10/26/15 06:38:07 /home/15504319/DSA120/DSAAssignment/TaskFunctions.java

```
  1 /*****************************************************************************
  2 *    FILE: TaskFunctions.java
  3 *    AUTHOR: Connor Beardsmore - 15504319
  4 *    UNIT: DSA120 Assignment S2- 2015
  5 *    PURPOSE: Handles and executes tasks on DC given a Taskline
  6 *    LAST MOD: 26/10/15
  7 *  REQUIRES: java.text.DateFormat, java.text.SimpleDateFormat
  8 *            java.util.Date, java.util.Iterator, connorLib
  9 *****************************************************************************/
 10 import java.text.DateFormat;
 11 import java.text.SimpleDateFormat;
 12 import java.util.Date;
 13 import java.util.Iterator;
 14 import connorLib.*;
 15
 16 public class TaskFunctions
 17 {
 18     //ALGORITHM CONSTANTS
 19     private final static int MONTH_URGENCY = 6;
 20     private final static int PREFERENCES = 3;
 21     private static int SHUFFLE_INCREMENT = 5;
 22     private static int MAX_SEARCH_PARAMS = 4;
 23
 24     //SEARCH CONSTANTS
 25     private final static int  NOTE = 0;
 26     private final static int  DATE = 1;
 27     private final static int  PRODUCT = 2;
 28     private final static int  WHOLESALER = 3;
 29 //---------------------------------------------------------------------------
 30     //addTask
 31     //IMPORT: indexArray (Carton), dc (DistroCentre), cartLine (String), avoid (int)
 32     //PURPOSE: Places new Carton in most appropriate spot in DC
 33
 34     public static void addTask(DistroCentre dc, CartonSearcher cs,
 35                                                 String cartLine, int avoid)
 36     {
 37         Carton item = new Carton(cartLine);
 38         DateClass warranty = item.getWar();
 39         DateFormat dF = new SimpleDateFormat("yyyy-MM-dd");
 40         DateClass curDate = new DateClass( dF.format( new Date() ) );
 41         int[] prefs = new int[PREFERENCES];
 42
 43         //Key con note isn't already in the DC
 44         if ( dc.getCartonIndex( item.getNote() ) != null )
 45         {
 46             throw new IllegalArgumentException("Carton already exists in DC: "
 47                                 + dc.getCartonIndex( item.getNote() ));
 48         }
 49         //Cannot add if DC is full
 50         if ( dc.isFull() )
 51         {
 52             System.out.println("FULL");
 53         }
 54         else
 55         {
 56             //All Call the same function, order of PREFERENCE is different
 57             //Firstmost parameters are higher pref, so are checked first
 58             //Only in worst case (i.e. nearly full dc), does 3rd pref get checked
 59             if ( warranty.isInfinite() )
 60             {
 61                 prefs[0] = dc.DEADEND;
 62                 prefs[1] = dc.ROLLING;
 63                 prefs[2] = dc.YARD;

 64                 processAdd(dc, cs, item, prefs, avoid);
 65             }
 66             //Item classed as urgent
 67             else if ( warranty.withinMonths(curDate, MONTH_URGENCY) )
 68             {
 69                 prefs[0] = dc.YARD;
 70                 prefs[1] = dc.ROLLING;
 71                 prefs[2] = dc.DEADEND;
 72                 processAdd(dc, cs, item, prefs, avoid);
 73             }
 74             //Item classed as nonurgent
 75             else
 76             {
 77                 prefs[0] = dc.ROLLING;
 78                 prefs[1] = dc.YARD;
 79                 prefs[2] = dc.DEADEND;
 80                 processAdd(dc, cs, item, prefs, avoid);
 81             }
 82             //User output given in specs
 83             if ( avoid == -1 )
 84             {
```

```
85                  System.out.println( item.getDIndex() + ":" + item.getRIndex() );
86              }
87          }
88      }
89  //-------------------------------------------------------------------------
90      //processAdd
91      //IMPORT: dc (DistroCentre), item (Carton), prefs (int[]), avoid (int)
92      //PURPOSE: Add's Cartons to first available slot in a room of
93      //           matching preference. Will never add to stockroom with index
94      //           avoid. avoid of -1 if this field not relevant
95
96      private static void processAdd(DistroCentre dc, CartonSearcher cs,
97                                     Carton item, int[] prefs, int avoid)
98      {
99          boolean done = false;
100
101         //Iterates over preferences from highest to lowest priority
102         for ( int ii = 0; ii < PREFERENCES; ii++ )
103         {
104             int jj = 0;
105
106             //Iterates over all stockrooms
107             while ( ( jj < dc.getCount() ) && ( done == false ) )
108             {
109                 //If stockroom matches current preference
110                 if ( ( dc.getType(jj) == prefs[ii] ) && ( jj != avoid ) )
111                 {
112                     IStockRoom room = dc.getStockRoom(jj);
