```
1 /** This class MyBag that implements the the interface from MulitSet
2  * this class implement the cardinality, multiplicity, add, isEmpty
3  * union and intersection of MyBag
4  *
5  *    Name: Long Nguyen: Student # 5427059
6  *
7  * @version  1.0 (April. 2014)                                    */
8
9
10 package MULTISET;
11
12 import java.io.Serializable;
13 import java.util.Arrays;
14 import java.util.Iterator;
15
16 import TestLists.*;
17
18 public class MyBag < E extends Keyed > implements MultiSet<E>, Serializable
   {
19
20    private E[]  items;   // the items in the Bag
21    private int  length;  // the length of the Bag
22    private int  cursor;  // the list cursor
23
24    // The default constructor
25    public MyBag ( ) {
26
27       //creates an object with a positive zero
28          this(0);
29
30       }; // constructor
31
32       public MyBag ( int size ) {
33
```

```
34          setItems((E[]) new Keyed[size]);
35          setLength(0);
36
37       }; // Constructor
38
39    //returns the number of elements in the collection
40    public int cardinality() {
41       return getLength();
42    }
43
44    //returns the number of elements that match item from this
45    public int multiplicity (E item) {
46
47       int numberOfItems = 0;
48
49       //Setting the cursor to the found
50       this.toFront();
51
52       //looping through to check if there are duplicates
53        while ( cursor < getLength() ) {
54
55           if((item.getKey().toString().compareTo(getItems()[cursor].getKey())) ==
   0){
56              numberOfItems++;
57           }
58             cursor = cursor + 1;
59           };
60
61        //System.out.println("Number of items  found " + numberOfItems);
62       return numberOfItems;
63    }
64
65    //Adds an Item to the collection; note this is a mutable operation
66    public void add(E anItem) {
```

```
67
68      int  j;
69
70       if ( getLength() >= getItems().length ) {
71         throw new NoSpaceException();
72       }
73       else {
74          for ( j = getLength()-1 ; j>=cursor ; j-- ) {
75             getItems()[j+1] = getItems()[j];
76          };
77          getItems()[cursor] = anItem;
78          setLength(getLength() + 1);
79       };
80
81  }
82
83  //returns true if this is empty
84  public Boolean isEmpty() {
85     return getLength() == 0;
86  }
87
88  //returns a new MultiSet by taking the union of this and aSet,
89  //the operation is immutable, neither this or aSet is modified
90  @SuppressWarnings("unchecked")
91  public MultiSet<E> union(MultiSet<E> aSet) {
92
93
94     char[] unsortedSet = null; // unsortedSet
95     String valuesOfString = "";  // initialize the string
96     E[] temp;
97     temp = (E[]) new Keyed[100]; //initialize the E array to 100
98
99     MultiSet <E> unionSet = new MyBag <E>(100); //creating a new MyBag
100    char b;
```

```
101     E values;
102     Object  element;
103
104     //iteratoring through the list to add in the string
105      Iterator itr = aSet.iterator();
106      while(itr.hasNext()) {
107          element = ((Keyed) itr.next()).getKey();
108          valuesOfString += element.toString();
109         }//end while
110
111      //looping through the "This" to get the values and adding it to the string
112     while ( cursor < getLength()) {
113        element = this.getItems()[cursor].getKey();
114       // System.out.println(element);
115        valuesOfString += element.toString();
116        b = (char) element.toString().charAt(0);
117        values = (E) new KeyedChar(b);
118           cursor = cursor + 1;
119       }//end while
120
121     //making the string into a char array and sorting it
122     unsortedSet = valuesOfString.toCharArray();
123      Arrays.sort(unsortedSet);
124
125      //adding the values to unionSet backward
126      for(int i =unsortedSet.length-1; i >= 0; i--){
127          b = (char) unsortedSet[i];
128          values = (E) new KeyedChar(b);
129          unionSet.add(values);
130
131        }
132
133     return unionSet; //returning the union of the collection
134  }
```

```
135
136   //returns true if this contains the same elements as aSet
137   public Boolean equal(MultiSet<E> aSet) {
138
139      //initialize the equals to false
140      boolean equals = false;
141
142      //pointing the cursor to the front
143         this.toFront();
144
