

10/26/15 01:55:30 /home/15504319/DSA120/DSAAssignment/connorLib/CartonSearcher.java

```

1  /*****
2  *   FILE: CartonSearcher.java
3  *   AUTHOR: Connor Beardsmore - 15504319
4  *   UNIT: DSA120 Assignment S2- 2015
5  *   PURPOSE: Sets up and utilizes a Search Environment with data structures
6  *             to allow for quick searching of items within the DC
7  *   LAST MOD: 25/10/15
8  *   REQUIRES: java.util.Iterator, connorLib
9  *****/
10 package connorLib;
11 import java.util.Iterator;
12
13 public class CartonSearcher
14 {
15     //CLASSFIELDS
16     private DSALinkedList dateList;
17     private BinarySearchTree<String, DSALinkedList> prodTree;
18     private BinarySearchTree<String, DSALinkedList> wholeTree;
19     //-----
20     //ALTERNATE Constructor
21     //IMPORT: dc (DistroCentre)
22
23     public CartonSearcher(DistroCentre dc)
24     {
25         buildSearchEnvironment(dc);
26     }
27     //-----
28     //buildSearchEnvironment
29     //IMPORT: dc (DistroCentre)
30     //PURPOSE: Set-ups Data structures required for searching DC
31
32     public void buildSearchEnvironment(DistroCentre dc)
33     {
34         setDateList(dc);
35         setProdTree(dc);
36         setWholeTree(dc);
37     }
38     //-----
39     //addSearchEnvironment
40     //IMPORT: item (Carton)
41     //PURPOSE: Adds new Carton to existing search data structures
42
43     public void addSearchEnvironment(Carton item)
44     {
45         updateDate(item);
46         updateProd(item);
47         updateWhole(item);
48     }
49     //-----
50     //removeSearchEnvironment
51     //IMPORT: item (Carton)
52     //PURPOSE: Removes a carton from existing search data structures
53
54     public void removeSearchEnvironment(Carton item)
55     {
56         removeDate(item);
57         removeProd(item);
58         removeWhole(item);
59     }
60     //-----
61     //setDateList
62     //IMPORT: dc (DistroCentre)
63     //PURPOSE: Adds all Cartons into the DC into a linked list, sorted by Date
64
65     private void setDateList(DistroCentre dc)
66     {
67         dateList = new DSALinkedList();
68
69         for (int ii = 1; ii < Carton.MAX_CON_NOTE + 1; ii++)
70         {
71             Carton item = dc.getCartonIndex(ii);
72             //Only Add to list if a Carton on that con note actually exists
73             if ( item != null )
74             {
75                 //If 0000-00-00, add to end of list
76                 if ( item.getWar().isInfinite() )
77                 {
78                     dateList.insertLast(item);
79                 }
80                 //Insert in sorted order
81                 else
82                 {
83                     dateList.insertSorted( item );
84                 }
85             }
86         }
87     }
88 }

```

```

85     }
86 }
87 }
88 //-----
89 //setProdTree
90 //IMPORT: dc (DistroCentre)
91 //PURPOSE: Creates a Binary Search Tree, where each Node is itself a
92 //         linked-list of items with the same product
93
94 private void setProdTree(DistroCentre dc)
95 {
96     prodTree = new BinarySearchTree<String, DSALinkedList>();
97     DSALinkedList newList = null;
98
99     for (int ii = 1; ii < Carton.MAX_CON_NOTE + 1; ii++)
100     {
101         Carton item = dc.getCartonIndex(ii);
102         //Only add if a carton actually exists in the DC
103         if ( item != null )
104         {
105             //Create a new Linked list and add item to it.
106             newList = new DSALinkedList();
107             newList.insertLast( item );
108
109             //Try to insert list into tree
110             //Will either succeed and return false, or fail and return true
111             int status = prodTree.insert( item.getProduct(), newList );
112             if ( status == -1 )
113             {
114                 //List with this Wholeuct already exists in the tree
115                 //So we find() to get the list, then add to the list
116                 newList = prodTree.find( item.getProduct() );
117                 newList.insertLast( item );
118             }
119         }
120     }
121 }
122 //-----
123 //setWholeTree
124 //IMPORT: dc (DistroCentre)
125 //PURPOSE: Creates a Binary Search Tree, where each Node is itself a
126 //         linked-list of items with the same wholesaler
127
128 private void setWholeTree(DistroCentre dc)
129 {
130     wholeTree = new BinarySearchTree<String, DSALinkedList>();
131     DSALinkedList newList = null;
132
133     for (int ii = 1; ii < Carton.MAX_CON_NOTE + 1; ii++)
134     {
135         Carton item = dc.getCartonIndex(ii);
136         //Only add if a carton actually exists in the DC
137         if ( item != null )
138         {
139             //Create a new Linked list and add item to it.
140             newList = new DSALinkedList();
141             newList.insertLast( item );
142
143             //Try to insert list into tree
144             //Will either succeed and return false, or fail and return true
145             int status = wholeTree.insert( item.getWhole(), newList );
146             if ( status == -1 )
147             {
148                 //List with this product already exists in the tree
149                 //So we find() to get the list, then add to the list
150                 newList = wholeTree.find( item.getWhole() );
151                 newList.insertLast( item );
152             }
153         }
154     }
155 }
156 //-----
157 //updateDate
158 //IMPORT: item (Carton)
159 //PURPOSE: Adds new item to sorted Date list
160
161 private void updateDate(Carton item)
162 {
163     if ( item.getWar().isInfinite() )
164     {
165         dateList.insertLast(item);
166     }
167     else
168     {
169         dateList.insertSorted( item );
170     }
171 }

