Homework: OOP in Java

This document defines the homework assignments from the "Java Basics" Course @ Software University. Please submit as homework a single `zip` / `rar` / `7z` archive holding the solutions (source code) of all below described problems. The solutions should be written in Java.

Problem 1. Geometry

Define a class structure that models a shape hierarchy.

- **Shape** – base class for any kind of shape, holds a list of **vertices**
  - **PlaneShape** – base class for all plane (2D) shapes, holds a list of **2D vertices** (holding **x** and **y**), implements **PerimeterMeasurable** and **AreaMeasurable** interfaces
    - **Triangle** – holds 3 vertices
    - **Rectangle** – holds 1 vertex, width, height
    - **Circle** – holds 1 vertex and radius
  - **SpaceShape** – base class for all three-dimensional shapes, holds a list of **3D vertices** (holding **x**, **y** and **z**), implements **AreaMeasurable** and **VolumeMeasurable** interfaces
    - **Square Pyramid** – holds 1 vertex (base center), base width, pyramid height
    - **Cuboid** – holds 1 vertex, width, height, depth
    - **Sphere** – holds 1 vertex and radius

A **vertex** is a point in 2D/3D space. The distance between two 2D vertices is calculated using the formula:

$$c = \sqrt{(x_A - x_B)^2 + (y_A - y_B)^2}$$

Define the following interfaces:

- **PerimeterMeasurable** – holds **double getPerimeter()**
- **AreaMeasurable** – holds **double getArea()**
- **VolumeMeasurable** – holds **double getVolume()**

Design the class hierarchy using proper inheritance and code reusability through abstraction. Each shape should implement its respective interfaces with proper formulas.

Override **toString()** to return information about each shape (shape type, each vertex's coordinates, perimeter/area/volume). Create objects of different classes and add them to a **single** array. Iterate through the array and print information about each shape.

Filter the existing array using **lambda expressions** by:

- **VolumeMeasurable** shapes whose **volume** is over 40.00
- **Plane shapes** and **sort** them by their **perimeter** in ascending order

Problem 2. 1lv Shop

Design a class hierarchy that models a shop.

- **Product** – base class for all products, holds **name**, **price**, **quantity** and **age restriction** (can be **None**, **Teenager** or **Adult**). Implements the **Buyable** interface.
  - **FoodProduct** – implements the **Expirable** interface. Returns 70% of the price if the product expires in 15 days time.
  - **ElectonicsProduct** – base class for electronics, holds guarantee period

    - **Computer** – has a guarantee period of 24 months. Returns 95% of the price if the quantity is over 1000.
    - **Appliance** – has a guarantee period of 6 months. Returns 105% of the price if the quantity is less than 50.
- **Customer** – holds **name**, **age** and **balance**

Define **properties** (getters and setters) for each class. Validate the data and throw **exceptions** where necessary.

Define the following interfaces**:**

- **Buyable** – holds **double getPrice()**
- **Expirable** – holds **Date getExpirationDate()**

Create a static class **PurchaseManager**. The class should hold the **processPurchase(Product product, Customer customer)** method that handles purchases (takes money from customer, reduces product quantity by 1).The **PurchaseManager** should throw exceptions with descriptive messages in the following situations:

- If the product is out of stock (i.e. no quantity)
- If the product has expired
- If the buyer does not have enough money
- If the buyer does not have permission to purchase the given product

Catch any exceptions in your **main()** method and print their message. Create several products of different types and add them to a list. Filter the list using **lambda expressions** by:

- **Expirable** products and get the **name** of the first product with the soonest date of expiration
- All products with **adult age restriction** and **sort** them by **price** in ascending order

Sample Input	Sample Output

`FoodProduct cigars = new FoodProduct("420 Blaze it fgt", 6.90, 1400, AgeRestriction.Adult);`
`Customer pecata = new Customer("Pecata", 17, 30.00);`
`PurchaseManager.processPurchase(pecata, cigars);`
`Customer gopeto = new Customer("Gopeto", 18, 0.44);`
`PurchaseManager.processPurchase(gopeto, cigars);`
	`You are too young to buy this product!`
	`You do not have enough money to buy this product!`

Exam problems.** - Java Basics Exam 8th February 2015

All of the problems below are given from the previous Java Basics exams. **You are not obligated** to submit any of them in your homework. We highly recommend you to try solving some or all of them so you can be well prepared for the upcoming exam. You need to learn how to use conditional statements, loops, arrays and other things (learn in internet how or read those chapters in the book "Fundamentals of computer programming with Java"). If you still find those problems too hard for solving it's very useful to **check** and **understand** the solutions. You can download all solutions and tests for this variant here or check all previous exams. You can also test your solutions in our automated judge system to see if you pass all tests.

