Homework 3

1. Algebra

The purpose of this exercise is to practice the art of writing and calling functions. You have to develop a little world in which the basic algebraic operations are expressed as functions:

$a + b$ is represented as `plus(a,b)`

$a - b$ is represented as `minus(a,b)`

$a * b$ is represented as `times(a,b)`

$a \hat{} b$ is represented as `pow(a,b)`

The integer part of a / b is represented as `div(a,b)`

a % b is represented as `mod(a,b)`

The integer part of $\sqrt{x}$ is represented as `sqrt(a)`

For example, the expression `2 * (4 + 3)` is expressed as `times(2, plus(4, 3))`.

Your task is to implement all the functions shown above, without using the Java operators `+`, `-`, `*`, `/`, `%`, and the functions `Math.pow` and `Math.sqrt`. The only algebraic operations that you are allowed to use are `++` (add 1), `x--` (subtract 1), `<`, `<=`, `>`, `>=`, `==`, and `!=`. You are allowed to use any other Java element that we learned, including `while`, `do while`, and `for`.

Inspect the given `Algebra.java` class, and implement all the functions. You are welcome to add more tests to the `main` function, as you see fit.

Implementation tips:

(1) To add `a` to `b`, we can add `1` to `a`, `b` times. This is the basic spirit of this exercise.

(2) When writing a function, try to use other functions that you've already implemented. For example, `times` can be implemented using `plus`.

(3) Implement the functions in the order in which they appear in the class.

2. Anagrams

An *anagram* is a word or a phrase formed by rearranging the letters of a different word or phrase, using every original letter exactly once. For example, the word "`listen`" can be rearranged into "`silent`". As we did with palindromes, we disregard spaces, punctuation marks, and upper/lower case letters. For example, "`anagram`" and "`Nag a Ram`" are anagrams.

Inspect the given `Anagram.java` class, and implement all its functions.

Implementation notes:

1. Start by reading the `main` function. Make sure that you understand all the tests.

2. Implement the `preProcess` function, and write tests that test it. You can add these tests to the `main` function, as you see fit.

3. Implement the `isAnagram` function. Start by pre-processing the two strings. Then check if the two resulting strings form an anagram. A natural implementation is to use a nested loop.

4. Implement `randomAnagram`. Note that this function is not supposed to return a word or phrase in the English language. Rather, it should return some random permutation of the characters in the given string. For example, a random anagram of the string "`java`" may be, say, "`ajva`". One way to implement this function is to use a loop that draws a random character from the string and then deletes the selected character from the string.

Submission

Submit the following files only:

- `Algebra.java`
- `Anagram.java`

Zip the five files into a file named `hw3.zip`, and submit it by following the submission instructions given in Moodle.

Get feedback: To get feedback about your programs before submitting them, use GETFEED, as many times as you want.

Deadline: Submit Homework 3 no later than Sunday, November 22, 2020, 23:55. You are welcome to submit earlier.

Preparation: You have what it takes to complete the `Algebra` class and the `randomAnagram` function in `Anagram`. By the end of Friday, November 13, you will have the knowledge necessary for completing the rest of the `Anagram` class.