Exercises: Defining Classes

This document defines the exercises for the "Java Advanced" course @ Software University. Please submit your solutions (source code) to all below-described problems in Judge.

Note: For these exercises, you are allowed to have multiple classes in the same file.

1. Opinion Poll

Create a Person class with two fields **String name** and **int age**. Write a program that reads from the console **N** lines of personal information and then prints all people whose **age** is **more than 30** years, **sorted in alphabetical order**. Print in the following format **"{name} - {age}"**.

Note: you can use the **`stream()`** to filter people.

Examples

Input	Output
3 Peter 12 Steven 31 John 48	John - 48 Steven - 31
5 Sofia 33 Thomas 88 Camilla 22 Robert 44 Owen 11	Robert - 44 Sofia - 33 Thomas - 88

2. Company Roster

Define a class **Employee** that holds the following information: **name, salary, position, department, email,** and **age**. The **name, salary**, **position,** and **department** are **mandatory,** while the rest are **optional**.

Your task is to write a program that takes **N** lines of information about employees from the console and calculates the department with the highest average salary, and prints for each employee in that department his **name, salary, email, and age** - **sorted by salary in descending order**. If an employee **doesn't have** an **email** – in place of that field, you should print "**n/a**" instead, if he doesn't have an **age** – print "**-1**" instead. Print in the following format:

"Highest Average Salary: {department}

{name$_1$} {salary$_1$} {email$_1$} {age$_1$}

{name$_2$} {salary$_2$} {email$_2$} {age$_2$}

…

{name$_n$} {salary$_n$} {email$_n$} {age$_n$}"

The **salary** should be printed to **two decimal places** after the separator.

Hint: You can define a **Department** class that holds a list of employees.

Examples

Input	Output

4 Peter 120.00 Dev Development peter@abv.bg 28 Tina 333.33 Manager Marketing 33 Sam 840.20 ProjectLeader Development sam@sam.com George 0.20 Freeloader Nowhere 18	Highest Average Salary: Development Sam 840.20 sam@sam.com -1 Peter 120.00 peter@abv.bg 28
6 Silver 496.37 Temp Coding silver@yahoo.com Sam 610.13 Manager Sales John 609.99 Manager Sales john@abv.bg 44 Venci 0.02 Director BeerDrinking beer@beer.br 23 Andre 700.00 Director Coding Popeye 13.3333 Sailor SpinachGroup popeye@pop.ey	Highest Average Salary: Sales Sam 610.13 n/a -1 John 609.99 john@abv.bg 44

3. * Speed Racing

Your task is to implement a program that keeps track of cars and their fuel and supports methods for moving the cars. Define a class **Car** that keeps track of a car information **Model, fuel amount, fuel cost for 1 kilometer,** and **distance traveled**. A Car Model is **unique** - there will never be 2 cars with the same model.

On the first line of the input, you will receive a number **N** - the number of cars you need to track, on **each** of the next **N** lines you will receive information for a car in the following format "**`{Model} {FuelAmount} {FuelCostFor1km}`**", all **cars start at 0 kilometers traveled**.
After the **N** lines, until the command "**End**" is received, you will receive commands in the following format "**`Drive {CarModel} {amountOfKm}`**", implement a method in the **Car** class to calculate whether a car **can** move that distance or **not**. If yes, the car **fuel amount** should be **reduced** by the amount of used fuel, and its **distance traveled** should be increased by the amount of kilometers traveled, otherwise, the car should not move (Its fuel amount and distance traveled should stay the same) and you should print on the console "**`Insufficient fuel for the drive`**". After the "**End**" command is received, print each car in order of appearing in input, and its current fuel amount and distance traveled in the format "**`{Model} {fuelAmount} {distanceTraveled}`**", where the fuel amount should be printed to **two decimal places** after the separator.

Examples

Input	Output
2 AudiA4 23 0.3 BMW-M2 45 0.42 Drive BMW-M2 56 Drive AudiA4 5 Drive AudiA4 13 End	AudiA4 17.60 18 BMW-M2 21.48 56
3 AudiA4 18 0.34 BMW-M2 33 0.41 Ferrari-488Spider 50 0.47 Drive Ferrari-488Spider 97 Drive Ferrari-488Spider 35 Drive AudiA4 85 Drive AudiA4 50 End	Insufficient fuel for the drive Insufficient fuel for the drive AudiA4 1.00 50 BMW-M2 33.00 0 Ferrari-488Spider 4.41 97

4. Raw Data

You are the owner of a courier company, and you want to make a system for tracking your cars and their cargo. Define a class **Car** that holds information about the **model, engine, cargo,** and a **collection of exactly 4 tires**. The engine, cargo, and tire **should be separate classes**, create a constructor that receives all information about the Car and creates and initializes its inner components (engine, cargo, and tires).

