# Exercises: Stream API

This document defines the exercises for ["Java Advanced" course @ Software University](https://softuni.bg/courses/java-advanced). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/).

# Part I: Initial Setup

You are given a text file **"StudentData.txt"** with data about students.

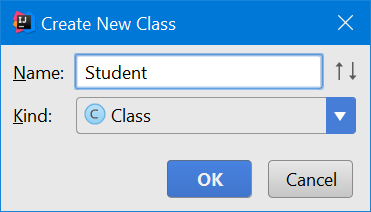
You need to create a **class Student**, then **read the file** and **create** as many instances of the class as there are **students** in the file.

## Create Student Class

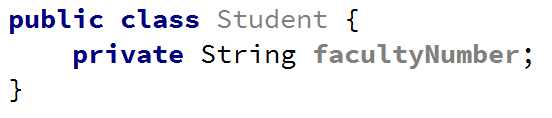
The class should have these properties:

* **Faculty Number - String**
* **First Name - String**
* **Last Name - String**
* **Email - String**
* **Age - Integer**
* **Group - Integer**
* **Grades - List<Integer>**
* **Phone - String**

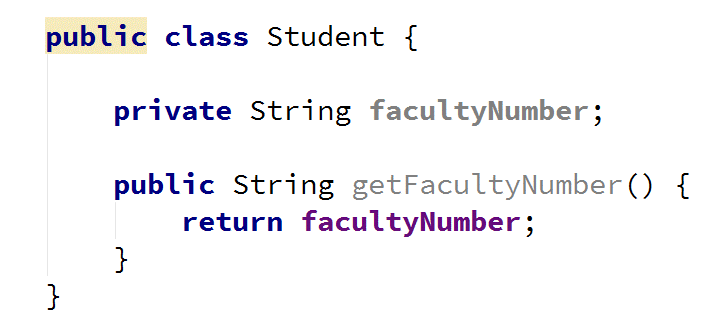
First of all, **create a class** in your project:



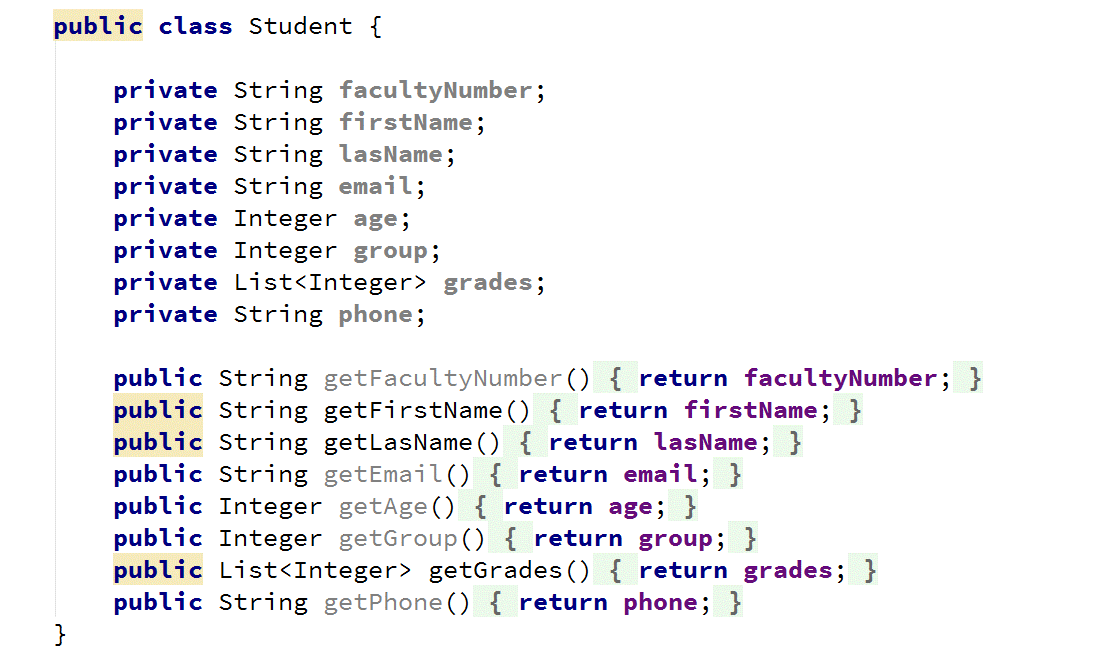
Create a private String field facultyNumber:



