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

```java
/***************************************************************************
*    FILE: TaskFunctions.java
*    AUTHOR: Connor Beardsmore - 15504319
*    UNIT: DSA120 Assignment S2- 2015
*    PURPOSE: Handles and executes tasks on DC given a Taskline
*    LAST MOD: 26/10/15
*  REQUIRES: java.text.DateFormat, java.text.SimpleDateFormat
*            java.util.Date, java.util.Iterator, connorLib
***************************************************************************/
import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import connorLib.*;

public class TaskFunctions
{
    //ALGORITHM CONSTANTS
    private final static int MONTH_URGENCY = 6;
    private final static int PREFERENCES = 3;
    private static int SHUFFLE_INCREMENT = 5;
    private static int MAX_SEARCH_PARAMS = 4;

    //SEARCH CONSTANTS
    private final static int  NOTE = 0;
    private final static int  DATE = 1;
    private final static int  PRODUCT = 2;
    private final static int  WHOLESALER = 3;
//---------------------------------------------------------------------------
    //addTask
    //IMPORT: indexArray (Carton), dc (DistroCentre), cartLine (String), avoid (int)
    //PURPOSE: Places new Carton in most appropriate spot in DC

    public static void addTask(DistroCentre dc, CartonSearcher cs,
                                              String cartLine, int avoid)
    {
        Carton item = new Carton(cartLine);
        DateClass warranty = item.getWar();
        DateFormat dF = new SimpleDateFormat("yyyy-MM-dd");
        DateClass curDate = new DateClass( dF.format( new Date() ) );
        int[] prefs = new int[PREFERENCES];

        //Key con note isn't already in the DC
        if ( dc.getCartonIndex( item.getNote() ) != null )
        {
            throw new IllegalArgumentException("Carton already exists in DC: "
                                    + dc.getCartonIndex( item.getNote() ));
        }
        //Cannot add if DC is full
        if ( dc.isFull() )
        {
            System.out.println("FULL");
        }
        else
        {
            //All Call the same function, order of PREFERENCE is different
            //Firstmost parameters are higher pref, so are checked first
            //Only in worst case (i.e. nearly full dc), does 3rd pref get checked
            if ( warranty.isInfinite() )
            {
                prefs[0] = dc.DEADEND;
                prefs[1] = dc.ROLLING;
                prefs[2] = dc.YARD;

                processAdd(dc, cs, item, prefs, avoid);
            }
            //Item classed as urgent
            else if ( warranty.withinMonths(curDate, MONTH_URGENCY) )
            {
                prefs[0] = dc.YARD;
                prefs[1] = dc.ROLLING;
                prefs[2] = dc.DEADEND;
                processAdd(dc, cs, item, prefs, avoid);
            }
            //Item classed as nonurgent
            else
            {
                prefs[0] = dc.ROLLING;
                prefs[1] = dc.YARD;
                prefs[2] = dc.DEADEND;
                processAdd(dc, cs, item, prefs, avoid);
            }
            //User output given in specs
            if ( avoid == -1 )
            {
                System.out.println( item.getDIndex() + ":" + item.getRIndex() );
            }
        }
    }
//---------------------------------------------------------------------------
    //processAdd
    //IMPORT: dc (DistroCentre), item (Carton), prefs (int[]), avoid (int)
    //PURPOSE: Add's Cartons to first available slot in a room of
    //          matching preference. Will never add to stockroom with index
    //          avoid. avoid of -1 if this field not relevant

    private static void processAdd(DistroCentre dc, CartonSearcher cs,
                                    Carton item, int[] prefs, int avoid)
    {
        boolean done = false;

        //Iterates over preferences from highest to lowest priority
        for ( int ii = 0; ii < PREFERENCES; ii++ )
        {
            int jj = 0;

            //Iterates over all stockrooms
            while ( ( jj < dc.getCount() ) && ( done == false ) )
            {
                //If stockroom matches current preference
                if ( ( dc.getType(jj) == prefs[ii] ) && ( jj != avoid ) )
                {
                    IStockRoom room = dc.getStockRoom(jj);
