COMPUTER SCIENCE 21A (FALL TERM, 2015)
DATA STRUCTURES

PROGRAMMING ASSIGNMENT 1

Due Tuesday, September 29 @ 11:00pm

IMPORTANT

1. Your code should be well commented:
   - Add your name and email address at the beginning of each .java file.
   - Write comments within your code when needed.
     - You must use Javadoc comments for your classes and methods.
     - You must write each method's running time in your Javadoc comments for that method.
2. Before submitting your assignment, you must zip all your files.
   - The ZIP file must have the naming convention LastnameFirstname-PA1.zip
3. Submit your assignment via LATTE using the assignment submission link.
4. Excluding arrays, you may only use data structures that you implement yourself.
   - You must justify your choices of data structures for each question.
   - Your data structures should not have hardcoded data types. For example, your implementation of a stack should be able to handle Integers, Doubles, Strings, or any sort of Java object.

Question 1

In **infix notation**, the operator (+, - , *, or /) appears between the operands. For example:

- 2 + 2
- 3 / 5

An alternative is **postfix notation**, where the operator appears at the end. The expression above would be written as:

- 2 2 +
- 3 5 /

The advantage of postfix notation is that complicated expressions do not require parentheses.
For example, using infix notation, we need parentheses for the expression: (5 – 2) * 4. The value of this expression is 12. If we leave out the parentheses, this becomes: 5 – 2 * 4, which is -3. Using postfix notation, the expression (5 – 2) * 4 becomes: 5 2 – 4 * and the second expression 5 – 2 * 4 becomes: 5 2 4 * -

No parentheses, no ambiguity!

Write a postfix calculator.

- Implement the ideal data structure for solving this problem.
- Using this implementation, finish `PostfixCalculator.java`. Your calculator should accept expressions one operator or number at a time in postfix order.
- Any time an operator is entered, the current value of the equation should be displayed to the user. The program continues with the current equation unless the user enters `clear`, which clears the current equation and starts a new one.
- Your calculator should support operators `+`, `*`, `-` , and `/`.
- Use doubles, not integers. Convert input into a double with the method `Double.parseDouble()`

Question 2

The mailroom in the Usdan Student Center is very busy. Many students walk by it on their way to class in the hopes of picking up their packages. However, sometimes the line is too long for the student to wait, and they must forgo waiting in line in order to get to class on time.

Your task is to create a model of this behavior and calculate the average waiting time per student. **You must implement a queue using arrays**.

For the purpose of this problem, you must make a few assumptions:

- It takes 5 units of time for the student being served to receive their packages.
- A maximum of 10 students can wait in line to collect their packages.
    - If the line is already full and a student arrives at the mailroom, the student will permanently leave. If the line later becomes not full and a different student arrives, that new student will join the line.

Problem: Given the arrival times at the mailroom, calculate the average waiting time per student.

Arrival times will be provided as a text file of integers. The first integer in the file is the arrival time of the first student to get their package, the second integer is the arrival time of the second student… the nth integer is the arrival time of the nth student.

Your code must support input of arbitrary size. That is, the input file may have any number of arrival times in it.

Your program should look like this (underlined is user input):

```
Usdan Mailroom Simulation:
      Please specify the input file: MailroomTest.txt

Here are the results of the simulation:
      Average waiting time: 49
```

Question 3

The children's game, "Duck, Duck, Goose," is played by having a group of children sit in a circle. One child who has been elected "it", walks around the outside of the circle. When this child visits each other child in the circle, he or she will designate that child as either "duck", or "goose". By labeling a child as "duck", nothing happens, and the child who is "it" moves on to visit the next child in the circle. However, after labeling a child as "goose", the two children, "it" and "goose", must race around the circle to try to get to the goose's former spot in the circle. Whoever reaches the spot in the circle first will sit down, while whoever is left standing becomes the new "it".

Your task is to simulate this game in Java.
Choose and implement the appropriate structure (or structures) for solving this problem.

Your program is expected to behave as follows:

- Process a provided input file, generating the circle of children in the order specified in the file.
- The child specified as the first line in the provided file Game.txt will be the initial child designated as "it".
- All subsequent elements in the file will be integers specifying how many "ducks" there are before the next goose.
- When a child is designated "goose", the child who was "it" will sit in the goose's former spot in the circle, and the goose becomes "it".

Example inputs:
Circle.txt:

```
Frida
Pablo
Caravaggio
Artemisia
Leonardo
```

Game.txt:

```
Sandro
1 2 2 5 100 2000 0 -3
```

Sandro will start next to Frida (as she is the first person in the circle), and he will move around the circle one position, where he will designate Pablo as "goose". Thus, Sandro sits down, and Pablo becomes "it". Pablo moves two positions around the circle. He passes Caravaggio, "duck", and then Artemisia, "goose!" Thus, he sits down and Artemisia becomes it. Artemisia moves two positions around the circle. She passes Leonardo, "duck", and then Frida, "goose!"

At this point, the circle looks like:
[Artemisia, Sandro, Caravaggio, Pablo, Leonardo]

This continues until the end of the file.

Once this processing is completed, whoever is still "it", and the final order of the circle should be printed out to the console.

```
Input circle file: circle.txt
Input game file: duckduckgoosetest.txt
Caravaggio is still it!
[Frida, Pablo, Leonardo, Artemisia, Sandro]
```

Your implementation must work on input of arbitrary size. Thus, the circle file will have some arbitrary quantity of Strings (each on a different line), and the game file will have a String on the first line, and then some arbitrary quantity of Integers following.