

## 10 Assignment

### Portfolio Problems: Loops

Finish the first part of Lab 10 that deals with loops.

#### Problem 1: Selection Sort

Complete the second part of **Lab 10** that introduces the *selection sort*. Add an example that sorts a list of `Strings` by their length.

#### Problem 2: Insertion Sort

We have seen the recursively defined *insertion sort* algorithm both in the first semester and also recently, using the recursively defined lists in Java.

The main idea behind the insertion sort was that each new item has been inserted into the already sorted list. We can modify this as follows:

1. Design the method `sortedInsert` that consumes a sorted `ArrayList<T>`, a `Comparator<T>` that has been used to define the sorted order for the given list, and an item of the type `T`. It modifies the given `ArrayList<T>` by adding the given item to the `ArrayList<T>`, preserving the ordering.

*Note:* Be careful to make sure it works correctly when the given `ArrayList` is empty, and when the item is inserted at the end of the `ArrayList`.

2. Design the method `insertionSort` that consumes an arbitrary (unsorted) `ArrayList<T>` and a `Comparator<T>` and produces a new sorted `ArrayList<T>` as follows:

It starts with an empty sorted list and inserts into it one by one all the elements of the given `ArrayList<T>`.

*Note:* It is a bit more difficult to define the insertion sort algorithm so that it mutates the existing `ArrayList` *in place*.

3. **Extra Credit** Design an in-place `insertionSort` method. You will get the credit only if the design is neat and clearly organized.

### Problem 3: ArrayList of Strings

Design the following two variants of a method in the `Algorithms` class that reverses the order of the words in the given `ArrayList<String>` (we assume each `String` is one word):

1. Design the method `reverseProduce` that takes one argument of the type `ArrayList<String>` and produces a new `ArrayList<String>` that contains all words from the given list, but in a reverse order.

So, if the original `ArrayList<String>` contained the Strings `["who" "what" "why"]` the new `ArrayList<String>` will contain the Strings `["why" "what" "who"]`.

2. Design the method `reverseInSitu` with the `void` return type that mutates the given `ArrayList<String>` by reversing the words it contains.
3. Finally design the method `printWords` that consumes an argument of the type `ArrayList<String>` and prints each entry on a new line, traversing the `ArrayList<String>` using the 'for-each' loop that uses the Java `Iterator`. Use

```
System.out.println(someString);
```

to print each line.

### Problem 4: Working with the `StringTokenizer`

Continue working with the same project, designing your solutions in the `Algorithms` class.

1. Look up the `StringTokenizer` class in JavaDocs. The methods there allow you to traverse over a `String` and produce one word at a time delimited by the selected characters. Read the examples. Then write the method `makeWords` that consumes one `String` (that represents a sentence with several words, commas, and other *delimiters* and produces an `ArrayList<String>` of words (Strings that contain only letters — we ignore the possibility of words like "don't"). The delimiters you should recognize are the comma, the semicolon, and the question mark.

2. The text in the `ArrayList<String>` `words` in the class `Words` is a secret encoding. It represents verses from a poem - if you read only the first words. Design the method `firstWord` that produces the first word from a given `String`. Use it to decode the poem.