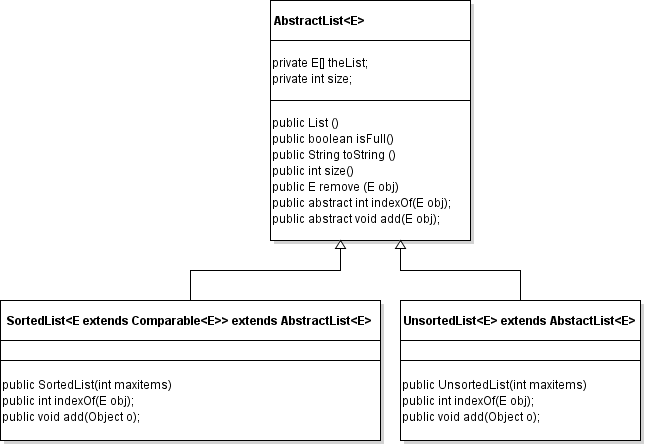
**List Assignment**

In this assignment you will work with a list hierarchy of classes which include: AbstractList, SortedList and UnsortedList. This design structure promotes code reusability and a shared interface between the two types of lists. The lists were purposely made simple to keep the implementation time low and keep the focus on the design aspects. The UML diagram below provides an overview of the classes, methods, instance fields and the relationships between classes. Review the UML diagram.



**ASSIGNMENT TASKS**

1. Review the UML diagram and the helper code for this assignment.
2. Create a new project with the helper code, then add a text file to your project called "Answers.txt" and place answers to the following questions in this file:
   1. What is the purpose for the <E> notation in the class signature line of AbstractList?
   2. Why is indexOf()method defined as an abstract method in AbstractList?
   3. Why is toString()method defined as a concrete method in AbstractList?
   4. When would you use a SortedList over an UnsortedList and vice versa? Provide specific application examples. Does it make sense to have two different types of lists or should you only have one?
3. Implement the **toString** method of AbstractList.
4. Implement the **add** and **indexOf** methods of UnsortedList. Test your changes.
5. Implement the **add** method in SortedList. When you make this method, use the indexOf method (which is in SortedList) to help you. There are detailed comments on this method that explain how it works. Also, you may find it helpful to use the System.arraycopy method to make room for the new item to be added into the list. See example usage below. Test your changes.

The below statement opens a new space at "index" for an insertion.

System.arraycopy(myArray, index, myArray, index + 1, size – index);

1. Create a **Driver** class with the below functionality. You may find it helpful to view example input/output on Mimir to get a conceptual idea of how the Driver will function.

*Scanner inFile = null; // scanner for file input*

*Scanner in = new Scanner(System.in); // scanner for console input*

*AbstractList<String> myList = null;*

*// Prompt the user for the file to load*

*do*

*{*

*display “Filename: “ and get filename from user*

*try {*

*attempt to assign inFile to a new Scanner with the filename specified*

*} catch the FileNotFoundException*

*{*

*display “Error – try again.”*

*}*

*} while the file is not found*

*do*

*{*

*// display “List type (1-2): “ and get the list type from the user. 1 is for sorted, 2 is for unsorted.*

*if the user response is invalid (not a “1” or “2”)*

*display “Error – try again.”*

*else*

*assign myList to the appropriate type of list with a max size of 20*

*} while the user input is invalid*

*// The below while loop loads data from the file*

*while the file hasNext() and the list is not full {*

*read a line of data from the file entered by the user; each line will contain a single word*

*add the line of data to the list*

*if hasNext() and the list is full*

*display “Warning – list did not load completely.”*

*}*

*do {*

*do {*

*display the following menu to the user:*

1. *Display all*
2. *Add item*
3. *Get item*
4. *Exit*

*Selection (1-4):*

*if the user response is invalid (not 1-4)*

*display “Error – try again.”*

*} while the user response is invalid*

*if the user option was 1*

*{*

*if the list is empty*

*display “List empty.”*

*else*

*display all the items in the list*

*}*

*else if the user option was 2*

*{*

*if the list is full*

*display “List full.”*

*else*

*{*

*display “Enter item: “ and get input from user*

*add the user data to the list*

*}*

*}*

*else if the user option was 3*

*{*

*display “Enter item: “ and get input from user*

*int index = myList.indexOf(…)*

*if the user data was found*

*display “Found”*

*else*

*display “Not found”*

*}*

*} while the user does not exit*

1. Test the runtime of calling indexOf for both a SortedList and UnsortedList. Follow the below steps:
   * 1. Modify your code to set the max size of myList to 127,142 (the number of words in "speedtest.txt", which we'll use for this exercise).
     2. Modify your code to print the runtime of calling indexOf (when the user selects get from the menu option) as follows:

long startTime = System.currentTimeMillis();

int index = myList.indexOf(…);

long endTime = System.currentTimeMillis();

System.out.println(endTime – startTime);

* + 1. When you run your program, use the “speedtest.txt” file and look for the indexOf “redness”. Test the runtime for both sorted and unsorted list. Record your results in your "Answers.txt" file.
    2. Undo your code modification from “a”, “b” and "c" above. These changes should NOT be uploaded to Mimir.

1. Add a **remove** method to your AbstractList and test its functionality in a driver class called "DriverRemoveTester.java". Below are the details for this method:

boolean remove(E obj) – Removes obj from the list, if present. Returns true if removed, false otherwise. There will NOT be any duplicates in the list, so you do not need to worry about that scenario.