Министерство цифрового развития, связи и массовых коммуникаций

Российской Федерации

Ордена Трудового Красного Знамени

Федерально государственное бюджетное образовательное учреждение

высшего образования

МОСКОВСКИЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ СВЯЗИ И ИНФОРМАТИКИ

Кафедра «Математической кибернетики и информационных технологий»

Лабораторная работа №6­­­

По дисциплине

«Информационные технологии и программирование»

Выполнил:

Студент группы БВТ2203

Бородин К.Н.

Москва 2023

ЗАДАНИЯ

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.

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.

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.

ЗАДАНИЕ 1.

Сначала мы указываем путь к файлу, затем создаём объект класса File, после с помощью сканнера открываем этот файл, указывая кодировку. Далее Мы создаём хэшмап, в котором будем хранить пары слово-количество его повторений.

Итеративно будем проходиться по каждом слову и обновлять значение хешмпапа. После цикла закрываем сканер.

Далее создаём массив, в котором элементами будут являться пары из хешмапа. Затем мы сортируем этот массив с помощью компаратора. В компараторе мы переопределяем функцию для сравнения таким образом, чтобы она сравнивала количество слов, после выводим первые 10 слов, если уникальных больше 10, или столько же слов, сколько уникальных. Код программы представлен далее.

import java.io.\*;

import java.util.\*;

import java.util.regex.Pattern;

/\*

 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.

 \*/

public class TopWords{

    public static void main(String[] args){

        //Указываем путь к файлу

        String filePath = "C:\\Users\\Kirill\\Desktop\\mtuci\\itip\\6lab\\book.txt";

        //создаём объект file

        File file = new File(filePath);

        // Создаём объект scanner для чтения файла

        // Сканнер плохо работает на русском

        Scanner scanner = null;

        try {

            scanner = new Scanner(file, "UTF-8");

            //сканнер разбивает текст на токену по разделителю, который по умолчанию это пробел

        }

        catch (FileNotFoundException e){

            //вывод исключения в стандартный поток ошибок, в нашем случае это будет консоль

            e.printStackTrace();

        }

        //Создаём объект Map для хранения слов и их количества

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

        while (scanner.hasNext()){

            String word = scanner.next().toLowerCase();//чтение следующее элемента

            int countCurrentWord = dict.getOrDefault(word, 0)+1;//получаем элемент, если он есть, или ноль, если его нет и увеличиваем на единицу

            dict.put(word, countCurrentWord);//Добавляем или обновляем элемент в количестве слов

        }

        //закрываем сканнер

        scanner.close();

        //создаём список из элементов Map

        ArrayList<HashMap.Entry<String, Integer>> elementsDict = new ArrayList<>(dict.entrySet());

        // сортируем список по убыванию количества повторенией

        Collections.sort(elementsDict, new Comparator<Map.Entry<String, Integer>>(){ // компаратор - умная штука, которая сравнивает элементы. В наешм случае мы создаём такой компаратор

            @Override //переопределение его метода для сравнения

            public int compare(Map.Entry<String, Integer> o1, Map.Entry<String, Integer> o2){ //функция сравнения, будет принимать два элемента и сравнивать их значения

                return o2.getValue().compareTo(o1.getValue()); // если o2>o1=>+1 если o2<01=>-1 если o2=o1=>0

            }

        });

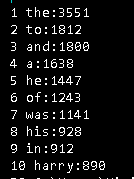
        //выводим 10 слов или меньше, если в исходном тексте их меньше

        for (int i =0; i<Math.min(10, elementsDict.size()); i++){

            Map.Entry<String, Integer> temp = elementsDict.get(i);

            System.out.println((i+1) +" " + temp.getKey() + ":" + temp.getValue());

        }



ЗАДАНИЕ 2.

Создаём класс для стека. Элемента стека будут иметь параметризованный класс. В этом стеке мы будем иметь массив из элементов, количество элементов, и вместимость по умолчанию. Это значение нужно при инициализации по умолчанию.

Далее создаём два конструктра – параметризованный и по умолчанию.

Функция для добавления элемента в конец будет сначала вызывать функцию для увеличения стека, а затем добавлять в новую ячейку элемент, при этом увеличиваем переменную размера стека.

Функция для удаления элемента будет сначала вызывать функцию для проверки стека на пустоту, а затем, если стек пустой, выбрасывать исключения. Затем мы сохраняем элемент, уменьшаем вместимость и на месте этого элемента ставим null и возвращаем элемент

Функция для возврата последнего элемента работает аналогичным образом, но без удаления элемента.

