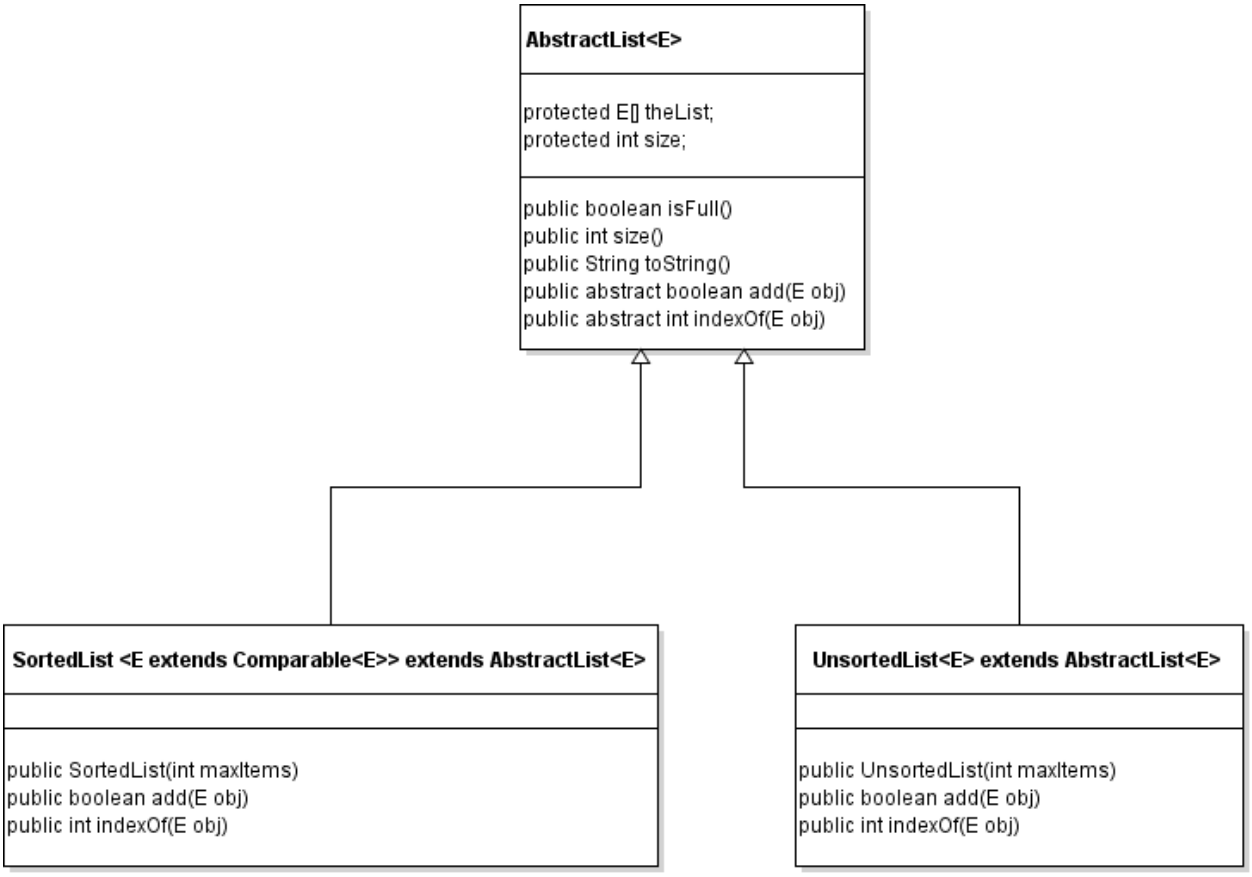
List Design

In this assignment you will create classes that yield a `SortedList` and `UnsortedList` in a design structure that promotes code reusability and a shared interface between the two types of lists. These lists are 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, consider possible implementations for the methods and answer the questions below.



Questions for you to answer before you begin programming are outlined below.

1) What is the purpose for the `<E>` notation in the class signature line of `AbstractList`?
2) The class signature line of `SortedList` states: "`SortedList<E extends Comparable<E>> extends AbstractList<E>`". Java's `Comparable` interface uses generics (e.g. `Comparable<E>`) to indicate the data type, `E`, that will be used to compare one object to another. "`SortedList<E extends`

`Comparable<E>>"` indicates `SortedList` will store `E` data types that implement the `Comparable<E>` interface. Note that Java uses the "extends" keyword (instead of "implements") to indicate `E` must implement an interface. The last part of the signature line, `extends AbstractList<E>,` indicates that the `SortedList` inherits from `AbstractList<E>`. Why does `SortedList` store `E` data types that implement the `Comparable<E>` interface?

3) Why is `add()` method defined as an abstract method in `AbstractList`?
4) Why is `toString()` method defined as a concrete method in `AbstractList`?
5) 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?
6) What is the benefit of using generics (i.e. the `<E>` notation) instead of having the lists store `Object` data types?
7) How would you go about adding `remove(E obj)` method functionality for the lists?

Driver pseudo-code

```
Scanner inFile = null; // scanner for file input
Scanner in = new Scanner(System.in); // scanner for console input
AbstractList<String> myList;

// Prompt the user for the file to load
do
{
    prompt the user for a file to load data into the list

    try {
        attempt to assign inFile to a new Scanner with the file specified
    } catch the FileNotFoundException
    {
        display an appropriate error message to the user
    }

} while the file is not found

// Prompt the user for the type of list they want
do
{
    prompt the user for the type of list they want, "1" for Sorted or "2" for Unsorted

    if the user response is invalid (not a "1" or "2")
        display an appropriate error message to the user
    else
        assign myList to the appropriate type of list with a max size of 50,000

} while the user input is invalid

// Load the data from the file into the list

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

    if the list is full
        display a warning message to the user indicating that the data did not load entirely
    else
        add the line of data to the list
}

do {

    do {
        display a menu option to the user similar to the below:
        1)   Display All Items
        2)   Add an Item
        3)   Get an Item
        4)   Exit
        Enter your selection (1-4):

        if the user response is invalid (not 1-4)
            display an appropriate message

    } while the user response is invalid

    if the user option was 1
    {
        if the list is empty
            display an appropriate message
        else
            display all the items in the list
    }
    else if the user option was 2
    {
        if the list is full
            display an appropriate message
        else
        {
            prompt the user for a String to add to the list (any String is valid)
            add the user data to the list
        }
    }
    else if the user option was 3
    {
        prompt the user for a String to lookup (any String is valid)

        startTime = System.currentTimeMillis()
        myList.indexOf(...)
        endTime = System.currentTimeMillis()

        if the user data was found
            display a message indicating it was found
            display the elapse time it took to perform the search
        else
            display a message indicating it was not found

} while the user does not want to exit
```