МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
Ордена трудового Красного Знамени федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе №6

“Работа с коллекциями”

Выполнил: студент группы БПИ2401

ФИО: Орлов Даниил Дмитриевич

Проверил: Харрасов Камиль Раисович

Москва, 2025

Цель работы:

Изучить работу с коллекциями Java, использование интерфейсов Map, List, Set, применение дженериков, обработку данных с использованием коллекций, методы перебора элементов коллекций, сортировку и анализ данных

Задание 1:

```
1   import java.io.File;
2   import java.io.FileNotFoundException;
3   import java.util.*;
```

Делаем необходимые импорты

```
4   public class TopWords {
        Run | Debug
5       public static void main(String[] args) {
6           String filePath = "C:\\JavaLabs\\laba6\\text.txt";
7           File file = new File(filePath);
```

Создаем публичный класс TopWords, обьявляем переменную, в которой будет храниться путь к текстовому файлу. Далее создаем обьект File, который указывает на файл по заданному пути.

```
8           Scanner scanner = null;
9           try {
10              scanner = new Scanner(file);
11          } catch (FileNotFoundException e) {
12              e.printStackTrace();
13              return;
14          }
```

Обьявляем и создаем обьект scanner, который читает содержимое файла. Ловим ошибку в том случае, если файл не найден, далее выводим сообщение об этой ошибке.

```
15          Map<String, Integer> wordCount = new HashMap<>();
```

Создаем Map, где ключ String – слово, значение Integer – количество повторений.

```
16          while (scanner.hasNext()) {
17              String word = scanner.next()
18                      .toLowerCase()
19                      .replaceAll(regex: "[^a-za-яё]", replacement: "");
```

Создаем цикл, который будет работать пока в файле есть слова. Читаем следующее слово, приводим его к нижнему регистру и удаляем все символы кроме букв.

```
20              if (!word.isEmpty()) {
21                  wordCount.put(word, wordCount.getOrDefault(word, defaultValue: 0) + 1);
22              }
23          }
```

Проверяем, что слово не пустое. Если слова нет, то возвращаем 0, увеличиваем количество повторений на 1 и сохраняем в Map.

```
24          scanner.close();
25          List<Map.Entry<String, Integer>> list = new ArrayList<>(wordCount.entrySet());
```

Закрываем scanner и создаем список из всех пар ключ-значение карты.

```
26          Collections.sort(list, new Comparator<Map.Entry<String, Integer>>() {
```

Делаем сортировку списка.

```
27              @Override
28              public int compare(Map.Entry<String, Integer> o1,
29                                 Map.Entry<String, Integer> o2) {
```

Создаем метод, который сравнивает два элемента.

```
30                  return o2.getValue().compareTo(o1.getValue());
31              }
32          });
```

Делаем сортировку по убыванию количества повторений.

```
33          System.out.println(x: "Топ-10 самых частых слов:");
34          for (int i = 0; i < Math.min(a: 10, list.size()); i++) {
35              System.out.println(list.get(i).getKey() + " — " +
36                                 list.get(i).getValue());
37          }
38      }
39  }
```

Выводим сообщение и создаем цикл, который будет идти по элементам в списке. Выводим каждое слово и количество его повторений.

Задание2:

```
1   public class Stack<T> {
2       private T[] data;
3       private int size;
```

Создаем публичный класс с обобщенным параметром типа <T>. Далее создаем массив элементов типа T и переменную для подсчета количества элементов в строке.

```
4       @SuppressWarnings("unchecked")
5       public Stack(int capacity) {
6           data = (T[]) new Object[capacity];
7           size = 0;
8       }
```

Создаем аннотацию, которая будет подавлять предупреждение о небезопасном приведении типов. Создаем конструктор Stack, где capacity – максимальный размер стека. Далее создаем массив обьектов, где T() – приведение типов, после этого указываем, что начальный размер стека равен нулю.

```
9       public void push(T element) {
10          if (size == data.length) {
11              throw new RuntimeException(message: "Стек переполнен");
12          }
13          data[size] = element;
14          size++;
15      }
```

