# Exercises: Defining Classes

This document defines the **exercise assignments** for the ["CSharp DB Advanced" course @ Software University](https://softuni.bg/trainings/1741/databases-advanced-entity-framework-october-2017).

## Define a class Person

Define a class **Person** with **public** properties for **Name** and **Age**.

### Note

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

|  |
| --- |
| static void Main(string[] args)  {  Type personType = typeof(Person);  PropertyInfo[] properties = personType.GetProperties  (BindingFlags.Public | BindingFlags.Instance);  Console.WriteLine(properties.Length);  } |

If you defined the class correctly, the test should pass.

### 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. Your fields should be **public**, otherwise you won’t see them.

## Creating Constructors

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

1. The first should take **no arguments** and produce a person with **name** “**No name**” and **age** = **1**.
2. 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.
3. 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**.

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

|  |
| --- |
| Type personType = typeof(Person);  ConstructorInfo emptyCtor = personType.GetConstructor(new Type[] { });  ConstructorInfo ageCtor = personType.GetConstructor(new[] { typeof(int) });  ConstructorInfo nameAgeCtor = personType.GetConstructor(new[] { typeof(string), typeof(int) });  bool swapped = false;  if (nameAgeCtor == null)  {  nameAgeCtor = personType.GetConstructor(new[] { typeof(int), typeof(string) });  swapped = true;  }  string name = Console.ReadLine();  int age = int.Parse(Console.ReadLine());  Person basePerson = (Person)emptyCtor.Invoke(new object[] { });  Person personWithAge = (Person)ageCtor.Invoke(new object[] { age });  Person personWithAgeAndName = swapped ? (Person)nameAgeCtor.Invoke(new object[] { age, name }) :(Person)nameAgeCtor.Invoke(new object[] { name, age });  Console.WriteLine("{0} {1}", basePerson.Name, basePerson.Age);  Console.WriteLine("{0} {1}", personWithAge.Name, personWithAge.Age);  Console.WriteLine("{0} {1}", personWithAgeAndName.Name, personWithAgeAndName.Age); |

If you defined the constructors correctly, the test should pass.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho  20 | No name 1  No name 20  Pesho 20 |
| Gosho  18 | No name 1  No name 18  Gosho 18 |
| Stamat  43 | No name 1  No name 43  Stamat 43 |

## 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

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’s **Model, fuel amount, fuel consumption for 1 kilometer** and **distance traveled**. 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 for 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 a command in the following format “**Drive <CarModel> <amountOfKm>**”. Implement a method in the **Car** class to calculate whether 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 **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** 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 ride  AudiA4 1.00 50  BMW-M2 33.00 0  Ferrari-488Spider 4.41 97 |

## 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 which takes **N** lines of 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. The **salary** should be printed to **two decimal places** after the seperator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Pesho 120.00 Dev Development pesho@abv.bg 28  Toncho 333.33 Manager Marketing 33  Ivan 840.20 ProjectLeader Development ivan@ivan.com  Gosho 0.20 Freeloader Nowhere 18 | Highest Average Salary: Development  Ivan 840.20 ivan@ivan.com -1  Pesho 120.00 pesho@abv.bg 28 |
| 6  Stanimir 496.37 Temp Coding stancho@yahoo.com  Yovcho 610.13 Manager Sales  Toshko 609.99 Manager Sales toshko@abv.bg 44  Venci 0.02 Director BeerDrinking beer@beer.br 23  Andrei 700.00 Director Coding  Popeye 13.3333 Sailor SpinachGroup popeye@pop.ey | Highest Average Salary: Sales  Yovcho 610.13 n/a -1  Toshko 609.99 toshko@abv.bg 44 |