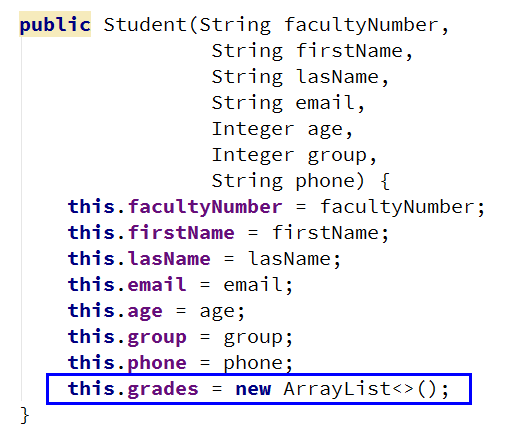
Create a **getter** for the facultyNumber field:



Now, **do the same for all other properties**:



When you are done creating fields and getters for all properties, **create a constructor for the class**, but be mindful about the List:

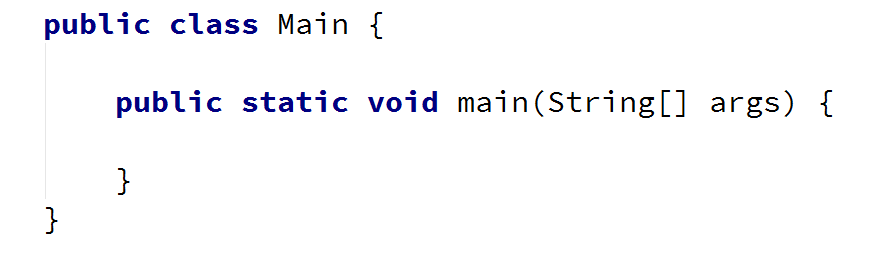


## Create Student Objects

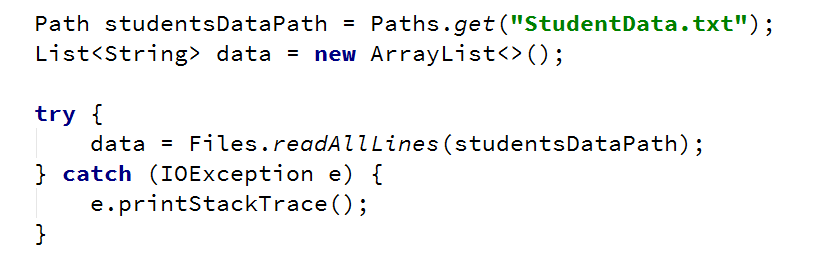
Create another class for the main method:



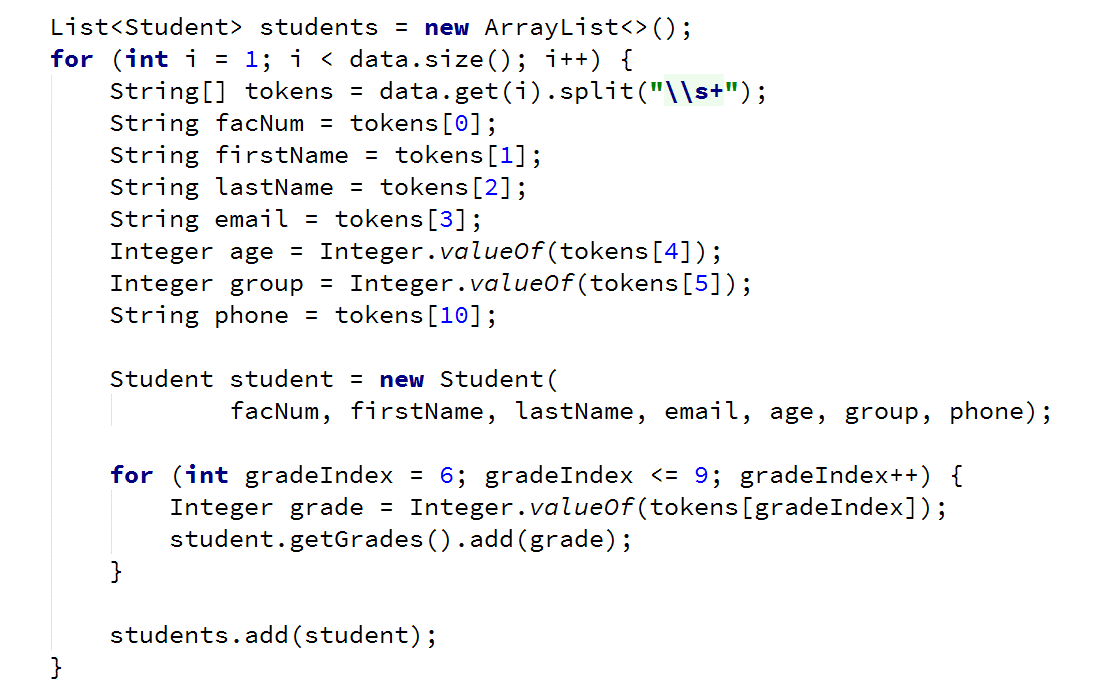
Crete your main method:



