Exercise: OOP Principles

This exercise is part of the “Databases Frameworks” course @ SoftUni.

Problem 1. Define an Interface Person

Define an interface **Person** with properties for **Name** and **Age**. Define a class **Citizen** which implements **Person** and has a constructor which takes a **string** name and an **int** age.

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

```
public static void main(String[] args) {
    Class[] citizenInterfaces = Citizen.class.getInterfaces();
    if(Arrays.asList(citizenInterfaces).contains(Person.class)){
        Method[] fields = Person.class.getDeclaredMethods();
        System.out.println(fields.length);
        Scanner scanner = new Scanner(System.in);
        String name = scanner.nextLine();
        int age = Integer.parseInt(scanner.nextLine());
        Person person = new Citizen(name,age);
        System.out.println(person.getName());
        System.out.println(person.getAge());
    }
}
```

If you defined the interface and implemented it correctly, the test should pass.

Examples

Input	Output
Pesho 25	2 Pesho 25

Problem 2. Multiple Implementation

Using the code from the previous task, define an interface **Identifiable** with a **string** property **Id** and an interface **Birthable** with a **string** property **Birthdate** and implement them in the **Citizen** class. Rewrite the Citizen constructor to accept the new parameters.

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

```
public static void main(String[] args) {
    Class[] citizenInterfaces = Citizen.class.getInterfaces();
    if (Arrays.asList(citizenInterfaces).contains(Birthable.class)
            && Arrays.asList(citizenInterfaces).contains(Identifiable.class)) {
        Method[] methods = Birthable.class.getDeclaredMethods();
        System.out.println(methods.length);
        System.out.println(methods[0].getReturnType().getSimpleName());
        methods = Identifiable.class.getDeclaredMethods();
        System.out.println(methods.length);
        System.out.println(methods[0].getReturnType().getSimpleName());
        Scanner scanner = new Scanner(System.in);
        String name = scanner.nextLine();
        int age = Integer.parseInt(scanner.nextLine());
        String id = scanner.nextLine();
        String birthdate = scanner.nextLine();
        Identifiable identifiable = new Citizen(name,age,id,birthdate);
        Birthable birthable = new Citizen(name,age,id,birthdate);
    }
}
```

If you defined the interfaces and implemented them, the test should pass.

Examples

Input	Output
Pesho 25 9105152287 15/05/1991	1 String 1 String

Problem 3. Ferrari

Model an application which contains a **class Ferrari** and an **interface**. Your task is simple, you have a **car - Ferrari**, its model is **"488-Spider"** and it has a **driver**. Your Ferrari should have functionality to **use brakes** and **push the gas pedal**. When the **brakes** are pushed down **print "Brakes!"**, and when the **gas pedal** is pushed down - **"Zadu6avam sA!"**. As you may have guessed this functionality is typical for all cars, so you should **implement an interface** to describe it.

Your task is to **create a Ferrari** and **set the driver's name** to the passed one in the input. After that, print the info. Take a look at the Examples to understand the task better.

Input

On the **single input line**, you will be given the **driver's name**.

Output

On the **single output line**, print the model, the messages from the brakes and gas pedal methods and the driver's name. In the following format:

`<model>/<brakes>/<gas pedal>/<driver's name>`

Constraints

The input will always be valid, no need to check it explicitly! The Driver's name may contain any ASCII characters.

Example

Input	Output
Bat Giorgi	488-Spider/Brakes!/Zadu6avam sA!/Bat Giorgi
Dinko	488-Spider/Brakes!/Zadu6avam sA!/Dinko

Note

To check your solution, copy the code below and paste it to the bottom of the code in your main method.

Reflection

```
String ferrariName = Ferrari.class.getSimpleName();
String carInterface = Car.class.getSimpleName();

boolean isCreated = Car.class.isInterface();

if (!isCreated) {
    throw new IllegalClassFormatException("No interface created!");
}
```

Problem 4. Telephony

You have a business - **manufacturing cell phones**. But you have no software developers, so you call your friends and ask them to help you create a cell phone software. They agree and you start working on the project. The project consists of one main **model - a Smartphone**. Each of your smartphones should have functionalities of **calling other phones** and **browsing in the world wide web.**

Your friends are very busy, so you decide to write the code on your own. Here is the mandatory assignment:

You should have a **model** - **Smartphone** and two separate functionalities which your smartphone has - to **call other phones** and to **browse in the world wide web**. You should end up with **one class** and **two interfaces**.

Input

The input comes from the console. It will hold two lines:

- **First line**: **phone numbers** to call (String), separated by spaces.
- **Second line: sites** to visit (String), separated by spaces.

Output

- First **call all numbers** in the order of input then **browse all sites** in order of input
- The functionality of calling phones is printing on the console the number which are being called in the format:

  `Calling... <number>`
- The functionality of the browser should print on the console the site in format:

  `Browsing: <site>!`
- If there is a number in the input of the URLs, print: **"Invalid URL!"** and continue printing the rest of the URLs.
- If there is a character different from a digit in a number, print: **"Invalid number!"** and continue to the next number.

