**More Exercises: Objects and Classes**

Problems for exercise and homework for the ["C# Fundamentals" course @ SoftUni](https://softuni.bg/trainings/3606/programming-fundamentals-with-csharp-january-2022)  
You can check your solutions in [Judge](https://judge.softuni.org/Contests/1325/Objects-and-Classes-More-Exercise)

* **Company Roster**

Define a class **Employee** that holds the following information: a **name, a salary,** and a **department**.

Your task is to write a program, which takes **N** lines of employees from the console, calculates the department with the highest average salary, and prints for each employee in that department his **name and salary** – **sorted by salary in descending order**. The **salary** should be rounded to **two digits** after the decimal separator.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Peter 120.00 Development  Tony 333.33 Marketing  Jony 840.20 Development  George 0.20 Nowhere | Highest Average Salary: Development  Jony 840.20  Peter 120.00 |
| 6  Suny 496.37 Coding  Johny 610.13 Sales  Teo 609.99 Sales  Vlady 0.02 BeerDrinking  Andrey 700.00 Coding  Popeye 13.3333 SpinachGroup | Highest Average Salary: Sales  Johny 610.13  Teo 609.99 |

* **Oldest Family Member**

Create two classes – **Family** and **Person**. The **Person** class should have **Name** and **Age** properties. The **Family** class should have a **list of people**, a method for adding members (**void AddMember(Person member)**), and a method, which returns the oldest family member(**Person GetOldestMember()**). Write a program that reads the names and ages of **N** people and **adds them to the family**. Then **print** the **name** and **age** of the oldest member.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Peter 3  George 4  Annie 5 | Annie 5 |
| 5  Steve 10  Christopher 15  Annie 4  John 35  Maria 34 | John 35 |

* **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 a track of a car’s **model, fuel amount, fuel consumption per kilometer,** and **traveled distance**. A Car’s 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 about cars in the following format "**<Model> <FuelAmount> <FuelConsumptionFor1km>**". 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 or not a car can move that distance. If it can, the car’s **fuel amount** should be **reduced** by the amount of **used** **fuel** and its **traveled** **distance** should be increased by the number of the **traveled kilometers**. Otherwise, the car should not move (its fuel amount and the traveled distance 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**, its **current fuel amount** and the **traveled** **distance** in the format **"<Model> <fuelAmount> <distanceTraveled>"**. Print the fuel amount rounded to **two digits** after the decimal 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 |

* **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 a piece of information about the **model, engine,** and **cargo**. The **Engine** and **Cargo** shouldbe **separate classes**. Create a constructor that receives all of the information about the **Car** and creates and initializes its inner components (engine and cargo).

On the first line of input, you will receive a number **N** – the number of cars you have. On each of the next **N** lines, you will receive the following information about a car: **"<Model> <EngineSpeed> <EnginePower> <CargoWeight> <CargoType>"**, where the **speed**, **power** and **weight** are all **integers.**

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 **cargo,** whose **weight** **< 1000**. If the command is **"flamable"** print all of the cars whose **Cargo Type is "flamable"** and has **Engine Power > 250**. The cars should be printed in order of appearing in the input.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  ChevroletAstro 200 180 1000 fragile  Citroen2CV 190 165 900 fragile  fragile | Citroen2CV |
| 4  ChevroletExpress 215 255 1200 flamable  ChevroletAstro 210 230 1000 flamable  DaciaDokker 230 275 1400 flamable  Citroen2CV 190 165 1200 fragile  flamable | ChevroletExpress  DaciaDokker |

* **Shopping Spree**

Create two classes: **class** **Person** and **class** **Product**. Each person should have a **name**, **money,** and a **bag** **of products**. Each product should have a **name** and a **cost**.

Create a program, in which **each command** corresponds to a **person buying a product**. If the person can **afford** a product, **add** it to his bag. If a person **doesn’t have enough** money, print an **appropriate** **message**: **"{Person} can't afford {Product}"**.

On the **first two lines,** you are given **all people** and **all products**. After all, purchases, print **every person** in the order of **appearance** and **all products** that he has **bought,** also in order of **appearance**. If **nothing was bought**, print the name of the person followed by **"Nothing bought"**.

**Examples**

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter=11;George=4  Bread=10;Milk=2;  Peter Bread  George Milk  George Milk  Peter Milk  END | Peter bought Bread  George bought Milk  George bought Milk  Peter can't afford Milk  Peter - Bread  George - Milk, Milk |
| Maria=0  Coffee=2  Maria Coffee  END | Maria can't afford Coffee  Maria - Nothing bought |