Practical task №4

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Notes.

1. If the application reads data from a file, you need to specify the same encoding using which the data is stored. Use encoding Ср1251.

2. If the language of the text information is not indicated specifically, choose a text that can contain both Cyrillic (Russian and Ukrainian) and Latin letters (English).

3. Create Demo class in the root package that demonstrates the actions of all the written functionality.

4. The names of the input and output files are relative to the value of user.dir ( = the root directory of the project) system property.

5. Make sure to check the project build logs (Jenkins), the output should coincide with the output on your local computer.

6. When displaying information, use a platform independent line terminator, otherwise if to try running the application using another OS, you might not see what you expect.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Task 1

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice4.Part1

The input data should be uploaded from the 'part1.txt' file

———————————————————————————————-

Create a class that displays the content of the text file to console, deleting the first two characters of each word with the length of more than 4 characters (example: original 'qwerty', new 'erty').

The size of the file should not be more than 1Kb (several lines are enough).

A 'word' should be considered a continuous sequence of Cyrillic or Latin characters.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Task 2

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice4.Part2

The input data should be uploaded from the 'part2.txt' file

The output data should be uploaded to the 'part2\_sorted.txt' file

———————————————————————————————-

Create a class that creates and fills the file 'part2.txt' with random whole numbers from 0 to 50 (10 numbers in total), then reads the file and writes its content to the other file ('part2\_sorted.txt') having sorted the numbers in the ascending order.

Display the content of both files in the console (the numbers should be separated with a space).

Write your own sorting method that uses some algorithm for that (bubble sort, for example). The output file should be readable.

Display the content of both the input and output files in the console.

Example of the console output

————————————————————————-

input ==> 30 23 16 16 9 23 3 18 21 29

output ==> 3 9 16 16 18 21 23 23 29 30

————————————————————————-

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Task 3

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice4.Part3

The input data should be uploaded from the 'part3.txt' file

———————————————————————————————-

The file contains characters, whole numbers and floating point numbers. Write a class that has the following functionality: in a loop, a user enters the data type (one of them: char, int, double, String), in response, the application prints out to the console all the values of the corresponding types that exist in the file. If a user enters the word 'stop', then the application exits the loop.

Resolve the task using regular expressions.

Note: A 'String' should be considered a sequence of two and more characters.

Characters - Latin or Cyrillic letters letters in upper or lower case (make sure to envisage that Cyrillic letters may be in the input file).

Example of the original file:

————————————————————————-

a bcd 43.43 432 и л фвыа 89 .98

————————————————————————-

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Task 4

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice4.Part4

The input data should be uploaded from the 'part4.txt' file

———————————————————————————————-

Create a class that implements interface java.lang.Iterable<String>. The class should parse the text file and return sentences. The method 'iterator' of the given class should return the iterator object - an instance of the inner class. The method Iterator#remove should throw an exception UnsupportedOperationException.

You are not allowed to use the existing implementations of the iterators from the container classes!

Note. Write a regular expression that 'cuts' sentences from the text, then use the Matcher object when implementing the methods of the Iterator interface.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Task 5

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice4.Part5

The input resource package, ru locale: resources\_ru.properties

The input resource package, en locale: resources\_en.properties

The resource packages are to be placed in the src/main catalog.

———————————————————————————————-

Create resource packages (.properties files) for two locales: ru and en. The packages should contain at least two records.

Example of the resources\_en.properties content

————————————————————————-

table = table

apple = apple

————————————————————————-

Example of the resources\_ru.properties content

————————————————————————-

table = стол

apple = яблоко

————————————————————————-

Write a class that in a loop reads the key and the localization name from the console and, in response, prints out the corresponding value to the console. The sign of the input end is the word 'stop'.

Reading from console and writing to console are required!

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Task 6

———————————————————————————————-

Class name: com.epam.rd.java.basic.practice4.Part6

The input file - 'part6.txt'

———————————————————————————————-

The file contains words written with Cyrillic and Latin letters.

Create a class that reads the user input from console and in response prints out all specific words from the file.

If user enters "cyrl" then all words with Russian or Ukrainian symbols are printed. If user enters "latn" then all words with Latin set of symbols are printed. The input with any case (upper or lower) must be recognized.

If a user enters the word "stop" then the application quits.

In other cases program prints "Incorrect input".

Example of the input file:

————————————————————————-

Is there anybody going to listen to my story

Лягає день. Він віддає свої надії ночі.

Робітники

Заморились працювати.

С барабаном ходит ёжик. Бум-бум-бум.

————————————————————————-

Example of the console output:

————————————————————————-

asdf<Enter>

asdf: Incorrect input

Latn<Enter>

Latn: Is there anybody going to listen to my story

Cyrl<Enter>

Cyrl: Лягає день Він віддає свої надії ночі Робітники Заморились працювати С барабаном ходит ёжик Бум бум бум

stop

————————————————————————-

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Demo class content

The Demo class should be placed in the root package (com.epam.rd.java.basic.practice4), it should demonstrate the actions of all the functionality.

For those subtasks that require console input, reset the standard input stream so that the input occurred from a certain string (after running your code, you need to envisage restoration of the standard streams).

Demo.main should run without user participation, there should be no expectation of the console input during execution of this method. Refer to the example of streams reset in the stub method.

In case the application freezes expecting the console input, it will stop running not more than in 2 minutes (the timeout is set for all the tasks).

Stub method of Demo.main

————————————————————————-

package com.epam.rd.java.basic.practice4;

import java.io.ByteArrayInputStream;

import java.io.InputStream;

import java.nio.charset.StandardCharsets;

public class Demo {

    private static final InputStream STD\_IN = System.in;

    public static void main(String[] args) {

        System.out.println("=========================== PART1");

        Part1.main(args);

        System.out.println("=========================== PART2");

        Part2.main(args);

        System.out.println("=========================== PART3");

        // set the mock input

        System.setIn(new ByteArrayInputStream(

                "char^String^int^double^stop".replace("^", System.lineSeparator()).getBytes(StandardCharsets.UTF\_8)));

        Part3.main(args);

        // restore the standard input

        System.setIn(STD\_IN);

        System.out.println("=========================== PART4");

        Part4.main(args);

        System.out.println("=========================== PART5");

        // set the mock input

        System.setIn(new ByteArrayInputStream(

                "table ru^table en^apple ru^stop".replace("^", System.lineSeparator()).getBytes(StandardCharsets.UTF\_8)));

        Part5.main(args);

        // restore the standard input

        System.setIn(STD\_IN);

        System.out.println("=========================== PART6");

        // set the mock input

        System.setIn(new ByteArrayInputStream(

                "Latn^Cyrl^asdf^latn^cyrl^stop".replace("^", System.lineSeparator()).getBytes(StandardCharsets.UTF\_8)));

        Part6.main(args);

        // restore the standard input

        System.setIn(STD\_IN);

    }

}

————————————————————————-