                    //If room isn't fully, add carton to it
                    if ( !room.isFull() )
                    {
                        room.addCarton(item);
                        item.setDIndex(jj);
                        dc.setCartonIndex(item);
                        //Update search environment to ensure search is quick
                        cs.addSearchEnvironment(item);
                        done = true;
                        //break; (more efficient to exit loop ASAP after add)
                    }
                }
                jj++;
            }
        }
    }
//---------------------------------------------------------------------------
    //removeTask
    //IMPORT: dc (DistroCentre), cs (CartonSearcher) product (String)
    //PURPOSE: Removes matching Carton from most appropriate spot in DC

    public static void removeTask(DistroCentre dc, CartonSearcher cs,
                                                        String product)
    {
        boolean done = false;
        String searchLine = "::" + product + ":";
        Carton[] matchArray = searchTask(dc, cs, searchLine);
        DSALinkedList dateList = new DSALinkedList();
        int steps, maxShuffles = 0;

        //Max possible shuffles that can ever be attained
        int freeSlots = dc.totalCapacity() - dc.totalItems();
        //Copy all matching products into a linkedlist, sorted by date asc.
        dateList = arrayToList( matchArray );

        do
        {
            Iterator iter = dateList.iterator();
            //Incremenet shuffles, check to see its not greater than freeSlots
            maxShuffles += SHUFFLE_INCREMENT;
            if ( maxShuffles > freeSlots )
            {
                maxShuffles = freeSlots;
            }

            while ( (iter.hasNext() ) && ( done == false) )
            {
                //Get next item in list, calculate steps needed to remove
                Carton item = (Carton)iter.next();
                steps = calcSteps(dc, item);

                //If it's considered "effecient", remove it
                if ( steps < maxShuffles )
                {
                    executeRemove(dc, cs, item, steps);
                    done = true;
                }
            }
        //If list is empty (i.e.No Matches), will iterate once and then stop
        } while ( !( dateList.isEmpty() ) && ( maxShuffles < freeSlots )
                                          && ( done == false ) );

        //If matching elements where found, but no remove happened
        if ( !( dateList.isEmpty() ) && (done == false) )
        {
            System.out.println("STUCK");
        }
    }
//---------------------------------------------------------------------------
    //calcSteps
    //IMPORT: dc (DistroCentre), item (Carton)
    //EXPORT: steps (int)
    //PURPOSE: Calculates the number of steps to remove an item from a stockroom

    private static int calcSteps(DistroCentre dc, Carton item)
    {
        int steps = 0;
        IStockRoom itemRoom = dc.getStockRoom( item.getDIndex() );

        //Stack steps = Count - Location - 1
        if ( itemRoom instanceof DeadEnd )
        {
            steps = itemRoom.getCount() - item.getRIndex() - 1;
        }
        //Queue steps = Index
        else if ( itemRoom instanceof Rolling )
        {
            steps = item.getRIndex();
        }
        return steps;
    }
//---------------------------------------------------------------------------
    //shuffleCartons
    //IMPORT: dc (DistroCentre), item (Carton)
    //PURPOSE: Shuffles cartons in the DC to allow allow to the required item

    private static void shuffleCartons(DistroCentre dc, CartonSearcher cs,
                                                              Carton item)
    {
        String cartLine = null;
        String searchResults = null;
        //Get stockroom required to shuffle items from and remove one item
        IStockRoom room = dc.getStockRoom( item.getDIndex() );
        Carton removedItem = room.removeCarton();

        //Loop until the item we remove is what we want
        while ( removedItem != item )
        {
            //Remove from DC index array and Search Environment
            dc.nullCartonIndex( removedItem.getNote() );
            cs.removeSearchEnvironment(removedItem);
            //Since the item is not the one we require, we add back
            cartLine = removedItem.toString();