Now read the contents of the file, making sure the path is valid:



Make a List<Student>, iterate over the lines that you have just read and create an object for each line, except the first (which holds the header) and add them to the List<Student>:



You are now set. You can start practicing your Stream API skills.

# Part II: Exercises

## Students by Group

Print all **students from group number 2**, using a Stream API query. Order the students by **first name**.

Submit your output in Judge as **Plain Text**.

### Examples

|  |
| --- |
| **Sample Output** |
| Albert Cruz  Amanda Hernandez  Arthur James  … |

### Solution

* Create a stream from the list of students that you have created in the previous section:



* Filter the students by their group



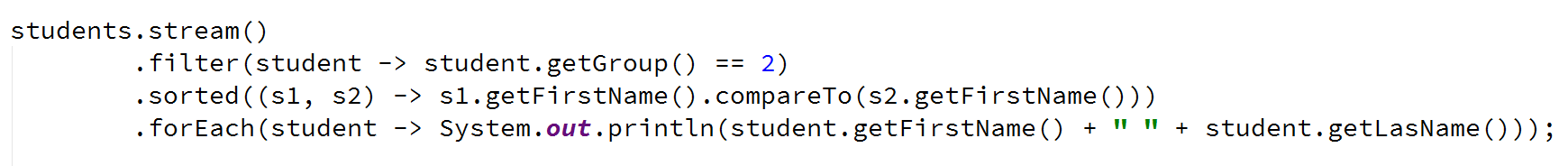
* Sort the filtered students by their first name



* Print each student's first and last name



* In the end, you should have a query that looks like this:



* Run you program and copy the results directly to Judge:



## Students by First and Last Name

Print all students whose **first name is before their last name lexicographically**, using a Stream query.

Print students in order of appearance.

### Examples

|  |
| --- |
| **Sample Output** |
| Aaron Gibson  Antonio Gonzalez  Cheryl Gray  Craig King  … |

### Hints

* Just use the filter function and compareTo()String method:



## Students by Age

Write a stream query that finds all **students with age in the range** **[18, 24]**.

The query should return the **first name**, **last name** and **age**.

Print students in order of appearance.

### Examples

|  |
| --- |
| **Sample Output** |
| Mildred Hansen 22  Craig Ellis 21  Andrea Harper 23  Richard Edwards 20  … |

