```java
package MULTISET;

/**
 * This class is a bag implementation of MultiSet using a contiguous array. Duplicates are allowed in
 * All items will be inserted into their appropriate place when added.
 *
 * @author Matt Laidman
 * @version 1.0 (March 2014)
 */

public class MyBag<E extends Keyed> implements MultiSet<E> {

    private E[] conSet; // array to store items
    private int count; // count of items in bag


    public MyBag ( ) { // default constructor creates MyBag size 100
        this(100);
    }


    @SuppressWarnings("unchecked")
    public MyBag(int capacity) { // creates MyBag of specified size
        conSet = (E[]) new Keyed[capacity];
        count = 0;
    }


    @SuppressWarnings("unchecked")
    public MyBag(E[] A) { // creases new MyBag from Keyed array
        conSet = (E[]) new Keyed[Math.max(2*A.length, 100)];
        count = 0;
        for (E a : A) {
            this.add(a);
        }
    }


    public int cardinality() { // returns number of items in 'this'
        return count;
    }


    public int multiplicity(E item) { // returns number of times item exists in 'this'
        int mult = 0;
        for (int i = 0 ; i < count ; i++) { // for each item in 'this'
            if (item.getKey().compareTo(conSet[i].getKey()) == 0) { // compare to item
                mult++;
            }
        }
        return mult;
    }


    public void add(E anItem) { // adds an item to 'this' in order, duplicates are allowed
        if (count >= conSet.length) { // check if there is enough space in array
            throw new NoSpaceException("Not enough space to add item.");
        }
        if (count == 0) { // if first item
            conSet[0] = anItem;
            count++;
        } else {
            for (int i = count - 1; i >= 0; i--) { // else loop through array
                if (anItem.getKey().compareTo(conSet[i].getKey()) >= 0) { // if insertion location
                    conSet[i + 1] = anItem;
                    count++;
                    break;
                } else { // move items over
                    conSet[i + 1] = conSet[i];
                }
            }
        }
    }


    public Boolean isEmpty() { // returns true if 'this' empty
        return count == 0;
```

```
78      }
79
80
81      @SuppressWarnings("unchecked")
82      public MultiSet<E> union(MultiSet<E> aSet) { // returns a new MyBag containing all items in '
    this' and aSet
83          MultiSet newSet = new MyBag();
84          for (int i = 0 ; i < count ; i++) { // add all items from 'this'
85              newSet.add(conSet[i]);
86          }
87          Iterator bagIterator = aSet.iterator(); // add all items from aSet
88          while (bagIterator.hasNext()) {
89              Keyed value = bagIterator.next();
90              newSet.add(value);
91          }
92          return newSet;
93      }
94
95
96      public Boolean equal(MultiSet<E> aSet) { // returns true if 'this' == aSet
97          int i = 0;
98          Iterator bagIterator = aSet.iterator(); // for each item in aSet
99          while (bagIterator.hasNext()) {
100             Keyed value = bagIterator.next();
101             if (!value.getKey().equals(conSet[i].getKey())) { // if item != 'this' item of same key
102                 return false;
103             }
104             i++;
105         }
106         return conSet[i] == null; // true if none left in either and all are equal
107     }
108
109
110     @SuppressWarnings("unchecked")
111     public MultiSet<E> intersection(MultiSet<E> aSet) { // returns new set containing minimum
    number of items in both 'this' and aSet
112         MultiSet newSet = new MyBag();
113         int c, j;
114         for (int i = 0 ; i < count ; i++) { // for each different item in 'this'
115             c = 1;
116             j = 1;
117             while (i < count-1 && conSet[i].getKey().equals(conSet[i+j].getKey())) { // get number
    of times item appears
118                 c++;
119                 j++;
120             }
121             for (int k = 0 ; k < Math.min(c, aSet.multiplicity(conSet[i])) ; k++) { // add items min(
    c, multiplicity of item in aSet)
122                 newSet.add(conSet[i]);
123             }
124             i = i + j - 1;
125         }
126         return newSet;
127     }
128
129
130     public Iterator<E> iterator ( ) { // returns an iterator for 'this'
131         return new Iterator<E>(conSet, count);
132     }
133
134     @Override
135     public String toString ( ) { // returns string representation of 'this'
136         String temp = "";
137         for (E a : conSet) {
138             temp = temp + a.getKey();
139         }
140         return temp;
141     }
142 }
```