# Exercises: Methods

This document defines the exercises for ["Java OOP Basics" course @ Software University](https://softuni.bg/trainings/1375/java-basics-oop-june-2016). Please submit your solutions (source code) of all below described problems in [Judge](https://judge.softuni.bg/).

## Method Says Hello!

You will receive the person name as an input. Write a class **Person** that only has a name and a **method**. The method should describe a greeting by the person, returning a String "<Person name> says Hello!". Print the result of the method call.

### Note

Add the following code to your main method and submit it to Judge.

|  |
| --- |
| Field[] fields = Person.**class**.getDeclaredFields(); Method[] methods = Class.*forName*(**"Person"**).getDeclaredMethods(); **if** (fields.**length** != 1 || methods.**length** != 1) {  **throw new** ClassFormatError(); } |

If you’ve defined the class correctly, the test should pass.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter | Peter says "Hello"! |

## Oldest Family Member

Create class **Person** with fields **name** and **age**. Create a class **Family**. The class should have **list of people**, method for adding members (**void addMember(Person member)**) and a method returning the oldest family member(**Person getOldestMember())**. Write a program that reads name and age for **N** people and **adds them to the family**. Then **print** the **name** and **age** of the oldest member.

### Note

Add the following code to your main method and submit it to Judge.

|  |
| --- |
| Method getOldestMethod = Class.*forName*(**"Family"**).getMethod(**"getOldestMember"**); Method addMemberMethod =Class.*forName*(**"Family"**).getMethod(**"addFamilyMember"**,Person.**class**); |

If you’ve defined the class correctly, the test should pass.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  Pesho 3  Gosho 4  Annie 5 | Annie 5 |  | 5  Steve 10  Christopher 15  Annie 4  Ivan 35  Maria 34 | Ivan 35 |

## Last Digit Name

Write a class **Number** that will hold an integer number. Write a **method** in the class that returns the **English name** of the last digit of the given number. Write a program that reads an integer and prints the returned value from this method.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1024 | four |  | 512 | two |

## Number in Reversed Order

Write a class **DecimalNumber** that has a method that **prints all its digits** in **reversed order**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 256 | 652 |  | 1.12 | 21.1 |

## Fibonacci Numbers

Define a class **Fibonacci**. It should keep a **list** of all **Fibonacci numbers** starting from **0, 1 until Nth** number in the sequence. Write a **method** in the Fibonacci class that receives as parameters **start position** and **end position** and returns **part of the sequence** starting from **start position (inclusive)** until **end position (exclusive)**.   
**ArrayList<Long> getNumbersInRange(int startPosition, int endPosition)**.

Write a program that reads **start position** and **end position** and prints the **Fibonacci numbers in that range**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0  6 | 0, 1, 1, 2, 3, 5 |
| 6  7 | 8 |
| 17  20 | 1597, 2584, 4181 |

## Prime Checker

Create a class **Number**. It should consist of an Integer and a Boolean. The integer is the actual value of the Number instance itself and the Boolean is representing – is it prime or not. They should be passed as parameters to the constructor (**Note there could be a case in which a passed Boolean value does not match**). The class should have a functionality to return the values of the Integer and the Boolean. Write another method whose goal is to return the next prime number as **new instance of the class**.

**You will be given an input** – the integer **“n”** of the class. Your task is to print on the console the next prime number and the Boolean value of the **current instance**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0 | 1, true |
| 1 | 2, true |
| 2 | 3, true |
| 14 | 17, false |

### Note

Add the following code to your main method and submit it to Judge.

|  |
| --- |
| Field[] fields = Number.**class**.getDeclaredFields();  ***List***<Field> filedsDeclared = Arrays.*stream*(fields)  .filter(f -> f.getName().contains(**"prime"**) || f.getName().contains(**"number"**))  .collect(Collectors.*toList*());  ***List***<Constructor<?>> constructors = Arrays.*stream*(Number.**class**.getDeclaredConstructors())  .filter(c -> c.getParameterCount() > 1)  .collect(Collectors.*toList*());  **if** (filedsDeclared.size() <= 1 || constructors.size() < 1) {  **throw new** ClassFormatException(); } |

If you’ve defined the class correctly, the test should pass.

## Immutable List

Create a class **ImmutableList**. It should consist of a collection of integers and a function to return them. You **should not** be able to modify the collection (e.g. every time you try to get the current collection, you should get a new instance of the class, **and never the collection itself**).

### Note

Add the following code to your main method and submit it to Judge.

|  |
| --- |
| Field[] fields = ImmutableList.**class**.getDeclaredFields(); **if** (fields.**length** < 1) {  **throw new** ClassFormatException(); }  java.lang.reflect.Method[] methods = ImmutableList.**class**.getDeclaredMethods(); ***List***<java.lang.reflect.Method> methodsReturnTypes = Arrays.*stream*(methods).filter(m -> {  **if** (!m.getReturnType().getName().equalsIgnoreCase(**"ImmutableList"**)) {  **return false**;  }   **return true**; }).collect(Collectors.*toList*());  **if** (methodsReturnTypes.size() < 1) {  **throw new** ClassFormatException(); } |

If you’ve defined the class correctly, the test should pass.