```

```

172     }
173 }
174 //-----
175 //updateProd
176 //IMPORT: item (Carton)
177 //PURPOSE: Adds new item to correct node in Binary Tree
178
179 private void updateProd(Carton item)
180 {
181     DSALinkedList newList = new DSALinkedList();
182     newList.insertLast( item );
183
184     //If insert worked, it was the first item and now we have a new node.
185     //If not, it failed because the key already exists
186     //We get the existing linked list, and add our item to the end
187     int status = prodTree.insert( item.getProduct(), newList );
188     if ( status == -1 )
189     {
190         newList = prodTree.find( item.getProduct() );
191         newList.insertLast( item );
192     }
193 }
194 //-----
195 //updateWhole
196 //IMPORT: item (Carton)
197 //PURPOSE: Adds new item to correct node in Binary Tree
198
199 private void updateWhole(Carton item)
200 {
201     DSALinkedList newList = new DSALinkedList();
202     newList.insertLast( item );
203
204     //If insert worked, it was the first item and now we have a new node.
205     //If not, it failed because the key already exists
206     //We get the existing linked list, and add our item to the end
207     int status = wholeTree.insert( item.getWhole(), newList );
208     if ( status == -1 )
209     {
210         newList = wholeTree.find( item.getWhole() );
211         newList.insertLast( item );
212     }
213 }
214 //-----
215 //removeDate
216 //IMPORT: item (Carton)
217 //PURPOSE: Remove matching item from sorted Date list
218
219 private void removeDate(Carton item)
220 {
221     dateList.removeSpecific(item);
222 }
223 //-----
224 //removeProd
225 //IMPORT: item (Carton)
226 //PURPOSE: Remove matching item from linked list within binary tree
227
228 private void removeProd(Carton item)
229 {
230     DSALinkedList prodList = prodTree.find( item.getProduct() );
231     prodList.removeSpecific(item);
232     if ( prodList.getLength() == 0 )
233     {
234         prodTree.delete( item.getProduct() );
235     }
236 }
237 //-----
238 //removeWhole
239 //IMPORT: item (Carton)
240 //PURPOSE: Remove matching item from linked list within binary tree
241
242 private void removeWhole(Carton item)
243 {
244     DSALinkedList wholeList = wholeTree.find( item.getWhole() );
245     wholeList.removeSpecific(item);
246     if ( wholeList.getLength() == 0 )
247     {
248         wholeTree.delete( item.getWhole() );
249     }
250 }
251 //-----
252 //searchProd
253 //IMPORT: product (String)
254 //PURPOSE: Returns linked List if Cartons with matching 'product'
255
256 public DSALinkedList searchProd(String product)
257 {
258     //Find list from Binary Tree

```

```

259     DSALinkedList matches = prodTree.find(product);
260     //If it's null, no list exists. Return an empty list
261     if ( matches == null )
262     {
263         matches = new DSALinkedList();
264     }
265     return matches;
266 }
267 //-----
268 //searchWhole
269 //IMPORT: wholesaler (String)
270 //PURPOSE: Returns linked List if Cartons with matching 'wholesaler'
271
272 public DSALinkedList searchWhole(String wholesaler)
273 {
274     //Find list from Binary Tree
275     DSALinkedList matches = wholeTree.find(wholesaler);
276     //If it's null, no list exists. Return an empty list
277     if ( matches == null )
278     {
279         matches = new DSALinkedList();
280     }
281     return matches;
282 }
283 //-----
284 //searchDate
285 //IMPORT: date (String)
286 //PURPOSE:
287
288 public DSALinkedList searchDate(String date)
289 {
290     DateClass maxDate = new DateClass(date);
291     DSALinkedList matches = new DSALinkedList();
292
293     //Iterate across the list
294     Iterator iter = dateList.iterator();
295     Carton item = (Carton)iter.next();
296
297     //Use compareTo to check that item date is less than the max date
298     while ( ( iter.hasNext() ) && ( item.getWar().compareTo(maxDate) <= 0 ) )
299     {
300         //Add any matching values to a new linked list
301         matches.insertFirst( item );
302         item = (Carton)iter.next();
303     }
304
305     return matches;
306 }
307 //-----
308 //printTree
309 //PURPOSE: Debugging Uses. Print's Both Tree's To Standard Out
310
311 public void printTree()
312 {
313     prodTree.printTree();
314     wholeTree.printTree();
315 }
316 //-----
317 }

```