OutLab 6: Basic Java

General instructions:

- Make sure you know what you write, you might be asked to explain your code at a later point in time.
- The submission will be graded automatically, so stick to the naming conventions and follow the directory structure strictly.
- The deadline for this lab is **Saturday, 02nd October, 11:59 PM.**
- After creating your directory, package it into a tarball: **rollno1-rollno2-outlab6.tar.gz**
- **Command to generate tarball:**

  ```
  tar -czvf rollno1-rollno2-outlab6.tar.gz rollno1-rollno2-outlab6/
  ```
- **Incorrect directory structure will lead to marks penalty**. **Please make sure we see the exact directory structure as shown above when we decompress the archive.**
- Submit once only per team from the moodle account of the smallest roll number.
- **The directory structure should be as follows (nothing more nothing less).**
- Quote major references in `references.txt`.

Tips:

- In Java, the filename and class name should be the same, otherwise, you will get errors while compiling!
- Just like in C/C++, java programs need to be compiled, then executed
  eg,
  ```
  $ javac <filename>.java
  $ java <filename>
  ```

Problem 1 (Marks: 10)

Write a java program, that takes command-line arguments, where the arguments are numbers.

Input:

$ java q1 arg1 arg2 arg2 …….. argN

The program should output the number of arguments, their sum, and their product printed simultaneously separated by commas.

Output:
$ N,<sum of N numbers>,<product of N numbers>

Note: Make sure there are NO SPACES in output!

<u>Problem 2 (Marks: 20)</u>

Write a Java program “q1.java” which takes a filename as a command line argument, eg:

$ java q1 input.txt

and reads the text inside the file to find out the character(s) which has the highest frequency in each line (you can assume every line will have some alphabets and there will be no empty lines). However, you must calculate frequencies for alphabet characters only. Also, all capital letters must be converted to small letters while counting. If more than 1 alphabets have the highest frequency then print all in lexicographical order separated by comma (NO SPACE). Output all these frequencies for each line in a new file named “output.txt”

NOTE: Implement using **HashMaps**!

Example:

input.txt:

```
...
Java is a high-level, class-based, object-oriented programming language.
Hello! My name is Slim Shadey…
...
...
```

output.txt:

```
...
a=8
e=3,l=3,m=3,s=3
...
...
```

File Structure:

```
q2
├── q2.java
├── input.txt
└── output.txt
```

Problem 3:

Remember the concept of "Regular Languages" from your CS228 course. These are formal languages that can be expressed using regular expressions. The following questions are all about using **Regex**. Create methods in class q3 named as func1-4. Their functionalities are explained as follows:

NOTE: most of these subparts can be done in a single line

func1: (5 marks)

Takes a string as an input and return a boolean value specifying whether or not the string meets the following conditions:
1. Only consist of letters from English Alphabet(both lower & upper case) or digits.
2. Has length <= 5

Sample input 1:
j@va2
Output : false

Sample input 2:
jAva7
Output : true

Sample input 3:
jva723
Output : false

func2: (5 marks)

Takes a string as input and returns a boolean value specifying whether or not the string meets the following pattern:

A string that starts from 0 or more "a", followed by at least one "b" and ending with "c".

Sample input 1:
aaaaaabbc
Output : true

Sample input 2:
bcc
Output : false

Sample input 3:
bc
Output : true

<u>func3: (5 marks)</u>

Another concept in Automata Theory is "Context-Free Grammars". CFGs are more expressive languages than RL (Regular Languages). Define func3 which takes a string as input and returns a boolean value specifying whether or not the string meets the following CFG:

{a^nb^n; n>=1}

Sample input 1:
ccaaaabbb
Output : false

Sample input 2:
aaaabbbb
Output : true

Sample input 3:
<Empty string>
Output : false

func4: (5 marks)

Takes two strings as input (an input string and a pattern string). The function should return a list of non-overlapping substrings extracted from the input string which matches the given pattern.

