**Homework 1 – Super List & Shapes (Report)**

Edward R. Gonzalez

4999406

**Introduction:**

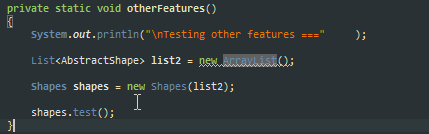
For the first homework assignment, the students were dispersed code as a skeleton, containing a file related to an ArrayList (termed SuperList), and a few files related to shapes. The goal was to complete the unimplemented functions of the Super List, and finish or create the necessary files pertaining to shapes, and test the results using a proper testing classes (Driver and Shapes). These actions are regulated based on the HW1.pdf provided with the skeleton file for the assignment.

**Approach:**

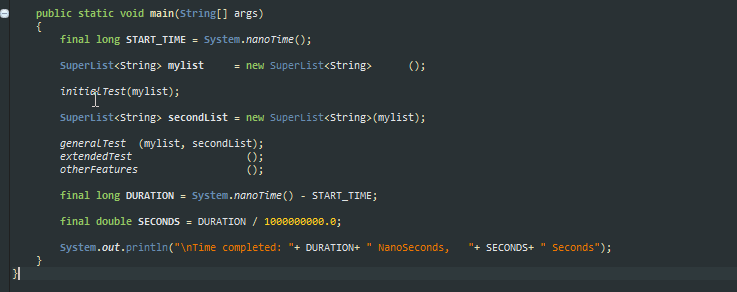
Initially the files in the skeleton were looked through and attempted to understand the base structure of the files and code. First the Driver class was looked at, then SuperList. Both files were refactored in order to both better understand everything, and have code laid out out in a manner more manageable for my mind. For each function unimplemented, the summary for the function was glossed over and then began looking at ways of implementing the proper code for the function on the fly. If syntax was unsure, the proposal mini window as well as various references for libraries imported, and the provided java documentation links. After completing one or many functions (depending on confidence and complexity) the implementation was tested with the “initialTest”. If the initialTest driver did not use the implemented function, it was tested in “generalTest”.

When SuperList was completed and functioned well enough to successfully complete generalTest, the shapes section of the assignment was looked into. Just as before with the main driver file and SuperList, all flies pertaining to shapes were refactored if necessary to an ideal spacial format with intentions to to scan and understand per-written code. The creation and implementation of the various classes and functions for shapes largely went in the order as described in the pdf provided. The circle and triangle super class were created, each with similar instance variables unique to those shape types with influence on their design from the rectangle class. Every super class and subclass created thereafter was given a constructor that could crate a new instance of the same class from a preexisting object. When all super classes and sub classes were created, the necessary functions were implemented in order of rectangle, circle, and triangle. The only exception was toString which, was implemented last and was done for every class at once. Initially just overwriting completely the AbstractShape's toString for ease of testing. With all initial implementation complete, each shape was tested and added one by one into the shapes tester function. The pre-coded shapes were tested first with each uncommented one by one with each a successful pass. Then the other shapes were added in and tested one by one as well. Each having their exceptions and logical errors ironed out after a couple of hours.

At this point the assignment was complete until the release of the second driver and shapes class along with the accidental release of the more complete super List. The second driver was implemented into my driver as extendedTest done right after general test with slight modifications for the sake of less scrolling through console. The unaware of the purpose of the completed SuperList, it was used to compare my implementation to those use and saw if it could fix problems encountered with the default expandList among other things that were forced to be changed to get through the generalTest (These are discussed in Discussion.).

**Results:**

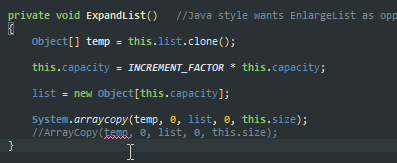
The code was tested using the driver class constructed while completing the assignment with the System's nanoTime function used to gauge performance. Three different cases were completed, the first two revolving the use of SuperList’s or ArrayList’s ArrayCopy function for ExpandList, and then another involving the use of SuperList or ArrayList for the shape list (SuperList using its own function for ExpandList.) Each case had 5 runs done with average calculated for those five. CPU: OC AMD FX8350 4.7ghz.

Screenshot (Taken with share X) Description: Time test code.

Case 1 (SuperList using its ArrayCopy):

Average: 0.0806065788 Seconds.

Case 2 (SuperList using System’s ArrayCopy):

  
Change in superList

Average: 0.0790150108 Seconds

Case 3 (otherFeatures Test function using ArrayList instead of SuperList):



Average: 0.076197587 Seconds

**Discussion:**

The performance difference in the average between -

cases 1 and 2: 0.0015915680 Seconds

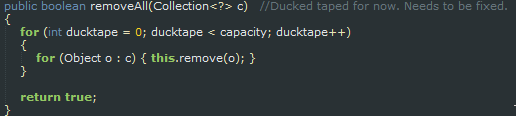
cases 2 and 3: 0.0028174238 Seconds

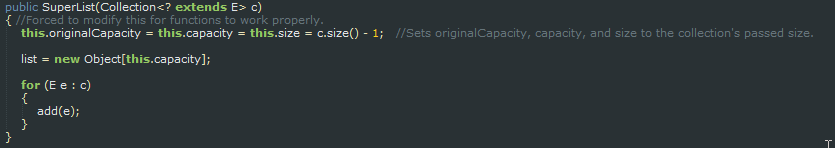
cases 1 and 3: 0.0044089918 Seconds

The is an always a reduction in time taken when either the ArrayCopy is used in SuperList originated from the System’s function or if ArrayList is used over SuperList. Although this time is in the milliseconds, it is enough to provide a significant performance boost in resource heavy programs especially since this is running on a VM.

On the my satisfaction with my implementation of SuperList. It worked out, and was a relief to get the entire thing working with all three test functions within the driver class. When completing SuperList, there was many difficulties involving the nullPointer exceptions and ArrayIndexOutOfBounds exceptions that only occurred with the general list. In order to work around this a few changes had to made to pre-implemented code, and an inefficient fix had to be used on the removeALl() function to work properly.

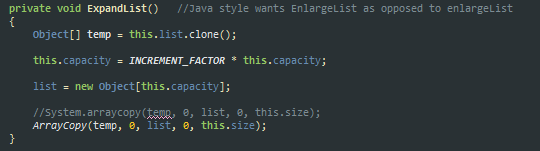
For the third SuperList constructor, its implementation had to be changed to this:



For removeAll, a for loop with a sentinent value of around the size of the list’s capacity had to be used to make sure every single item was removed. Otherwise the first indexed item may be not removed for some reason.

For removeLast, the parameter used needed to be size instead of list.length – 1 (Related to expand list’s problem).

For ExpandList, the last parameter for ArrayCopy in both System’s and SuperList’s had to be changed to use the size parameter over list.length, otherwise there would be an exception in generalTest.



**References:**

Foreach in C# vs Java:

<http://www.javacamp.org/javavscsharp/foreach.html>

Generics:

<https://en.wikipedia.org/wiki/Generics_in_Java>

Formula for Shapes:

<http://www.mathsisfun.com/area.html>

<http://www.mdk12.msde.maryland.gov/instruction/curriculum/hsa/geometry/math_reference_sheet.html>

Group Member: Matthew Garcia

Helped with triangle angle formula implementation.