# Exercises: Classes and Objects

You can check your solutions in **Judge system**: <https://judge.softuni.bg/Contests/3161/Classes-and-Objects>

## Car

**Note**: you need a **public** StartUp class with the namespace CarManufacturer.

Create a **public** **class** named Car. The class should have **private** **fields** for:

* make: string
* model: string
* year: int

The class should also have **public** **properties** for:

* Make: string
* Model: string
* Year: int

You should be able to use the class like this:

Graphical user interface, text, application

Description automatically generated

## Car: Add More Members

**Note**: you need a **public** StartUp class with the namespace CarManufacturer.

Create a class **public** Car with additional members (you can use the class from the previous task).

The class should have private fields for:

* make: string
* model: string
* year: int
* fuelQuantity: double
* fuelConsumption: double

The class should also have properties for:

* Make: string
* Model: string
* Year: int
* FuelQuantity: double
* FuelConsumption: double

The class should also have methods for:

* Drive(double distance): void – this method checks if the car fuel quantity minus the distance multiplied by the car fuel consumption is bigger than zero. If it is remove from the fuel quantity the result of the multiplication between the distance and the fuel consumption. Otherwise write on the console the following message:

"Not enough fuel to perform this trip!"

* WhoAmI(): string – returns the following message:

"Make: {this.Make}\nModel: {this.Model}\nYear: {this.Year}\nFuel: {this.FuelQuantity:F2}L"

You should be able to use the class like this:

Text

Description automatically generated

## Car Constructors

Using the class from the previous problem create one parameterless constructor with default values:

* Make – VW
* Model – Golf
* Year – 2025
* FuelQuantity – 200
* FuelConsumption – 10

Create a second constructor accepting make, model and year upon initialization and calls the base constructor with its default values for fuelQuantity and fuelConsumption.

Text, letter

Description automatically generated

Create a third constructor accepting make, model, year, fuelQuantity and fuelConsumption upon initialization and reuses the second constructor to set the make, model and year values.

Text

Description automatically generated with medium confidence

Go to **StartUp.cs** file and make 3 different instances of the **Class Car**, using the **different** overloads of the constructor.

Text

Description automatically generated

## Car Engine and Tires

Using the Car class, you already created, define another class Engine.

The class should have private fields for:

* horsePower: int
* cubicCapacity: double

The class should also have properties for:

* HorsePower: int
* CubicCapacity: double

The class should also have a constructor, which accepts **horsepower** and **cubicCapacity** upon initialization:

Text

Description automatically generated

Now create a class Tire.

The class should have private fields for:

* year: int
* pressure: double

The class should also have properties for:

* Year: int
* Pressure: double

The class should also have a constructor, which accepts **year** and **pressure** upon initialization:

Text

Description automatically generated with medium confidence

Finally, go to the Car class and create **private fields** and **public properties** for **Engine** and **Tire[]**.Create another constructor, which accepts make, model, year, fuelQuantity, fuelConsumption, Engine and Tire[] upon initialization:

Text

Description automatically generated

You should be able to use the classes like this:

Text

Description automatically generated

## Special Cars

This is the final and most interesting problem in this lab. Until you receive the command "No more tires", you will be given tire info in the format:

{year} {pressure}

{year} {pressure}

…

"No more tires"

You have to collect all the tires provided. Next, until you receive the command "Engines done" you will be given engine info and you also have to collect all that info.

{horsePower} {cubicCapacity}

{horsePower} {cubicCapacity}

…

The final step - until you receive "Show special", you will be given information about cars in the format:

{make} {model} {year} {fuelQuantity} {fuelConsumption} {engineIndex} {tiresIndex}

…

Every time you have to create a **new Car** with the information provided. The car engine is the provided **engineIndex** and the tires are **tiresIndex**. Finally, collect all the created cars. When you receive the command "Show special", drive 20 kilometers all the cars, which were manufactured during 2017 or after, have horse power above 330 and the sum of their tire pressure is between 9 and 10. Finally, print information about each special car in the following format:

"Make: {specialCar.Make}"

"Model: {specialCar.Model}"

"Year: {specialCar.Year}"

"HorsePowers: {specialCar.Engine.HorsePower}"

"FuelQuantity: {specialCar.FuelQuantity}"

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2 2.6 3 1.6 2 3.6 3 1.6  1 3.3 2 1.6 5 2.4 1 3.2  No more tires  331 2.2  145 2.0  Engines done  Audi A5 2017 200 12 0 0  BMW X5 2007 175 18 1 1  Show special | Make: Audi  Model: A5  Year: 2017  HorsePowers: 331  FuelQuantity: 197.6 |

## Employees

**NOTE**: You need a **public** class StartUp in the namespace ClassesEmployee.

Define a **public** classEmployee in the namespace ClassesEmployee with **private** fields for **name** and **age** and **public** properties Name and Age.

Create a few objects of type Employee:

|  |  |
| --- | --- |
| **Name** | **Age** |
| Dan | 20 |
| Joey | 18 |
| Tommy | 43 |

Use both the inline initialization and the default constructor.

## Creating Constructors

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

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

1. The first should take no arguments and produce an employee with name “**No name**” and age = **1**.
2. The second should accept only an integer number for the age and produce an employee 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 an employee with the given name and age.

