1 Overview

This project brings together several essential Java constructions and some important concepts that should be part of every student’s understanding of Computer Science. Sets, as we have consistently used that term in class, are an unordered collections of unique objects, called its “elements.” \{1, 2, 3\} is an example of a set containing three integers. Sets are deceptively simple, but a lot of very important results in computer science follow from a deeper study and understanding of them.

1.1 Our focus

For this project, you will implement two kinds of sets:

**Unordered Sets** These are exactly what we think of when we think of sets. In principle, the unordered set may contain any conceivable object type—including other sets.

**Ordered Sets** These are what we suspect: they are sets whose elements have some kind of ordering relationship—i.e., they can be compared with each other and placed in some kind of order.

The only “complication” is coming to terms with Java’s syntax that is used to specify the definitions that have been prepared for you. Remember: you never change any of these definitions, but it’s helpful if you understand what they mean when you see them.

1.2 Java syntax

Examine the interface definitions that are used in the implementation our set objects:

```java
public interface ISet<T> {
    public int size();
    public boolean isMember( T ele );
    public boolean isEmpty();
    public boolean adjoin( T ele );
    ....
}
```

The < T > is basically a “type variable,” i.e., a place holder that can be set to any valid first-class Java type, such as `String` or a class that you define. You will spend more time with this particular syntax and construction in the next course. For the time being, just treat every occurrence of the symbol that appears between the angle brackets as placeholders for type names at runtime.
2 What you need to do

You will implement the required methods of the ISet interface on the UnorderedSet class, and you will then rely upon Java’s inheritance mechanism to share a critical subset of these definitions on OrderedSet. You should note that most of the definitions from the interface that you define on UnorderedSet are the same as they would be for OrderedSet, such as size(), isEmpty, etc. This means, among other things, that when we extend the UnorderedSet class we can take advantage of these similarities and implement only the methods that are different for OrderedSets—adjoin() and equals(), and the constructors for each class.

2.1 The methods you must implement

Let’s focus first on the methods that properly belong to the sets interface.

- **size()** The size of a set is the number of elements it contains.
- **isEmpty()** Returns true if the set is empty—i.e., contains no elements.
- **adjoin( T ele )**. To “adjoin” an element is to add it to the set only if that element is not already in the set. Because elements in sets are unique, this method is used to add objects to a set only if they are not already found.
- **union()** The union of two sets contains all of the elements that appear in either set A or B. In the case of Ordered sets, moreover, newly adjoined elements must be added in ascending order.
- **intersect**. The intersection of two sets contain only the elements that appear in both.
- **difference**. The difference of two sets is a new set consisting of the elements in the first that are not in the second.
- **isMember(T ele)** Returns true if the element ele is found in set.
- **isSubset( ISet<T> other)** Returns true if all elements of the this set are members of the other set: thisSet ⊆ otherSet.

Most of these operations are identical for both unordered and ordered sets, except that the union, intersection, and difference methods need to respect the order of the underlying sets for OrderedSets. For example: the union of \{a, b, c\} and \{c, d, e\} might be \{c, d, e, a, b\} in the case of unordered sets, but must be \{a, b, c, d, e\} in the case of ordered sets.

If you think about this, it makes sense. Assuming that we use the adjoin method to construct new sets when computing their unions, intersections, and differences, then only the adjoin method differs from ordered and unordered sets. By implementing that method first, for both the ordered and unordered set classes, the union, intersection, and difference methods that we implement on UnorderedSets now just work for OrderedSets as well!
2.2 Additional methods

Naturally, we have to implement two different equals methods because unordered sets will be considered equal when they both contain the same elements, irrespective of their order. The order is important, however, in the case of ordered sets.

The toString() method is the same for both. And, finally, we need at least two constructors:

Default Ctors These make “empty” sets by creating a new ArrayList<T> for the class.

Copy Ctors These take as parameters sets of the same type as the class and create shallow copies of their parameters.

3 Helping yourself

The Internet contains many references to sets. Some of these might be helpful to you—especially if you are a little unsure about the details of union, intersection, and set-differences. I doubt that you will find appropriate code examples, because this specification is not standard. But, the methods that you need to write a pretty short if you understand what needs to be done.

3.1 Grading, testing, etc

I have made all of the tests Public so that you don’t have to worry about tokens. I also suggest that you make your own StudentTests. I have provided a placeholder for those.

Finally, follow the documentation in the source-code listing as it provides local hints, etc., to guide you in your work. Whenever possible, I use methods such as adjoin(), isMember() or isSubset() to implement my versions of union, intersect, and difference, instead of trying to retrieve the underlying ArrayList and operate on it using get(), contains, etc., methods because this approach is often tedious and error-prone. Worse, if you do not use adjoin() then you will not be able to reuse the union() method, and maybe not the intersect or the difference methods either—depending.

3.2 Prohibited constructions

Use only the get(), add() and add( int, object), and toString() from the ArrayList class.