Constraints

- Each site's URL should consist only of letters and symbols (**No digits are allowed** in the URL address)

Examples

Input	Output
`0882134215 0882134333 08992134215 0558123 3333 1`	`Calling... 0882134215`

`http://softuni.bg http://youtube.com`
`http://www.g00gle.com`
	`Calling... 0882134333`
	`Calling... 08992134215`
	`Calling... 0558123`
	`Calling... 3333`
	`Calling... 1`
	`Browsing: http://softuni.bg!`
	`Browsing: http://youtube.com!`
	`Invalid URL!`

Problem 5. Border Control

It’s the future, you’re the ruler of a totalitarian dystopian society inhabited by **citizens** and **robots**, since you’re afraid of rebellions you decide to implement strict control of who enters your city. Your soldiers check the **Id**s of everyone who enters and leaves.

You will receive from the console an unknown amount of lines until the command “**End**” is received, on each line there will be the information for either a citizen or a robot who tries to enter your city in the format **“<name> <age> <id>**” for citizens and “**<model> <id>**” for robots. After the end command on the next line you will receive a single number representing **the last digits of fake ids**, all citizens or robots whose **Id** ends with the specified digits must be detained.

The output of your program should consist of all detained **Id**s each on a separate line (the order of printing doesn’t matter).

Examples

Input	Output

`Pesho 22 9010101122`
`MK-13 558833251`
`MK-12 33283122`
`End`
`122`
	`9010101122`
	`33283122`

`Toncho 31 7801211340`
`Penka 29 8007181534`
`IV-228 999999`
`Stamat 54 3401018380`
`KKK-666 80808080`
`End`
`340`
	`7801211340`

Problem 6. Birthday Celebrations

It is a well known fact that people celebrate birthdays, it is also known that some people also celebrate their pets birthdays. Extend the program from your last task to add **birthdates** to citizens and include a class **Pet**, pets have a **name** and a **birthdate**. Encompass repeated functionality into interfaces and implement them in your classes.

You will receive from the console an unknown amount of lines until the command "**End**" is received, each line will contain information in one of the following formats **"Citizen <name> <age> <id> <birthdate>**" for citizens, "**Robot <model> <id>**" for robots or "**Pet <name> <birthdate>**" for pets. After the end command on the next line you will receive a single number representing **a specific year**, your task is to print all birthdates (of both citizens and pets) in that year in the format **day/month/year** (the order of printing doesn't matter).

Examples

Input	**Output**

`Citizen Pesho 22 9010101122 10/10/1990`
`Pet Sharo 13/11/2005`
`Robot MK-13 558833251`
`End`
`1990`
	`10/10/1990`

`Citizen Stamat 16 0041018380 01/01/2000`
`Robot MK-10 12345678`
`Robot PP-09 00000001`
`Pet Topcho 24/12/2000`
`Pet Kosmat 12/06/2002`
`End`
`2000`
	`01/01/2000`
	`24/12/2000`

`Robot VV-XYZ 11213141`
`Citizen Penka 35 7903210713 21/03/1979`
`Citizen Kane 40 7409073566 07/09/1974`
`End`
`1975`
	`<no output>`

Problem 7. Mankind

Your task is to model an application. It is very simple. The mandatory models of our application are 3:

- Human
- Worker
- Student

The parent class – **Human** should have **first name** and **last name**. Every student has a **faculty number**. Every worker has a **week salary** and **work hours per day**. It should be able to calculate the money he earns by hour. You can see the constraints below.

Input

On the first input line, you will be given info about a single student - a **name** and **faculty number**.

On the second input line, you will be given info about a single worker - **first name**, **last name**, **salary** and **working hours**.

Output

You should first print the info about the student following a single blank line and the info about the worker in the given formats:

- Print the student info in the following format:
    - `First Name: {student's first name}`
    - `Last Name: {student's last name}`
    - `Faculty number: {student's faculty number}`

- Print the worker info in the following format:
    - `First Name: {worker's first name}`
    - `Last Name: {worker's second name}`
    - `Week Salary: {worker's salary}`
    - `Hours per day: {worker's working hours}`
    - `Salary per hour: {worker's salary per hour}`

Print exactly two digits after every double value's decimal separator (e.g. 10.00)

Constraints

Parameter	Constraint	Exception Message
Human first name	Should start with a capital letter	"Expected upper case letter!Argument: firstName"
Human first name	Should be more than 4 symbols	"Expected length at least 4 symbols!Argument: firstName"
Human last name	Should start with a capital letter	"Expected upper case letter!Argument: lastName"
Human last name	Should be more than 3 symbols	"Expected length at least 3 symbols!Argument: lastName "
Faculty number	Should be in range [5..10] symbols	"Invalid faculty number!"
Worker last name	Should be more than 3 symbols	"Expected length more than 3 symbols!Argument: lastName"
Week salary	Should be more than 10	"Expected value mismatch!Argument: weekSalary"
Working hours	Should be in the range [1..12]	"Expected value mismatch!Argument: workHoursPerDay"

