# Lab: Stream API

This document defines the lab overview for the ["Java Advanced" course @ Software University](https://softuni.bg/trainings/1377/advanced-java-may-2016). Please submit your solutions (source code) of all below described problems at the end of the course at [softuni.bg](https://softuni.bg/trainings/1377/advanced-java-may-2016).

# Introduction

In the current piece we are going to jump from one place to another in our code and see if we can do some of the things with less code. This is our goal, because that can improve the readability of the project and that’s something you are obliged to do.

## Using Stream API Aggregate Operations

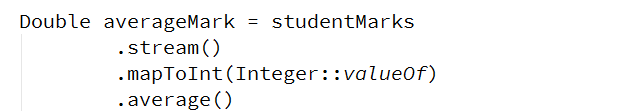
What you can do now is head to the RepositoryFilters class and delete the getStudentAverageGrade() method so that we can replace it with the one that comes from the Stream Api. Next look in the printFilteredStudents() and find average mark variable:



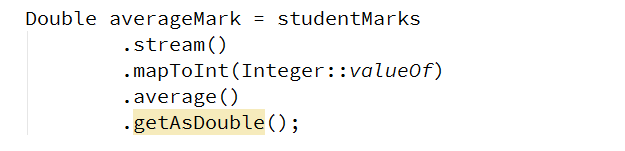
We can now just use the built in methods to get the average of a collection and then calculate the rest. You should already know that to start a streaming pipeline you need to use the stream() method on a collection. We can use the studnetsMarks ArrayList:



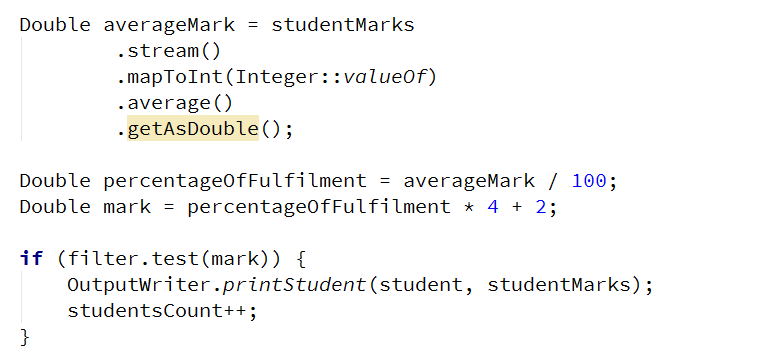
Next, if we want to use the average() operation we must map all values in the stream to integers, so we can use the .mapToInt() operatins and .average() right after that:



As you can see we want to save the result of this as a Double variable, so we can end this stream with the getAsDouble() and we should get a result like this:



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 everything should look:



## Using Stream API Custom Sorting

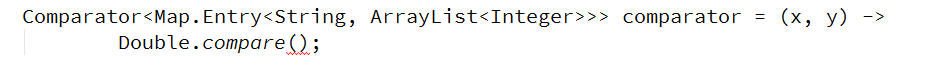
I know you may not like that but now we’ve done the sorting the hard way, we can easily replace it with the easy and more readable way. For that reason, we are going to delete most of the code we typed the last time in the RepositorySorters and replace it with a more compact and generally more readable version. For starters you can safely delete the get totalScore() and createComparator() methods:



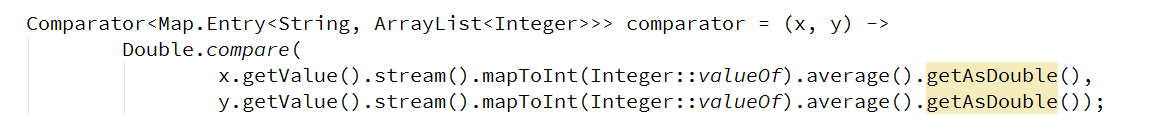
Then you can start to replace code. We are still going to use a comparator, that we will create, but we will be using only one, and we are going to define it just after the assign operator using a single lambda expression. Let’s start with the parameters. Since for a comparing operation we need two parameters we will use a lambda that takes to parametes:



