CIS 232 – Assignment 1

(125 points)
Due: Mon. 10/2 at 11:59p (code uploaded to Canvas)

In this assignment you will get practice working with the *book's* (p. 586) `ArrayList` implementation. You will amend the basic `ArrayList` class provided in the `weiss.util` package. All existing functionality must be preserved unless explicitly specified otherwise. Remember that `ArrayList` is generic.

Implement the Java class depicted below, using your own code. You may not use the API's `ArrayList` class or any other API collection save arrays. Be sure to follow the specs exactly. Please keep with the spirit of the assignment or risk a significant penalty.

You will revise the Weiss `ArrayList` class with the following modifications (Note that from this point onward substitute `"Whatever"` with your class name – details below):

1) The list must be sorted in ascending order at all times, with no null references in the middle of the data. The list must have this order maintained at the end of every public method call, so some existing methods will need to be modified.
2) Modify the `add()` (and any other supporting methods as you see fit) method of the class so that each new item is inserted in sorted (ascending) order.
3) Add a new public. method, `addAt()`, that takes the object to add and the index to insert it at, in that order. No items are replaced, just relocated. Note that if the index parameter is not the proper one for that item based on ordering the method should insert the item at the proper index that places it in order, not the requested one.
4) Modify the `set()` method so that it replaces and returns the item at the specified index while putting the new item in its correct location in the list as with `addAt()`.
5) Adjust the constructor that takes a `Collection` parameter so that it creates the `Whatever` with the same size as the `Collection` passed to it. The constructor still initializes the `Whatever` with the contents of the `Collection` parameter.
6) Add another constructor that takes in a lone integer parameter. The `Whatever` is initially created to this size rather than the default size.
7) Add an instance method named `getMode()` that returns the most-occurring value in the list. The method's return type is `Result`, a generic interface that has two methods: `AnyType mode()`, which returns the most-occurring item in the list, and `int count()`, which returns how many times that item occurs. Tiebreak on first-occurring item of the highest frequency. The code for the `Result` interface will be provided to you to incorporate into your package. Do not send this interface file with your code. Note that the object you return needs to be a very simple container with the values necessary to support the interface methods. It does not include any computation logic – all work should be done by your list class.
8) Add a generic `static` method that takes in a (your) `Whatever` and performs a binary search on it. The method takes exactly two parameters: a `Whatever`, and the item to search for. The type of the second parameter needs to reflect what could possibly be in the `Whatever` instance. This method returns an int that is the index in the list where the item resides, or –1 if not found. The method header will look *similar* (same name but you need to revise it to make it use generics) to:

```
public static int binSearch(Whatever list, AnyType item)
```

Note that in order to make these changes work, your data must be `Comparable`.

A few thoughts/tips:

- Copy/revise the Weiss `ArrayList`, but don't subclass - avoids visibility issues.
  - Have your class "extend" (implement really, but generics use the keyword "extend" for a parameterized type) `Comparable` …example:

```
public class Whatever<AnyType extends Comparable<? super AnyType>>
   extends AbstractCollection<AnyType> implements List<AnyType>
```

    - Where `Whatever` is the name of your class
- To use a `Comparable [ ]` within the class, try this (it's also kinda sideways):
    - When creating the array, make it `Comparable` and then cast:

```
theItems = (AnyType []) new Comparable[ DEFAULT_CAPACITY ];
```

I will test your `Whatever` with an appropriate data type. Be sure that you aggressively test your code to make sure it works as specified.

Your grade on this assignment will be largely determined by the correctness of your code (i.e. does it function as specified and give correct results.) Efficiency is a lesser concern (than it will be later) but code with notable redundancies and other correctable inefficiencies will be penalized. Documentation (see below) is 15% of the grade.

Additional notes:

1. Note: This is an individual assignment. The code, strategies, and approach must come solely from you with the exception of code provided by the course textbook or the instructor. There is to be no discussion or other exchange of information relating to this assignment with other students (present, past, or future.)
2. Code that does not compile as submitted results in a 0 for the assignment.
3. Assign your classes to this package: `cis232.a1`. `Result` is also in this package.
4. Be sure to import the Weiss classes rather than those in the API – there's less to implement, but definitely enough to get things done.
5. All instance variables must be `private`.
6. You do not have to do anything with the `SubList` inner class. You can delete that code, comment it out, or leave it in.

Various Administrative:

1. Name your source code file (and the java class) as follows: A1232, the first letter of your first name followed by your last name, up to a maximum of nine characters - e.g. `A1232RSco.java`
2. Upload a zip file consisting of a single `.java` source file and a PDF of your documentation (using a descriptive name that includes your file prefix).
3. Documentation requirements:
   - Easily readable source code. Comment where necessary and use unambiguous names for classes, methods, and variables. Use good coding style and ample (but not excessive or inconsistent) white space. Ragged code and other poor style will result in a noticeable deduction.
   - Write a short but complete (and typed) overview of the project, explaining your choice of design and algorithms as well as how/why you decided to implement `Result` the way that you did. Be clear in your explanation and use correct spelling and punctuation. Your writing needs to be college-level and written in a professional manner assuming an audience with at least equivalent knowledge of the subject area. Spelling, grammar, and tone absolutely count.