Создаем метод, который будет добавлять элемент в стек. Проверяем не переполнен ли стек. Если переполнен – выбрасываем исключение при ошибке. Далее кладем элемент в вершину стека и увеличиваем размер стека.

```
16      public T pop() {
17          if (size == 0) {
18              throw new RuntimeException(message: "Стек пуст");
19          }
20          T element = data[size - 1];
21          data[size - 1] = null;
22          size--;
23          return element;
24      }
```

public T pop() - возвращает элемент типа T и удаляет верхний элемент. Проверяем пуст ли стек. Если пуст – выбрасываем исключение. T element = data[size - 1] - получаем верхний элемент. data[size - 1] = null - очистка ссылки. Далее уменьшаем размер и возвращаем удаленный элемент.

```
25      public T peek() {
26          if (size == 0) {
27              throw new RuntimeException(message: "Стек пуст");
28          }
29          return data[size - 1];
30      }
31  }
```

public T peek() – возвращает верхний элемент без удаления. Проверяем не пуст ли стек. Если пуст – выбрасываем исключение. return data[size - 1] – возврат элемента.

Файл для теста:

```
1   public class StackTest {
        Run | Debug
2       public static void main(String[] args) {
```

Создаем публичный класс для теста.

```
3           Stack<Integer> stack = new Stack<>(capacity: 10);
4           stack.push(element: 1);
5           stack.push(element: 2);
6           stack.push(element: 3);
```

Создаем стек на 10 элементов и добавляем несколько.

```
7           System.out.println(stack.pop());
8           System.out.println(stack.peek());
```

Удаляем верхний элемент и выводим. Далее осуществляем просмотр верхнего элемента без удаления.

```
 9          stack.push(element: 4);
10          System.out.println(stack.pop());
11      }
12  }
```

Добавляем новый элемент в стек и снова осуществляем удаление верхнего элемента и его вывод.

Задание 3:

```
1   import java.util.LinkedHashMap;
2   import java.util.Map;
3   public class ShopSales {
```

Делаем необходимые импорты и создаем публичный класс ShopSale.

```
4       private LinkedHashMap<String, Integer> sales = new LinkedHashMap<>();
5       private LinkedHashMap<String, Double> prices = new LinkedHashMap<>();
```

Создаем LinkedHashMap с названием sales, где ключ-значение это название товара – количество продаж и LinkedHashMap с названием prices, где пара ключ-значение это товар-цена.

```
6       public void addProduct(String name, double price) {
7           prices.put(name, price);
8       }
```

Создаем метод, который будет добавлять товар, принимаем название товара и его цену.

```
9       public void sellProduct(String name) {
10          sales.put(name, sales.getOrDefault(name, defaultValue: 0) + 1);
11      }
```

Создаем метод, который будет регистрировать продажу товара. Есди товара нет, то берется 0 и добавляется 1.

```
12      public void printSales() {
13          System.out.println(x: "Проданные товары:");
```

Создаем метод для вывода информации выводим сообщение о проданных товарах.

```
14          for (Map.Entry<String, Integer> entry : sales.entrySet()) {
15              System.out.println(entry.getKey() + " - " + entry.getValue());
16          }
17      }
```

Создаем цикл, который проходит по всем парам ключ-значение, будет выводить название товара и количество продаж.

```
18      public double getTotalSum() {
19          double sum = 0;
```

Создаем метод для подсчета общей суммы продаж и задаем переменную для суммы.

```
20          for (Map.Entry<String, Integer> entry : sales.entrySet()) {
21              sum += prices.get(entry.getKey()) * entry.getValue();
22          }
23          return sum;
24      }
```

Создаем цикл, который будет проходиться по всем парам ключ-значение, затем добавляем к общей сумме количество проданных товаров, умноженное на его стоимость. Далее возвращаем сумму.

```
25      public String getMostPopularProduct() {
26          String popular = null;
27          int max = 0;
```

Создаем метод для поиска самого популярного товара, задаем переменную popular и приравниваем ее к null и переменную max, которую приравниваем к нулю.

```
28          for (Map.Entry<String, Integer> entry : sales.entrySet()) {
29              if (entry.getValue() > max) {
30                  max = entry.getValue();
31                  popular = entry.getKey();
32              }
33          }
34          return popular;
35      }
```

Создаем цикл, который будет проходиться по всем парам ключ-значение. Делаем проверку на максимальное значение, обновляем величину максимального значения и сохраняем название товара. Далее возвращаем значение переменной popular.

```
36      public static void main(String[] args) {
37          ShopSales shop = new ShopSales();
```