113                     //If room isn't fully, add carton to it
114                     if ( !room.isFull() )
115                     {
116                         room.addCarton(item);
117                         item.setDIndex(jj);
118                         dc.setCartonIndex(item);
119                         //Update search environment to ensure search is quick
120                         cs.addSearchEnvironment(item);
121                         done = true;
122                         //break; (more efficient to exit loop ASAP after add)
123                     }
124                 }
125                 jj++;
126             }
127         }
128     }
129 //-------------------------------------------------------------------------
130     //removeTask
131     //IMPORT: dc (DistroCentre), cs (CartonSearcher) product (String)
132     //PURPOSE: Removes matching Carton from most appropriate spot in DC
133
134     public static void removeTask(DistroCentre dc, CartonSearcher cs,
135                                                         String product)
136     {
137         boolean done = false;
138         String searchLine = "::" + product + ":";
139         Carton[] matchArray = searchTask(dc, cs, searchLine);
140         DSALinkedList dateList = new DSALinkedList();
141         int steps, maxShuffles = 0;
142
143         //Max possible shuffles that can ever be attained
144         int freeSlots = dc.totalCapacity() - dc.totalItems();
145         //Copy all matching products into a linkedlist, sorted by date asc.
146         dateList = arrayToList( matchArray );
147
148         do
149         {
150             Iterator iter = dateList.iterator();
151             //Incremenet shuffles, check to see its not greater than freeSlots
152             maxShuffles += SHUFFLE_INCREMENT;
153             if ( maxShuffles > freeSlots )
154             {
155                 maxShuffles = freeSlots;
156             }
157
158             while ( (iter.hasNext() ) && ( done == false) )
159             {
160                 //Get next item in list, calculate steps needed to remove
161                 Carton item = (Carton)iter.next();
162                 steps = calcSteps(dc, item);
163
164                 //If it's considered "effecient", remove it
165                 if ( steps < maxShuffles )
166                 {
167                     executeRemove(dc, cs, item, steps);
168                     done = true;
169                 }
170             }
171         //If list is empty (i.e.No Matches), will iterate once and then stop
```

```
172         } while ( !( dateList.isEmpty() ) && ( maxShuffles < freeSlots )
173                                           && ( done == false ) );
174
175         //If matching elements where found, but no remove happened
176         if ( !( dateList.isEmpty() ) && (done == false) )
177         {
178             System.out.println("STUCK");
179         }
180     }
181 //-------------------------------------------------------------------------
182     //calcSteps
183     //IMPORT: dc (DistroCentre), item (Carton)
184     //EXPORT: steps (int)
185     //PURPOSE: Calculates the number of steps to remove an item from a stockroom
186
187     private static int calcSteps(DistroCentre dc, Carton item)
188     {
189         int steps = 0;
190         IStockRoom itemRoom = dc.getStockRoom( item.getDIndex() );
191
192         //Stack steps = Count - Location - 1
193         if ( itemRoom instanceof DeadEnd )
194         {
195             steps = itemRoom.getCount() - item.getRIndex() - 1;
196         }
197         //Queue steps = Index
198         else if ( itemRoom instanceof Rolling )
199         {
200             steps = item.getRIndex();
201         }
202         return steps;
203     }
204 //-------------------------------------------------------------------------
205     //shuffleCartons
206     //IMPORT: dc (DistroCentre), item (Carton)
207     //PURPOSE: Shuffles cartons in the DC to allow allow to the required item
208
209     private static void shuffleCartons(DistroCentre dc, CartonSearcher cs,
210                                                         Carton item)
211     {
212         String cartLine = null;
213         String searchResults = null;
214         //Get stockroom required to shuffle items from and remove one item
215         IStockRoom room = dc.getStockRoom( item.getDIndex() );
216         Carton removedItem = room.removeCarton();
217
218         //Loop until the item we remove is what we want
219         while ( removedItem != item )
220         {
221             //Remove from DC index array and Search Environment
222             dc.nullCartonIndex( removedItem.getNote() );
223             cs.removeSearchEnvironment(removedItem);
224             //Since the item is not the one we require, we add back
225             cartLine = removedItem.toString();