Problem 3.* – Gandalf`s Stash

Gandalf the Gray is a great wizard but he also loves to eat. When he is hungry he gets angry. These are some of Gandalf's favorite types of food:

- **Cram**: **2** points of happiness;
- **Lembas**: **3** points of happiness;
- **Apple**: **1** point of happiness;
- **Melon**: **1** point of happiness;
- **HoneyCake**: **5** points of happiness;
- **Mushrooms**: **-10** points of happiness;
- Everything else: **-1** point of happiness;

Gandalf moods are:

- **Angry** - below -5 points of happiness;
- **Sad** - from -5 to 0 points of happiness;
- **Happy** - from 0 to 15 points of happiness;
- **Special JavaScript mood** - when happiness points are more than 15;

Input

The input comes from the console. It will holds two lines: first - Gandalf`s first mood.

On the second line will be described the foods that Gandalf will eat, separated with different symbols or whitespace(s). Comparing the input is **case-insensitive**. The input data will always be valid and in the format described. There is no need to check it explicitly.

Output

Print on the console Gandalf`s happiness points and mood after he drinks his beers.

Constraints

- The characters in the input string will be no more than: **1000.**
- The food count would be in the range **[1…100]**.
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output
`-10` `Cram, banica,Melon!_,HonEyCake,     !HoneYCake,hoNeyCake_;`	`7` `Happy`
`-30` `gosho, pesho, meze, Melon, HoneyCake@;`	`-27` `Angry`
`-3` `HoneyCake honeyCake honeyCake HoneyCakE HoneYCake HonEyCake HoneyCake HoneyCake HoneyCake HoNeyCake`	`47` `Special JavaScript mood`
`-2` `mELon, AMelon, beer,cRam, nacepin`	`-2` `Sad`

Problem 4.* – Letters change Numbers

Nakov likes Math. But he also likes the English alphabet a lot. He invented a game with numbers and letters from the **English** alphabet. The game was simple. You get a string consisting of a **number between two letters**. Depending on whether the letter was in front of the number or after it you would perform different mathematical operations on the number to achieve the result.

First you start with the letter **before** the number. If it's **Uppercase** you **divide** the number by the letter's **position** in the alphabet. If it's **lowercase** you **multiply** the number with the letter's position. **Then** you move to the **letter after** the number. If it's **Uppercase** you **subtract** its position from the resulted number. If it's **lowercase** you **add** its position to the resulted number. But the game became too easy for Nakov really quick. He decided to complicate it a bit by doing the same but with **multiple** strings keeping track of only the **total sum** of all results. Once he started to solve this with more strings and bigger numbers it became quite hard to do it only in his mind. So he kindly asks you to write a program that **calculates the sum of all numbers after the operations on each number have been done**.

For example, you are given the sequence "**A12b s17G**". We have two strings – **"A12b"** and **"s17G"**. We do the operations on each and sum them. We start with the letter before the number on the first string. **A is Uppercase** and its position in the alphabet is **1**. So we divide the number 12 with the position 1 (**12/1 = 12)**. Then we move to the letter after the number. **b is lowercase** and its position is 2. So we add 2 to the resulted number (**12+2=14)**. Similarly for the second string **s is lowercase** and its position is 19 so we multiply it with the number (**17*19 = 323)**. Then we have Uppercase G with position 7, so we subtract it from the resulted number (**323 – 7 = 316)**. Finally we sum the 2 results and we get **14 + 316=330**;

Input

The input comes from the console as a **single line, holding the sequence of strings**. Strings are separated by **one or more white spaces**.

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

Output

Print at the console a single number: the **total sum of all processed numbers** rounded up to **two digits** after the decimal separator.

Constraints

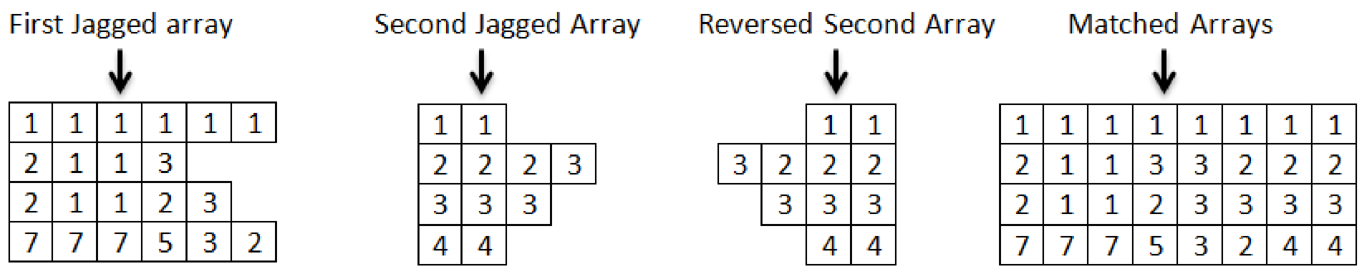
- The **count** of the strings will be in the range **[1…10].**
- The numbers between the letters will be integers in range **[1…2,147,483,647].**
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output	Comment
`A12b s17G`	`330.00`	`12/1=12, 12+2=14, 17*19=323, 323–7=316,` **`14+316=330`**
`P34562Z q2576f   H456z`	`46015.13`	
`a1A`	`0.00`	

Problem 5.* – Lego Blocks

You are given two **jagged arrays**. Each array represents a **Lego block** containing integers. Your task is first to **reverse** the second jagged array and then check if it would **fit perfectly** in the first jagged array.