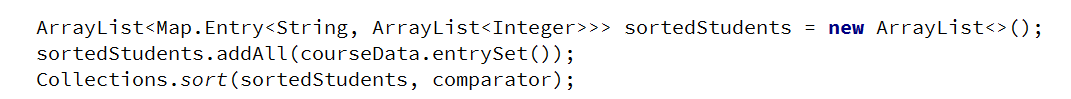
Next, since we are going to compare the averages of two different ArrayLists, which will be reduced to doubles, we can use the Double.compare() method:



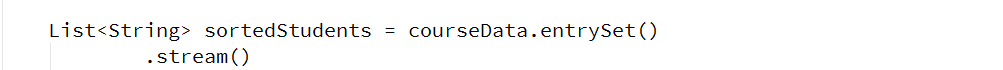
And inside the brackets we need to type the values that will be compared. These are the averages of the two ArrayLists, which we can again get with the help of the Stream API:



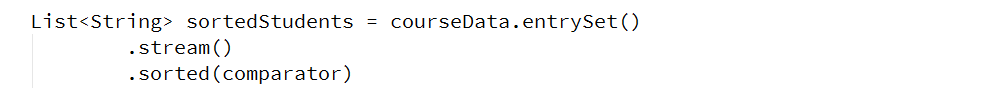
After we are ready with this, we can again safely delete the next few lines of code:



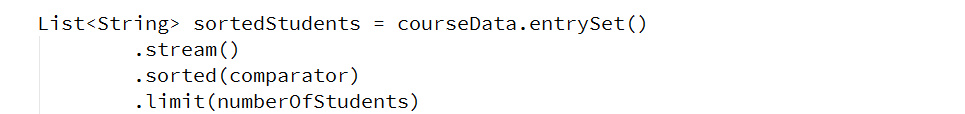
And replace them with a single query in which we will stream the entry set of the course data:



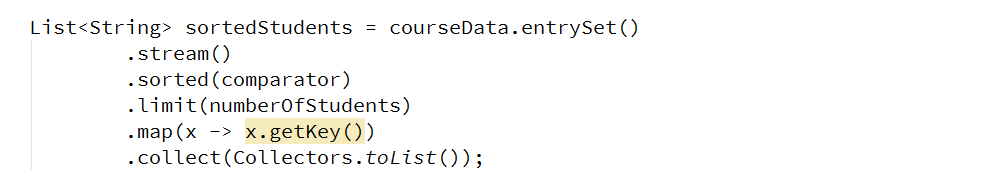
Next we want to sort this stream with the comparator that we have just created:



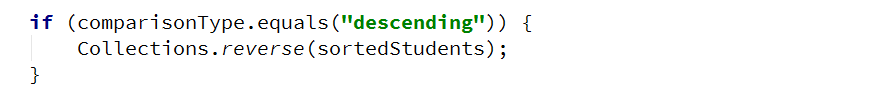
Remember that we had to sort out how much students to take if we have additional condition in the user input? We can do this too with a single functional method:



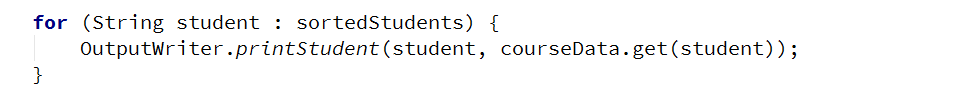
The only thing left to do is to map (transform) every object in the stream to a data type which can be useful to us. We can use Strings because we can then easily get every student by his username from the HashMap, which in turn will get them in a sorted order. And after we map the stream object we need to collect them in a collection:



Probably you are now wondering what happened with the second type of sorting, namely sort in descending order. Well as we now have our students ordered in a List, we can just reverse it if we need to:

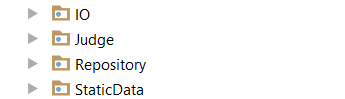


And then just print the results:

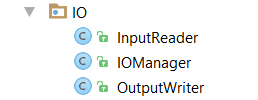


## Creating Folder Structure

There are no more things we can change using the Stream API so I suggest we use the current piece so that we can at least order the structure of our project at least a little bit.