            addTask( dc, cs, cartLine, item.getDIndex() );
            //Remove the next item, but store details just prior so we can
            //print them to the user
            searchResults = stringSearchResults(item);
            removedItem = room.removeCarton();
        }

        //Once removed correctly, output to user the details
        System.out.println( searchResults );
    }
//---------------------------------------------------------------------------
    //executeRemove
    //IMPORT: dc (DistroCentre), cs (CartonSearcher), item (Carton), steps (int)

    private static void executeRemove(DistroCentre dc, CartonSearcher cs,
                                                  Carton item, int steps)
    {
        //If 0, must be in Yard/top of stack/front of queue
        if ( steps == 0 )
        {
            System.out.println( stringSearchResults(item) );
            IStockRoom room = dc.getStockRoom( item.getDIndex() );
            if ( room instanceof Yard )
            {
                //Can remove straight from any index
                ((Yard)room).removeCarton( item.getRIndex() );
            }
            else
            {
                //No shuffling require
                room.removeCarton();
            }
        }
```

```java
        else
        {
            //Shuffle items away to allow us to remove
            shuffleCartons(dc, cs, item);
        }
        //Update Links to cartons in both Search and DC index array
        dc.nullCartonIndex( item.getNote() );
        cs.removeSearchEnvironment(item);
    }
//--------------------------------------------------------------------------
    //searchTask
    //IMPORT: dc (DistroCentre), cartLine (String)
    //PURPOSE: Finds all instances of matching Carton in DC

    public static Carton[] searchTask(DistroCentre dc, CartonSearcher cs,
                                                        String cartLine)
    {
        String[] tokens = cartLine.split(":", -1);
        int searchParams = getParamNum(tokens);
        DSALinkedList matches = null;
        Carton[] matchArray = null;

        //Validate there are 4 fields total, and at least one search param given
        if ( (tokens.length != MAX_SEARCH_PARAMS) || (searchParams == 0) )
        {
            throw new IllegalArgumentException("Invalid Search Task");
        }
        //If con note is in search, O(1) straight into index array
        if ( !(tokens[NOTE].equals("")) && ( searchParams == 1 ) )
        {
            matchArray = searchConNote( dc, tokens[NOTE] );
        }
        //If param is 1, we search appropriate data structure, no cross
        //    referencing is required to confirm matches.
        else if ( searchParams == 1 )
        {
            matchArray = searchSingleParam(cs, tokens);
        }
        //If param is 2, search by one paramater, and then cross reference
        else if ( searchParams == 2 )
        {
            if ( (!(tokens[PRODUCT].equals("")))
                                    && (!(tokens[WHOLESALER].equals(""))) )
            {
                matchArray = searchProdWhole(cs, tokens);
            }
            else if ( (!(tokens[PRODUCT].equals("")))
                                    && (!(tokens[DATE].equals(""))) )
            {
                matchArray = searchProdDate(cs, tokens);
            }

            else if ( (!(tokens[WHOLESALER].equals("")))
                                    && (!(tokens[DATE].equals(""))) )
            {
                matchArray = searchWholeDate(cs, tokens);
            }
        }
        else
        {
            matchArray = searchAllParam(cs, tokens);
        }
        //If array is empty, no matching elements were found
        if ( matchArray.length == 0 )
        {
            System.out.println("NOT FOUND");
        }

        return matchArray;
    }
//--------------------------------------------------------------------------
    //searchConNote
    //IMPORT: dc (DistroCentre), note (String)
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Search DC for Carton matching String note, return the Carton

    private static Carton[] searchConNote(DistroCentre dc, String note)
    {
        Carton[] matchArray = null;
        int conNote = Integer.parseInt(note);

        if ( ( conNote < 1 ) || ( conNote > 1023 ) )
        {
            throw new IllegalArgumentException("Invalid Task File : Con Note");