Example

Input	Output

`Ivan Ivanov 08`
`Pesho Kirov 1590 10`
	`Invalid faculty number!`

`Stefo Mk321 0812111`
`Ivcho Ivancov 1590 10`
	`First Name: Stefo`
	`Last Name: Mk321`
	`Faculty number: 0812111`
	
	`First Name: Ivcho`
	`Last Name: Ivancov`
	`Week Salary: 1590.00`
	`Hours per day: 10.00`
	`Salary per hour: 22.71`

Problem 8. Vehicles

Write a program that models 2 vehicles (**Car** and **Truck**) and will be able to simulate **driving** and **refueling** them. **Car** and **truck** both have **fuel quantity**, **fuel consumption in liters per km** and can be **driven given distance** and **refueled with given liters.** But in the summer both vehicles use air conditioner and their **fuel consumption** per km is **increased** by **0.9** liters for the **car** and with **1.6** liters for the **truck**. Also the **truck** has a tiny hole in his tank and when it gets **refueled** it gets only **95%** of given **fuel**. The **car** has no problems when refueling and adds **all given fuel to its tank.** If vehicle cannot travel given distance its fuel does not change.

Input

- On the **first line** - information about the car in format **`{Car {fuel quantity} {liters per km}}`**
- On the **second line** – info about the truck in format **`{Truck {fuel quantity} {liters per km}}`**
- On third line - **number of commands N** that will be given on the next **N** lines
- On the next **N** lines – commands in format
  - **`Drive Car {distance}`**
  - **`Dive Truck {distance}`**
  - **`Refuel Car {liters}`**
  - **`Refuel Truck {liters}`**

Output

After each **Drive command** print whether the Car/Truck was able to travel given distance in format if it's successful. **Print the distance with all digits after the decimal separator except trailing zeros.** Use the **DecimalFormat** class:

```
Car/Truck travelled {distance} km
```

Or if it is not:

```
Car/Truck needs refueling
```

Finally print the **remaining fuel** for both car and truck rounded **2 digits after floating point** in format:

```
Car: {liters}
Truck: {liters}
```

Example

Input	Output

`Car 15 0.3`
`Truck 100 0.9`
`4`
`Drive Car 9`
`Drive Car 30`
`Refuel Car 50`
`Drive Truck 10`
	`Car travelled 9 km`
	`Car needs refueling`
	`Truck travelled 10 km`
	`Car: 54.20`
	`Truck: 75.00`

`Car 30.4 0.4`
`Truck 99.34 0.9`
`5`
`Drive Car 500`
`Drive Car 13.5`
`Refuel Truck 10.300`
`Drive Truck 56.2`
`Refuel Car 100.2`
	`Car needs refueling`
	`Car travelled 13.5 km`
	`Truck needs refueling`
	`Car: 113.05`
	`Truck: 109.13`

Problem 9. Animals

Create a hierarchy of **Animals**. Your task is simple: there should be a base class which all others derive from. Your program should have 3 different animals – **Dog**, **Frog** and **Cat**.

Let's go deeper in the hierarchy and create two additional classes – **`Kitten`** and **`Tomcat`**. **`Kittens` are female and `Tomcats` are male!**

We are ready now, but the task is not complete. Along with the animals, there should be and a class which classifies its derived classes as sound producible. You may guess that all animals are sound producible. The only one mandatory functionality of all sound producible objects is to **`produceSound()`**. For instance, the dog should bark.

Your task is to model the hierarchy and test its functionality. Create an animal of all kinds and make them produce sound.

On the console, you will be given some lines of code. Each two lines of code, represents animals and their names, age and gender. On the first line, there will be the kind of animal, you should instantiate. And on the next line, you will be given the name, the age and the gender. Stop the process of gathering input, when the command **"Beast!"** is given.

Output

- On the console, print for each animal you've instantiated, its info on two lines. On the first line, print: **`{Kind of animal} {name} {age} {gender}`**
- On the second line, print: **`{produceSound()}`**

Constraints

- Each **Animal** should have **name**, **age** and **gender**
- **All properties**' values should **not be blank** (e.g. name, age and so on...)
- If you enter invalid input for one of the properties' values, throw exception with message: **"Invalid input!"**
- Each animal should have a functionality to **`produceSound()`**
- Here is example of what each kind of animal should produce when, **`produceSound()`** is called
  - **Dog: "BauBau"**

- **Cat: “MiauMiau”**
- **Frog: “Frogggg”**
- **Kittens: “Miau”**
- **Tomcat: “Give me one million b***h”**
- **Message from the Animal class: "Not implemented!"**

Examples

Input	Output

`Cat`
`Macka 12 Female`
`Dog`
`Sharo 132 Male`
`Beast!`
	`Cat Macka 12 Female`
	`MiauMiau`
	`Dog Sharo 132 Male`
	`BauBau`

`Frog`
`Sashky 12 Male`
`Beast!`
	`Frog Sashky 12 Male`
	`Frogggg`

`Frog`
`Sashky -2 Male`
`Beast!`
	`Invalid input!`