226             addTask( dc, cs, cartLine, item.getDIndex() );
227             //Remove the next item, but store details just prior so we can
228             //print them to the user
229             searchResults = stringSearchResults(item);
230             removedItem = room.removeCarton();
231         }
232
233         //Once removed correctly, output to user the details
234         System.out.println( searchResults );
235     }
236 //-------------------------------------------------------------------------
237     //executeRemove
238     //IMPORT: dc (DistroCentre), cs (CartonSearcher), item (Carton), steps (int)
239
240     private static void executeRemove(DistroCentre dc, CartonSearcher cs,
241                                             Carton item, int steps)
242     {
243         //If 0, must be in Yard/top of stack/front of queue
244         if ( steps == 0 )
245         {
246             System.out.println( stringSearchResults(item) );
247             IStockRoom room = dc.getStockRoom( item.getDIndex() );
248             if ( room instanceof Yard )
249             {
250                 //Can remove straight from any index
251                 ((Yard)room).removeCarton( item.getRIndex() );
252             }
253             else
254             {
255                 //No shuffling require
256                 room.removeCarton();
257             }
258         }
```

```
259         else
260         {
261             //Shuffle items away to allow us to remove
262             shuffleCartons(dc, cs, item);
263         }
264         //Update Links to cartons in both Search and DC index array
265         dc.nullCartonIndex( item.getNote() );
266         cs.removeSearchEnvironment(item);
267     }
268 //-------------------------------------------------------------------------
269     //searchTask
270     //IMPORT: dc (DistroCentre), cartLine (String)
271     //PURPOSE: Finds all instances of matching Carton in DC
272 
273     public static Carton[] searchTask(DistroCentre dc, CartonSearcher cs,
274                                                         String cartLine)
275     {
276         String[] tokens = cartLine.split(":", -1);
277         int searchParams = getParamNum(tokens);
278         DSALinkedList matches = null;
279         Carton[] matchArray = null;
280 
281         //Validate there are 4 fields total, and at least one search param given
282         if ( (tokens.length != MAX_SEARCH_PARAMS) || (searchParams == 0) )
283         {
284             throw new IllegalArgumentException("Invalid Search Task");
285         }
286         //If con note is in search, O(1) straight into index array
287         if ( !(tokens[NOTE].equals("")) && ( searchParams == 1 ) )
288         {
289             matchArray = searchConNote( dc, tokens[NOTE] );
290         }
291         //If param is 1, we search appropriate data structure, no cross
292         //    referencing is required to confirm matches.
293         else if ( searchParams == 1 )
294         {
295             matchArray = searchSingleParam(cs, tokens);
296         }
297         //If param is 2, search by one paramater, and then cross reference
298         else if ( searchParams == 2 )
299         {
300             if ( (!(tokens[PRODUCT].equals("")))
301                                     && (!(tokens[WHOLESALER].equals(""))) )
302             {
303                 matchArray = searchProdWhole(cs, tokens);
304             }
305             else if ( (!(tokens[PRODUCT].equals("")))
306                                     && (!(tokens[DATE].equals(""))) )
307             {
308                 matchArray = searchProdDate(cs, tokens);
309             }
310 
311             else if ( (!(tokens[WHOLESALER].equals("")))
312                                     && (!(tokens[DATE].equals(""))) )
313             {
314                 matchArray = searchWholeDate(cs, tokens);
315             }
316         }
317         else
318         {
319             matchArray = searchAllParam(cs, tokens);
320         }
321         //If array is empty, no matching elements were found
322         if ( matchArray.length == 0 )
323         {
324             System.out.println("NOT FOUND");
325         }
326 
327         return matchArray;
328     }
329 //-------------------------------------------------------------------------
330     //searchConNote
331     //IMPORT: dc (DistroCentre), note (String)
332     //EXPORT: matchArray (Carton[])
333     //PURPOSE: Search DC for Carton matching String note, return the Carton
334 
335     private static Carton[] searchConNote(DistroCentre dc, String note)
336     {
337         Carton[] matchArray = null;
338         int conNote = Integer.parseInt(note);
339 
340         if ( ( conNote < 1 ) || ( conNote > 1023 ) )
341         {
342             throw new IllegalArgumentException("Invalid Task File : Con Note");
343         }
344 
