Assignment #3
2002-91 (2019 Spring)
3/8/19

175 Possible Points -- 100 for sorting solutions, 25 for timings table, 50 for User Docs

See Course Notes -- Part B: Sorting

1 of 6 -- Required Interfaces

The interfaces are included in the ZIP file "src.zip".

Generics are optional. If **not** using generics, then please use interfaces from these two packages:

- interfaces
- sortspecs

If using generics, then please use interfaces from these two packages:

- genericinterfaces
- genericsortspecs

2 of 6 -- IndexableList and Linked List

From assignment #2, you already have an implementation of the IndexableList interface. You will use it for this assignment.

From assignment #2, you have coded up a linked list or two (maybe a doubly linked). For this assignment, please make a small tweak and have these classes now "implement" the ListDs interface.

It is cool to do your "arrayed" sorts using a Java "List/ArrayList". Simply construct a IndexableList on your resulting sorted list when needed (one-liner).

3 of 6 -- Implement Sort Classes

Code **four** sort classes. Each must implement the *SortMethod* interface. See "SortMethod.java" for a description of the two methods that the interface declares.

Here are the sorts to solve:

- Implement merge sort on an indexable type of list. **(one java file)**
- Implement merge sort on a linked list. **(one java file)**
- Implement two more sorts **(two java files)**. Choose any two from the list below:
    - Selection Sort
    - Bubble Sort
    - Quick Sort

For the sorts that sort an indexable list you may use Java's ArrayList. For sorts that sort a linked list,

you'll use your own linked list (from Assignment #2). And again, please see “SortMethod.java” for the two methods that must be overridden in each sorter to satisfy the interface.

4 of 6 -- SorterFactory

Please provide a SorterFactory. Do not worry about this until the very end. This is just icing to help your customer. Please use the naming convention as shown below for your four sorts. Again, you may, or may not, use generics (your choice). Naturally, if your sorters use generics, then use generics here too.

Without Generics

```
public class SorterFactory {

	public final static SortMethod newLinkedInsertionSorter(Object... args) {
		BODY
	}

	public final static SortMethod newArrayedInsertionSorter(Object... args) {
		BODY
	}

	Etc.
}
```

With Generics

```
public class SorterFactory {

	public final static <DT> SortMethod newLinkedInsertionSorter(DT... args) {
		BODY
	}

	public final static <DT> SortMethod newArrayedInsertionSorter(DT... args) {
		BODY
	}

	Etc.
}
```

5 of 6 -- Timings

When you are done with your timings, prepare a table (and submit as **CSV**). Please convert to CSV with name SortTimings.csv. This does not have to be automated. It is suggested just to manually compose a spreadsheet table in the format shown below, and then save off as CSV file to submit.

Here is the table with the required column headings with a couple example rows

Sort Method	Data Structure	Data Size	Sort Time
Insertion Sort	Indexable List	75,000	20,495 ms (20.5 sec)
Insertion Sort	Linked List	75,000	15 ms (0 sec)

6 of 6 -- User Docs

Please submit user docs in a similar fashion as we did for “Assignment #2, Part II (User Docs)”. Keep it as simple as possible. the docs do not need to be long and verbose. Think concise, clear, simple, and complete. Like Assignment #2, here is the user docs checklist:

1. README-FIRST.TXT -- 2 Points
2. Installation Guide (txt or pdf) -- 4 Points
3. User Guide (txt or pdf) -- 4 Points
4. JAVA files that demonstrate your software -- Nine or more JAVA files -- 20 Points
5. Example Output -- Nine or more TXT files -- 20 Points

Total = 50

Reference

Below is our revised data structure interface hierarchy. The hierarchy now has room for tree data structures (future work). These changes will impact your "Assignment 2" code very little.

For this assignment, the one change you will make as mentioned in item 2 of 5 above, is that your linked list class or classes should now implement the "ListDs" interface. Your classes should already come close to having the necessary methods, with a few code tweaks to satisfy the spec (interface).

```
DataStructure
      LinearDs
            GeneralDs
                  ListDs
                        IndexableList
            SpecializedDs
                  Queue
                  Stack
                  etc.
      TreeDs
```