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 «Математическая Кибернетика и Информационные технологии»

Лабораторная работа №6

По дисциплине «Информационные технологии и программирование»

Выполнил: Студент группы

БПИ 2301

Антонова Ирина

Москва

2024

Цель:

Изучение и применение коллекций выражений в Java

Задание:

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.

Задание 2:

Написать обобщенный класс Stack, 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.

Задание 3:

Необходимо 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 TreeSet для хранения списка проданных товаров.

Ход работы:

Откроем папку LR6 в программе VSCode и начнем создавать файлы, необходимые для выполнения заданий.

Задание 1:

Создадим в папке файл file.txt и напишем там любой текст. Далее создаем файл TopWords.java для реализации методов.

Импорт необходимых классов:

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

Объявление класса и метода main:

```
public class TopWords {
    public static void main(String[] args) {
```

Определение пути к файлу и создание объекта File:

```
String filePath = "text.txt";
File file = new File(filePath);
```

Создание объекта Scanner для чтения файла:

```
Scanner scanner = null;
        try{
            scanner = new Scanner(file);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        }
```

Создание HashMap для хранения слов и их частот:

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

Настройка разделителя для Scanner и чтение слов из файла:

```
while (scanner.hasNext()){
            String word = scanner.next().toLowerCase();
            if (words.containsKey(word)){
                words.put(word, words.get(word)+1);
            } else {
                words.put(word, 1);
            }
        }
```

Закрытие Scanner:

```
scanner.close();
```

Создание списка из элементов HashMap:

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

Здесь создается список **list**, который будет содержать все элементы из **HashMap words**. Метод **entrySet()** возвращает набор всех пар "ключ-значение" из **HashMap**, и метод **addAll()** добавляет все эти пары в список **list**.

Сортировка списка по частоте слов в порядке убывания:

```
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()-o1.getValue();
            }
        });
```

Для сортировки списка используется метод **Collections.sort()**, который принимает два аргумента:

1. Список, который нужно отсортировать (**list**).
2. Объект, реализующий интерфейс **Comparator**, который определяет порядок сортировки.

Интерфейс Comparator

Интерфейс **Comparator** используется для определения порядка сортировки элементов. Он имеет один метод **compare**, который принимает два объекта и возвращает целое число:

- Если возвращаемое значение отрицательное, первый объект меньше второго.
- Если возвращаемое значение положительное, первый объект больше второго.
- Если возвращаемое значение равно нулю, объекты равны.

Map.Entry<String, Integer> o1 и Map.Entry<String, Integer> o2 — это два элемента списка, которые нужно сравнить.

o2.getValue() - o1.getValue() — это выражение, которое сравнивает частоты слов. Если частота второго слова больше, то возвращаемое значение будет положительным, и первое слово будет считаться меньшим (то есть оно будет идти после второго в отсортированном списке). Это обеспечивает сортировку в порядке убывания частот.

Вывод 10 самых часто встречающихся слов:

```
for (int i = 0; i < Math.min(10, list.size()); i++) {
            System.out.println(i+1 + ". " + list.get(i).getKey());
        }
```

Цикл **for** проходит по первым 10 элементам отсортированного списка (или по всем элементам, если их меньше 10) и выводит слова в консоль.
Math.min(10, list.size()) гарантирует, что не будет попытки выйти за пределы списка, если в нем меньше 10 элементов.

Ответ программы после java TopWords.java в терминале будет:

```
PS C:\Users\ira\OneDrive\Desktop\work\ИТИП\LR6> java TopWords.java
1. и
2. на
3. в
4. за
5. лишь
6. вечер,
7. бриллианты,
8. окутывала
9. берегу
10. океана.
PS C:\Users\ira\OneDrive\Desktop\work\ИТИП\LR6>
```

Задание 2:

Класс **Stack** представляет собой обобщенную реализацию стека, который поддерживает операции добавления, удаления и просмотра элементов.

```
public class Stack<T> {
```

Поля класса

1. Массив **data**:

```
private T[] data;
```

Этот массив используется для хранения элементов стека. Он является обобщенным, что позволяет хранить элементы любого типа.

2. Переменная **size**:

```
private int size;
```

Эта переменная отслеживает текущий размер стека, то есть количество элементов, находящихся в стеке.

Конструктор

Конструктор инициализирует стек с заданной емкостью.

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

- Аннотация **@SuppressWarnings("unchecked")**: Эта аннотация подавляет предупреждения компилятора о непроверенных операциях приведения типов. В данном случае она используется для подавления предупреждения при создании обобщенного массива.
- Инициализация массива **data**:

Создается новый массив объектов с заданной емкостью и приводится к типу **T[]**. Это позволяет хранить элементы любого типа в массиве.

