PCS4 Assignments – Week 2

Version: 2016, April

[](https://www.google.nl/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRxqFQoTCKelttzKnMkCFYM-DwodxIoJeg&url=https://www.cartoonstock.com/directory/s/sorting_office.asp&bvm=bv.107763241,d.ZWU&psig=AFQjCNEh3N8GNdBaRjQeTLZKZVlgVDxp1g&ust=1448025819337986)

Table of Contents

[\* Assignment 2.1: Delivering packages 2](#_Toc450047082)



[Assignment 2.2: Insert data in a sorted list 6](#_Toc450047083)



[Assignment 2.3: Merging two lists which are already sorted 6](#_Toc450047084)



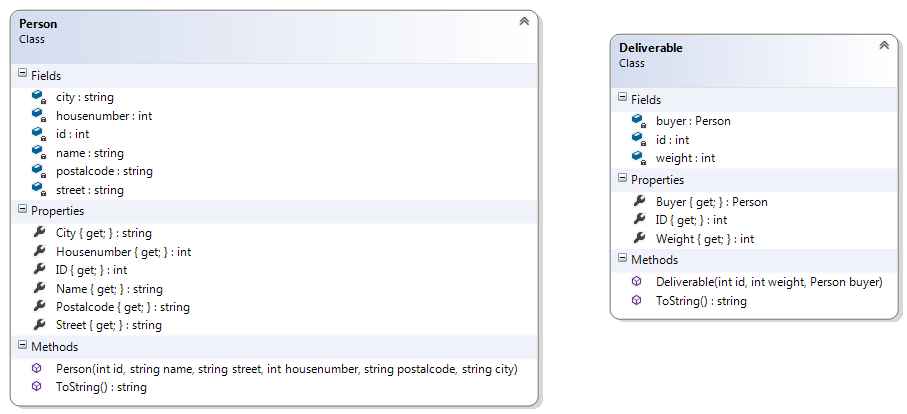
**NOTE**: The assignments marked with an asterisk are the most important ones.   
 The others serve as extra practice.

## **\* Assignment 2.1: Delivering packages**

In this practicum you will practice with **sorting**. The idea is that you implement sorting algorithms by yourself.

A list has a built-in method to sort. If you implement sorting algorithms first by yourself, you will understand the built-in sorting of a list better. So first: do it yourself; next week we will continue sorting in other ways.

A transport company delivers packages (deliverables) to clients. A deliverable has a weight and a person who bought it (the buyer). A person lives on a certain address, where a package should be delivered. In the class diagram below you see an overview of the classes Deliverable and Person. The class Person is implemented in the startup-project. The class Deliverable is implemented except the ToString-method.



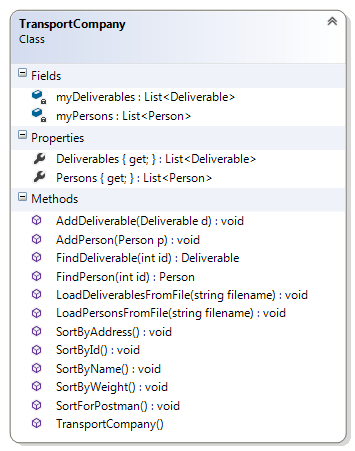
**PART 1:**

Be sure that the ToString method from the Deliverable-class displays besides its id and its weight also some fields of its buyer. So a call to this method should result in something like:

*"Id:10, weight:350, to be delivered at: Sven Kramer - Mainstreet 33 - 5688GE Eindhoven"*

**PART 2:**

The class TransportCompany (class diagram below) holds a list of persons and a list of deliverables. Have a look at it. Some members are implemented and some are not.



There is a folder "data" in the root-folder of the solution. This folder contains two files:   
- persons.txt, containing several persons  
- deliverables.txt, containing several deliverables.

For more information about these files: see appendix A at the end of this assignment.

Implement the method

public void LoadDeliverablesFromFile(String filename)

in the TransportCompany-class.

Implement the “LOAD ALL”-button. This event-handler must read all the information of both files and all this information must be stored in the 2 lists in a TransportCompany-object.

