# Lab: Java-OOP-Fundamentals

# Problems for exercises and homework for the

# ["DB Advanced" course @ SoftUni](https://softuni.bg/courses/databases-advanced-hibernate).

# Part I: Stream<T> and Types of Streams

## Take Two

Read a **sequence of integers**, given on a single line separated by a space.

Finds all **unique** elements, such that **10 ≤ n ≤ 20** and print only **the first 2 elements**.

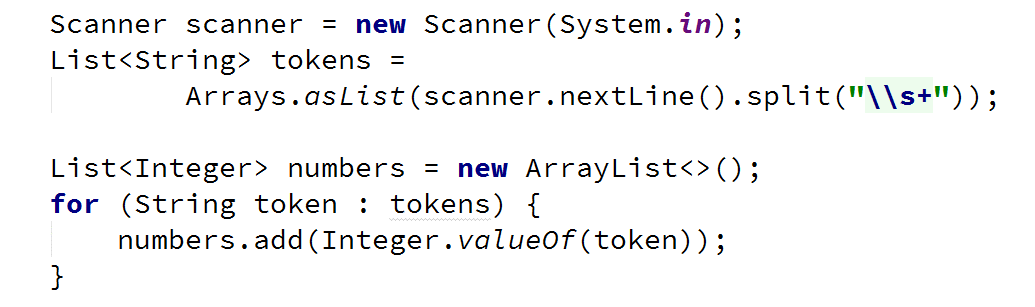
If there are **fewer than 2 elements**, print as much as there are. If there are **no elements**, print nothing.

### Examples

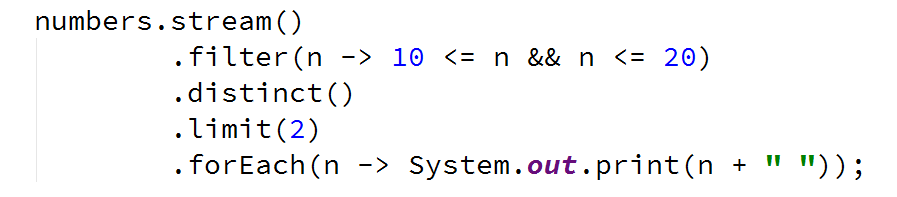
|  |  |
| --- | --- |
| **Input** | **Output** |
| 15 2 15 14 12 | 15 14 |
| 17 -2 3 | 17 |
| -2 3 | (*no output)* |

### Hints

* Read the input using a Scanner or a BufferedReader and parse the strings to a list of numbers:



* Filter the numbers with filter(), take the unique ones with distinct(), take only two from the stream with limit() and iterate over them while printing with forEach():



## Upper Strings

Read a sequence of strings, given on a single line separated with a space.

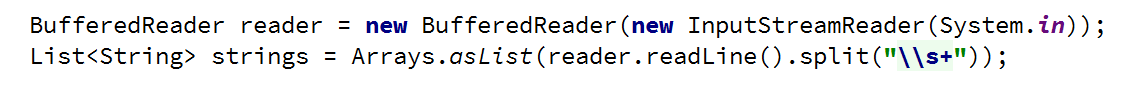
Map each to upper case and print them, using the Stream API.

### Examples

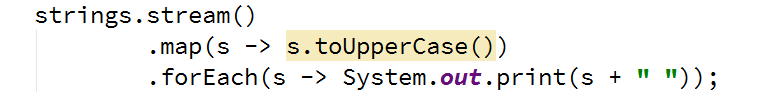
|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Gosho Stefan | PESHO GOSHO STEFAN |
| Soft Uni Rocks | SOFT UNI ROCKS |
| *(empty line)* | *(no output)* |

### Hints

* Read the input using a Scanner or a BufferedReader into a list of strings List<String>:



* Call a stream over the list and map every element to upper case. Iterate over the stream and print the result:



## First Name

Read a **sequence of names**, given on a single line, separated by a space.

Read a letter.

Of the names that start with the letter find the first name by lexicographical order

If there is **no name** that conforms to the requirement, **print "No match"**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Rado Plamen Pesho  p | Pesho |
| Plamen Gosho Rado  s | No match |

### Hints

* Make sure you are comparing letter with the same casing (lower or upper)
* Use filter(), sorted() and findFirst()
* Use Optional<T>

## Average of Doubles

Read a **sequence of double numbers**, given on a single line, separated by a space.

Find the **average of all elements**, using the Stream API.

**Round** the output to the second digit after the decimal separator.

If there are **no numbers** in the sequence, **print "No match"**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 4 5 6 | 4.50 |
| 3.14 5.2 6.18 | 4.84 |
| *(empty list)* | No match |

### Hints

* Use a primitive stream DoubleStream
* Use OptionalDouble
* Make sure to filter empty strings before transforming the stream

# Part II: Types of Operations

## Min Even Number

Read a sequence of numbers, given on a single line, separated by a space.

Find the smallest number of all even numbers, using the Stream API.

If there are **no numbers** in the sequence, **print "No match"**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 2 3 4 5 6 | 2.00 |
| 3.14 -2.00 1.33 | -2.00 |
| *(empty list)* | No match |

### Hints

* Use map function to map the objects to Double
* Make sure to filter empty strings
* Filter the even numbers
* Get the smallest number using Double.compare(x1, x2)

## Find and Sum Integers

Read a sequence of elements, given on a single line, separated by a space.