Функция увеличения ёмкости будет ей увеличивать только в случае, если размера недостаточно. Затем будет создавать новый массив с большей ёмкостью и в него копировать элементы предыдущего. Код программы представлен далее.

import java.util.Arrays;

/\*

 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

 \*/

public class Stack<T> {

    private T[] data;

    private int size;

    private static final int DEFAULT\_CAPACITY = 10;

    public Stack(){ //конструктор по умолчанию

        data = (T[]) new Object[DEFAULT\_CAPACITY];

        //(T[]) приведению к заданному типу того, что написано далее

        // создание новое массива типа Object и с заданным объёмом

        //В Java массивы обобщённых типов имеют ограничение из-за

        //стираний типов(type erasure). Во время компиляции обобщённые

        //типы заменяются их необобщёнными версиями, и во вермя выполнения

        //информация о типах стирается.

        //Создать напрямую массив обобщённого типа нельзя!

        size = 0;

    }

    public Stack(int capacity){ //конструктор с аргументами

        data = (T[]) new Object[capacity];

        size = 0;

    }

    public void push(T element){//Добавить элемент

        ensureCapacity();

        data[size++] = element;

    }

    public T pop(){//удалить и вернуть верхний элемент

        if (isEmpty()){

            throw new IllegalStateException("Stack is empty");

        }

        T element = data[--size];

        data[size] = null; //убираем элемент из стека

        return element;

    }

    public T peek(){//возврат верхнего элемента

        if (isEmpty()){

            throw new IllegalStateException("Stack is empty");

        }

        return data[size-1];

    }

    public boolean isEmpty(){//проверка на пустоту

        return size==0;

    }

    private void ensureCapacity(){//Увеличение размерности

        if (size == data.length){

            int newCapacity = data.length+1;

            data = Arrays.copyOf(data, newCapacity);

        }

    }

    public static void main(String[] args){

        Stack<Integer> stack = new Stack<>(1);

        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());

    }

}



ЗАДАНИЕ 3.

TreeSet в Java является реализацией интерфейса Set, который расширяет SortedSet. Он представляет собой набор уникальных элементов, отсортированных в порядке возрастания. Вот основные характеристики TreeSet:

1. **Уникальность элементов:** TreeSet не допускает наличие дубликатов. Если вы пытаетесь добавить элемент, который уже присутствует в множестве, операция добавления просто проигнорируется.
2. **Сортировка элементов:** Элементы в TreeSet автоматически сортируются в возрастающем порядке на основе их значений или с использованием компаратора, предоставленного при создании множества.
3. **Бинарное дерево поиска:** Внутри TreeSet использует структуру данных, называемую бинарным деревом поиска. Это эффективная структура для хранения и поиска отсортированных данных.
4. **Операции добавления, удаления и поиска:** Операции добавления, удаления и поиска элементов в TreeSet выполняются в среднем за логарифмическое время, что делает его эффективным для больших объемов данных.

С моим вариантом реализация поиска самого популярного варианта не возможна, поскольку TreeSet не может хранить пары значений.

Создаю класс, который будет реализовывать данное задание. Для хранение элементов, согласно варианту, используется TreeSet. Также буду использовать переменную для подсчёт количества проданных товаров.

Конструктор будет инициализировать трисет.

Метод добавления товара будет добавлять товар в список товаров, а также увеличивать переменную количества товаров.

Метод для отображения товаров будет итерироваться по списку товаров и выводить их.

Метод количества товаров будет возвращать соответствующую переменную

Реализация кода представлена далее.

import java.util.\*;

/\*

Программа должна позволять добавлять проданные товары в коллекцию,

выводить список проданных товаров, а также считать общую сумму

продаж

Использовать TreeSet для хранения списка проданных

товаров.

 \*/

public class SalesTracker {

    private TreeSet<String> soldProducts;

    private int counter = 0;

    public SalesTracker(){

        soldProducts = new TreeSet<>();

        //конструтктов

    }

    public void addSale(String product){

        soldProducts.add(product);

        counter++;

        //Добавление элемента

    }

    public void displaySales(){

        for (String i:soldProducts){

            System.out.println(i);

        }

        // Итерация по всем объектам и их вывод

    }

    public double calculateTotalSales(){

        return counter;

    }

    public static void main(String[] args) {

        SalesTracker salesTracker = new SalesTracker();

        salesTracker.addSale("book");

        salesTracker.addSale("jacket");

        salesTracker.addSale("cup");

        salesTracker.addSale("book"); // Продажа еще одной книги

        //список проданных товаров с количеством

        salesTracker.displaySales();

        // Общее количество продаж

        System.out.println(salesTracker.calculateTotalSales());

    }

}