Also add some code to this event-handler to display information about all these deliverables in the listbox.

**PART 3:**

The method SortByWeight() in the TransportCompany-class must sort the elements of the deliverables-list by weight (from a light weight to a heavy weight).

Implement this method SortByWeight().

Implement the event-handler for “sort-by-weight”-button. Clicking this button should result in sorting the deliverables by weight and displaying them in the listbox, so you can test if it works.

**PART 4:**

Now do something similar for the “sort-by-buyer’s name”-button. After clicking this button the deliverables-list must be sorted by name of its buyer. So, first implement the method SortByName() in the TransportCompany-class and use this method in the event-handler of the button. And again , the re-ordered list must be shown in the listbox.

**PART 5:**

Next step is to sort the deliverables-list in ascending order of their id-number.

And again, make the associated button do its work!

**PART 6:**

Now a bit more difficult: about the “sort by address”-button.

It must be possible to sort the elements of the deliverables-list by buyer-address (first alphabetically on street. If two deliverables should be delivered in the same street, then they must be sorted by house-number (from a small house-number to a large house-number). ( In appendix C you see the deliverables in this sorted order.)  
Make it work!

**PART 7:**

And now a very hard one: about the “sort for the postman”-button: difficult, but challenging.

It must be possible to sort the elements of the list for the postman. The deliverables should be first sorted alphabetically by streetname. And for deliverables for the same street: the postman first delivers on one side of the street (the odd house-numbers from small to large) and then the other side of the street (the even house-numbers from large to small).  
( In appendix C you see the deliverables in this sorted order.)

**Appendix A: contents of the files in the data-folder:**The folder "data" (in the root-folder of the solution) contains two files: one for persons and one for deliverables.

For every person the person's file contains 7 lines: name, street, house-number, postal code, city, id-number and a delimiter (line with stars).

The deliverables-file has for every deliverable one line with a unique id, its weight and the index of the person in the person's list.

Those two files describe the following situation:

Persons:

|  |  |  |
| --- | --- | --- |
| id | name | Street, house-number, postal code and city |
| 11 | Mr. Blue | Zaragossastraat 41, 5688DH Eindhoven |
| 22 | Sven Kramer | Mainstreet 33, 5688GE, Eindhoven |
| 33 | Irene Wust | Kerkstraat 20, 5693DE Eindhoven |
| 44 | Naomi van As | Mainstreet 33, 5688GE, Eindhoven |
| 55 | Anouk | Kerkstraat 10, 5693DE Eindhoven |
| 66 | Adele | Kerkstraat 13, 5693DE Eindhoven |
| 77 | Bruno Mars | Kerkstraat 39, 5693DE Eindhoven |
| 88 | RedNex | AAstraat 166, 5688BX, Eindhoven |

Deliverables:

|  |  |  |  |
| --- | --- | --- | --- |
| id | weight | for person with id | Person's name is |
| 1 | 350 | 22 | Sven Kramer |
| 2 | 700 | 66 | Adele |
| 3 | 360 | 88 | RedNex |
| 4 | 360 | 77 | Bruno Mars |
| 5 | 500 | 22 | Sven Kramer |
| 6 | 900 | 33 | Irene Wust |
| 7 | 750 | 44 | Naomi van As |
| 8 | 950 | 11 | Mr. Blue |
| 9 | 100 | 77 | Bruno Mars |
| 10 | 40 | 55 | Anouk |
| 11 | 60 | 55 | Anouk |
| 12 | 30 | 55 | Anouk |
| 13 | 200 | 55 | Anouk |
| 14 | 250 | 44 | Naomi van As |
| 15 | 600 | 33 | Irene Wust |

**Appendix B: Deliverables, after sorting by address, could be:**

Id: 3, weight: 360, should be delivered at: RedNex - AAstraat 166 - 5688BX Eindhoven

Id: 12, weight: 30, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 10, weight: 40, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 13, weight: 200, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 11, weight: 60, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 2, weight: 700, should be delivered at: Adele - Kerkstraat 13 - 5688DE Eindhoven

Id: 15, weight: 600, should be delivered at: Irene Wust - Kerkstraat 20 - 5693DE Eindhoven

