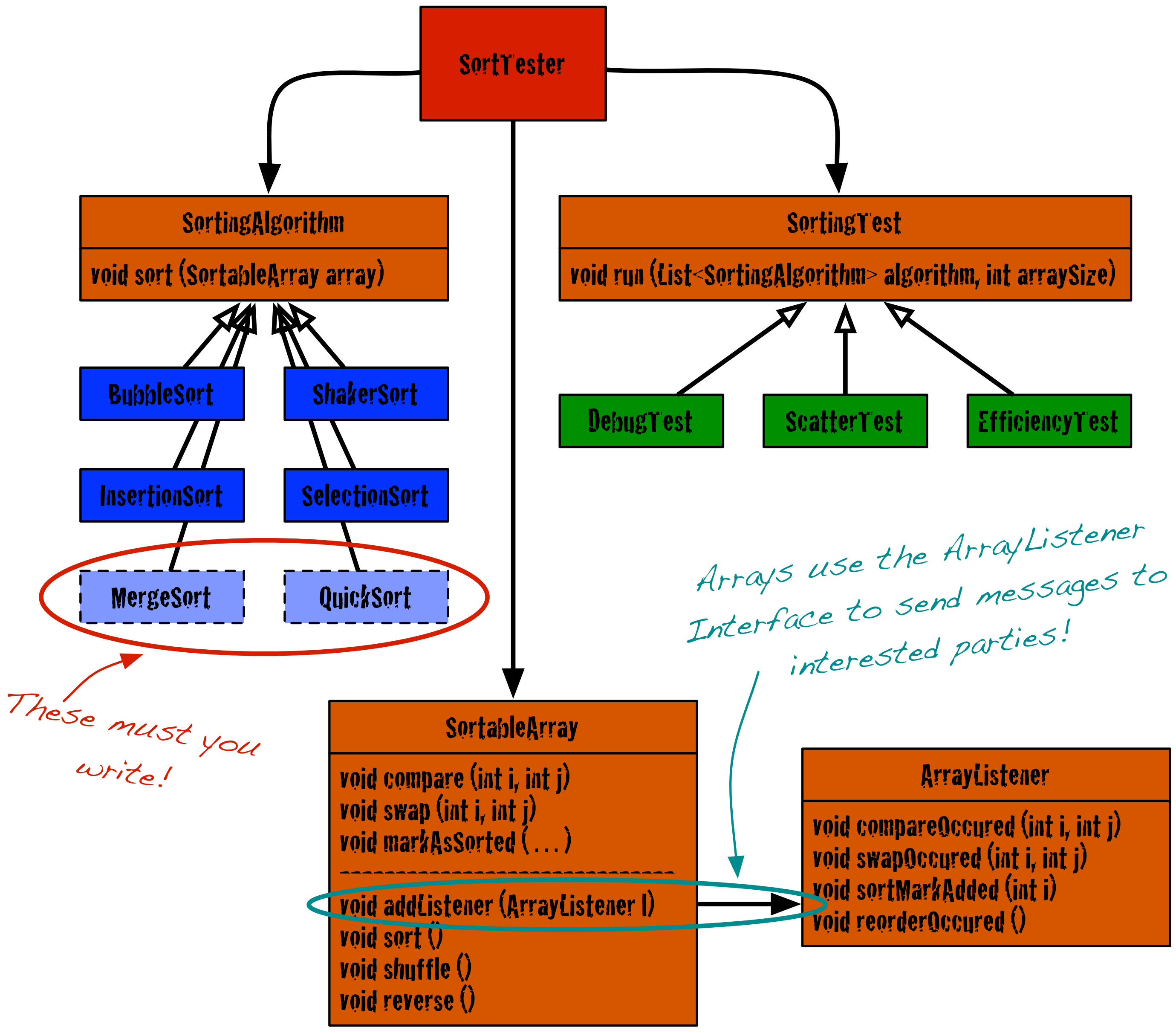
The *Even Cooler* Sorting Hat


SortTester
SortingAlgorithm
void sort (SortableArray array)
BubbleSort
ShakerSort
InsertionSort
SelectionSort
MergeSort
QuickSort
These must you write!
SortingTest
void run (List<SortingAlgorithm> algorithm, int arraySize)
DebugTest
ScatterTest
EfficiencyTest
Arrays use the ArrayListener Interface to send messages to interested parties!
SortableArray
void compare (int i, int j)
void swap (int i, int j)
void markAsSorted (. . .)
void addListener (ArrayListener l)
void sort ()
void shuffle ()
void reverse ()
ArrayListener
void compareOccured (int i, int j)
void swapOccured (int i, int j)
void sortMarkAdded (int i)
void reorderOccured ()

It seems you did too well last time. Dotted your i's, crossed your t's, even learned how to draw an ampersand. Because *Pen-Fifteen* is back. And this time it's with bad news.

One of their best people was working on a redesigned sorting visualizer when she fell prey to a gruesome sorting accident. No-one knows exactly what happened; they probably never will. All they ever found was her spleen and left humerus.

It is up to you to finish her work.
Please try not to suffer a similar fate.
It would be most inconvenient.

To: AGENT [REDACTED]
From: Pen-Fifteen
Subject: Re: Sorting Algorithms

Start with the top: SortTester.java. Make sure you understand every single line in it, because it makes everything else happen. You will need to modify it only slightly to add your own *SortingAlgorithms* and *SortingTests*.

Once you're ready, copy your code from the last assignment into the *ShakerSort*, *SelectionSort*, and *InsertionSort* classes sort() methods. Now you can begin testing. There are three built in test modules, and you will write a fourth. After compiling with

```
javac SortTester
```

you should take them each for a spin:

- **java SortTester debug bubble 8**
  Displays a step by step animation of any sort.
- **java SortTester scatter bubble:shaker:selection:insertion 128**
  Displays side-by-side scatter plots for each sort with a list size of 128 items.
