CS 0445 Spring 2026
Recitation Exercise 3

Introduction:

We have discussed the BagInterface<T> in lecture and recently (Lectures 5 and 6) looked at the ArrayBag<T> and ResizableArrayBag<T> implementations using an array. We'd like to add some functionality to the BagInterface<T> and its implementation in the ResizableArrayBag<T> class. This exercise will help you to better understand the implementation details of the array-based bag and to design your own implementations of methods not covered in lecture.

Details:

Exercises 1.5, 1.6 and 1.7 in the textbook consider 3 methods not in the BagInterface<T> but that could be added. Specifically, consider the following three methods:

```
// Creates a new bag that combines the contents of this bag
// and anotherBag.
//    @param anotherBag  The bag that is to be added.
//    @return  A combined bag.
public BagInterface<T> union(BagInterface<T> anotherBag);

// Creates a new bag that contains those objects that occur
// in both this bag and anotherBag.
//    @param anotherBag  The bag that is to be compared.
//    @return  A combined bag.
public BagInterface<T> intersection(BagInterface<T> anotherBag);

// Creates a new bag of objects that would be left in this bag
// after removing those that also occur in anotherBag.
//    @param anotherBag  The bag that is to be removed.
//    @return  A combined bag.
public BagInterface<T> difference(BagInterface<T> anotherBag);
```

These operations are usually associated with sets, where duplicates of items are not allowed. Since we are here considering bags rather than sets, duplicates are allowed and therefore the effect of these methods is also somewhat different from what they would be in a set. For example, given two bags with bag1 = { A, B, B, C } and bag2 = { B, C, D } the union bag1.union(bag2) would be { A, B, B, B, C, C, D }. Note that the order of the elements in the resulting bag is arbitrary (i.e. does not matter).

Also, the intersection of two bags will be the items that are contained in both bags, including duplicates. Thus, an item may occur more than one time within an intersection. For example, given the two bags above, bag1.intersection(bag2) would be { B, C }. However, now consider bag3 = { B, C, B}. Now, bag1.intersection(bag3) would be { B, B, C }.

Similarly, the difference of two bags must take duplicates into account. In the examples above, bag1.difference(bag2) = { A, B }, bag2.difference(bag1) = { D }, bag1.difference(bag3) = { A } and bag3.difference(bag1) = { }. As you can see, the difference() method is not symmetric (i.e. for two bags b1 and b2, b1.difference(b2) is not necessarily equal to b2.difference(b1)).

Your task is to implement the three new methods in the ResizableArrayBag<T> class so that they work as described. The interface (with the new methods) is available in the Canvas directory with this exercise [note: be sure to use the version of BagInterface<T> that contains the new methods]. The ResizableArrayBag<T> class (without the new methods) is also available in file ResizableArrayBag.java. Be sure to add the new methods to the class in order to get this to work. See Files/Recitation/Recitation3/.

Test your implementation with the main program CS445Rec3.java. The output contents should match that shown in Rec3Out.txt. Note however, that within any bag the order of the data does not matter – so if your bags show the contents in a different order that does not necessarily mean that they are incorrect.

Hints:

You will need to iterate through the contents of one or perhaps both of your ResizableArrayBag<T> objects to implement these methods. This can be done by accessing the underlying arrays in one or more loops.

The argument bag for each of these methods will be passed in via a BagInterface<T> parameter. This parameter type will restrict access to the methods in BagInterface<T> and will not allow direct access of the underlying data in the argument bag. To get this access (if you need it – which you probably will) you will need to cast the parameter to type ResizableArrayBag<T>. For example, if the parameter is

```
        BagInterface<T> anotherBag
```

you could cast it as follows

```
        ResizableArrayBag<T> argBag = (ResizableArrayBag<T>) anotherBag;
```

The above statement will work as long as the actual bag being passed into these methods is a ResizableArrayBag<T>. If the object were a different bag implementation then the cast above would result in a ClassCastException. If you wanted to avoid casting the argument another option would be to use the toArray() method to return the contents in an array. You could then access the array in your methods.

Be careful not to change the contents of the bags involved in these operations. The only effect of these methods should be to create and return a new bag – the bag from which the method is called and the argument bag should NOT be changed in any way. If you would find it easier to implement these methods IF you could change one or more of these bags, just make a copy of it before you do so. There is no copy constructor for the ResizableArrayBag<T> class but it is not difficult to make a copy directly.