        }

        Carton item = dc.getCartonIndex( conNote );
        if ( item != null )
        {
            matchArray = new Carton[1];
            matchArray[0] = item;
        }
        else
        {
            matchArray = new Carton[0];
        }
        return matchArray;
    }
//----------------------------------------------------------------------------
    //searchSingleParam
    //IMPORT: cs (CartonSearcher), tokens (String[])
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Call appropriate method to return an array of matching Cartons

    private static Carton[] searchSingleParam(CartonSearcher cs, String[] tokens)
    {
        DSALinkedList matches = null;
        Carton[] matchArray = null;

        //Search by product only
        if ( !(tokens[PRODUCT].equals("")) )
        {
            matches = cs.searchProd( tokens[PRODUCT] );
        }
        //Search by wholesaler only
        else if ( !(tokens[WHOLESALER].equals("")) )
        {
            matches = cs.searchWhole( tokens[WHOLESALER] );
        }
        //Search by date only
        else if ( !(tokens[NOTE].equals("")) )
        {
            matches = cs.searchDate( tokens[DATE] );
        }

        return listToArray(matches);
    }
//----------------------------------------------------------------------------
    //searchAllParam
    //IMPORT: cs (CartonSearcher), tokens (String[])
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Search DC for Carton matching String note, return the Carton

    private static Carton[] searchAllParam(CartonSearcher cs, String[] tokens)
    {
        //Get list of Cartons that match the specified Product type
        DSALinkedList matches = cs.searchProd( tokens[PRODUCT] );
        DateClass maxDate = new DateClass( tokens[DATE] );
        DSALinkedList crossRefed = new DSALinkedList();
        Iterator iter = matches.iterator();

        //Iterator across list of all matching Cartons
        while ( iter.hasNext() )
        {
            Carton item = (Carton)iter.next();
            //Cross-reference by checked other parameters, Add items
            //that meet all criteria to crossRefed list

            System.out.println( item.getWar().compareTo(maxDate) <= 0);

            if ( item.getWhole().equals(tokens[WHOLESALER]) )
            {
                if ( ( item.getWar().compareTo(maxDate) <= 0)
                                            || ( maxDate.isInfinite() ) )
                {
                    crossRefed.insertFirst( item );
                }
            }
        }
        return listToArray(crossRefed);
    }
//----------------------------------------------------------------------------
    //searchProdWhole
    //IMPORT: cs (CartonSearcher), tokens (String[])
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Search DC for Carton matching both product and wholesaler

    private static Carton[] searchProdWhole(CartonSearcher cs, String[] tokens)
    {
        //Get list of Cartons that match the specified Product type
        DSALinkedList matches = cs.searchProd( tokens[PRODUCT] );
        DSALinkedList crossRefed = new DSALinkedList();
        Iterator iter = matches.iterator();
        while ( iter.hasNext() )
        {
            Carton item = (Carton)iter.next();
            //Cross-reference by Wholesaler, Add items
            //that meet criteria to crossRefed list
            if ( item.getWhole().equals(tokens[WHOLESALER]) )
            {
                crossRefed.insertFirst( item );
            }
        }
        return listToArray(crossRefed);
    }
//---------------------------------------------------------------------------
    //searchProdDate
    //IMPORT: cs (CartonSearcher), tokens (String[])
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Search DC for Carton matching both product and date

    private static Carton[] searchProdDate(CartonSearcher cs, String[] tokens)
    {
        //Get list of Cartons that match the specified Product type
        DSALinkedList matches = cs.searchProd( tokens[PRODUCT] );
        DateClass maxDate = new DateClass( tokens[DATE] );
        DSALinkedList crossRefed = new DSALinkedList();
        Iterator iter = matches.iterator();
        while ( iter.hasNext() )
        {
            Carton item = (Carton)iter.next();
            //Cross-reference by Date, Add items
            //that meet criteria to crossRefed list