- Инициализация переменной **size**:

Переменная **size** устанавливается в 0, так как стек изначально пуст.

Метод push

Метод **push** добавляет элемент в стек.

```
    public void push(T element) {
        data[size] = element;
        size++;
    }
```

- Добавление элемента в массив:

Элемент добавляется в массив на позицию, указанную переменной **size**.

- Увеличение размера стека:

Переменная **size** увеличивается на 1, чтобы отразить добавление нового элемента.

Метод pop

Метод **pop** удаляет и возвращает верхний элемент стека.

```
    public T pop() {
        T element = data[size-1];
        data[size-1] = null;
        size--;
        return element;
    }
```

- Получение верхнего элемента:

Верхний элемент стека (последний добавленный элемент) извлекается из массива.

- Очистка позиции в массиве:

Позиция в массиве, где находился верхний элемент, очищается (устанавливается в **null**), чтобы освободить память.

- Уменьшение размера стека:

Переменная **size** уменьшается на 1, чтобы отразить удаление элемента.

- Возврат элемента:

Верхний элемент возвращается из метода.

Метод peek

Метод **peek** возвращает верхний элемент стека без его удаления.

- Возврат верхнего элемента:

```
    public T peek() {
        return data[size-1];
    }
```

Верхний элемент стека возвращается без его удаления из стека.

Файл StackMain.java будет содержать тест на программу, то есть одноименный класс и тестовый класс main

```
public class StackMain {
    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());
    }
}
```

Выводом программы будет:

```
4
PS C:\Users\ira\OneDrive\Desktop\work\ИТИП\LR6> java StackMain.java
3
2
4
```

Задание 3:

Создадим файл и в нем одноименный класс. Пропишем геттеры и сеттеры а так же переопределим метод toString

```
public class TreeProduct{
    private String name;
    private int price;
    private int quantity;

    public TreeProduct(String name, int price, int quantity){
        this.name = name;
        this.price = price;
        this.quantity = quantity;
    }

    public String getName(){
        return name;
    }

    public void setName(String name){
        this.name = name;
    }

    public int getPrice(){
        return price;
    }

    public void setPrice(int price){
        this.price = price;
    }
    public int getQuantity() {
        return quantity;
    }

    public void setQuantity(int quantity) {
        this.quantity = quantity;
    }

    @Override
    public String toString(){
        return "[" + name + ", " + price + "]";
    }
}
```

Далее создадим файл и класс в котором будем реализовывать все необходимые методы для выполнения задания
Класс **TreeShop** предназначен для управления проданными продуктами в магазине. Он использует структуру данных **TreeSet** для хранения продуктов,

что позволяет автоматически поддерживать их в отсортированном порядке. В данном отчете будет подробно рассмотрено, как реализован этот класс и какие функции он выполняет.

Импорт необходимых классов

```
import java.util.TreeSet;
```

Импортируется класс **TreeSet** из пакета **java.util**. **TreeSet** — это коллекция, которая хранит элементы в отсортированном порядке и не допускает дубликатов.

Определение класса TreeShop

```
public class TreeShop {

    private TreeSet<TreeProduct> soldProducts;
```

Класс **TreeShop** содержит одно поле **soldProducts**, которое является **TreeSet** объектов типа **TreeProduct**. Этот **TreeSet** будет использоваться для хранения проданных продуктов.

Конструктор класса TreeShop

```
    public TreeShop() {
        soldProducts = new TreeSet<>((p1, p2) -> {
            int quantityComparison = Integer.compare(p2.getQuantity(),
p1.getQuantity());
            if (quantityComparison != 0) {
                return quantityComparison;
            }
            return p1.getName().compareTo(p2.getName());
        });
    }
```

Конструктор инициализирует **TreeSet** с пользовательским компаратором. Компаратор сначала сравнивает продукты по количеству (в порядке убывания), а затем по имени (в лексикографическом порядке), если количество одинаково. Это позволяет автоматически поддерживать продукты в отсортированном порядке при их добавлении.

Метод для добавления продукта addProduct

```
    public void addProduct(TreeProduct product) {
        for (TreeProduct existingProduct : soldProducts) {
            if (existingProduct.getName().equals(product.getName())) {
```

```
                existingProduct.setQuantity(existingProduct.getQuantity() +
product.getQuantity());
                soldProducts.remove(existingProduct);
                soldProducts.add(existingProduct);
                return;
            }
        }
        soldProducts.add(product);
    }
```

Метод **addProduct** добавляет продукт в **TreeSet**. Если продукт с таким же именем уже существует, его количество увеличивается, и продукт перемещается в правильное место в **TreeSet**. Если продукта нет, он просто добавляется.