Создаем точку входа в программу и создаем обьект магазина.

```
38          shop.addProduct(name: "Хлеб", price: 40);
39          shop.addProduct(name: "Молоко", price: 60);
40          shop.addProduct(name: "Яблоки", price: 100);
```

Добавляем товары и цены на них.

```
41          shop.sellProduct(name: "Хлеб");
42          shop.sellProduct(name: "Хлеб");
43          shop.sellProduct(name: "Молоко");
44          shop.sellProduct(name: "Яблоки");
45          shop.sellProduct(name: "Яблоки");
```

Регистрируем продажи.

```
46          shop.printSales();
```

Выводим список проданных товаров.

```
47          System.out.println("Общая сумма: " + shop.getTotalSum());
48          System.out.println("Самый популярный товар: " + shop.getMostPopularProduct());
49      }
50  }
```

Выводим сумму, на которую были проданы товары и выводим самый продаваемый товар.

Вывод:

В ходе лабораторной работы были реализованы программы с использованием коллекций Java. Были изучены основные интерфейсы коллекций и их реализации. Рассмотрены принципы работы дженериков, методы сортировки, перебора элементов и хранения данных.

Контрольные вопросы:

1. Какие интерфейсы коллекций есть в Java?

Основные интерфейсы:

- Collection
- List
- Set

- Queue
- Deque
- Map (не наследуется от Collection, но входит в Collections Framework)

2. Какие классы коллекций есть в Java?

Наиболее используемые:

- ArrayList
- LinkedList
- HashSet
- LinkedHashSet
- TreeSet
- HashMap
- LinkedHashMap
- TreeMap
- PriorityQueue
- Stack

3. Что такое итератор?

Итератор — это объект, позволяющий **последовательно перебирать элементы коллекции**.

Основные методы:

- hasNext()
- next()
- remove()

4. Как работают коллекции на основе интерфейса Map?

- Хранят данные в виде **пар ключ–значение**
- Ключи **уникальны**
- Доступ к данным осуществляется **по ключу**
- Пример: HashMap, TreeMap

5. Как работают коллекции на основе интерфейса List?

- Хранят элементы **в определённом порядке**
- Поддерживают **индексацию**
- Допускают **повторяющиеся элементы**

6. Как работают коллекции на основе интерфейса Set?

- Хранят **уникальные элементы**
- Повторы запрещены
- Обычно не имеют индексов

7. Как можно синхронизировать коллекции в Java?

Способы:

1. Collections.synchronizedList(...)
2. Collections.synchronizedMap(...)
3. Использовать коллекции из пакета java.util.concurrent
    - ConcurrentHashMap
    - CopyOnWriteArrayList

8. Какие методы предоставляет интерфейс Collection?

Основные методы:

- add()
- remove()
- size()
- isEmpty()
- contains()
- iterator()
- clear()

9. Какие реализации интерфейса List вы знаете?

- ArrayList
- LinkedList
- Vector
- Stack

10. Какие реализации интерфейса Set вы знаете?

- HashSet
- LinkedHashSet
- TreeSet

11. Что такое Comparable и Comparator?

- Comparable — интерфейс **естественного порядка сортировки**
- Comparator — интерфейс **внешнего правила сортировки**

12. Что такое параметр типа?

Параметр типа — это **обобщённый тип**, обозначаемый:

<T>

<E>

<K, V>

13. Как параметризовать метод, класс?

Класс:

class Box<T> { }

Метод:

<T> void print(T value)

14. Что такое стирание типов?

Стирание типов — механизм Java, при котором:

- информация о дженериках **удаляется во время компиляции**
- в байткоде все типы становятся Object

15. Как можно обойти ограничения стирания типов?

- Передавать Class<T>
- Использовать фабричные методы
- Ограничивать типы (<T extends ...>)

16. Как работают дженерики с массивами?

- Массивы знают свой тип во время выполнения
- Дженерики — **нет**
- Поэтому возникают ограничения

17. Можно ли создать массив дженериков?

Нет

new T[10] // запрещено

18. Что такое wildcard тип?

Wildcard — это ?

Пример:

List<?> list

Означает: список с неизвестным типом.

19. В чем разница между <extends T> и <super T>?

extends – только чтение и верхняя граница

super – можно добавлять и нижняя граница

Ссылка на репозиторий: https://github.com/qpdle/ITiP.git