# Defining Classes

## Define a Class Person

**NOTE**: You need a StartUp class with the namespace DefiningClasses.

Define a class **Person** with **private** fields for **name** and **age** and **public** properties **Name** and **Age**.

### Bonus\*

Try to create a few objects of type Person:

|  |  |
| --- | --- |
| **Name** | **Age** |
| Pesho | 20 |
| Gosho | 18 |
| Stamat | 43 |

Use both the inline initialization and the default constructor.

## Creating Constructors

**NOTE**: You need a StartUp class with the namespace DefiningClasses.

Add **3** constructors to the **Person** class from the last task. Use constructor chaining to reuse code:

* The **first** should take **no arguments** and produce a person with name "**No name**" and **age** **= 1**.
* The **second** should accept only an integer **number** for the **age** and produce a person with name "**No name**" and **age** equal to the passed **parameter**.
* The **third** one should accept a **string** for the **name** and an integer for the **age** and should produce a person with the given **name** and **age**.

## Oldest Family Member

Use your **Person** **class** from the previous tasks. Create a class **Family**. The class should have a **list of people**, a method for adding members - **void AddMember(Person member)** and a method returning 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  Pesho 3  Gosho 4  Annie 5 | Annie 5 |
| 5  Steve 10  Christopher 15  Annie 4  Ivan 35  Maria 34 | Ivan 35 |

## Opinion Poll

Using the **Person** class, 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**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Pesho 12  Stamat 31  Ivan 48 | Ivan - 48  Stamat - 31 |
| 5  Nikolai 33  Yordan 88  Tosho 22  Lyubo 44  Stanislav 11 | Lyubo - 44  Nikolai - 33  Yordan - 88 |

## Speed Racing

Write a program that keeps track of **cars** and their **fuel** and supports methods for **moving** the cars. Define a class **Car**. Each Car has the following properties:

* **string Model**
* **double FuelAmount**
* **double FuelConsumptionPer**K**ilometer**
* **double Travelled 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 a car 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** and its **current fuel amount** and the **traveled** **distance** in the format:

**"{model} {fuelAmount} {distanceTraveled}**"

Print the fuel amount formatted **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

Write a program that tracks **cars** and their **cargo**. Define a class **Car** that holds an information about **model, engine, cargo** and a **collection of exactly 4 tires**. The **engine**, **cargo** and **tire** shouldbe **separate classes**. Create a **constructor** that receives all of the information about the **Car** and creates and **initializes** its inner **components** (**engine**, **cargo** and **tires**).

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 an information about each car in the format:

"**{model} {engineSpeed} {enginePower} {cargoWeight} {cargoType} {tire1Pressure} {tire1Age} {tire2Pressure} {tire2Age} {tire3Pressure} {tire3Age} {tire4Pressure} {tire4Age}"**

The **speed**, **power**, **weight** and **tire age** are **integers** and **tire** **pressure** is a **double.**

After the **N** lines, you will receive a single line with one of the following commands:

* "**fragile**" - print all cars whose **cargo** is **"fragile"** with **a tire**, whose **pressure is** **< 1**
* "**flamable**" - print all of the cars, whose **cargo** 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 |

## Car Salesman

Define two classes **Car** and **Engine.**

**Car** has the following properties:

* **Model**
* **Engine**
* **Weight**
* **Color**

Engine has the following properties:

* **Model**
* **Power**
* **Displacement**
* **Efficiency**

A Car’s **weight** and **color** and its Engine’s **displacement** 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, you will receive a number **M**. On each of the next **M** lines, an information about a **Car** will follow in the format:

**"{model} {engine} {weight} {color}"**

The engine 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** all the **cars** in the order they were received and their information in the format defined bellow. If any of the optional fields is missing, print "**n/a**" in its place:

{CarModel}:  
 {EngineModel}:  
 Power: {EnginePower}  
 Displacement: {EngineDisplacement}  
 Efficiency: {EngineEfficiency}  
 Weight: {CarWeight}  
 Color: {CarColor}

### Bonus\*

Override the classes’ **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 |

#### Submission

Zip all the files in the project folder except **bin** and **obj** folders.