## Oldest Employee

Use your **Employee class** from the previous tasks. Create a class **Department**. The class should have a **list of employees**, a method for adding members (**void AddMember(Employee member)**) and a method returning the oldest department member(**Employee GetOldest())**. Write a program that reads the names and ages of **N** people and **adds them to the department**. Then **print** the **name** and **age** of the oldest member.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  Nick 23  Jorge 34  Sophie 55 | Sophie 55 |  | 5  Steve 29  Christopher 25  Annie 24  Brian 35  Nicole 24 | Brian 35 |

## Opinion Poll

Using the **Employee** 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  Angela 22  Joshua 31  Connor 48 | Connor - 48  Joshua - 31 |
| 5  Molly 33  Peter 88  Paul 22  Johnny 44  Martin 21 | Johnny - 44  Molly - 33  Peter - 88 |

## Formula 1 Drivers

You need to write software that analyzes and provides information about Formula 1 drivers. On the first line you will read a number **N** which will specify how many lines with drivers you will receive. On each of the next **N** lines you will receive information about the driver. Define class **Driver.**

All drivers have:

* Name – a **string**
* Age – integer
* TotalTime – a **floating-point number**
* Speed – a **floating-point number**

You should print all of the information about the **driver with the best time** in the format defined bellow.

DriverName: { Name }

DriverAge: { Age }

Time: { TotalTime }

Speed: { Speed }

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Michael Schumacher 52 1.56 290  Lewis Hamilton 36 1.42 303  Sebastian Vettel 33 1.59 278 | DriverName: Lewis Hamilton  DriverAge: 36  Time: 1.42  Speed: 303 |
| 5  Kevin Magnussen 28 1.59 240  Nikita Mazepin 22 1.53 257  Charles Pic 31 1.47 265  Daniel Ricciardo 31 2.10 200  Nico Rosberg 35 1.50 260 | DriverName: Charles Pic  DriverAge: 31  Time: 1.47  Speed: 265 |
| 2  Alexander Rossi 29 1.37 273  Takuma Sato 44 1.54 262 | DriverName: Alexander Rossi  DriverAge: 29  Time: 1.37  Speed: 273 |

## Bank Account

You have the task to create a program that serves the bank accounts of bank customers. Create a new class named **BankAccount**.

This class will contains the following members:

AccountNumber – a **string**

OwnerName – a **string**

AccountBalance –decimal

Method public void MakeDeposit – This method accepts an amount, increases the balance and prints as string the new account balance in the following format:

* **"Account balance: { the new account balance }"**

Method public void MakeWithdrawal – This method accepts an amount, checks if there are enough funds for this withdrawal.

* In case there are not enough funds print: "Non-Sufficient Funds"
* If there are sufficient funds available print as string the new account balance in the following format: "Withdrawn funds: { amount of funds withdrawn}. Funds available on the account: { the amount of funds available }"

On the **first** **line** you will receive the **customer** **information**, separated by a single space. The account number will be first followed by the owner name and finally the account balance. Until the command “**End**” is received, you will receive commands for deposit or withdraw funds.

You will not receive invalid data or negative numbers.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3543653456 Jorge Cooper 587  Deposit 200  Withdrawal 100  End | Account balance: 787  Withdrawn funds: 100. Funds available on the account: 687 |
| 3963185629 Peter Davis 692  Withdrawal 700  Deposit 8  Withdrawal 40  Withdrawal 660  Withdrawal 200  End | Non-Sufficient Funds  Account balance: 700  Withdrawn funds: 40. Funds available on the account: 660  Withdrawn funds: 660. Funds available on the account: 0  Non-Sufficient Funds |

## Antique Bookstore

A friend of yours opens an antique bookstore and asks you to help him. He needs a program that store and provide information about the books. The program needs **class** **Book** that has:

Title – string

Author – string

PubDate – int (Publication Date)

Price – decimal

Publisher – string

Discount – string

**Books** **without** a **publisher** and **discount** can be created in the program. In these cases, the **default** **value** for the **publisher is** "Unknown" and for the **discount** is "Unavailable". There may be a book **with** a **publisher** and without a **discount**, but **not** a book **with** **no** **publisher** but **with** a **discount**.

Until you receive the "**End**" command, you will be given books with their information separated by ", ". First is the **title** followed by the **author**, **publication** **date**, **price** and **optional** **publisher** and **discount**.

After the "**End**" command is received you should print all of the book **sorted by the author** **in ascending order** in the format:

Title : {book title}

Author: {book author}

Publication Date: {PubDate}

Price: {book price}

Publisher: {publisher}

Discount: {discount}

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| The Years, Virginia Woolf, 1937, 195, Hogarth Press, 10%  The Lure of the Mask, Harold MacGrath, 1908, 200  Ask the Dust, John Fante, 1939, 185, Stackpole Sons, 15%  End | Title : The Lure of the Mask  Author: Harold MacGrath  Publication Date: 1908  Price: 200  Publisher: Unknown  Discount: Unavailable  Title : Ask the Dust  Author: John Fante  Publication Date: 1939  Price: 185  Publisher: Stackpole Sons  Discount: 15%  Title : The Years  Author: Virginia Woolf  Publication Date: 1937  Price: 195  Publisher: Hogarth Press  Discount: 10% |
| Star Maker, Olaf Stapledon, 1937, 215, Methuen & Co  Foundation, Isaac Asimov, 1951, 199, Gnome, 7%  Stuart Little, E.B. White, 1945, 259, Harper and Brothers, 5%  Moonraker, Ian Fleming, 1955, 199  End | Title : Stuart Little  Author: E.B. White  Publication Date: 1945  Price: 259  Publisher: Harper and Brothers  Discount: 5%  Title : Moonraker  Author: Ian Fleming  Publication Date: 1955  Price: 199  Publisher: Unknown  Discount: Unavailable  Title : Foundation  Author: Isaac Asimov  Publication Date: 1951  Price: 199  Publisher: Gnome  Discount: 7%  Title : Star Maker  Author: Olaf Stapledon  Publication Date: 1937  Price: 215  Publisher: Methuen & Co  Discount: Unavailable |

### **Hints**:

Use multiple constructors.