You will be creating a SuperList class.
You will need to imbed a ListNode class into your SuperList class. Remember that ListNode is necessary to fill your SuperList, but is not a stand-alone object. If it is separate, you will lose points.

You must declare your SuperList and ListNode object with the use of generics so that the user can specify variable typing when filling the list.

Your SuperList needs to function like a Stack, a Queue, or an ArrayList based strictly on method calls.

SuperList will hold your standard variable types, but not as those types. You will be creating the ListNode class to actually store those values.

Keep in mind that at no time are you allowed to utilize any pre-existing data structures. You *CANNOT* use an array, ArrayList, Stack, or Queue in your code.

The ListNode class must:

- Have a class attribute for value
- Be declared to function with generics (use type E).
- ***getValue*** method
- ***setPrevious*** and ***setNext*** methods as well their respective "***get***" accessor methods
- ***hasNext*** and ***hasPrevious*** methods (returns boolean).

The SuperList class must have:

- Class variables to store the root and the end value of the SuperList along with their respective accessor methods
- Be declared to function with generics (use type E).
- Create a constructor that will establish a new empty SuperList
- Create an overloaded constructor that will create a new SuperList with a new root value received as a parameter (value needs to be received as a generic type but stored as a new ListNode)
- Create an ***add*** method that will allow you to append a new value to the end of the SuperList.
- Create an overloaded ***add*** method that will allow you to insert a value into the SuperList at a specified index location. Your method must throw an ArrayIndexOutOfBoundsException exception if you try to access a location that is not possible. Must function as an O(n) efficiency operation.
- Create a ***push*** method that will allow you to "push" a new ListNode onto the SuperList. The push method needs to add a new ListNode at the end of the SuperList. Must function as an O(1) efficiency operation.
- Create a ***pop*** method that will allow you to remove a ListNode from the end of the SuperList. Must function as an O(1) efficiency operation.

- Create a ***poll*** method that will allow you to remove a ListNode from the start of the SuperList. Must function as an O(1) efficiency operation.
- Create a ***stackPeek*** method that will allow you to "get" the end value that is stored in the SuperList without removing it from the SuperList. Must function as an O(1) efficiency operation.
- Create a ***queuePeek*** method that will allow you to "get" the start value that is stored in the SuperList without removing it from the SuperList. Must function as an O(1) efficiency operation.
- Create a ***get*** method that will allow you to return a value from the SuperList at a specified index location. Your method must throw an ArrayIndexOutOfBoundsException exception if you try to access a location that is not possible. Must function as an O(n) efficiency operation.
- Create a ***size*** method that will return the number of objects stored in the list. Must function as an O(1) efficiency operation. May require a small change to the class
- Create a ***remove*** method that will delete a ListNode from the SuperList based on a received index location. This method should return that removed value on the way out of the method.
- Create an ***isEmpty*** method that will return a boolean of true or false depending on whether there are any ListNodes stored within the list. If there are more than 0 ListNodes in the SuperList, return false. If there are 0 ListNodes in the SuperList, return true.
- Create a ***clear*** method that will remove all of the ListNodes from the SuperList. Must function as an O(1) efficiency operation.
- Create a ***contains*** method that will check to see if a given ListNode exists within the SuperList. Must function with O(n) efficiency.
- ***toString*** method to output the list. It should have a square bracket to start, commas separating each value, and a square bracket to end.
    - ***Example:*** [1,2,3,4] or, if empty, []

//Next page has the runner requirements

Runner

- Create a new class that will fill a "SuperList" of Integer that contains 30 random Integers from 1 to 1000. SuperList must be declared the same way that you would declare an "ArrayList" of Integer. You cannot create ListNodes here. You just add Integers to the list and the SuperList will handle the rest.
- Output the list. (You must use the toString method of SuperList.)
- Output the size of the SuperList.
- Move all values from the "ArrayList" version of the SuperList into a new SuperList that functions like a "Stack". (You must use the remove and push methods of SuperList.)
- Display the "Stack" version of the SuperList and store each value in a new SuperList that functions like a "Queue". (You must use the pop and add methods of SuperList.)
- Display the "Queue" version of the SuperList and store each value back into the original "ArrayList" version of the SuperList but in random "ArrayList" positions. (Reuse the now empty "ArrayList" version of SuperList. You must use the poll and add by index position methods of SuperList.)
- Calculate the sum of the list
- Calculate the sum of the even indexed values
- Calculate the sum of the odd indexed values
- Create duplicates of all even values from the "SuperList". Add each new value to the end of the SuperList.
- Remove all instances of values that are divisible by 3.
- Insert the value 55555 into the 4^{th} position in the SuperList.
- Sort the SuperList in ascending order.
- Search through your SuperList to find the "median" value (if there are an even number of values in the list, find the average of the values at position n/2-1 and position n/2). Identify that value and list the values that fall before and after the "median".

- Instantiate a new SuperList of String.
- Store a "sentence" in a string and then store each of the words in the sentence in your SuperList.
- Remove any strings from your SuperList that have less than or equal to 3 characters.
- Sort the SuperList of Strings in ascending order. You must write an insertion sort method to sort the values.
- Find the average word length.