We can make 4 packages in the current project called IO, Judge, Repository and Static data. 

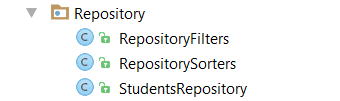
In the IO folder we can put all the following things:



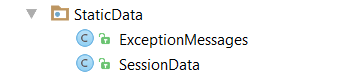
In the Judge folder as you can imagine, we’ll put the Tester class:



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



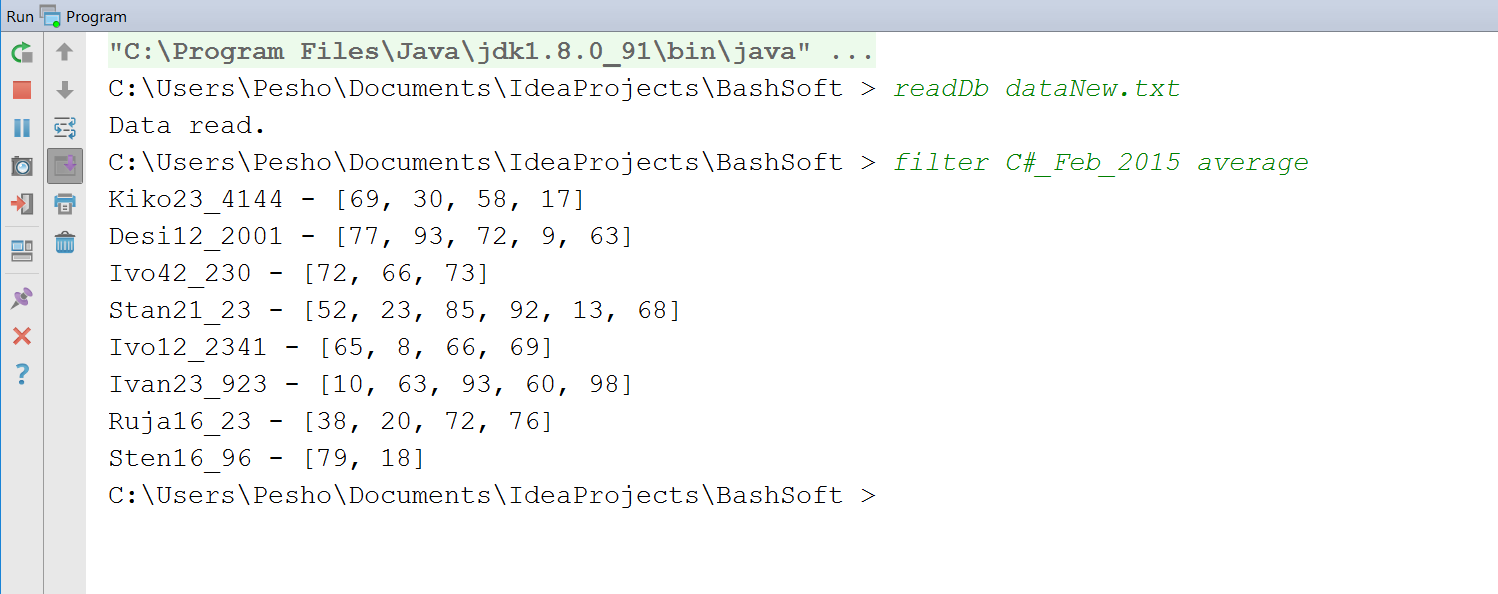
And finally in the static data we should put the ExceptionMessages and the SessionData.



You can leave everything else in the root package of your project.

## Test Your Code

Everything should be ok and we are ready to start reading from the input. Next thing to do is read the dataNew.txt from where you’ve saved it and apply one sorting and one filtering.



You can now play with and test the rest of the functionality that we have just implemented!

Congratulations! You’ve successfully completed the lab exercise for Stream API.