- **java SortTester efficiency bubble:shaker:selection:insertion 10000**
  Displays efficiency graphs and tables for each sort with a list size of 10,000 items.

The *SortableArray* class has also been improved since the last assignment:

1. Each *SortableArray* actually contains two arrays: a data array and a temporary array of the same size. This will be necessary for merge sort. You may copy values back and forth between them using the following methods:
   ```
   copyToTemp (int index, int tempIndex)
   copyFromTemp (int tempIndex, int index)
   ```
2. *DebugTest* allows you to pause at any point by pressing "p" or calling the array's **pause()** method in your *SortingAlgorithm.*

Armed with these tools, you should be ready to tackle the two Nlog(N) sorts: *merge sort* and *quick sort*. These are tricky, but your late colleague has already written a to-do list for each. Follow these carefully and you shouldn't befall a similar fate.

Once you've finished, print out each of your algorithms and annotate them to explain why they are the efficiency they are. As always, [properly explained] pictures are worth a thousand words.

Good luck, good skills, and goodbye.
We hope it is only temporary.

Grading

Algorithms	
Merge Sort	(5)
Quick Sort	(5)
Code Quality	
Neatness, consistency, readability	(5)
Dead Trees Portion	
Algorithm annotation	(5)
Extra Credit	
What's another cool Test to include?	(5) <-- credit depends on
Write it!	(5) <-- awesomeness

Super Duper IMPORTANT Things

Talk through the following with your partner and ask me about about any you don't understand. These aren't graded because that would be annoying for all involved, but not being able to answer any of them indicates some conceptual misunderstanding which is in your (and your family and grade's) best interest to resolve.

1. How does main() work? You should understand EVERY SINGLE LINE! :-)
2. In general, how do you calculate the efficiency of recursive algorithms?
3. Where did the ArrayView code from the last assignment migrate to?
4. Why do you pass start and end to sort() rather than actually splitting the array?