```java
package Project4;

import java.io.Serializable;

/**********************************************************************
 * This program creates a single Linked List. It has functions that allow
 * the user to add, remove, or get data from the linked list.
 *
 * @author Justin Von Kulajta Winn and Nick Layman
 *@version 1.9
 *********************************************************************/

public class MySingleLinkedList implements Serializable
{
    /** this is Node that represents the top of the linked list */
    private Node top;

    /** this is Node that represents the bottom of the linked list */
    private Node tail;

    /** this is the integer that holds the size of the linked list */
    public int size;

    /**********************************************************************
     * This constructor make an empty list and sets the size to 0
     *********************************************************************/
    public MySingleLinkedList() {
        top = null;
        size = 0;
    }

    /**********************************************************************
     * This function returns the size of the linked list
     * @return size is the size of the linked list
     *********************************************************************/
    public int size() {
        return size;
    }

    /**********************************************************************
     * This function clears the current linked list and sets the top to empty
     * and resets the size of the list
     *********************************************************************/
    public void clear () {
        top = null;
        size = 0;
    }

    /**********************************************************************
     * This function adds an auto to the linked list. It will sort each
     * addition first by whether it is a car or truck, then it checks each
     * date. THe list goes from earliest bought cars to most recently bought
     * trucks
     * @param s is the auto being passed. It is either a car or truck
     *********************************************************************/
    public void add(Auto s) {
```

```
57          //Case 0: No List exists therefore it is simply added to the top of the list
58          if (top == null){
59              top = tail = new Node(s, null);
60              size++;
61              return;
62          }
63
64          //If the item is a car, it will enter this 'if' statement
65          if (s.getClass().getName().equalsIgnoreCase("Project4.Car")) {
66              //Case 1 For Cars: No Previous Car Exists
67              if (top.getData().getClass().getName().equals("Project4.Truck")) {
68                  top = new Node(s, top);
69                  size++;
70                  return;
71              }
72
73              //Case 2 For Cars: Passed Auto has earliest date so gets first spot
74              if (top.getData().getBoughtOn().compareTo(s.getBoughtOn()) > 0){
75                  top = new Node(s, top);
76                  size++;
77                  return;
78              }
79
80              //Case 3 For Cars: Cars exist and next Node is not a truck
81              Node temp = top;
82              if (temp.getNext() == null){
83                  Node Insert;
84                  Insert = new Node(s, null);
85                  temp.setNext(Insert);
86                  tail = Insert;
87                  size++;
88                  return;
89              }
90
91              while (temp.getNext() != null &&
92                      temp.getNext().getData().getClass().getName()
93                              .equals("Project4.Car")) {
94                  //Case 4: The passed car's date is in the middle of the list
95                  if (temp.getNext().getData().getBoughtOn()
96                          .compareTo(s.getBoughtOn()) > 0) {
97                      Node InsertCar;
98                      InsertCar = new Node(s, temp.getNext());
99                      temp.setNext(InsertCar);
100                     size++;
101                     return;
102                 }
103                 temp = temp.getNext();
104             }
105             Node Insert;
106             //Case 5: Reaches end of list and no trucks exist
107             if (temp.getNext() == null){
108                 Insert = new Node(s, null);
109                 temp.setNext(Insert);
110                 tail = Insert;
111                 size++;
112                 return;
```

```
113             }
114             //Case 6: Reaches end of list and trucks exist
115             else {
116                 Insert = new Node(s, temp.getNext());
117                 temp.setNext(Insert);
118                 size++;
119                 return;
120             }
121 
122 
123         }
124 
125         //If this is reached, the auto 's' is a truck
126         else {
127             Node temp2 = tail;
128 
129             //Case 7: Auto is a Truck and no other trucks exist
130             if (temp2.getData().getClass().getName().equals("Project4.Car")){
131                 Node End = new Node(s, null);
132                 temp2.setNext(End);
133                 tail = End;
134                 size++;
135                 return;
136             }
137 
138             //Case 8: Only 1 Truck exists in the list
139             // it compares the incoming truck and the current trucks
140             if (top.getNext() == null){
141                 if (top.getData().getBoughtOn().compareTo(s.getBoughtOn()) > 0)
142                 {
143                     top = new Node(s, top);
144                     size++;
145                     return;
146                 }
147                 else {
148                     Node Second = new Node(s, null);
149                     top.setNext(Second);
150                     size++;
151                     return;
152                 }
153             }
154 
155             Node temp = top;
156             //This loop gets us to the last car in the list