345         Carton item = dc.getCartonIndex( conNote );
```

```
346          if ( item != null )
347          {
348              matchArray = new Carton[1];
349              matchArray[0] = item;
350          }
351          else
352          {
353              matchArray = new Carton[0];
354          }
355          return matchArray;
356      }
357  //-----------------------------------------------------------------------
358      //searchSingleParam
359      //IMPORT: cs (CartonSearcher), tokens (String[])
360      //EXPORT: matchArray (Carton[])
361      //PURPOSE: Call appropriate method to return an array of matching Cartons
362
363      private static Carton[] searchSingleParam(CartonSearcher cs, String[] tokens)
364      {
365          DSALinkedList matches = null;
366          Carton[] matchArray = null;
367
368          //Search by product only
369          if ( !(tokens[PRODUCT].equals("")) )
370          {
371              matches = cs.searchProd( tokens[PRODUCT] );
372          }
373          //Search by wholesaler only
374          else if ( !(tokens[WHOLESALER].equals("")) )
375          {
376              matches = cs.searchWhole( tokens[WHOLESALER] );
377          }
378          //Search by date only
379          else if ( !(tokens[NOTE].equals("")) )
380          {
381              matches = cs.searchDate( tokens[DATE] );
382          }
383
384          return listToArray(matches);
385      }
386  //-----------------------------------------------------------------------
387      //searchAllParam
388      //IMPORT: cs (CartonSearcher), tokens (String[])
389      //EXPORT: matchArray (Carton[])
390      //PURPOSE: Search DC for Carton matching String note, return the Carton
391
392      private static Carton[] searchAllParam(CartonSearcher cs, String[] tokens)
393      {
394          //Get list of Cartons that match the specified Product type
395          DSALinkedList matches = cs.searchProd( tokens[PRODUCT] );
396          DateClass maxDate = new DateClass( tokens[DATE] );
397          DSALinkedList crossRefed = new DSALinkedList();
398          Iterator iter = matches.iterator();
399
400          //Iterator across list of all matching Cartons
401          while ( iter.hasNext() )
402          {
403              Carton item = (Carton)iter.next();
404              //Cross-reference by checked other parameters, Add items
405              //that meet all criteria to crossRefed list
406
407              System.out.println( item.getWar().compareTo(maxDate) <= 0);
408
409              if ( item.getWhole().equals(tokens[WHOLESALER]) )
410              {
411                  if ( ( item.getWar().compareTo(maxDate) <= 0)
412                                          || ( maxDate.isInfinite() ) )
413                  {
414                      crossRefed.insertFirst( item );
415                  }
416              }
417          }
418          return listToArray(crossRefed);
419      }
420  //-----------------------------------------------------------------------
421      //searchProdWhole
422      //IMPORT: cs (CartonSearcher), tokens (String[])
423      //EXPORT: matchArray (Carton[])
424      //PURPOSE: Search DC for Carton matching both product and wholesaler
425
426      private static Carton[] searchProdWhole(CartonSearcher cs, String[] tokens)
427      {
428          //Get list of Cartons that match the specified Product type
429          DSALinkedList matches = cs.searchProd( tokens[PRODUCT] );
430          DSALinkedList crossRefed = new DSALinkedList();
431          Iterator iter = matches.iterator();
432          while ( iter.hasNext() )
```

```
433         {
434             Carton item = (Carton)iter.next();
435             //Cross-reference by Wholesaler, Add items
436             //that meet criteria to crossRefed list
437             if ( item.getWhole().equals(tokens[WHOLESALER]) )
438             {
439                 crossRefed.insertFirst( item );
440             }
441         }
442         return listToArray(crossRefed);
443     }
444 //-------------------------------------------------------------------------
445     //searchProdDate
446     //IMPORT: cs (CartonSearcher), tokens (String[])
447     //EXPORT: matchArray (Carton[])
448     //PURPOSE: Search DC for Carton matching both product and date
449 
450     private static Carton[] searchProdDate(CartonSearcher cs, String[] tokens)
451     {
452         //Get list of Cartons that match the specified Product type
453         DSALinkedList matches = cs.searchProd( tokens[PRODUCT] );
454         DateClass maxDate = new DateClass( tokens[DATE] );
455         DSALinkedList crossRefed = new DSALinkedList();
456         Iterator iter = matches.iterator();
457         while ( iter.hasNext() )
458         {
459             Carton item = (Carton)iter.next();
460             //Cross-reference by Date, Add items
461             //that meet criteria to crossRefed list