            if ( ( item.getWar().compareTo(maxDate) <= 0 )
                                    || ( maxDate.isInfinite() ) )
            {
                crossRefed.insertFirst( item );
            }
        }
        return listToArray(crossRefed);
    }
//---------------------------------------------------------------------------
    //searchWholeDate
    //IMPORT: cs (CartonSearcher), tokens (String[])
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Search DC for Carton matching both wholesaler and date

    private static Carton[] searchWholeDate(CartonSearcher cs, String[] tokens)
    {
        //Get list of Cartons that match the specified Wholesaler type
        DSALinkedList matches = cs.searchWhole( tokens[WHOLESALER] );
        DateClass maxDate = new DateClass( tokens[DATE] );
        DSALinkedList crossRefed = new DSALinkedList();
        Iterator iter = matches.iterator();
        while ( iter.hasNext() )
        {
            Carton item = (Carton)iter.next();
            //Cross-reference by Date, Add items
            //that meet criteria to crossRefed list
            if ( ( item.getWar().compareTo(maxDate) <= 0 )
                                    || ( maxDate.isInfinite() ) )
            {
                crossRefed.insertFirst( item );
            }
        }
        return listToArray(crossRefed);
    }
//---------------------------------------------------------------------------
    //getParamNum
    //IMPORT: tokens (String[])
    //EXPORT: searchParams (int)
    //PURPOSE: Get number of tokens that aren't null, excluding the first

    private static int getParamNum(String[] tokens)
    {
        int searchParams = 0;
        for ( int ii = 1; ii < tokens.length; ii++ )
        {
            //If token is empty, we don't search by that parameter
            if ( !(tokens[ii].equals("")) )
            {
                searchParams++;
            }
        }
        return searchParams;
    }
//---------------------------------------------------------------------------
    //arrayToList
    //IMPORT: array (Carton[])
    //EXPORT: newList
    //PURPOSE: Convert an array to a new sorted linked list, based on date

    private static DSALinkedList arrayToList(Carton[] array)
    {
        DSALinkedList newList = new DSALinkedList();

        //Iterate across array, insert elements into list
        for ( int ii = 0; ii < array.length; ii++ )
        {
            if ( array[ii].getWar().isInfinite() )
            {
                //Lifetime warranty items go to the end
                newList.insertLast( array[ii] );
            }
            else
            {
                //Everything else gets sorted appropriately
                newList.insertSorted( array[ii] );
            }
        }
        return newList;
    }
//-----------------------------------------------------------------------------
    //listToArray
    //IMPORT: matches (DSALinkedList)
    //EXPORT: matchArray (Carton[])
    //PURPOSE: Convert a linked list to a new array

    private static Carton[] listToArray(DSALinkedList matches)
    {
        Carton[] matchArray = new Carton[matches.getLength()];
        Iterator iter = matches.iterator();

        //Copy each element across individually
        for (int ii = 0; ii < matchArray.length; ii++)
        {
            matchArray[ii] = (Carton)iter.next();
        }
        return matchArray;
    }
//-----------------------------------------------------------------------------
    //printArray
    //IMPORT: matches (Carton[])
    //PURPOSE: Sort an array and output its contents

    public static void printArray(Carton[] matches)
    {
        //Sort cartons via distroIndex and roomIndex
        Sorts.quickSort( matches );

        for (int jj = 0; jj < matches.length; jj++)
        {
            System.out.println( stringSearchResults(matches[jj]) );
        }
    }
//-----------------------------------------------------------------------------
    //stringSearchResults
    //IMPORT: match (Carton)
    //EXPORT: statestring (String)
    //PURPOSE: Print Carton in format dIndex:rIndex:N:WA:P:WH

    private static String stringSearchResults(Carton match)
    {
        String statestring = match.getDIndex() + ":" + match.getRIndex();
        statestring += ":" + match.toString();
        return statestring;
    }
//-----------------------------------------------------------------------------
}
```