```
1
2   /*Alan Stoloff
3    * Dr Benjamin
4    * CSI220
5    * 21 September 20202
6    *   An implementation of the Map interface.
7    *
8    *   A doubly linked list implementation is used.
9    *   Front and rear sentinel nodes are used.
10   *
11   *   Author; Alan Stoloff
12   *   Date;15 September 2020
13   */
14
15  public class Map <K, V> implements MapInterface<K, V>
16  {
17      private Node header;   // Pointer to the front sentinel in the Map.
18
19
20      // The Constructor sets up the sentinel nodes with
21      // header pointing to the front.
22
23      public Map()
24      {
25          Node front = new Node();   // Construct the sentinel
26          Node rear = new Node();    // nodes.
27
28          front.prev = null;   // Link the sentinels.  All map
29          front.next = rear;   // items will be between the sentinels
30          rear.prev  = front;  // so each items is assured to have
31          rear.next  = null;   // a node in front and in back.
32
33          header = front;  // The header points to the front node.
34      }
35
36      /*
37       *  int getSize() − return the number of items in the Map
38       */
39
40      public int getSize()
41      {
42          //initializes counter
43          int count=0;
44          //initializes pointer
45          Node ptr=header.next;
46          //while the pointer is not at the end move the pointer and up count
47          while(ptr.next!=null)
48          {
49              count+=1;
50              ptr=ptr.next;
51          }
52
53          return count;   // Return the number of items in the Map.
54                          //0 if only the front and end sentinel
55      }
56
57      /*
58       *   void makeEmpty() − remove all the items from the Map
59       */
60
61      public void makeEmpty()
62      {
63          //initializes pointer
64          Node ptr=header.next;
65          while (ptr.next != null)  // Find the rear sentinel
66              ptr = ptr.next;
67          //set pointer to link front and rear sentinel
68          header.next=ptr;
69          ptr.prev=header;
```

```
70      }                       // All items between them are removed.
71
72
73      /*
74       *    void insert() − Insert a new (key,value), if the key is
75       *                    already in the Map just update the value.
76       */
77
78      public void insert(K key, V value)
79      {
80          //initialize pointer
81          Node ptr=header.next;
82          //goes through list checking if the key is there and updates its value
83          while(ptr.next!=null){
84              if(ptr.key.equals(key)){
85                  ptr.value=value;
86                  return;
87              }
88              else{
89                  ptr=ptr.next;
90              }
91          }
92          //if the key did not exist a new node is created and inserted in front
93          Node newNode=new Node();
94          newNode.key=key;
95          newNode.value=value;
96          header.next.prev=newNode;
97          newNode.next=header.next;
98          header.next=newNode;
99          newNode.prev=header;
100     }
101
102
103     /*
104      *   void remove() − remove the Map item with the given key.
105      *                   If the key is not in the Map − do nothing.
106      */
107
108     public void remove(K key)
109     {
110         //pointer is made
111         Node current=header.next;
112         //loop goes through list to find key to be removed
113         while(current.next!=null){
114             //if the key is a match the node is removed and the list in reconected
115             if(current.key.equals(key)){
116                 current.next.prev=current.prev;
117                 current.prev.next=current.next;
118             }
119             current=current.next;
120         }
121         return;
122     }
123
124     /*
125      *   V getValue() − return the value of the Map item given
126      *
127      *its key.  If the key is not in the Map
128      *                   return null.
129      */
130
131     public V getValue(K key)
132     {
133         //a pointer is made
134         Node current=header.next;
135         //loop traverses the list to find the key
136         while(current.next!=null){
137             //if the key matches the value is returned
138             if(current.key.equals(key)){
```

```
                return current.value;

            }
            current=current.next;
        }
        //if the list is done and the key is not found return null
        return null;
    }

    /*
     *   boolean isEmpty() − return whether or not the Map
     *                       is empty.
     */

    public boolean isEmpty()
    {
        //pointer is made
        Node ptr=header.next;
        while (ptr.next != null){  // Find the rear sentinel
            ptr = ptr.next;
        }
        //if the front points to the rear it is empty
        if(header.next.equals(ptr)){
            return true;
        }
        return false;
    }                                    // the doubly linked list.

    /*
     *   toString() − return a String representation
     *                of the map.
     */



    public String toString()
    {
        String str = "\nThe Map\n−−−−−−−−−−−−−−\n";

        Node ptr = header.next;

        while (ptr.next != null) {             // Create a String consisting
            str = str + "key: ";               // of all the (key, value)
            str = str + ptr.key.toString();    // pairs − return this String
            str = str + "   ";                 // as the value of the function.
            str = str + "value: ";
            str = str + ptr.value.toString();
            str = str + "\n";
            ptr = ptr.next;
        }

        str = str + "−−−−−−−−−−−−−−\n";

        return str;
    }



    /*
     *   toStringBkw() − return a String representation
     *                   of the map using blinks.
     */

    public String toStringBkw()
    {
        String str = "\nThe Map − displayed backwards\n−−−−−−−−−−−−−−\n";

        Node ptr = header.next;

```

```
208         while (ptr.next != null)  // Find the rear sentinel
209             ptr = ptr.next;
210         ptr = ptr.prev;     // Point to node before rear sentinel
211 
212         while (ptr.prev != null) {             // Create a String consisting
213             str = str + "key: ";               // of all the (key, value)
214             str = str + ptr.key.toString();    // pairs − return this String
215             str = str + "   ";                 // as the value of the function.
216             str = str + "value: ";
217             str = str + ptr.value.toString();
218             str = str + "\n";
219             ptr = ptr.prev;
220         }
221 
222         str = str + "−−−−−−−−−−−−−\n";
223 
224         return str;
225     }
226     /*
227      *    Inner Class − Node objects for Map items
228      *                  in a doubly linked list.
229      *
230      */
231 
232     private class Node{
233         public K key;
234         public V value;
235         public Node prev;
236         public Node next;
237     }
238 }
```