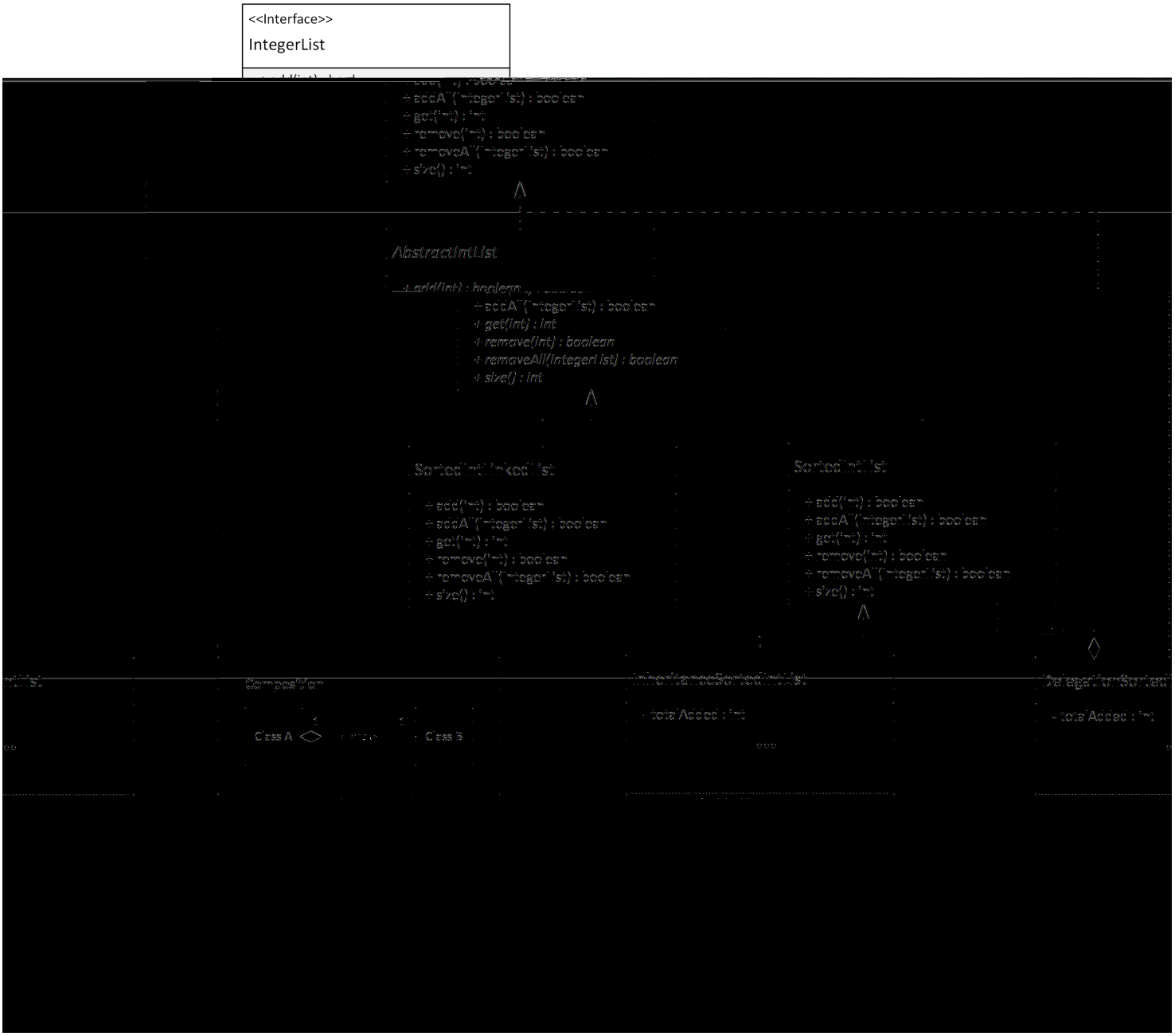
Inheritance and Delegation

In this recitation you will examine the relative strengths and weaknesses of inheritance and delegation by using both techniques to add a feature to a `SortedIntList` class. The `SortedIntList` class is one of a family of integer lists. It is very similar to the `AbstractIntList` that it extends, except it stores its elements in ascending order. The documentation in `recitation/04/doc/index.html` might be helpful.



Suppose we want to instrument a `SortedIntList` to count how many elements have been added since it was created. (This is not the same as its current size, which is reduced when an element is removed). To provide this functionality, you should count the number of attempted insertions and also provide an access method to get this count. Your solutions should use inheritance and delegation so you can reuse, but not modify, the original `SortedIntList` implementation.

Instrumentation with inheritance

One way to add this instrumentation is with inheritance. We have provided an empty `InheritanceSortedIntList` class where you should implement your inheritance-based solution.

Hints: `InheritanceSortedIntList` should extend `SortedIntList`. The `SortedIntList` contains two methods that add elements, `add` and `addAll`, so you should override both of these methods to track how many elements have been added.

After you have implemented the `InheritanceSortedIntList` class, open the `InheritanceSortedIntListTest` class and write some new tests that check the instrumentation you just added. Hint: make sure you check that `getTotalAdded` works with both `add` and `addAll`.

Run the tests and answer the following questions:

1. Did the test outcomes match your prediction? If not, why? (You might want to use the `printList` helper we provided you and read the documentation for `addAll` in the `AbstractIntList` class.)
2. If the tests failed, how could you change the `InheritanceSortedIntList` so it would no longer fail these tests?

Instrumentation with delegation

Another way to add this instrumentation is with delegation. We have provided an empty `DelegationSortedIntList` class where you should implement your delegation-based solution.

Hints: Give your new class a private field that references an instance of the `SortedIntList`. This design is called *composition* because the new class is composed of the existing one. Each instance method in the new class should invoke the corresponding method on the instance of the existing class and return the result; this is known as *forwarding*, and the methods in the new class are known as *forwarding methods*. The combination of composition and forwarding is loosely referred to as *delegation*.

After you have implemented the `DelegationSortedIntList` class, open the `DelegationSortedIntListTest` class and write some new tests that check the instrumentation you just added. Hint: make sure you check that `getTotalAdded` works with both `add` and `addAll`.

Run the tests and answer the following question:

1. Did the test outcomes match your prediction? If not, why?

Comparing delegation and inheritance

1. Which option, delegation or inheritance, is more dependent on the implementation details of the `SortedIntList`?
2. Can you think of a change to the implementation of `SortedIntList` that would affect one (or both) of your implementations?
3. Describe some problems with inheritance that using delegation solved in this example.
4. Describe some situations where it would be appropriate to use inheritance.