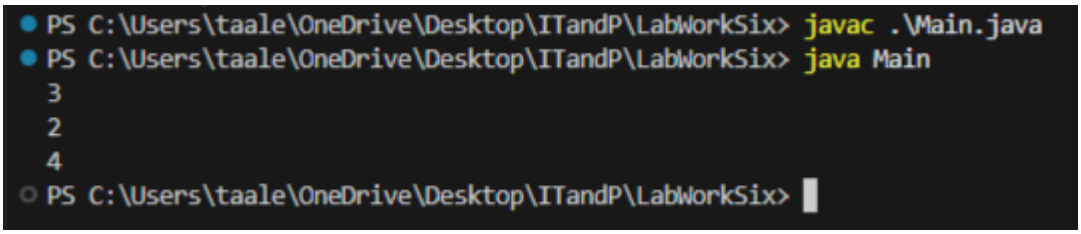
Прописываем метод push – добавление элемента, он кладёт элемент в стек, тип элемента – T (любой). Проверяем переполнение массива. data.length – максимальный размер массива, если size равен длине массива, значит места

больше нет. При добавлении элемент кладётся в конец массива, и size увеличивается на 1.

Метод pop – удаление элемента. Этот метод «снимает» верхний элемент и возвращает его. Сначала идёт проверка на пустоту массива. Верхний элемент лежит на позиции size – 1. Очищаем ячейку массива, уменьшаем размер стека, возвращаем удалённые элемент.

Метод peek – просмотр вершины, он показывает верхний элемент, но не удаляет его. Возвращаем элемент на позиции size – 1.

Стек используем в отдельном файле в классе Main. Добавляем элементы, удаляем вершину, возвращаем текущую вершину после удаления предыдущей.



Задание 3. Разработать программу для учета продаж в магазине. Программа должна позволять добавлять проданные товары в коллекцию, выводить список проданных товаров, а также считать общую сумму продаж и наиболее популярный товар. Использовать ArrayList для хранения списка проданных товаров.

```
public class Product {
    private String name;
    private double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    @Override
    public String toString() {
        return name + " - " + price + " руб.";
    }
```

```
}
```

```
public class MainTwo { import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

public class Shop {
    private ArrayList<Product> soldProducts = new ArrayList<>();

    public void addProduct(Product product) {
        soldProducts.add(product);
    }

    public void printProducts() {
        System.out.println("Проданные товары:");
        for (Product product : soldProducts) {
            System.out.println(product);
        }
    }

    public double getTotalSales() {
        double summa = 0;
        for (Product product : soldProducts) {
            summa += product.getPrice();
        }
        return summa;
    }

    public String getMostPopularProduct() {
        if (soldProducts.isEmpty()) {
            return "Нет продаж";
        }

        Map<String, Integer> counter = new HashMap<> ();

        for (Product product : soldProducts) {
            counter.put(product.getName(), counter.getOrDefault(product.getName(), 0) + 1);
        }

        String popular = null;
        int maCounter = 0;

        for (Map.Entry<String, Integer> entry : counter.entrySet()) {
            if (entry.getValue() > maCounter) {
                maCounter = entry.getValue();
                popular = entry.getKey();
            }
        }

        return popular;
```

```
    }
}

    public static void main(String[] args) {
        Shop shop = new Shop();

        shop.addProduct(new Product("Хлеб", 50));
        shop.addProduct(new Product("Молоко", 90));
        shop.addProduct(new Product("Хлеб", 50));
        shop.addProduct(new Product("Сыр", 300));
        shop.addProduct(new Product("Хлеб", 50));

        shop.printProducts();

        System.out.println("Общая сумма продаж = " + shop.getTotalSales() + " руб.");
        System.out.println("Самый популярный товар: " + shop.getMostPopularProduct());
    }

}
```

Создаём класс Product, в котором будем хранить название товара (поле name), его цену (price). Прописываем конструктор, чтобы создавать объекты класса Product. Также прописывает геттеры, чтобы другие классы могли получать имя и цену товара.

Создаём класс Shop, который будет хранить объекты типа класса Product в списке (динамическом классе). soldProducts – это список всех проданных товаров.

Метод addProduct добавляет объект типа Product в список soldProducts. Метод add у ArrayList добавляет элемент в конец списка.

