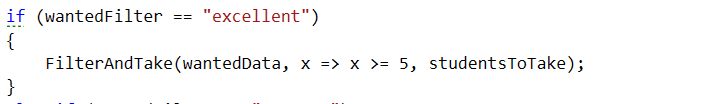
# Lab: LINQ

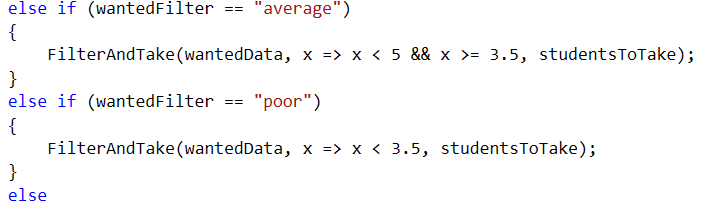
In the current piece we will see how we can implement some of the operations with less code thanks to the LINQ operators. In this way we will improve the readability of the project and that’s something you are obliged to do.

## Change Predicate Methods with Lambda Expressions

The first thing we are going to change though is not related to LINQ. All we want to change here is in the **RepositoryFilters** in the **public FilterAndTake**. In the previous piece we created 3 methods that filter the given data. However, we have the possibility to delete the 15 rows that these methods take in the code. In the first method, where the **wantedFilter** is excellent we can easily express it through a predicate using lambda. We have only one input parameter so our statement in the call of the **private** **FilterAndTake** should look something like this:



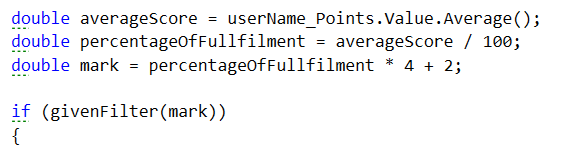
Next up are the average and the poor filter, however they are pretty much the same as the one above. For this reason we are not going to discuss them any further and just change the calls of the filter methods with the appropriate lambda expression:



Now we can easily delete the three methods from the repository filters class.

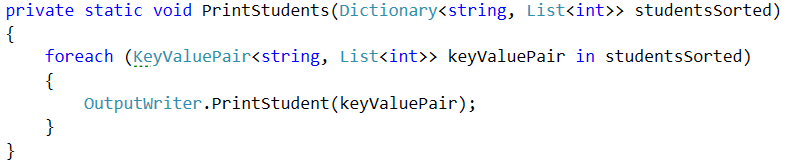
## Use LINQ Average Instead of a Custom One

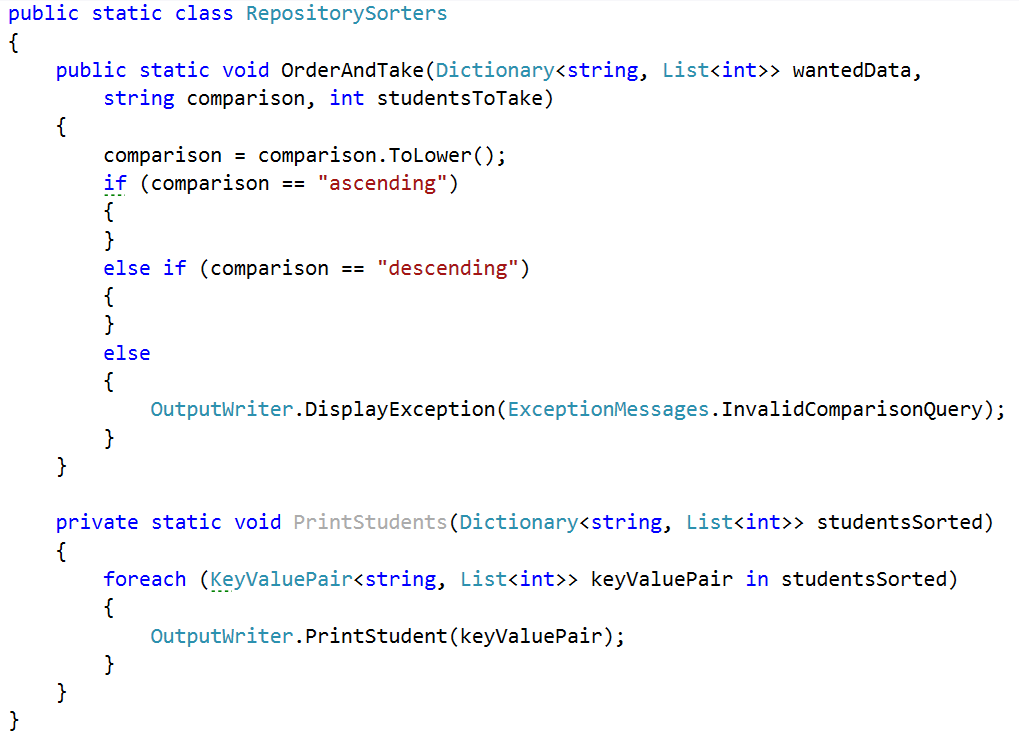
Another piece we can get rid of is the Average method and replace it with the operator that comes from the LINQ. In the **private FilterAndTake** we should change the **averageMark’s** name to **averageScore** and its value equal to the output of the just mentioned above operator. Next thing we need to make is a new variable called **percentageOfFullfillments** equal to the average score devided by the maximal score on a task which is 100. Finally, we should make one last variable that is the actual mark and it is equal to the percentage of fulfillment multiplied by 4 and summed with 2 after that. Here is how the **private** **FilterAndTake** method should look:



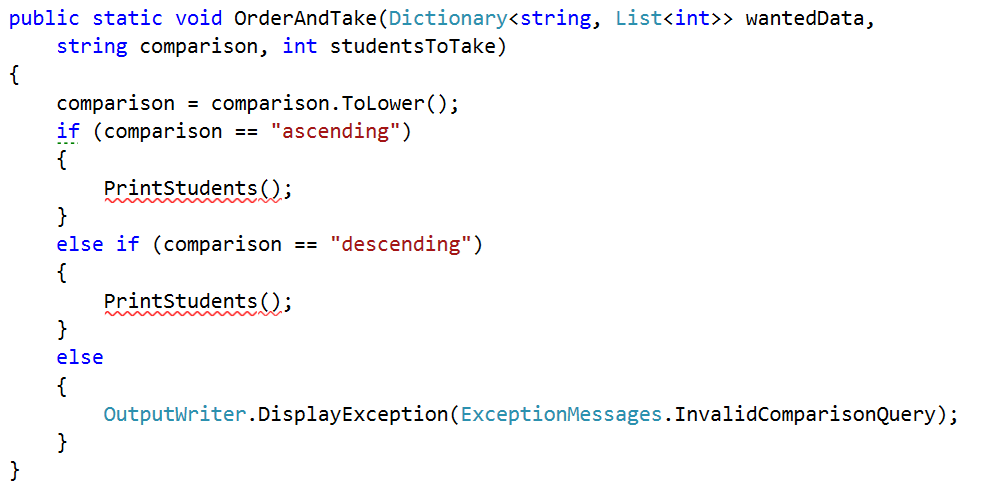
## Changing Structure in Repository Sorters

I guess you wouldn’t be quite happy to hear that we’ve done the sorting the hard way. But now we can appreciate LINQ’s easiness and replace it with the easy and more readable way. This means that we can delete the **private** **OrderAndTake** method but just before that let’s extract only one method that we can reuse for our functionality. The new method for printing the sorted students declaration’s and implementation should look like this:

  
  
Now you can easily delete all the methods except the **public OrderAndTake** and the one we just extracted. So after the deletions of all occurrences, the class should look like this:

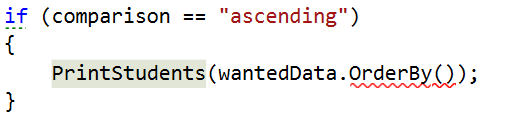


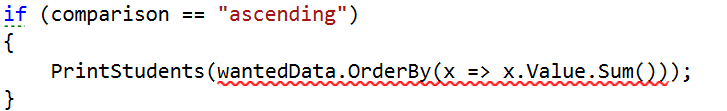
Whether the wanted filter is ascending or descending we want to call the **PrintStudents** method passing an already sorted dictionary.

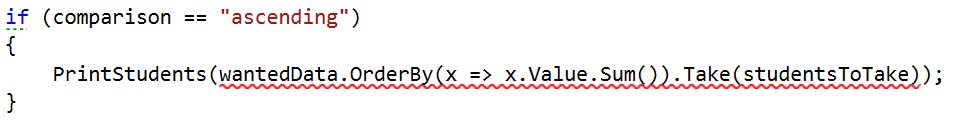


First we are going to implement the ascending comparison and then we’ll need to only change one word and copy the rest of the sorting so that it works for descending as well.

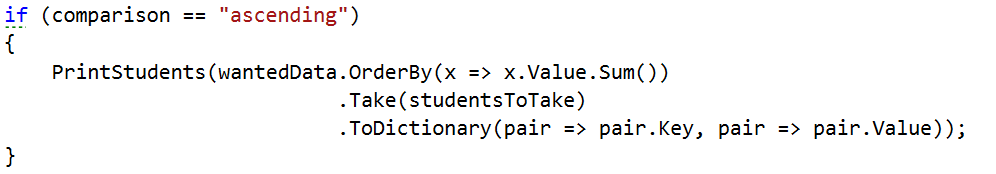
Since we have a method for ordering a collection from LINQ, we are going to use it out of the box.

  
  
What the function wants is the criteria which to use for sorting and in our case we’ll pass to it the sum of the scores on the tasks.

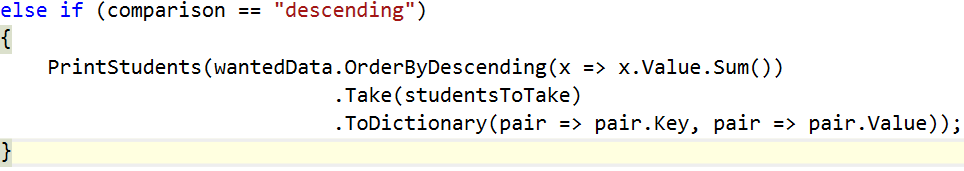
  
  
Now we should take only the needed number of results:



Finally, we should convert it to a dictionary since after ordering it’s a collection of different type:



As we said we will copy this piece and only change the **OdrerBy** with a **OrderByDescending**.

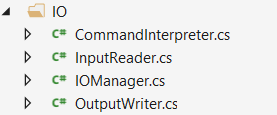


Now you can test the functionality of the filters and sorters and see if they still work.

## Creating folder structure

There are no more places where we can use LINQ so I suggest we use the current piece to put the structure of our project in order at least a little bit.

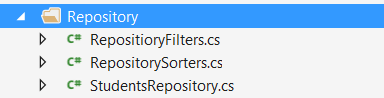
Let’s create 4 folders in the current project called **IO**, **Judge**, **Repository**, **Static** **data**. In the IO folder put all the following things:



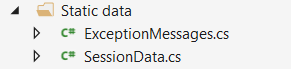
In the Judge folder as you can imagine, we’ll put the Tester class:



In the repositories folder we’ll put everything related to the repository:



And finally in the Static data we should put the ExceptionMessages and the SessionData.



Well done! You have completed the tutorial for your BashSoft a.k.a DIY Judge. Though you may feel free to continue explore and experiment with adding new features.