## Car

Create a class **Car**. Every car has a **speed, fuel** and **fuel economy** (given in the same order on the first line). They should be stored in the class. Your task is to create a program which executes one of the commands:

* **Travel <distance>** – makes the car travel the specified <distance>

If you are given a distance which you don't have enough fuel to travel, just go as far as you can.

* **Refuel <liters>** – refuels the car with the specified <fuel>
* **Distance** – gets the total travel distance
* **Time** – get the total travel time
* **Fuel** – gets the remaining fuel
* **END** – stops the program

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 100 20 20  Travel 100  Distance  Time  Fuel  END | Total distance: 100.0 kilometers  Total time: 1 hours and 0 minutes  Fuel left: 0.0 liters |

## Pizza Time

Create a class **Pizza**. Every Pizza has a name (e.g. “Peperoni”) and a group. You should make it have a functionality to return its name and group.

Write a method (**in the class Pizza**), whose parameters are Strings and the result is a Map of Integers and Strings - the processed input. On the single input line, you will receive some Strings. Every String consists of two elements – first – the **group** and the second – the pizza’s **name**.

Your task is to get the input from the console and create a collection of pizza instances. Set their names and their groups to correspond the input. Make a Map (e.g. HashMap), consisting of the group and all pizza names of that group. After you collect the input, print the groups and their pizzas. **You must use Varargs!**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4Peperoni 2Margarita 2RunningChiken 4DonVito | 2 – Margarita, RunningChiken  4 – Peperoni, DonVito |

### Note

Add the following code to your main method and submit it to Judge.

|  |
| --- |
| Class<?> pizzaClass = Pizza.**class**; Method[] methods = pizzaClass.getDeclaredMethods(); ***List***<Method> checkedMethods = Arrays.*stream*(methods)  .filter(m -> m.getReturnType().getName().contains(**"Map"**))  .collect(Collectors.*toList*());  **if** (checkedMethods.size() < 1) {  **throw new** ClassFormatException(); } |

If you’ve defined the class correctly, the test should pass.

### Hint

Try using regex for processing the input.

## Date Modifier

Create a class **DateModifier** which stores the difference of the days between two Dates. It should have a method which takes two String parameters representing a date as Strings and **calculates the difference in the days between them.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1992 05 31  2016 06 17 | 8782 |
| 2016 05 31  2016 04 19 | 42 |

### Hint

Use the **Calendar** class.

## Rectangle Intersection

### Create a class Rectangle. It should consist of an id, width, height and the coordinates of its top left corner (horizontal and vertical). Create a method which receives as a parameter another Rectangle, checks if the two rectangles intersect and returns true or false.

### On the first line you will receive the number of rectangles – N and the number of intersection checks – M. On the next N lines, you will get the rectangles with their ID, width, height and coordinates. On the last M lines, you will get pairs of IDs which represent rectangles. Print if each of the pairs intersect.

### You will always receive valid data. There is no need to check if a rectangle exists.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 1  Pesho 2 2 0 0  Gosho 2 2 0 0  Pesho Gosho | true |

## \*Print People

Create a class **Person**. Every person should have name, age and occupation. Your task is to create the class and read some people, while adding them to a collection. Sort them by age and print them in the given format. **Override the toString() and compareTo() methods.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Gosho 22 Dentist  Mimi 13 Student  END | Mimi - age: 13, occupation: Student  Gosho - age: 22, occupation: Dentist |

## Problem 13. \*\*Drawing tool

You are young programmer and your Boss is giving you a task to create a tool which is drawing figures on the console. He knows you are not so good at OOP tasks so he told you to create only single class - CorDraw. Its task is to draw rectangular figures on the screen.

CorDraw’s constructor should take as parameter a Square instance or a Rectangle instance, extract its characteristics and draw the figure. Like we said your Boss is a good guy and he has some more info for you:

One of your classes should be a class named **Square** that should have only one method – **Draw()** which uses the length of the square’s sides and draws them on the console. For horizontal lines, use dashes ("-") and spaces (" "). For vertical lines – pipes ("|"). If the size of the figure is 6, dashes should also be 6.

### Hint

Search in internet for abstract classes and try implementing one. This will help you to reduce input parameter in the CorDraw’s constructor to a single one.

### Examples

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| Square  3 | |- - -|  | |  |- - -| | Square’s size is 3 so we draw 3 pipes down and 3 dashes across |

|  |  |  |
| --- | --- | --- |
| **Input** | **Output** | **Comment** |
| Rectangle  7  3 | |- - - - - - -|  | |  |- - - - - - -| | The Rectangle’s width is 7 and the length is 3 |