Id: 6, weight: 900, should be delivered at: Irene Wust - Kerkstraat 20 - 5693DE Eindhoven

Id: 4, weight: 360, should be delivered at: Bruno Mars - Kerkstraat 39 - 5688DE Eindhoven

Id: 9, weight: 100, should be delivered at: Bruno Mars - Kerkstraat 39 - 5688DE Eindhoven

Id: 14, weight: 250, should be delivered at: Naomi van As - Mainstreet 33 - 5688GE Eindhoven

Id: 5, weight: 500, should be delivered at: Sven Kramer - Mainstreet 33 - 5688GE Eindhoven

Id: 7, weight: 750, should be delivered at: Naomi van As - Mainstreet 33 - 5688GE Eindhoven

Id: 1, weight: 350, should be delivered at: Sven Kramer - Mainstreet 33 - 5688GE Eindhoven

Id: 8, weight: 950, should be delivered at: Mr. Blue - Zaragossastraat 41 - 5688DH Eindhoven

**And deliverables, after sorting for the postman, could be:**

Id: 3, weight: 360, should be delivered at: RedNex - AAstraat 166 - 5688BX Eindhoven

Id: 2, weight: 700, should be delivered at: Adele - Kerkstraat 13 - 5688DE Eindhoven

Id: 4, weight: 360, should be delivered at: Bruno Mars - Kerkstraat 39 - 5688DE Eindhoven

Id: 9, weight: 100, should be delivered at: Bruno Mars - Kerkstraat 39 - 5688DE Eindhoven

Id: 15, weight: 600, should be delivered at: Irene Wust - Kerkstraat 20 - 5693DE Eindhoven

Id: 6, weight: 900, should be delivered at: Irene Wust - Kerkstraat 20 - 5693DE Eindhoven

Id: 12, weight: 30, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 13, weight: 200, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 10, weight: 40, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 11, weight: 60, should be delivered at: Anouk - Kerkstraat 10 - 5693DE Eindhoven

Id: 14, weight: 250, should be delivered at: Naomi van As - Mainstreet 33 - 5688GE Eindhoven

Id: 5, weight: 500, should be delivered at: Sven Kramer - Mainstreet 33 - 5688GE Eindhoven

Id: 7, weight: 750, should be delivered at: Naomi van As - Mainstreet 33 - 5688GE Eindhoven

Id: 1, weight: 350, should be delivered at: Sven Kramer - Mainstreet 33 - 5688GE Eindhoven

Id: 8, weight: 950, should be delivered at: Mr. Blue - Zaragossastraat 41 - 5688DH Eindhoven

Assignment 2.2: Insert data in a sorted list – Real Time Sorting

Can you change the PART 2 from the previous assignment (the reading of the files) is such a way that the deliverables-list gets sorted by weight while reading the data. So, each next deliverable that is being read from the file, must be inserted in the list immediately at the right place. (So you first start with the first deliverable at place 0, then the second one, which must be placed before or after the first one dependent on their weights. And so on.)



Assignment 2.3: Merging two lists which are already sorted  
A transporting company made a deal with two webshops. At the end of the day the webshops sends their list of deliverables to the transporting company. The webshops already sorted their lists by address.

The transporting company wants to merge the lists to a final list in such a way, that this merged final list is also sorted.

One way of merging is adding all separate elements of every original list to the final list and then sort this big final list. This way of working is not very efficient. A better way (but still not efficient) is based on:

1. Look at the first elements of both original list. The element, that should be first in the merged list will be added to the merged list and removed from its original list.



1. Repeat the former step until all elements of the two original lists are in the merged list.



This idea is good, but not efficient. By removing an element from a list all other elements are shifted back in the list. A better way is to not remove the elements from the original lists, but have two indices, say index1 and index2, one for every original list. All elements in the first original list with an index lower than index1 are already placed in the merged list and the same for index2. The algorithm will be something like:

1. Look at the elements on position index1 and index2 in the original lists. Compare these elements. The element that should be first will be added to the merged list and the corresponding index will be incremented by 1.



1. Repeat the former step until all elements of the two original lists are in the merged list.