**CS 9053**
Tuesday, March 31, 2020
Prof. Dean Christakos

**Assignment 6**
**Due: April 7, 2020**

How to submit: create a zip file or tar file with the following format:

```
Assignment6/
        PartI/

        PartII/

        PartIII/
```

**Part I: Events**

The code in SimpleAddition.java will create a window with two text fields containing numbers and a JLabel that displays the sum. Any time either text field is changed, an event is generated. Create event listeners for each text field such that the value of the Sum data field is updated to have the latest sum of the two addends.

You do not have to worry about any of the code in setupComponentValues(), nor do you have to understand any GUI development.

You must add the correct arguments to `addend1Field.addActionListener()` and `addend2Field.addActionListener()`. Minimize redundancy of code.

**Part II: Lambda Expressions, Inner Classes**

In part three, there's a class called "RandomWords". There's a file in the "data" directory called "words" with 100 different words. Your assignment is to randomly take 10 of those words. Use the Java methods `Math.random`, `java.util.Random.nextInt` or `java.util.Random.ints` to pick them.

The other thing you will have to learn about is Pair from the javafx library. A Pair is data type that contains two values, a key and a value. You can create a pair of words using the method add(word). **Note: when using Eclipse, you may get an error when you import javafx.util.Pair, telling you there is an API restriction. See here to fix it:** https://stackoverflow.com/questions/25222811/access-restriction-the-type-application-is-not-api-restriction-on-required-l.

You would parameterize the Pair like so:

```
Pair<String,String> wordPair = new Pair<String, String>(word1, word2);
```

And retrieve data from it like so:

```
wordPair.getKey();
wordPair.getValue();
```

It's up to you to ensure that there are no pairs that are the same and no pairs of both (s1, s2) and (s2, s1).

You want to choose every pair of the 10 words you choose out of the 100. This is "10 choose 2" pairs = 45 pairs, from the formula $\binom{n}{k} = \frac{n(n-1)\cdots(n-k+1)}{k(k-1)\cdots 1}$, or $\frac{n!}{k!(n-k)!}$.

You are going to calculate the Levenshtein Distance between each of the 45 pairs of words https://en.wikipedia.org/wiki/Levenshtein_distance. This is a number that represents the "difference" between two words, describing how many edits it takes to convert one word to another. **NOTE: YOU MAY NOT USE A LIBRARY THAT IMPLEMENTS THE LEVENSHTEIN DISTANCE. YOU HAVE TO DO IT YOURSELF.**

If you can calculate the Levenshtein distance between all these pairs of words, then you can sort the list of pairs.

Use the ArrayList class to store a list of Pairs.

Sort the list of Pairs by the Levenshtein distance and return a list of Pairs. You will do three things:

- Implement the Levenshtein distance between two Strings. This is a static method in `RandomWords`. It will be callable from anywhere by calling `RandomWords.levenshteinDistance`
- Create a Comparator as an Inner Class and pass the comparator into `ArrayList.sort()`
    - The `Comparator<Pair<String,String>>` will have a method `compare(Pair<String,String> p1, Pair<String,String> p2)`
- Write a Lambda function as an argument to `ArrayList.sort()` that sorts the pairs by the Levenshtein distance in ascending order

**Part III Single Abstract Method Interfaces and Lambdas:**

In part III, there are two files. The first is called `MathOperation.java`, a Single Abstract Method Interface which has one method, "`operation`," which returns a double and takes two doubles as parameters. The other is `ResultPrinter.java`. It prints out the result of

`MathOperation`, which you can pass into the constructor, on two arguments, `a` and `b`. There are also two static methods: `go(double a, double b, MathOperation op)`, which takes in two arguments and a `MathOperation` and `go(Collection<Pair<Double,Double>> c, MathOperation op)`, which takes in some kind of collection of pairs of doubles which will be applied to the `MathOperation`.

Implement the three versions of "`go`".

Pass in the `MathOperation` parameter to the constructor, and to the static `go` methods as a Lambda.

Instantiate `ResultPrinter` with a simple math operation that does addition. Execute `rp.go()` and show that it gives the correct result.

Implement and execute the static method `ResultPrinter.go` with two arguments and a Lambda function that does multiplication.

Implement and execute the static method `FunctionResultPrinter.go` with the array list of doubles and a lambda function that does division and have the method loop through the array list of pairs and execute that operation on the two members of the pair as arguments.

Show what the output printed out is.