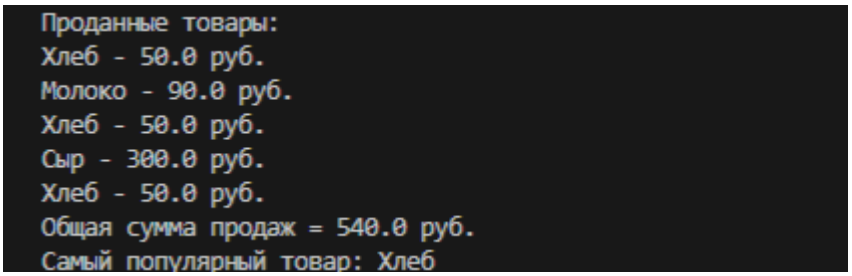
Чтобы вывести все проданные товары, проходимся циклом for-each по каждому товару в списке, выводим на консоль, используя метод toString() у объекта Product.

Чтобы подсчитать общую сумму продаж, объявляем переменную summa. Проходимся по каждому товару в списке и прибавляем его цену к summa. Возвращаем итоговую сумму.

Метод нахождения самого популярного товара. Сначала проверяем, не пустой ли список продаж, если товаров нет, выводим соответствующее сообщение. Создаём таблицу (HashMap) для подсчёта количества каждого товара: ключ – название товара (String), значение – сколько раз он продан (Integer). Заполняем таблицу counter.

Чтобы найти товар с наибольшим количеством продаж создаём переменные popular (название товара) и maxCounter (сколько раз встречается). Проходимя по каждой паре (товар, количество), если текущее количество больше maxCounter, обновляем popular и maxCounter.

В классе MainTwo показываем работу программы.



Вывод

В ходе выполнения лабораторной работы была достигнута цель – изучение и практическое освоение работы с коллекциями языка Java, а также приобретение навыков использования стандартных коллекций Map и ArrayList, а также обобщенных классов для хранения, обработки и анализа данных.

Ответы на контрольные вопросы

1. Основные интерфейсы коллекций в Java: Navigable Set, SortedSet, Deque, List, Collection, Iterable; Navigable Map, SortedMap, Map.
2. Основные классы коллекций в Java: Linked HashSet, HashMap, TreeMap, AbstractMap; Linked List, Linked HashSet, Abstract SecuentialList, ArrayList, MathSet, Priority Queue, Abstract List, Abstract Set, Abstract Queue, Array Queue, Abstract Collection.
3. Итератор в Java – это объект, который позволяет последовательно проходить (перебирать) элементы коллекции, не раскрывая внутреннюю структуру самой коллекции. (hasNext() – проверяет, есть ли ещё элементы для перебора, next() – возвращает следующий элемент коллекции и продвигает итератор вперёд, remove() – удаляет элемент, на который указывает итератор).
4. Map - представляет ассоциативный массив, где каждому ключу соответствует одно значение. Основные реализации: HashMap, TreeMap. Ключ – уникальный идентификатор элемента, не может быть два одинаковых ключа, значение – данные, связанные с ключом.
5. List – представляет упорядоченную коллекцию элементов, допускающую дубликаты. Основные реализации: ArrayList, LinkedList. Элементы можно получать по индексу.

6. Set — представляет коллекцию, в которой каждый элемент уникален. Основные реализации: HashSet, TreeSet. Порядок не сохраняется.
7. Есть несколько способов синхронизации коллекций в Java: классе Collections есть методы для создания синхронизированных обёрток над коллекциями – Collections.synchronizedXXX(), Java предоставляет специализированные конкурентные коллекции, которые уже безопасны для многопоточного доступа, можно синхронизировать доступ к коллекции вручную через блок synchronized.
8. Интерфейс Collection<E> в Java является базовым для большинства коллекций (кроме Map) и задаёт общие методы, которые позволяют работать с элементами коллекции: boolean add(E, e) – добавляет элемент в коллекцию, возвращает true, если элемент был добавлен, boolean remove(Object o) – удаляет один экземпляр указанного элемента, void clear() – удаляет все элементы коллекции, boolean contains(Object o) – проверяет, есть ли указанный элемент в коллекции, boolean isEmpty() – проверяет, пустая ли коллекция, int size() – возвращает количество элементов в коллекции, Object[] toArray() – возвращает массив всех элементов коллекции.
9. ArrayList - основан на массиве, поддерживает быстрый доступ по индексу (get(index)), но вставка или удаление в середине списка медленнее, динамически расширяется при добавлении новых элементов, подходит, когда нужно часто читать элементы, а вставки/удаления в середине встречаются редко.
   LinkedList – основан на двусвязном списке, быстрое добавление и удаление элементов в начале, середине или конце списка, медленный доступ по индексу (get(index) требует перебора элементов), может использоваться как List и как Queue.
   Vector – похож на ArrayList, но синхронизированный (подходит для многопоточного доступа), реже используется в современных приложениях, так как синхронизация делает его медленнее, методы: add(), get(), remove() и т.д.
   Stack – наследуется от Vector и реализует стек (LIFO), методы: push(), pop(), peek().
10. HashSet – основан на хеш-таблице, быстрое добавление, удаление и поиск элементов (O(1) в среднем), порядок элементов не сохраняется, подходит, когда важна только уникальность элементов, а порядок не имеет значения.
    LinkedHashSet – наследуется от HashSet, сохраняет порядок добавления элементов, немного медленнее HashSet из-за хранения порядка, используется, когда нужно сохранить уникальность и порядок добавления.

