Assignment 2: ***Supermarket statistics***

**Student 1  
Name**: Bram Janssen  
**StudentNr**: 500748908

**Student 2  
Name**: Maud de Jong  
**StudentNr**: 500855247

**Education**: Software Engineering   
**Domain**: HBO-ICT   
**Faculty**: Hogeschool van Amsterdam   
**Assessor**: Mark Smit **Occasion**: 1st Chance

Contents

[Introduction 3](#_Toc85308888)

[Snippet 1 4](#_Toc85308889)

[Sample indexOfByRecursiveBinarySearch () 4](#_Toc85308890)

[Function of code 4](#_Toc85308891)

[Justification why it is on the list 4](#_Toc85308892)

[Snippet 2 6](#_Toc85308893)

[Sample fromLine() 6](#_Toc85308894)

[Function of code 6](#_Toc85308895)

[Justification why it is on the list 6](#_Toc85308896)

[Snippet 3 7](#_Toc85308897)

[Sample mergePurchasesFromFile 7](#_Toc85308898)

[Function of code 7](#_Toc85308899)

[Justification why it is on the list 7](#_Toc85308900)

[Snippet 4 8](#_Toc85308901)

[Sample remove 8](#_Toc85308902)

[Function of code 8](#_Toc85308903)

[Justification why it is on the list 8](#_Toc85308904)

[Snippet 5 9](#_Toc85308905)

[Sample mergePurchasesFromFileRecursively() 9](#_Toc85308906)

[Function of code 9](#_Toc85308907)

[Justification why it is on the list 9](#_Toc85308908)

[Snippet 6 10](#_Toc85308909)

[Sample findUnsortedItem() 10](#_Toc85308910)

[Function of code 10](#_Toc85308911)

[Justification why it is on the list 10](#_Toc85308912)

[Snippet 7 11](#_Toc85308913)

[Sample canCompareProduct() 11](#_Toc85308914)

[Function of code 11](#_Toc85308915)

[Justification why it is on the list 11](#_Toc85308916)

# Introduction

This assignment was called ‘supermarket statistics’ and contains the methods product, purchase, purchaseTracker, orderedList and orderedArrayList. In this program you can load files containing product information and transform them into products. You can also load files containing a barcode and an amount which can we transformed into a purchase containing a product and a count of the amounts sold. You can get insight in what products do best/worst and merge multiple files into a summary of all sales.

There also are two list implementations: the interface orderedlist which defines what a orderedlist implementation should contain and has a aggregate method which uses a mapper to aggregate all values.

The other implementation is an implementation of the orderedlist interface which has a number of sorted elements and a ordening the sorted elements have. It has all functionalities of a arraylist plus two kinds of searching the position of a specific item (recursive and iterative).

# Snippet 1

## Sample indexOfByRecursiveBinarySearch ()



## Function of code

In this method we recursively seek a searchitem in a sorted list. When we don’t find the item we call findUnsortedItem which searches the same item in the unsorted part of the list.

## Justification why it is on the list

Recursion is one of the subjects that were introduced in the lessons meant for this assignment. So the idea of recursion was new for us and it took some time to figure out how to get the right result back. The method begins in the middle of the sorted part of the list. This middle item is compared to the item you search. If it is the item we seek we return the place of the item. If the compare result is positive it means the item we search is earlier on the list and this method is called with the end changed to the place we checked minus one. If the compare result is negative it means the item we search is later on the list and this method is called with the beginning changed to the place we checked plus one. This way we can find the item we seek without having to go by every single item on the list, as we can scrap half of the range after every compare. If the item is not found in the sorted part of the array another method is called which searches the unsorted part of the list for the item.

Another solution would be to recursively go through each item of the list, but this would almost always take longer, especially if the item we seek is more towards the back of the list.

# Snippet 2

## Sample fromLine()



## Function of code

fromLine makes a purchase out of a line containing the barcode and amount of the product.

## Justification why it is on the list

We decided to add this snippet because it shows we understand techniques like lambda functions. This method splits a string into two parts: the barcode and amount. The barcode is parsed to a long so it can be found in the products list. After that we use a filter to find the product with the barcode we got from the string. If the barcode isn’t found in the products list a new product is made with the unfound barcode. Finallt, a new purchase is made containing the found/made product and the count parsed to a integer.

Another solution would be to make a loop over the products list in order to find the product with the right barcode, but filter is the faster and cleaner solution.

# Snippet 3

## Sample mergePurchasesFromFile



## Function of code

In mergePurchasesFromFile all purchases inside a file are merged with the existing purchases, meaning the count is updated by adding the new sale data.

## Justification why it is on the list

This method shows we completely understand how we create a merge function. In this function a temporary list containing the same ordening as the old ones. Then the old list is sorted so we can efficiently search the list. Then we use our importitemsFromFile method to import all items from the given filepath into the temporary list using the fromline function as a converter. Then we loop over all purchases in our new purchases array. If an product exists (and thus title isn’t null) the item is merged into our purchases list by adding the count of the new product to the count of the old product.

Another solution would be to add the previous counts to the new list and replacing the current purcheses list with the new one, but that would just require a unnecessary extra step.

# Snippet 4

## Sample remove



## Function of code

These remove methods remove an item form the list

## Justification why it is on the list

This snippet shows how the representation invariant is sustained, because when a object was in the sorted section of the list the length of the sorted section is reduced by one. The sorted part of the list is still sorted once you remove this item, but it is one item shorter.

Another solution would be to sort the whole list once you remove an item, but this would cost unnecessary time.

# Snippet 5

## Sample mergePurchasesFromFileRecursively()



## Function of code

This function merges the purchases from the given file and all its subdirectories.

## Justification why it is on the list

This method shows once again that we understand recursion. In this method all purchases within a given file and its subdirectories are added to the purchase list. First, the file matching the filepath is saved. Then If the file is a directory we recursively call this method for every child in this directory. If the file is a purchase file a method is called which can get all purchases from this file into the purchase list.

Another way of doing this would be to iteratively go into the directive but this would be difficult because you have to know the (maximum) number of directives inside each directive and how deep you had to go. You can do this if you know you have a set number of depth but it isn’t reusable and you have to change the code if you change the file structure.

# Snippet 6

## Sample findUnsortedItem()



## Function of code

Finds an item in the unsorted part of the list

## Justification why it is on the list

This method shows we know how to make code according to the DRY (don’t repeat yourself) principle. This piece of code is the same in both the iterative and the recursive finditem methods, so we decided to make a new method out of it so we don’t have to do the same thing twice. In this method every unsorted element in the list is checked to see if it is the same as the item that is sought. When the item is found the index of the item is returned. If the item isn’t found -1 is returned.

An alternative solution would be to do this piece of code in both the iterative and recursive approach of the finditem. But this would just be repeated unnecessary code.

# Snippet 7

## Sample canCompareProduct()



## Function of code

This test tests the equals method of product

## Justification why it is on the list

This method shows we understand how to test our code. In this test we test if it is correct that the equals method only compares the barcode and it doesn’t matter what the title and price are.

Another way of doing this would be to make one product and make the other products out of this initial product but this would make the test way less readable.