Practice6

Везде в коде, при реализации функциональных интерфейсов использовать лямбда выражения или ссылки на методы.

Помните, что контейнеры Hash/TreeSet, Hash/TreeMap не контролируют потенциальное появление дубликатов при изменении никак не отслеживают изменения, то есть, если изменение состояния ключа какой ни будь пары может произойти, то контейнер это никак не отслеживает.

Задание 1

Пакет: ua.nure.name.Practice6
Класс:
Part1 – содержит моделирование консольного ввода/вывода и вызов метода WordContainer.main
WordContainer – содержит требуемую по условию функциональность
Word – поставлен в соответствие слову с информацией о его частоте в тексте

Разработать приложение, которое считывает текст из консольного ввода и выводит слова в порядке убывания их частоты появления в тексте (при совпадении частот порядок - лексикографичекий). Под словами понимать непрерывную последовательность непробельных символов.
Решить задачу с применением ООП подхода. Класс **Word**, содержит строковое поле **content** и целое поле **frequency**, контейнер **WordContainer** агрегирует объекты **Word**. При использовании контейнерных классов из ядра грамотно реализовать методы **Word#equals** / **Word#hashCode** / **Word#compareTo** (если они будут нужны).

Если вход такой:

```
asd asdf asd asdf asdf 43 asdsf 43 43 434
```

То выход должен быть таким:

```
43 : 3
asdf : 3
asd : 2
434 : 1
asdsf : 1
```

При запуске **WordContainer.main** осуществляет чтение из стандартного потока ввода (консоли). Консольный ввод в общем случае может содержать несколько строк, в каждой строке может быть несколько слов. Признаком завершения консольного ввода служит слово **stop**.

Продемонстрировать работу приложения (Par1.main).

Задание 2

Пакет: **ua.nure.name.Practice6**
Класс: **Part2**

Написать программу, которая моделирует следующий процесс. В кругу стоят n человек, пронумерованных от 0 до n-1. При ведении счета по кругу вычеркивается каждый k-й человек (0<k<n), пока не останется один. Решить задачу двумя способами - с использованием класса ArrayList и с использованием класса LinkedList. Определить и сравнить время выполнения каждой из двух программ на одинаковых входных данных (задавать в Part2.main).
Продемонстрировать работу приложения (**Par2.main**)

Задание 3

Пакет: ua.nure.name.Practice6
Класс: Part1, Parking

Реализовать класс **Parking**, моделирующий работу n-местной автостоянки. Машина подъезжает к определенному месту и едет вправо, пока не встретится свободное место. Класс должен поддерживать методы, обслуживающие приезд и отъезд машины. Определить метод, который выводит в консоль текущее состояние стоянки. Продемонстрировать работу приложения (**Part3.main**).

Задание 4

Пакет: ua.nure.name.Practice6
Класс: Part4, Graph

Реализовать класс **Graph**, представляющий собой неориентированный граф. В конструкторе класса передаётся количество вершин в графе. Методы должны поддерживать быстрое добавление и удаление рёбер.
Продемонстрировать работу приложения (**Part4.main**).

Задание 5

Пакет: **ua.nure.name.Practice6**
Класс: **Part5**, **Tree**

Создать generic класс **Tree**, который реализует структуру данных "двоичное дерево поиска". Контейнерные классы не использовать.

Tree.java

```
public class Tree<E extends Comparable<E>> {
        // добавляет элемент в контейнер
        // если в контейнере есть элемент равный по compareTo добавляемому,
        // то добавления не происходит и метод возвращает false
        // в противном случае элемент попадает в контейнер и метод возвращает true
        // первый добавляемый элемент становится корнем дерева
        public boolean add(E element) {...}

        // добавляет все элементы из массива в контейнер (вызов в цикле метода add, см. выше)
        public void add(E[] elements) {...}

    // удаляет элемент из контейнера
        // если удаляемого элемента в контейнере нет, то возвращает false
        // в противном случае удаляет элемент и возвращает true
        // ВАЖНО! при удалении элемента дерево не должно потерять свойства
бинарного дерева поиска
        public boolean remove(E element) {...}

        // распечатывает дерево, так чтобы было видно его древовидную структуру,
см. ниже пример
        public void print() {...}

        // вложенный класс, объекты этого класса составляют дерево
        private static class Node<E> {...}
}
```