On the first line of the input, you will receive a number **N** - the number of cars you have, on each of the next **N** lines, you will receive information about a car in the format:

"**`{Model} {EngineSpeed} {EnginePower} {CargoWeight} {CargoType} {Tire1Pressure} {Tire1Age} {Tire2Pressure} {Tire2Age} {Tire3Pressure} {Tire]'3Age} {Tire4Pressure} {Tire4Age}`**", where the speed, power, weight and tire age are **integers**, tire pressure is a **double.**

After the **N** lines, you will receive a single line with one of 2 commands "**`fragile`**" or "**`flamable`**", if the command is "**`fragile`**" print all cars whose **Cargo Type is** "**`fragile`**" with a **tire** whose **pressure is < 1** if the command is "**`flamable`**" print all cars whose **Cargo Type is** "**`flamable`**" and have **Engine Power > 250**. The cars should be printed in order of appearing in the input.

Examples

Input	Output
2 ChevroletAstro 200 180 1000 fragile 1.3 1 1.5 2 1.4 2 1.7 4 Citroen2CV 190 165 1200 fragile 0.9 3 0.85 2 0.95 2 1.1 1 fragile	Citroen2CV
4 ChevroletExpress 215 255 1200 flamable 2.5 1 2.4 2 2.7 1 2.8 1 ChevroletAstro 210 230 1000 flamable 2 1 1.9 2 1.7 3 2.1 1 DaciaDokker 230 275 1400 flamable 2.2 1 2.3 1 2.4 1 2 1 Citroen2CV 190 165 1200 fragile 0.8 3 0.85 2 0.7 5 0.95 2 flamable	ChevroletExpress DaciaDokker

5. Car Salesman

Define two classes **Car** and **Engine.** A **Car** has a **model, engine, weight,** and **color**. An Engine has a **model**, **power, displacement,** and **efficiency**. A Car's **weight**, color, Engine displacements, and **efficiency** are **optional**.

On the first line, you will read a number **N** which will specify how many lines of engines you will receive, on each of the next **N** lines, you will receive information about an **Engine** in the following format "**`{Model} {Power} {Displacement} {Efficiency}`**". After the lines with engines, on the next line, you will receive a number **M** – specifying the number of Cars that will follow, on each of the next **M** lines the information about a **Car** will follow in the following format "**`{Model} {Engine} {Weight} {Color}`**", where the engine in the format will be the **model of an existing Engine**. When creating the object for a **Car**, you should keep a **reference to the real engine** in it, instead of just the engine's model, **note** that the optional properties **might be missing** from the formats.

Your task is to print each car (in the **order** you **received** them) and its information in the format defined below. If any of the optional fields have not been given, print "**`n/a`**" in its place instead of:

`"{CarModel}:`
`{EngineModel}:`
`Power: {EnginePower}`
`Displacement: {EngineDisplacement}`

Efficiency: {EngineEfficiency}
Weight: {CarWeight}
Color: {CarColor}"

Optional

Override the class **ToString()** methods to have a reusable way of displaying the objects.

Examples

Input	Output
2 V8-101 220 50 V4-33 140 28 B 3 FordFocus V4-33 1300 Silver FordMustang V8-101 VolkswagenGolf V4-33 Orange	FordFocus: V4-33: Power: 140 Displacement: 28 Efficiency: B Weight: 1300 Color: Silver FordMustang: V8-101: Power: 220 Displacement: 50 Efficiency: n/a Weight: n/a Color: n/a VolkswagenGolf: V4-33: Power: 140 Displacement: 28 Efficiency: B Weight: n/a Color: Orange
4 DSL-10 280 B V7-55 200 35 DSL-13 305 55 A+ V7-54 190 30 D 4 FordMondeo DSL-13 Purple VolkswagenPolo V7-54 1200 Yellow VolkswagenPassat DSL-10 1375 Blue FordFusion DSL-13	FordMondeo: DSL-13: Power: 305 Displacement: 55 Efficiency: A+ Weight: n/a Color: Purple VolkswagenPolo: V7-54: Power: 190 Displacement: 30 Efficiency: D Weight: 1200 Color: Yellow VolkswagenPassat: DSL-10: Power: 280 Displacement: n/a Efficiency: B Weight: 1375 Color: Blue FordFusion: DSL-13:

	Power: 305 Displacement: 55 Efficiency: A+ Weight: n/a Color: n/a

6. Pokémon Trainer

You want to be the best pokemon trainer, like no one ever was, so you set out to catch pokemon. Define a class **Trainer** and a class **Pokemon**. The trainer has a **name**, a **number of badges,** and a **collection of pokemon**. Pokemon has a **name**, an **element,** and **health**, all values are **mandatory**. Every Trainer **starts with 0 badges**.

From the console, you will receive an unknown number of lines until you receive the command "**Tournament**", each line will carry information about a pokemon and the trainer who caught it in the format "**{TrainerName} {PokemonName} {PokemonElement} {PokemonHealth}**" where **TrainerName** is the name of the Trainer who caught the pokemon, names are **unique** there cannot be 2 trainers with the same name. After receiving the command "**Tournament**" an unknown number of lines containing one of three elements "**Fire**", "**Water**", and "**Electricity**" will follow until the command "**End**" is received. For every command, you must check if a trainer has **at least 1** pokemon with the given element, if he does he receives 1 badge, otherwise, all his pokemon **lose 10 health**, if a pokemon falls **to 0 or less health, he dies** and must be deleted from the trainer's collection. After the command "**End**" is received, you should print all trainers **sorted by the amount of badges they have in descending order** (if two trainers have the same amount of badges, they should be sorted by order of appearance in the input) in the format "**{TrainerName} {Badges} {NumberOfPokemon}**".

Examples

Input	Output
Peter Charizard Fire 100 George Squirtle Water 38 Peter Pikachu Electricity 10 Tournament Fire Electricity End	Peter 2 2 George 0 1
Sam Blastoise Water 18 Alex Pikachu Electricity 22 John Kadabra Psychic 90 Tournament Fire Electricity Fire End	Alex 1 1 Sam 0 0 John 0 1

7. Google

Google is always watching you, so it should come as no surprise that they know everything about you (even your pokemon collection), since you're really good at writing classes, Google asked you to design a Class that can hold all the information they need for people.

You will receive an unknown number of rows from the console until you receive the "**End**" command, on each of those lines, there will be information about a person in one of the following formats:

- "**{Name} company {companyName} {department} {salary}**"
- "**{Name} pokemon {pokemonName} {pokemonType}**"
- "**{Name} parents {parentName} {parentBirthday}**"
- "**{Name} children {childName} {childBirthday}**"
- "**{Name} car {carModel} {carSpeed}**"

You should structure all information about a person in a class with nested subclasses. People's names are **unique** - there won't be 2 people with the same name, a person can also have **only 1 company** and **car**, but can have **multiple parents, children,** and **pokemons**. After the command "**End**" is received on the next line, you will receive a **single** name, you should **print** all information about that person. **Note** that information can change **during** the **input**, for instance, if we receive multiple lines which specify a person's company, only the **last one** should be the one remembered. The salary must be formatted to **two decimal places** after the separator.

Note: print the information in the format:

"**{personName}**

Company:

{companyName} {companyDepartment} {salary}

…

Children:

{childName} {childBirthday}

…"

Examples

Input	Output
PeterSmith company PeshInc Management 1000.00 JohnJohnson car Trabant 30 PeterSmith pokemon Pikachu Electricity PeterSmith parents PoshoPeshev 22/02/1920 JohnJohnson pokemon Electrode Electricity End JohnJohnson	JohnJohnson Company: Car: Trabant 30 Pokemon: Electrode Electricity Parents: Children:
GeorgeJohnson pokemon Onyx Rock GeorgeJohnson parents SaraJohnson 13/03/1933 AlexAlexson pokemon Moltres Fire GeorgeJohnson company JeleInc Jelior 777.77 GeorgeJohnson children SamJohnson 01/01/2001 SmithSmithon pokemon Blastoise Water GeorgeJohnson car AudiA4 180 GeorgeJohnson pokemon Charizard Fire End GeorgeJohnson	GeorgeJohnson Company: JeleInc Jelior 777.77 Car: AudiA4 180 Pokemon: Onyx Rock Charizard Fire Parents: SaraJohnson 13/03/1933 Children: SamJohnson 01/01/2001

8. *** Family Tree

You want to build your family tree, so you went to ask your grandmother, sadly your grandmother keeps remembering information about your predecessors in pieces, so it falls to you to group the information and build the family tree.

