1P03 Assignment 5

(Due Date April 9th 4:00 p.m.)

Objective

To practice creating an ADT using constrained parametric types.

Background

In mathematics, the notion of **multiset** (or **bag**) is a generalization of the notion of set in which members are allowed to appear more than once. For example, there is a unique set that contains the elements *a* and *b* and no others, but there are many multisets with this property, such as the multiset that contains two copies of *a* and one of *b* or the multiset that contains three copies of both *a* and *b*. The term "multiset" was coined by Nicolaas Govert de Bruijn in the 1970s.[1] The use of multisets in mathematics and beyond predates the name "multiset" by many centuries: Knuth (1998) attributes the first study of multisets to the Indian mathematician Bhascara Acharya (circa 1150), who described permutations of multisets. {http://en.wikipedia.org/wiki/Multiset}

The Assignment

You are given the following interface which describes a Multiset interface, and typical operations on a multiset. This is not a complete list, but will suffice for the purpose of this assignment. Since a Bag (duplicates allowed, also called a multiset) and a Set (no duplicates) are cousins, we can define one interface which represents the implementations of both a Bag and a Set.

```
package MULTISET;
public interface MultiSet<E extends Keyed> extends Iterable<E> {

      // returns the number of elements in the collection
      public int cardinality();

      // returns the number of elements that match item from this
      public int multiplicity(E item);

      // Adds anItem to the collection; note this is a mutable operation
      public void add(E anItem);

      // returns true if this is empty
      public Boolean isEmpty();

      // returns a new MultiSet by taking the union of this and aSet,
      // the operation is immutable, neither this or aSet is modified
      public MultiSet<E> union(MultiSet<E> aSet);

      // returns true if this contains the same elements as aSet
      public Boolean equal(MultiSet<E> aSet);
```

```
	// returns a new MultiSet by taking the intersection of this and aSet,
	// the operation is immutable, neither this or aSet is modified
	// the operation can be views as min(|S|,|T|)
	public MultiSet<E> intersection (MultiSet<E> aSet);

	// Returns an interator over the collection this.
	public Iterator<E> iterator();
}
	// MultiSet
```

Using the interface, define two implementations: The first is to implement MyBag which creates a collection where duplicates are allowed. This implementation should use contiguous memory, arrays to store the collection. Since the add operator is mutable, this implied the array is sufficiently large to accommodate a large collections which can be added to.

The second implementation is to implement MySet, which creates a collection using a Linked Lists to hold the collection. All elements in the collection are unique (only 1 occurrence) corresponding to the definition of a Set.

The operation of union (S union T) is defined for a bag as simply all the elements in S and all elements in T. For a set, the union consists of at most 1 occurrence of each element in S and T.

Intersection: In a normal set (no duplicates), where we have S intersect T, the resultant set would include 1 occurrence those members which appear in both S and T. For a bag, this is defined as the minimum of the member elements which are in common between S and T. E.g. {a,a,a,b,b,c,d} and {a, a, b, b, c, c}, the intersection would be {a, a, b, b, c}.

It is envisioned that when a set or bag is created, it is empty and the add method is used to populate the collection.

Since operations such as union and intersect require a new object to be created, it makes sense that there are constructors present which accept arrays or nodes for the express purpose of creating a new object. Consider the below constructor:

```
	private MyBag(E[] A){
```

If the union is created between this and aSet, it is expected that the elements will be stored in a temporary array of type E[]. The above constructor would accept the array to create a new object. E.g.

```
	@SuppressWarnings("unchecked")
	public MultiSet<E> union(MultiSet<E> aSet){
		E[]  temp;
		temp = (E[]) new Keyed[size of the array];
		some code that you will write.
		return new MyBag<E>(temp);
```

```
        }
```
Some important points to note: firstly, the constructor is listed as private. Thus only accessible from within the class. When coding MySet, you will likely need a similar one which accepts Node<E>.

`@SuppressWarnings("unchecked")` is used to pacify the compiler which would complain about temp = (E[]) new Keyed[...], since the compiler would view the cast as illegal. These are traditionally called pragmas or compiler directives. Any method in your solution which uses a cast as exemplified would need a pragma.

It is easy to access the elements of this, since the data structure, be it array or linked list is directly accessible. However, how do you get to the elements of the parameter aSet? This is where you will find the iterator useful.

In addition, implement toString to aid you in debugging. It will have no use other then debugging.

When dealing with MultiSets you will need to supply restricted types to those sets. Much like lists, in fact any element you put into the set should be Keyed so that the getKey method is accessible. This will be needed when doing comparisons for the various operations. The type of element you put in is up to you. You can use a Keyed char or Keyed int. You will need to create one.

The iterator will require you to write an Exception.

How to Start

1. The assignment can generate up to 10 classes.
2. Start with implementing MyBag. It is very similar in structure to the ConList example.
3. Start by implementing the methods add, cardinality and iterator, thus implying you need to create the iterator class. This will be very similar (try 95% the same) as the ConListIterator discussed in class.
4. Write a small test harness which creates a new MyBag, adds some Keyed elements to the Bag, and then using the iterator, printing them to System.out. If you can't get Iterator to work consider the toString method.
5. Implement the other methods 1 at a time, testing as you go.
6. Write the class MySet and implement it in small stages. This class is to use linked lists, thus a node<e> class.
7. Start early, please don't start the day before it is due.

Testing

Write a comprehensive test harness to ensure that all constructors and methods do what they are supposed to do. It is your responsibility to ensure completeness. Since MyBag and MySet are type compatible, it does make some sense that you can take the union and intersection using mixed types. However, for the purpose of this assignment we will ignore this fact and not require implementation nor testing in this realm.

Submission

Refer to the course website for details on submission.
Also include sample output from your test harness.