Код

```
Tree<Integer> tree = new Tree<>();

System.out.println(tree.add(3));
System.out.println(tree.add(3));

System.out.println("~~~~~~~");
tree.add(new Integer[] {1, 2, 5, 4, 6, 0});
tree.print();

System.out.println("~~~~~~~");
System.out.println(tree.remove(5));
System.out.println(tree.remove(5));

System.out.println("~~~~~~~");
tree.print();
```

Вывод

```
true
false
~~~~~~~
        0
  1
        2
3
        4
  5
        6
~~~~~~~
true
false
~~~~~~~
        0
    1
        2
4
    5
        6
```

Продемонстрировать работу приложения (**Part5.main**).

Задание 6

Пакет: ua.nure.name.Practice6
Класс: **Part6** (принимает параметры командной строки)
Входную информацию загружать из файла который лежит в корне проекта, назвать **part6.txt**

Реализовать консольное приложение, (**ua.nure.name.Practice6.Part6**) которое анализирует текст. Формат входных параметров командной строки для приложения (программа должна понимать как **короткие**, так и **длинные** опции):

-i (или --input) путь к входному файлу;
-t (или --task) наименование подзадачи.

Всего параметров командной строки 4, т.е. две пары <**имя_опции значение_опции**>. Порядок следования опций с их значениями прооизвольный.
Примеры параметров командной строки для запуска приложения

```
-i input.txt -t frequency
 --input input.txt --task length
```

В методе **Demo.main** смоделировать пеердачу параметров командной строки в метод **Part6.main**, пример:

```
System.out.println("~~~~~~~~~~~~Part6");
Part6.main(new String[] {"--input", "part6.txt", "--task", "length"});
Part6.main(new String[] {"--input", "part6.txt", "--task", "frequency"});
Part6.main(new String[] {"--input", "part6.txt", "--task", "duplicates"});
```

Подзадач всего три **frequency**, **length**, **duplicates**.

Под словами понимать непрерывную последовательность букв **латинского** алфавита.

Part 1 (frequency)

Во входном файле найти три слова, которые встречаются наиболее часто (**при совпадении частот – те, которые встречаются раньше**), и распечатать их отсортированными по алфавиту в обратном порядке в формате: слово ==> частота

Пример вывода

```
panda ==> 15
ezhik ==> 20
apple ==> 19
```

Part 2 (length)

Во входном файле найти три самых длинных слова и распечатать их в формате: слово ==> количество букв в слове. Список должен быть отсортирован по убыванию количества букв в слове.

> Если у двух слов количество букв совпадает, то слово, которое было в исходном файле раньше, должно в результирующем списке находиться также раньше.

Пример вывода

```
anesthetist ==> 11
kitchen ==> 7
bird ==> 4
```

Part 3 (duplicates)

Во входном файле найти первые три слова, которые имеют дубликаты, и напечатать их инверсию в верхнем регистре.

Пример вывода

```
ADNAP
TAC
ENIGREBUA
```

Задание 7

Пакет: ua.nure.name.Practice6
Класс: **Part7**, **Range**

Написать класс **Range**, который бы представлял собой промежуток чисел **[n, m]**, где **n<m**. Класс должен реализовывать интерфейс **Iterable**. Итератор реализовать таким образом, чтобы он проходил он начала до конца промежутка. В конструктор передавать дополнительный параметр **reverse**.

Класс **Range** должен иметь два конструктора:

```
public Range(int n, int m) { ... }
public Range(int n, int m, boolean reverse) { ... }
```

Пример.

```
Range range = new Range(3, 10);
for (Integer el : range) {
	System.out.printf("%d ", el);
}
System.out.println();
// result: 3 4 5 6 7 8 9 10

range = new Range(3, 10, true);
for (Integer el : range) {
	System.out.printf("%d ", el);
}
System.out.println();
// result: 10 9 8 7 6 5 4 3
```

Продемонстрировать работу приложения (**Part7.main**).