Filter all elements that are integers and calculate their sum, using the Stream API.

If there are **no numbers** in the sequence, **print "No match"**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Sum 3 and 4 | 7 |
| Sum -3 and -4 | -7 |
| Sum three and four | No match |

### Hints

* Use filter 🡪 map 🡪 reduce pattern
* Check if element's char at index 0 is a sign (+ or -)
* Check if all else element's chars are digits

## Bounded Numbers

On the first line, read two numbers, a **lower** and **an upper bound**, separated by a space.

On the second line, read a sequence of numbers, separated by a space.

Print all numbers, such that [lower bound] ≤ n ≤ [upper bound].

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 7  1 2 3 4 5 6 7 8 9 | 5 6 7 |
| 7 5  9 5 7 2 6 8 | 5 7 6 |
| 3 4  5 6 7 8 | *(no output)* |

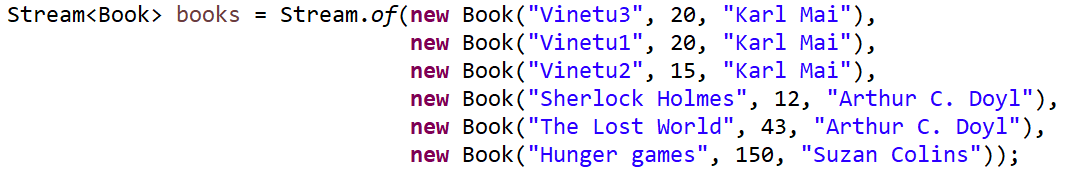
### Hints

* Use collect(Collectors.toList())

## \*Book Store

Create class **Book** with **name:String**, **author:String** and **price:double** fields

Create a hardcoded Stream of Books :



### For each distinct author find the sum of prices of his books

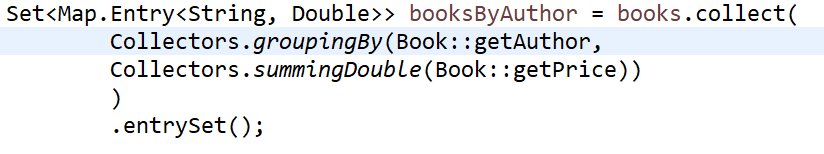
### Sort result Map by decreasing sum of prices, then by author name alphabetically

### Examples

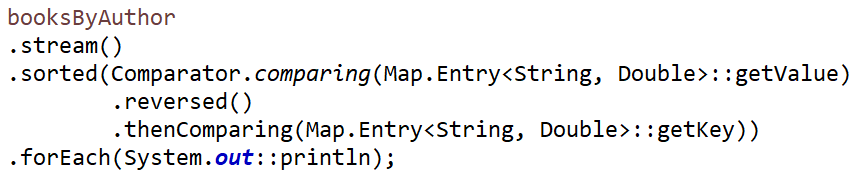
|  |
| --- |
| **Output** |
| Suzan Colins=150.0  Arthur C. Doyl=55.0  Karl Mai=55.0 |

### Hints

* Transform the Stream of Books into Map<String, Double> by grouping by author name and summing book prices. Then Work with the Entry Set of the Map :



* Then sort the Set of Map.Entry by Entry’s value in decreasing order first and then by Entry’s key alphabetically. At the end print the result :



## \*Print File

Write a program which takes a path to a file. The path should be **full path**, not **relative to the project**.

The program should read the file and print its content to the console.

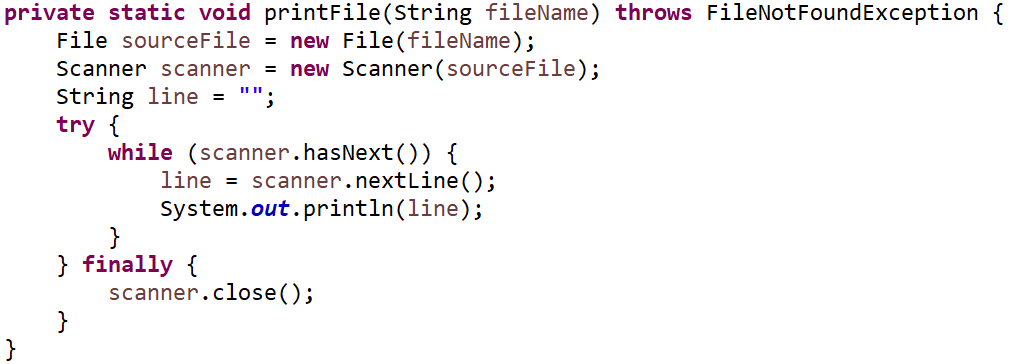
Make sure the program is closing all resources it uses – all file descriptors should be released (closed).

Inform the user in case of file printed successfully.

Inform the user for the case of reading operation failed. Ask the user to try again with a new file it is not found.

### Hints

Create method **void printFile(String filePath) throws FileNotFoundException**. User Scanner object to read from a file :



Use **printFile** method in **main** method. Do not let **FileNotFoundException** to pass through main method:

