```java
package MULTISET;

/**
 * This class is a set implementation of MultiSet using a linked node list. No duplicate items are allowed in a MySet.
 * All items will be inserted into their appropriate place when added.
 *
 * @author Matt Laidman
 * @version 1.0 (March 2014)
 */

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

    private Node<E> linkSet; // Linked List to store set
    private int count; // count of items in set

    public MySet() { // default constructor, create list count 0
        count = 0;
    }


    @SuppressWarnings("unchecked")
    public MySet(E[] A) { // create list from array
        count = 0;
        for (E a : A) {
            this.add(a);
        }
    }


    public int cardinality() { // return num items in list
        return count;
    }


    public int multiplicity(E item) { // returns the number times an items occurs in a set, should always be 1 for a MySet
        Node<E> p = linkSet;
        while (p != null && !p.item.getKey().equals(item.getKey())) { // find first
            p = p.next;
        }
        if (p == null) { // return 0 if didn't find
            return 0;
        } else {
            return 1;
        }
    }


    public void add(E anItem) { // adds an items to the set in order, no duplicates
        Node<E> q = null;
        Node<E> p = linkSet;
        while (p != null) { // checks if items already exists
            if (p.item.getKey().equals(anItem.getKey())) {
                return;
            }
            p = p.next;
        }
        p = linkSet;
        while (p != null && anItem.getKey().compareTo(p.item.getKey()) >= 0) { // get insertion location of item
            q = p;
            p = p.next;
        }
        if (q == null) { // if first item
            linkSet = new Node<E>(null, anItem, p);
            count++;
            if (p != null) {
                p.prev = linkSet;
            }
        } else { // if not first item
            q.next = new Node<E>(q, anItem, p);
            count++;
            if (p != null) {
                p.prev = q.next;
            }
        }
```

```
    }


    public Boolean isEmpty() { // returns true if the set contains no items
        return count == 0;
    }

    @SuppressWarnings("unchecked")
    public MultiSet<E> union(MultiSet<E> aSet) { // returns a new set containing all items present in 'this' and aSet
        MultiSet newSet = new MySet();
        Node<E> p = linkSet;
        while (p != null) { // add all items from 'this'
            newSet.add(p.item);
            p = p.next;
        }
        Iterator setIterator = aSet.iterator(); // add all items from aSet
        while (setIterator.hasNext()) {
            Keyed value = setIterator.next();
            newSet.add(value);
        }
        return newSet;
    }

    @SuppressWarnings("unchecked")
    public Boolean equal(MultiSet<E> aSet) { // returns true if 'this' == aSet
        Node<E> p = linkSet;
        Iterator setIterator = aSet.iterator();
        while (p != null && setIterator.hasNext()) {
            if (!p.item.getKey().equals(setIterator.next().getKey())) { // if items are not equal
                return false;
            }
            p = p.next;
        }
        return true;
    }

    @SuppressWarnings("unchecked")
    public MultiSet<E> intersection(MultiSet<E> aSet) { // returns a new set containing all items in both 'this' and aSet
        MultiSet newSet = new MySet();
        Node<E> p = linkSet;
        while (p != null) { // for each items in 'this'
            Iterator setIterator = aSet.iterator();
            while (setIterator.hasNext()) {
                Keyed value = setIterator.next();
                if (p.item.getKey().equals(value.getKey())) { // compare to each item in aSet, add () if they're the same
                    newSet.add(value);
                }
            }
            p = p.next;
        }
        return newSet;
    }


    public Iterator<E> iterator() { // return an iterator for 'this'
        return new Iterator<E>(linkSet, count);
    }

    @Override
    public String toString() { // return a string representation of 'this'
        String temp = "";
        Node<E> p = linkSet;
        while (p != null) {
            temp = temp + p.item.getKey();
            p = p.next;
        }
        return temp;
    }
}
```