On the first line of the input, you will receive either a name or a birthdate in the format "**{FirstName} {LastName}**" or "**day/month/year**" - your task is to find information about the person in the family tree. On the next lines, until the command "**End**" is received, you will receive information about your predecessors that you will use to build the family tree.

The information will be in one of the following formats:

- "**{FirstName} {LastName} - {FirstName} {LastName}**"
- "**{FirstName} {LastName} - {day/month/year}**"
- "**{day/month/year} - {FirstName} {LastName}**"
- "**{day/month/year} - {day/month/year}**"
- "**{FirstName} {LastName} {day/month/year}**"

The first 4 formats reveal a family tie – **the person on the left is the parent of the person on the right** (as you can see the format does **not** need to contain names, for example, the 4th format means the person born on the left date is parent to the person born on the right date). The last format ties **different** information together – i.e. **the person with that name was born on that date**. **Names** and **birthdates** are **unique** – there won't be 2 people with the same name or birthdate, there will **always** be enough entries to construct the family tree (all people's names and birthdates are known, and they have **at least one** connection to another person in the tree).

After the command "**End**" is received, you should print all information about the person whose name or birthdate you received on the first line – his **name, birthday, parents, and children** (check the examples for the format). The people in the parents and children lists should be **ordered** by their **first appearance** in the input (regardless if they appeared as a birthdate or a name, for example, in the first input, Peter is before Sara because he first appeared in the first line, while she appears in the third).

Examples

Input	Output
Peter Johnson 11/11/1951 - 23/5/1980 Sara Johnson - 23/5/1980 Sara Johnson 9/2/1953 Peter Johnson - George Johnson George Johnson 1/1/2005 Sam Johnson 11/11/1951 Peter Johnson 23/5/1980 End	Peter Johnson 23/5/1980 Parents: Sam Johnson 11/11/1951 Sara Johnson 9/2/1953 Children: George Johnson 1/1/2005
13/12/1993 25/3/1934 - 4/4/1961 Peter Samthon 25/3/1934 4/4/1961 - George Samthon Sam Samthon - Sara Samthon George Samthon 13/12/1993 Sara Samthon 7/7/1995 Sam Samthon 4/4/1961 End	George Samthon 13/12/1993 Parents: Sam Samthon 4/4/1961 Children:

9. ** Cat Lady

Sara has many cats in her house of various breeds, since some breeds have specific characteristics, Sara needs some way to catalog the cats. Help her by creating a class hierarchy with all her breeds of cats so that she can easily check their characteristics.

Sara has 3 specific breeds of cats "**Siamese**", "**Cymric**" and the very famous Bulgarian breed "**Street Extraordinaire**", each breed has a **specific** characteristic about which information should be kept. For the Siamese cats their **ear size** should be kept, for Cymric cats - the **length** of **their fur** in millimeters and for the Street Extraordinaire, the **decibels** of their **meowing** during the night.

From the console, you will receive lines of cat information until the command "**End**" is received, the information will come in one of the following formats:

- "**Siamese {name} {earSize}**"
- "**Cymric {name} {furLength}**"
- "**StreetExtraordinaire {name} {decibelsOfMeows}**"

On the last line, after the "**End**" command, you will receive the **name** of a cat, and you should print that cat. Round the numbers **two digits** after the decimal separator.

Examples

Input	Output
StreetExtraordinaire Maca 85 Siamese Sim 4 Cymric Tom 28 End Tom	Cymric Tom 28.00
StreetExtraordinaire Koti 80 StreetExtraordinaire Maca 100 Cymric Tim 31 End Maca	StreetExtraordinaire Maca 100.00

Hint

Use class inheritance to represent the cat hierarchy and override the **ToString()** methods of concrete breeds to allow for easy printing of the cat, regardless of what breed it is.