## Sort Students

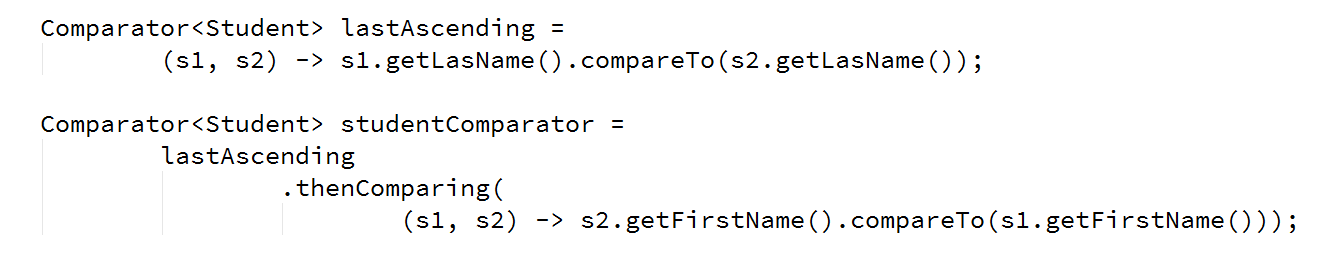
Sort the students first by **last name** in **ascending** order and then by **first name** in **descending** order.

### Examples

|  |
| --- |
| **Sample Output** |
| Roger Adams  William Alexander  Charles Alvarez  … |

### Hints

* You need to create a second comparator with .thenComparing():



* Pass the comparator to the sorted() method

## Filter Students by Email Domain

Print all students that have email **@gmail.com**.

Print students in order of appearance.

### Examples

|  |
| --- |
| **Sample Output** |
| Mildred Hansen mhansen4@gmail.com  Marie Simpson msimpson13@gmail.com  Frances Peters fpeters1p@gmail.com  … |

## Filter Students by Phone

Filter all students with phones in Sofia (starting with **02** / **+3592**). Print their first and last names along with their phone.

Print students in order of appearance.

### Examples

|  |
| --- |
| **Sample Output** |
| Ashley Lee +35920412542  Michelle Daniels +35922450998  Charles Alvarez 02913942  … |

## Excellent Students

Print all students that have **at least one mark (6)**.

Print the students in order of appearance, print their marks sorted in descending order.

### Examples

|  |
| --- |
| **Sample Output** |
| Sara Mills 6 5 4 3  Daniel Carter 6 4 3 2  Aaron Gibson 6 5 3 3  Mildred Hansen 6 5 5 2  … |

## Weak Students

Write a similar program to the previous one to extract the **students with at least 2 marks under or equal to (3)**.

Sort students from the weakest to the strongest. Print their marks in ascending order.

### Examples

|  |
| --- |
| **Sample Output** |
| Jeffrey Gibson 2 2 3 3  Daniel Ferguson 2 2 3 3  Lawrence Welch 2 2 3 4  Michelle Daniels 2 3 3 3  … |

### Hints

## \* Students by Enrollment Year

Extract and print all **students by year of enrollment**.

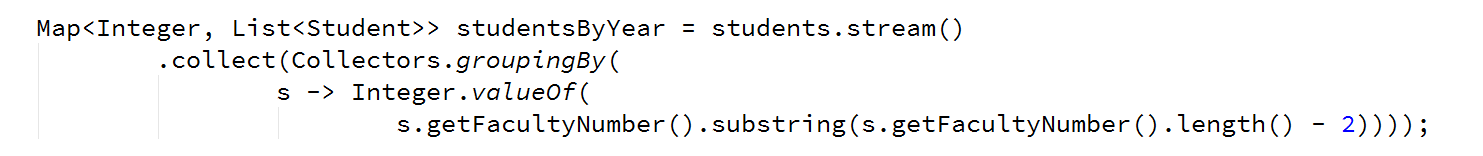
**Sort years** in ascending order. **Sort all students** in a given year by **Full Name (First Name + Last Name)**.

### Examples

|  |
| --- |
| **Sample Output** |
| 2011:  -- Aaron Gibson  -- Anne Freeman  -- Beverly Clark  …  2012:  -- Catherine Diaz  -- Craig Flores  …  2013:  … |

### Hints

* Collect the students by using groupingBy() of the Collectors class   
  in a Map<Integer, List<Student>>:



* Sort the map by year and then print each year's sorted students

## \* Little John

As you probably know Little John is the right hand of the famous English hero - Robin Hood.A little known fact isthat Little John can't handle Math very well. Before Robin Hood left to see Marry Ann, he asked John to **count** his hay of arrowsand send him an **encrypted** message containing thearrow's count**.** The message should be encrypted since it can be intercepted by the Nottingham’s evil Sheriff. Your task is to help Little John before it is too late (0.10 sec).