145         Iterator itr = aSet.iterator();
146
147            //looping through to check if there are duplicates
148            while ( cursor < getLength() && itr.hasNext() ) {
149
150               //Iterator through the aSet to get the key
151               Object  element = ((Keyed) itr.next()).getKey();
152
153               //if comparing each values one at a time to check if they are equal
154               if(this.getItems()
    [cursor].getKey().toString().compareTo(element.toString()) == 0){
155                  //System.out.println(this.items[cursor].getKey());
156                  //System.out.println(element.toString());
157                  cursor = cursor + 1;
158                  equals = true;
159                  //else set it to false if they are not equal
160                 }else{
161                    equals = false;
162                    break;
163                 }//end else
164
165              }//end while loop
166
167            //return the value if it's true or false
```

```
168     return equals;
169   }
170
171   /*This method gets the intersection between the two sets */
172   @SuppressWarnings("unchecked")
173   @Override
174   public MultiSet<E> intersection(MultiSet<E> aSet) {
175
176     char[] unsortedSetA = null; // unsortedSetA char array
177     char[] unsortedSetB = null; // unsortedSetB char array
178
179     String valuesOfStringSetA = "";  // initialize the string
180     String valuesOfStringSetB = "";  // initialize the string
181
182     E[] temp;
183     temp = (E[]) new Keyed[100];
184
185     MultiSet <E> intersectionSet = new MyBag <E>(100);  //creating a new
    MyBag
186     char b;
187     E values;
188     Object  element;
189
190     //iteratoring through the aSet and adding it to a string
191      Iterator itr = aSet.iterator();
192      while(itr.hasNext()) {
193          element = ((Keyed) itr.next()).getKey();
194          valuesOfStringSetB += element.toString();
195        }
196
197     //looping through the values of "This " and getting the values and adding
    it to string
198      while ( cursor < getLength()) {
199        element = this.getItems()[cursor].getKey();
```

```
200          valuesOfStringSetA += element.toString();
201             cursor = cursor + 1;
202          }
203
204       /*Sorting the Array of both sets*/
205       unsortedSetA = valuesOfStringSetA.toCharArray();
206       unsortedSetB = valuesOfStringSetB.toCharArray();
207        Arrays.sort(unsortedSetA);
208        Arrays.sort(unsortedSetB);
209
210       //Checking the Array and check for duplicate values for the intersection
211       for(int i =unsortedSetA.length-1; i>=0; i--){
212          for(int j = unsortedSetA.length-1; j >=0; j--){
213             if(unsortedSetA[i] == unsortedSetB[j]){
214                   b = (char) unsortedSetA[i];
215                   values = (E) new KeyedChar(b);
216                   //adding the value to the intersectionSet
217                   intersectionSet.add(values);
218
219             }//end if
220          }//end for
221          }
222       //return the intersectionSet
223       return intersectionSet;
224
225
226    }; // advance
227
228    @Override
229    public Iterator<E> iterator() {
230       // TODO Auto-generated method stub
231       return new ListIterator<E>(this);
232       //return null;
233    }
```

```
234
235    //pointing the cursor to the front
236 public void toFront ( ) {
237        cursor = 0;
238    }; // toFront
239
240    //checking if the cursor is bigger then lenght
241    public E get ( ) {
242        if ( cursor >= getLength() ) {
243          throw new NoItemException();
244        }
245        else {
246           return getItems()[cursor];
247        }
248        //return null;
249
250    }; // get
251
252    public void advance( ) {
253
254        if ( cursor < getLength() ) {
255           cursor = cursor + 1;
256        }
257    }//end if statement
258
259    public boolean offEnd ( ) {
260        return cursor >= getLength();
261
262    }
263
264    public int getLength() {
265        return length;
266    }
267
```

```
268    public void setLength(int length) {
269        this.length = length;
270    }
271
272    public E[] getItems() {
273        return items;
274    }
275
276    public void setItems(E[] items) {
277        this.items = items;
278    }; // offEnd
279
280 }
281
```