TreeSet- реализует интерфейсы NavigableSet и SortedSet, элементы хранятся в отсортированном порядке (по возрастанию или по компаратору), поиск, вставка и удаление занимают O(log n), подходит, когда важен отсортированный набор уникальных элементов.

11. В Java Comparable и Comparator — это интерфейсы, которые используются для сортировки объектов. Они решают задачу, как сравнивать объекты между собой. Но работают немного по-разному.
Comparable – интерфейс, который реализует сам класс объекта, если объекты этого класса можно сравнивать между собой. Позволяет определить естественный порядок объектов. Содержит один метод: int compareTo(T o). Возвращает: < 0, если текущий объект меньше o; 0, если объекты равны; > 0, если текущий объект больше o.
Comparator – отдельный объект, который задаёт правила сортировки для стороннего класса или для альтернативного способа сортировки. Содержит один метод: int compare(T o1, T o2). Возвращает: < 0, если o1 меньше o2; 0, если равны; > 0, если o1 больше o2.
12. Тип-параметр — имя, используемое для обозначения типа, который будет заменен конкретным типом при использовании параметризованного типа. Например, в List< E> E — это тип-параметр. Это способ сделать класс, интерфейс или метод обобщённым (generic), чтобы он мог работать с разными типами данных, не привязываясь к конкретному типу. Другими словами, параметр типа — это «заглушка» для типа данных, который будет указан позже, при создании объекта или вызове метода.
13. Чтобы сделать класс обобщённым, после имени класса указывается параметр типа в угловых скобках <T>:

```
public class Box<T> {
	private T item;

	public void setItem(T item) {
		this.item = item;
	}

	public T getItem() {
		return item;
	}
}
```

Метод тоже может быть обобщённым, даже если сам класс не параметризован. Для этого перед типом возвращаемого значения указывается параметр типа <T>:

```
public static <T> T getFirst(List<T> list) {
	return list.get(0);
```

}

14. Стирание типов (Type Erasure) в Java — это механизм, с помощью которого информация о параметрах типа дженериков удаляется во время компиляции. На этапе выполнения JVM не знает, какой конкретный тип был указан (T превращается в Object или в ограниченный тип T extends ...). Это позволяет обеспечить совместимость дженериков с "обычными" классами, которые существовали до введения generics.
15. В Java стирание типов (type erasure) накладывает некоторые ограничения на использование дженериков: нельзя создавать массивы параметризованных типов, проверять тип через instanceof для T, использовать T.class и т.д. Есть несколько способов обойти эти ограничения: Использовать ограничение типа (Bounded Type), Передавать Class<T> в конструктор, использовать коллекции или Object[] вместо массива дженериков.
16. В Java дженерики и массивы работают с ограничениями из-за стирания типов (type erasure). Прямое создание массивов параметризованных типов запрещено, но есть способы обхода.
17. Прямое создание массива дженериков в Java невозможно из-за стирания типов (type erasure).
18. В Java wildcard тип (подстановочный тип) — это способ работать с обобщёнными типами без точного указания типа параметра. Он обозначается знаком «?» и позволяет создавать более гибкие обобщённые методы и коллекции.
19. В Java <T extends ...> и <T super ...> (или <? extends T> / <? super T>) используются в дженериках для ограничения типов, но применяются по-разному. Главная разница — это направление ограничения и возможность чтения/записи элементов.
    ? extends T — верхняя граница (extends). Ограничивает тип T и его подклассы. Используется, когда коллекция произведена (producer), т.е. мы читаем элементы. Можно безопасно читать, но нельзя добавлять (кроме null).
    ? super T — нижняя граница (super). Ограничивает тип T и его супертипы. Используется, когда коллекция потребитель (consumer), т.е. мы добавляем элементы. Можно безопасно добавлять элементы T или его подклассы, но чтение даёт только Object.