NOTE: the substrings are not overlapping each other

Sample input 1:
fun4("aaaaaabbccccaaaaaabbcqqq bc ccc acc", "a*b+c")
Output :
[aaaaaabbc, aaaaaabbc, bc]

Sample input 2:
fun4("aaaaaabbccccaaaaaabbcqqq bc ccc acc", "aa+bbb+cc")
Output :
[ ]

Problem 4 (Marks: 25)

Create a class called *Matrix.* The matrix or its elements should not be accessible without using the methods provided by the class. Implement the following methods in your class:

1. **Constructor:**
    a. integer n, float v: create an n x n matrix with all elements equal to v
    b. integer n, integer m, float v: create an n x m matrix with all elements equal to v
    c. integer n, integer m: create an n x m matrix with all elements equal to 0.0
    d. integer n: create an n x n matrix with all elements equal to 0.0
2. **add():** will take a *Matrix* object as argument and return the sum of the matrix object and the argument matrix object (return type should be Matrix). If the matrices are not addable, print "Matrices cannot be added" and return a 1x1 matrix with all values equal to 0.
3. **matmul():** will take a Matrix object as an argument and return the product of matrix object and the argument matrix object(return type will be Matrix). If the matrices cannot be multiplied, print "Matrices cannot be multiplied" and return a 1x1 matrix with all values equal to 0.
4. **scalarmul():** will take an integer as input and update the current matrix by multiplying all its elements by the scalar. Won't return anything.

5. **getrows():** returns the number of rows in the matrix
6. **getcols():** returns the number of columns in the matrix
7. **getelem():** takes in two integer values as arguments and returns the value at that index (first integer corresponds to the row and the second corresponds to the column - 0-based indexing). If the indices are out of bounds, print "Index out of bound"and return -100.
8. **setelem():** takes in two integers and a floating point number as arguments (in that exact order) and sets that value at the specified index to the float argument. (first integer corresponds to the row and the second corresponds to the column, both 0-based indexing). If the indices are out of bounds, print "Index out of bound". Won't return anything.
9. **printmatrix():** prints (not returns) the matrix with elements in a row separated by spaces and rows separated by newlines. **(check thread on piazza for output format)**

Your solution file should contain nothing but the class definition. NO main method. (It's recommended that you make a separate java file for the main method during testing, which imports the class file and calls these methods. This is only to avoid submitting a wrong file which consists the main method itself. Do not submit any extra files.)
Also, make sure to name the class methods EXACTLY as given in the problem statement as well as make them PUBLIC.

Problem 5 (Marks: 25)

Java is an Object Oriented Programming language and the concept of classes is deeply rooted in its implementation. Leverage this to implement a teacher-student relationship as follows. You are given 3 empty classes in the example submission folder for this question, namely: *Person.java, Teacher.java,* and *Student.java*

Follow these instructions EXACTLY as given, otherwise, your output might not be valid according to the auto-grader.

The elements of the classes should not be accessible without using the methods provided by the class. Implement the following functionalities for the given classes:

1. Structure of **Person** class:
    a. Constructor:
        i. String name, int age: to initialize values of members

   b. **getName():** returns the name of the person
   c. **getAge():** returns the age of the person
   d. **intro():** prints an introduction in the following format
      "I am a person, <name>, <age>"

2. Structure of **Teacher** class:
   a. This class inherits from **Person** class
   b. Constructor:
      i. String name, int age, int salary: to initialize values of members
      ii. String name, int age: to initialize values of members with default salary as 10000
      iii. Person p, int salary: to initialize values of members using Person object
   c. **getSalary():** returns the salary earned by the teacher
   d. **addStudent(Student stud):** add a new student to the current list of students under this teacher (use ArrayList)
   e. **getStudents():** returns an "ArrayList" of students under this teacher
   f. **intro():** prints an introduction in the following format
      "I am a Teacher, <name>, <age>, <salary>"

3. Structure of **Student** class:
   a. This class inherits from **Person** class
   b. Constructor:
      i. String name, int age, int rollno: to initialize values of members
      ii. Person p, int rollno: to initialize values of members using Person object
   c. **getRollNo():** returns the rollno of the student
   d. **addTeacher(Teacher teachr):** add a new teacher to the current list of teachers who are his/her guides (use ArrayList)
   e. **getTeachers():** returns an "ArrayList" of teachers who are guides of this student
   f. **intro():** print an introduction in the following format
      "I am a Student, <name>, <age>, <rollno>"

Your solution file should contain nothing but the class definition. NO main method. (It's recommended that you make a separate java file for the main method during testing, which imports the class file and calls these methods. This is only to avoid submitting a wrong file which consists of the main method itself. Do not submit any extra files.)

Also, make sure to name the class methods EXACTLY as given in the problem statement as well as make them PUBLIC.

File Structure:

```
q5
├── Person.java
├── Teacher.java
└── Student.java
```

Final Submission Structure:

```
rollno1-rollno2-outlab6/
├── references.txt
├── q1
│    └── q1.java
├── q2
│    ├── q2.java
│    ├── input.txt
│    └── output.txt
├── q3
│    └── q3.java
├── q4
│    └── Matrix.java
└── q5
     ├── Person.java
     ├── Teacher.java
     └── Student.java
```