157             while (temp.getNext().getData().getClass().getName()
158                     .equals("Project4.Car")){
159                 temp = temp.getNext();
160             }
161 
162             //Case 9: If Incoming Truck was bought earlier than the first truck
163             if (temp.getData().getClass().getName().equals("Project4.Car")){
164                 if (temp.getNext().getData().getBoughtOn()
165                         .compareTo(s.getBoughtOn()) > 0){
166                     Node Check = new Node(s, temp.getNext());
167                     temp.setNext(Check);
168                     size++;
```

```
169                     return;
170                 }
171             } else{
172                 if (temp.getData().getBoughtOn().compareTo(s.getBoughtOn()) > 0){
173                     top = new Node(s, top);
174                     size++;
175                     return;
176                 }
177             }
178 
179 
180 
181             while (temp.getNext() != null){
182                 //Case 10: Incoming Truck should be in the middle of the list
183                 if (temp.getNext().getData().getBoughtOn()
184                         .compareTo(s.getBoughtOn()) > 0) {
185                     Node InsertTruck;
186                     InsertTruck = new Node(s, temp.getNext());
187                     temp.setNext(InsertTruck);
188                     size++;
189                     return;
190                 }
191                 temp = temp.getNext();
192             }
193 
194             //Case 11: Truck goes one the end of the list
195             Node End = new Node(s, null);
196             temp.setNext(End);
197             tail = End;
198             size++;
199             return;
200         }
201         //  Order is: (First) List all Cars in bought by date order
202         //  followed by (second) List all Trucks in bought by order.
203     }
204 
205     /**************************************************************************
206      * This function removes an auto from a linked list. It searches the
207      * list for an identical auto within the list. It then removes said auto
208      * @param auto is the auto being searched for in the list.
209      *************************************************************************/
210     public void remove(Auto auto){
211         //Case 0: There is no list
212         if (top == null)
213             return;
214 
215         //Case 1: remove the top
216         if (top.getData().equals(auto)) {
217             top = top.getNext();
218             size--;
219             return;
220         }
221 
222         //Case 3: remove from middle or end
223 
224         //find the auto before the one to remove
```

```
225         Node temp = top;
226         while(temp.getNext() != null && !temp.getNext().getData().equals(auto))
227             temp = temp.getNext();
228 
229         if (temp.getNext() == null)
230             return;
231         else
232             temp.setNext(temp.getNext().getNext());
233 
234         size--;
235     }
236 
237     /**************************************************************************
238      * This function removes an auto from the list based on the index.
239      * It will return the auto at the given index if the index is within
240      * range of the size of the list
241      * @param index is the location of the desired auto that is to be
242      *              removed from the list
243      * @throws IllegalArgumentException if given index is out of bounds
244      * @return the auto being removed from the list
245      *************************************************************************/
246     public Auto remove(int index) {
247         if (index < 0 || index >= size) {
248             if (size == 0 && index == 0)
249                 return null;
250 
251             throw new IllegalArgumentException();
252         }
253 
254         //valid index and list exists
255 
256         Auto data;
257 
258         //remove the top
259         if (index == 0) {
260             data = top.getData();
261             top = top.getNext();
262             size--;
263             return data;
264         }
265 
266         //remove anything else
267         Node temp = top;
268         for(int i = 0; i < index - 1; i++){
269             temp = temp.getNext();
270         }
271         data = temp.getNext().getData();
272         temp.setNext(temp.getNext().getNext());
273         size--;
274         return data;
275     }
276 
277     /**************************************************************************
278      * This function gets the auto at the request index location. Unlike
279      * the remove function that is based off of an index, it does not remove
280      * the auto from the list
```

```
281      * @param index is the location of the desired auto that is to be
282      *              returned from the list
283      * @throws IllegalArgumentException if the index is not within the size
284      *                                  of the linked list
285      * @return the auto at the index
286      ***************************************************************************/
287     public Auto get(int index) {
288         if (index < 0 || index >= size) {
289             if (size == 0 && index == 0)
290                 return null;
291 
292             throw new IllegalArgumentException();
293         }
294 
295         //valid index and list exists
296 
297         Node temp = top;
298         for(int i = 0; i < index; i++){
299             temp = temp.getNext();
300         }
301         return temp.getData();
302     }
303 
304     /***************************************************************************
305      * This function was used to check the linked list. This is not used.
306      ***************************************************************************/
307     public String toString() {
308         return null;
309     }
310 }
311 
```