462             if ( ( item.getWar().compareTo(maxDate) <= 0 )
463                                     || ( maxDate.isInfinite() ) )
464             {
465                 crossRefed.insertFirst( item );
466             }
467         }
468         return listToArray(crossRefed);
469     }
470 //-------------------------------------------------------------------------
471     //searchWholeDate
472     //IMPORT: cs (CartonSearcher), tokens (String[])
473     //EXPORT: matchArray (Carton[])
474     //PURPOSE: Search DC for Carton matching both wholesaler and date
475 
476     private static Carton[] searchWholeDate(CartonSearcher cs, String[] tokens)
477     {
478         //Get list of Cartons that match the specified Wholesaler type
479         DSALinkedList matches = cs.searchWhole( tokens[WHOLESALER] );
480         DateClass maxDate = new DateClass( tokens[DATE] );
481         DSALinkedList crossRefed = new DSALinkedList();
482         Iterator iter = matches.iterator();
483         while ( iter.hasNext() )
484         {
485             Carton item = (Carton)iter.next();
486             //Cross-reference by Date, Add items
487             //that meet criteria to crossRefed list
488             if ( ( item.getWar().compareTo(maxDate) <= 0 )
489                                     || ( maxDate.isInfinite() ) )
490             {
491                 crossRefed.insertFirst( item );
492             }
493         }
494         return listToArray(crossRefed);
495     }
496 //-------------------------------------------------------------------------
497     //getParamNum
498     //IMPORT: tokens (String[])
499     //EXPORT: searchParams (int)
500     //PURPOSE: Get number of tokens that aren't null, excluding the first
501 
502     private static int getParamNum(String[] tokens)
503     {
504         int searchParams = 0;
505         for ( int ii = 1; ii < tokens.length; ii++ )
506         {
507             //If token is empty, we don't search by that parameter
508             if ( !(tokens[ii].equals("")) )
509             {
510                 searchParams++;
511             }
512         }
513         return searchParams;
514     }
515 //-------------------------------------------------------------------------
516     //arrayToList
517     //IMPORT: array (Carton[])
518     //EXPORT: newList
519     //PURPOSE: Convert an array to a new sorted linked list, based on date
```

```
520
521     private static DSALinkedList arrayToList(Carton[] array)
522     {
523         DSALinkedList newList = new DSALinkedList();
524
525         //Iterate across array, insert elements into list
526         for ( int ii = 0; ii < array.length; ii++ )
527         {
528             if ( array[ii].getWar().isInfinite() )
529             {
530                 //Lifetime warranty items go to the end
531                 newList.insertLast( array[ii] );
532             }
533             else
534             {
535                 //Everything else gets sorted appropriately
536                 newList.insertSorted( array[ii] );
537             }
538         }
539         return newList;
540     }
541 //---------------------------------------------------------------------------
542     //listToArray
543     //IMPORT: matches (DSALinkedList)
544     //EXPORT: matchArray (Carton[])
545     //PURPOSE: Convert a linked list to a new array
546
547     private static Carton[] listToArray(DSALinkedList matches)
548     {
549         Carton[] matchArray = new Carton[matches.getLength()];
550         Iterator iter = matches.iterator();
551
552         //Copy each element across individually
553         for (int ii = 0; ii < matchArray.length; ii++)
554         {
555             matchArray[ii] = (Carton)iter.next();
556         }
557         return matchArray;
558     }
559 //---------------------------------------------------------------------------
560     //printArray
561     //IMPORT: matches (Carton[])
562     //PURPOSE: Sort an array and output its contents
563
564     public static void printArray(Carton[] matches)
565     {
566         //Sort cartons via distroIndex and roomIndex
567         Sorts.quickSort( matches );
568
569         for (int jj = 0; jj < matches.length; jj++)
570         {
571             System.out.println( stringSearchResults(matches[jj]) );
572         }
573     }
574 //---------------------------------------------------------------------------
575     //stringSearchResults
576     //IMPORT: match (Carton)
577     //EXPORT: statestring (String)
578     //PURPOSE: Print Carton in format dIndex:rIndex:N:WA:P:WH
579
580     private static String stringSearchResults(Carton match)
581     {
582         String statestring = match.getDIndex() + ":" + match.getRIndex();
583         statestring += ":" + match.toString();
584         return statestring;
585     }
586 //---------------------------------------------------------------------------
587 }
```