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

```java
/***************************************************************************
*    FILE: CartonSearcher.java
*    AUTHOR: Connor Beardsmore - 15504319
*    UNIT: DSA120 Assignment S2- 2015
*    PURPOSE: Sets up and utlizes a Search Environment with data structures
*           to allow for quick searching of items within the DC
*    LAST MOD: 25/10/15
*  REQUIRES: java.util.Iterator, connorLib
**************************************************************************/
package connorLib;
import java.util.Iterator;

public class CartonSearcher
{
    //CLASSFIELDS
    private DSALinkedList dateList;
    private BinarySearchTree<String, DSALinkedList> prodTree;
    private BinarySearchTree<String, DSALinkedList> wholeTree;
//---------------------------------------------------------------------------
    //ALTERNATE Constructor
    //IMPORT: dc (DistroCentre)

    public CartonSearcher(DistroCentre dc)
    {
        buildSearchEnvironment(dc);
    }
//---------------------------------------------------------------------------
    //buildSearchEnvironment
    //IMPORT: dc (DistroCentre)
    //PURPOSE: Set-ups Data structures required for searching DC

    public void buildSearchEnvironment(DistroCentre dc)
    {
        setDateList(dc);
        setProdTree(dc);
        setWholeTree(dc);
    }
//---------------------------------------------------------------------------
    //addSearchEnvironment
    //IMPORT: item (Carton)
    //PURPOSE: Adds new Carton to existing search data structures

    public void addSearchEnvironment(Carton item)
    {
        updateDate(item);
        updateProd(item);
        updateWhole(item);
    }
//---------------------------------------------------------------------------
    //removeSearchEnvironment
    //IMPORT: item (Carton)
    //PURPOSE: Removes a carton from existing search data structures

    public void removeSearchEnvironment(Carton item)
    {
        removeDate(item);
        removeProd(item);
        removeWhole(item);
    }
//---------------------------------------------------------------------------
    //setDateList
    //IMPORT: dc (DistroCentre)
    //PURPOSE: Adds all Cartons into the DC into a linked list, sorted by Date


    private void setDateList(DistroCentre dc)
    {
        dateList = new DSALinkedList();

        for (int ii = 1; ii < Carton.MAX_CON_NOTE + 1; ii++)
        {
            Carton item = dc.getCartonIndex(ii);
            //Only Add to list if a Carton on that con note actually exists
            if ( item != null )
            {
                //If 0000-00-00, add to end of list
                if ( item.getWar().isInfinite() )
                {
                    dateList.insertLast(item);
                }
                //Insert in sorted order
                else
                {
                    dateList.insertSorted( item );
                }
            }
        }
    }
//---------------------------------------------------------------------------
    //setProdTree
    //IMPORT: dc (DistroCentre)
    //PURPOSE: Creates a Binary Search Tree, where each Node is itself a
    //            linked-list of items with the same product

    private void setProdTree(DistroCentre dc)
    {
        prodTree = new BinarySearchTree<String, DSALinkedList>();
        DSALinkedList newList = null;

        for (int ii = 1; ii < Carton.MAX_CON_NOTE + 1; ii++)
        {
            Carton item = dc.getCartonIndex(ii);
            //Only add if a carton actually exists in the DC
            if ( item != null )
            {
                //Create a new Linked list and add item to it.
                newList = new DSALinkedList();
                newList.insertLast( item );

                //Try to insert list into tree
                //Will either succeed and return false, or fail and return true
                int status = prodTree.insert( item.getProduct(), newList );
                if ( status == -1 )
                {
                    //List with this Wholeuct already exists in the tree
                    //So we find() to get the list, then add to the list
                    newList = prodTree.find( item.getProduct() );
                    newList.insertLast( item );
                }

            }
        }
    }
//---------------------------------------------------------------------------
    //setWholeTree
    //IMPORT: dc (DistroCentre)
    //PURPOSE: Creates a Binary Search Tree, where each Node is itself a
    //            linked-list of items with the same wholesaler

    private void setWholeTree(DistroCentre dc)
    {
        wholeTree = new BinarySearchTree<String, DSALinkedList>();
        DSALinkedList newList = null;

        for (int ii = 1; ii < Carton.MAX_CON_NOTE + 1; ii++)
        {
            Carton item = dc.getCartonIndex(ii);
            //Only add if a carton actually exists in the DC
            if ( item != null )
            {
                //Create a new Linked list and add item to it.