You are given **4 input** strings(hay). Those strings **may or may not** contain arrows. The arrows can be of different type as follows:

* ">----->" – a small arrow
* ">>----->" – a medium arrow
* ">>>----->>" – a large arrow

Note that the **body** of each arrow will always be **5 dashes long**. The **difference** between the arrows is in their **tip** and **tail**. The given 3 types are the only ones you should count, the **rest should be ignored** (Robin Hood does not like them). You should start searching the hays **from the largest** arrow type down **to the smallest** arrow type.

After you find the **count** of each arrow type you should **concatenate** them into one number in order: small,medium, large arrow(even if the arrow count is 0). Then you **convert** the number in **binary** representation, **reverse** it and **concatenate it again** with the initial binary representation of the number. You **convert** the final binary number again **back to decimal**. This is the encrypted message you should send to Robin Hood.

### Input

The input will be read from the console. The **data** will be received from 4 input **lines containing strings**.

### Output

The output should be a decimal number, representing the encrypted count of arrows.

### Constraints

* The input strings will contain any ASCII character.
* Allowed working time: 0.1 seconds. Allowed memory: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| >>>----->>abc>>>----->>  >>>----->>  >----->s  >>-----> | 14535  *The count is: 1 small, 1 medium and 3 large arrows*  *113(dec) = 1110001(bin) -> reversed is 1000111(bin)*  *11100011000111(bin) = 14535(dec)* |

## \* Office Stuff

You are given a sequence of **n** companies in format |**<company> - <amount> - <product>|**. Example:

* |SoftUni - 600 - paper|
* |Vivacom - 600 - pen|
* |XS - 20 - chair|
* |Vivacom - 200 - chair|
* |SoftUni - 40 - chair|
* |XS - 40 - chair|
* |SoftUni - 1 - printer|

Write a program that prints **all companies** in **alphabetical** order. For each company print the product type and their aggregated ordered amounts. Order the products by **order of appearance**. **Print** the result in the following format: **<company>: <product>-<amount>, <product>-<amount>,…** For the orders above the output should be:

* SoftUni: paper-600, chair-40, printer-1
* Vivacom: pen-600, chair-200
* XS: chair-60

### Input

The input comes from the console. At the first line the number **n** stays alone. At the next **n** lines, we have **n** orders in format |**<company> - <amount> - <product>|**.

The input data will always be valid and in the format described. There is no need to check it explicitly.

### Output

Print **one line for each company**. Company lines should be ordered in **alphabetical** **order**. For each company print the **products** ordered by this company in **order** of **appearance**, along with the total amount for the given product. Each line should be in format **<company>: <product>-<amount>, <product>-<amount>, … <product>-<amount>**

### Constraints

* The **count** of the lines **n** will be in the range [1 … 100].
* The **<company>** and **<product>** will consist of only of **Latin characters**, with length of [1 … 20].
* The **<amount>** will be an integer number in the range [1 … 1000].
* Time limit: 0.1 sec. Memory limit: 16 MB.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 7  |SoftUni - 600 - paper|  |Vivacom - 600 - pen|  |XS - 20 - chair|  |Vivacom - 200 - chair|  |SoftUni - 40 - chair|  |XS - 40 - chair|  |SoftUni - 1 - printer| | SoftUni: paper-600, chair-40, printer-1  Vivacom: pen-600, chair-200  XS: chair-60 |
| 5  |SoftUni - 200 - desk|  |SoftUni - 40 - PC|  |SoftUni - 200 - desk|  |SoftUni - 600 - paper|  |SoftUni - 600 - textbook| | SoftUni: desk-400, PC-40, paper-600, textbook-600 |

## \*\* Export to Excel

Write a program to create an Excel file like the one below using an external library. Such as [Apache POI](http://howtodoinjava.com/apache-commons/readingwriting-excel-files-in-java-poi-tutorial/) for Java.

You are given as **input** course data about **1000 students** in a **.txt** file (tab-separated values). Each line in the input holds **ID**, **first name**, **last name**, **email**, **gender**, **student type**, **exam result**, **homework sent**, **homework** **evaluated**, **teamwork score**, **attendances count**, **bonus**.