1. Поиск существующего продукта:
    - Используется цикл **for** для перебора всех элементов в **TreeSet**.
    - Если имя текущего продукта совпадает с именем добавляемого продукта, то количество существующего продукта обновляется.
2. Обновление количества существующего продукта:
    - Количество существующего продукта увеличивается на количество добавляемого продукта.
    - Существующий продукт удаляется из **TreeSet** и снова добавляется, чтобы он был отсортирован заново.
3. Добавление нового продукта:
    - Если существующий продукт не найден, новый продукт просто добавляется в **TreeSet**.

Метод для печати проданных продуктов printSoldProducts

```
    public void printSoldProducts() {
        for (TreeProduct product : soldProducts) {
            System.out.println("[" + product.getName() + ", " +
product.getPrice() + ", " + product.getQuantity() + "]");
        }
    }
```

Метод **printSoldProducts** выводит информацию о всех проданных продуктах в консоль. Он перебирает все элементы в **TreeSet** и выводит их имена, цены и количества.

Метод для расчета общей суммы продаж calculateTotalSales

```
    public int calculateTotalSales() {
        int totalSales = 0;
        for (TreeProduct product : soldProducts) {
            totalSales += product.getPrice() * product.getQuantity();
        }
        return totalSales;
    }
```

Метод **calculateTotalSales** рассчитывает общую сумму продаж. Он перебирает все элементы в **TreeSet**, умножает цену каждого продукта на его количество и суммирует результаты.

Метод для получения самого популярного продукта getMostPopularProduct

```
    public TreeProduct getMostPopularProduct() {
        if (soldProducts.isEmpty()) {
            return null;
        }
        TreeProduct mostPopular = soldProducts.first();
        System.out.println("Most Popular Product: [" + mostPopular.getName() + ",
" + mostPopular.getPrice() + ", " + mostPopular.getQuantity() + "]");
        return mostPopular;
    }
```

Метод **getMostPopularProduct** возвращает самый популярный продукт (тот, который был продан в наибольшем количестве). Если **TreeSet** пуст, метод возвращает **null**. В противном случае, он возвращает первый элемент **TreeSet**, так как продукты отсортированы по количеству в порядке убывания.

Также создадим файл и класс для тестирования программы

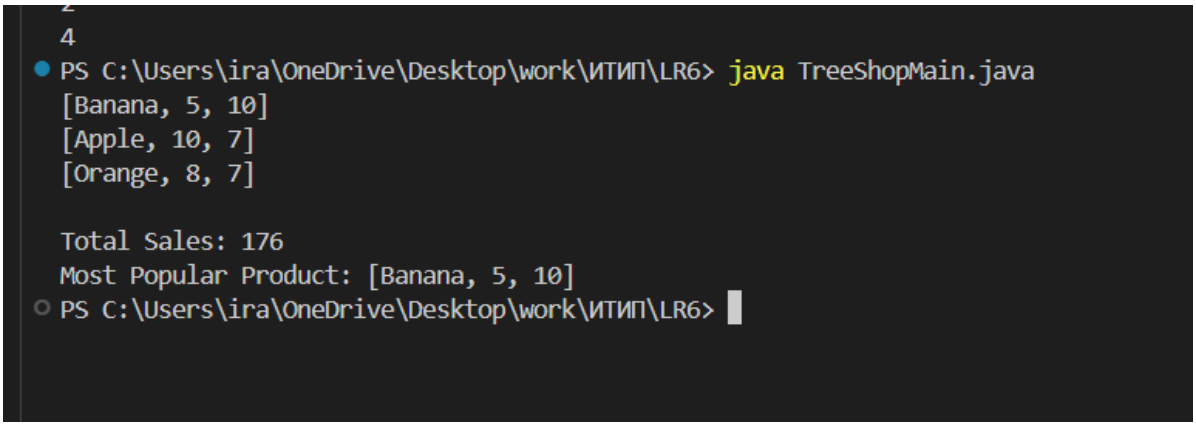
```
public class TreeShopMain {
    public static void main(String[] args) {
        TreeShop shop = new TreeShop();
        shop.addProduct(new TreeProduct("Apple", 10, 5));
        shop.addProduct(new TreeProduct("Banana", 5, 10));
        shop.addProduct(new TreeProduct("Apple", 10, 2));
        shop.addProduct(new TreeProduct("Orange", 8, 7));

        shop.printSoldProducts();

        System.out.println("\nTotal Sales: " + shop.calculateTotalSales());

        shop.getMostPopularProduct();
    }
}
```

После запуска программы ответом будет



Вывод:

Корректно выполнены все задания и изучены коллекции в java.

Ссылка на репозиторий с кодом
https://github.com/k00kzaAntonovaIra/ITIP_2024.git