The picture above shows exactly what fitting arrays mean. If the arrays fit perfectly you should **print out** the newly made rectangular matrix. If the arrays do not match (they do not form a rectangular matrix) you should print out the **number of cells** in the first array and in the second array combined. The examples below should help you understand more the assignment.

Input

The first line of the input comes as an **integer number n** saying how many rows are there in both arrays. Then you have **2 * n** lines of numbers separated by whitespace(s). The first **n** lines are the rows of the first jagged array; the next **n** lines are the rows of the second jagged array. There might be leading and/or trailing whitespace(s).

Output

You should print out the combined matrix in the format:

```
[elem, elem, …, elem]
[elem, elem, …, elem]
[elem, elem, …, elem]
```

If the two arrays do not fit you should print out : `The total number of cells is: count`

Constraints

- The number n will be in the range [2…10].
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output

2
1 1 1 1 1 1
2 1 1 3
1 1
2 2 2 3
	[1, 1, 1, 1, 1, 1, 1, 1]
	[2, 1, 1, 3, 3, 2, 2, 2]

2
1 1 1 1 1
1 1 1
1
1 1 1 1 1
	The total number of
	cells is: 14

Problem 6.* – User Logs

Marian is a famous system administrator. The person to overcome the security of his servers has not yet been born. However, there is a new type of threat where users flood the server with messages and are hard to be detected since they change their IP address all the time. Well, Marian is a system administrator and is not so into programming. Therefore, he needs a skillful programmer to track the user logs of his servers. You are the chosen one to help him!

You are given an input in the format:

IP=(IP.Address) message=(A&sample&message) user=(username)

Your task is to parse the ip and the username from the input and for **every user**, you have to display **every ip** from which the corresponding user has sent a message and the **count of the messages** sent with the corresponding ip. In the output, the usernames must be **sorted alphabetically** while their IP addresses should be displayed in the **order of their first appearance.** The output should be in the following format:

```
username:
IP => count, IP => count.
```

For example, given the following input - **IP=192.23.30.40 message='Hello&derps.' user=destroyer**, you have to get the username **destroyer** and the IP **192.23.30.40** and display it in the following format:

```
destroyer:
192.23.30.40 => 1.
```

The username destroyer has sent a message from ip 192.23.30.40 once.

Check the examples below. They will further clarify the assignment.

Input

The input comes from the console as **varying number** of lines. You have to parse every command until the command that follows is **end.** The input will be in the format displayed above, there is no need to check it explicitly.

Output

For every user found, you have to display each log in the format:

```
username:
IP => count, IP => count…
```

The IP addresses must be split with a comma, and each line of IP addresses must end with a dot.

Constraints

- The number of commands will be in the range [1..50]
- The IP addresses will be in the format of either **IPv4** or **IPv6.**
- The messages will be in the format: **This&is&a&message**
- The username will be a string with length in the range [3..50]
- Time limit: 0.3 sec. Memory limit: 16 MB.

Examples

Input	Output

`IP=192.23.30.40 message='Hello&derps.' user=destroyer`
`IP=192.23.30.41 message='Hello&yall.' user=destroyer`
`IP=192.23.30.40 message='Hello&hi.' user=destroyer`
`IP=192.23.30.42 message='Hello&Dudes.' user=destroyer`
`end`
	`destroyer:`
	`192.23.30.40 => 2,`
	`192.23.30.41 => 1,`
	`192.23.30.42 => 1.`

`IP=FE80:0000:0000:0000:0202:B3FF:FE1E:8329 message='Hey&son'`
`user=mother`
`IP=192.23.33.40 message='Hi&mom!' user=child0`
`IP=192.23.30.40 message='Hi&from&me&too' user=child1`
`IP=192.23.30.42 message='spam' user=destroyer`
`IP=192.23.30.42 message='spam' user=destroyer`
`IP=192.23.50.40 message='' user=yetAnotherUsername`
`IP=192.23.50.40 message='comment' user=yetAnotherUsername`
`IP=192.23.155.40 message='Hello.' user=unknown`
`end`
	`child0:`
	`192.23.33.40 => 1.`
	`child1:`
	`192.23.30.40 => 1.`
	`destroyer:`
	`192.23.30.42 => 2.`
	`mother:`
	`FE80:0000:0000:0000:0202`
	`:B3FF:FE1E:8329 => 1.`
	`unknown:`
	`192.23.155.40 => 1.`
	`yetAnotherUsername:`
	`192.23.50.40 => 2.`