                newList = new DSALinkedList();
                newList.insertLast( item );

                //Try to insert list into tree
                //Will either succeed and return false, or fail and return true
                int status = wholeTree.insert( item.getWhole(), newList );
                if ( status == -1 )
                {
                    //List with this product already exists in the tree
                    //So we find() to get the list, then add to the list
                    newList = wholeTree.find( item.getWhole() );
                    newList.insertLast( item );
                }

            }
        }
    }
//---------------------------------------------------------------------------
    //updateDate
    //IMPORT: item (Carton)
    //PURPOSE: Adds new item to sorted Date list

    private void updateDate(Carton item)
    {
        if ( item.getWar().isInfinite() )
        {
            dateList.insertLast(item);
        }
        else
        {
            dateList.insertSorted( item );
        }
    }
//--------------------------------------------------------------------------
    //updateProd
    //IMPORT: item (Carton)
    //PURPOSE: Adds new item to correct node in Binary Tree

    private void updateProd(Carton item)
    {
        DSALinkedList newList = new DSALinkedList();
        newList.insertLast( item );

        //If insert worked, it was the first item and now we have a new node.
        //If not, it failed because the key already exists
        //We get the existing linked list, and add our item to the end
        int status = prodTree.insert( item.getProduct(), newList );
        if ( status == -1 )
        {
            newList = prodTree.find( item.getProduct() );
            newList.insertLast( item );
        }
    }
//--------------------------------------------------------------------------
    //updateWhole
    //IMPORT: item (Carton)
    //PURPOSE: Adds new item to correct node in Binary Tree

    private void updateWhole(Carton item)
    {
        DSALinkedList newList = new DSALinkedList();
        newList.insertLast( item );

        //If insert worked, it was the first item and now we have a new node.
        //If not, it failed because the key already exists
        //We get the existing linked list, and add our item to the end
        int status = wholeTree.insert( item.getWhole(), newList );
        if ( status == -1 )
        {
            newList = wholeTree.find( item.getWhole() );
            newList.insertLast( item );
        }
    }
//--------------------------------------------------------------------------
    //removeDate
    //IMPORT: item (Carton)
    //PURPOSE: Remove matching item from sorted Date list

    private void removeDate(Carton item)
    {
        dateList.removeSpecific(item);
    }
//--------------------------------------------------------------------------
    //removeProd
    //IMPORT: item (Carton)
    //PURPOSE: Remove matching item from linked list within binary tree

    private void removeProd(Carton item)
    {
        DSALinkedList prodList = prodTree.find( item.getProduct() );
        prodList.removeSpecific(item);
        if ( prodList.getLength() == 0 )
        {
            prodTree.delete( item.getProduct() );
        }
    }
//--------------------------------------------------------------------------
    //removeWhole
    //IMPORT: item (Carton)
    //PURPOSE: Remove matching item from linked list within binary tree

    private void removeWhole(Carton item)
    {
        DSALinkedList wholeList = wholeTree.find( item.getWhole() );
        wholeList.removeSpecific(item);
        if ( wholeList.getLength() == 0 )
        {
            wholeTree.delete( item.getWhole() );
        }
    }
//--------------------------------------------------------------------------
    //searchProd
    //IMPORT: product (String)
    //PURPOSE: Returns linked List if Cartons with matching 'product'

    public DSALinkedList searchProd(String product)
    {
        //Find list from Binary Tree
        DSALinkedList matches = prodTree.find(product);
        //If it's null, no list exists. Return an empty list
        if ( matches == null )
        {
            matches = new DSALinkedList();
        }
        return matches;
    }
//------------------------------------------------------------------------------
    //searchWhole
    //IMPORT: wholesaler (String)
    //PURPOSE: Returns linked List if Cartons with matching 'wholesaler'

    public DSALinkedList searchWhole(String wholesaler)
    {
        //Find list from Binary Tree
        DSALinkedList matches = wholeTree.find(wholesaler);
        //If it's null, no list exists. Return an empty list
        if ( matches == null )
        {
            matches = new DSALinkedList();
        }
        return matches;
    }
//------------------------------------------------------------------------------
    //searchDate
    //IMPORT: date (String)
    //PURPOSE:

    public DSALinkedList searchDate(String date)
    {
        DateClass maxDate = new DateClass(date);
        DSALinkedList matches = new DSALinkedList();

        //Iterate across the list
        Iterator iter = dateList.iterator();
        Carton item = (Carton)iter.next();

        //Use compareTo to check that item date is less than the max date
        while ( ( iter.hasNext() ) && ( item.getWar().compareTo(maxDate) <= 0 ) )
        {
            //Add any matching values to a new linked list
            matches.insertFirst( item );
            item = (Carton)iter.next();
        }

        return matches;
    }
//------------------------------------------------------------------------------
    //printTree
    //PURPOSE: Debugging Uses. Print's Both Tree's To Standard Out

    public void printTree()
    {
        prodTree.printTree();
        wholeTree.printTree();
    }
//------------------------------------------------------------------------------
}
```