# Exercises: OOP Principles Part 2

Problems for the ["Fund. Of Programming" course @ Software University](https://softuni.bg/trainings/2133/fundamentals-of-programming-with-c-sharp-november-2018)

You can check your solutions here: [Judge](https://judge.softuni.bg/Contests/1456/OOP-Principles)

# Part III: Interfaces & Abstraction

## 10. Shapes

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

Build hierarchy of interfaces and classes:



You should be able to use the class like this:

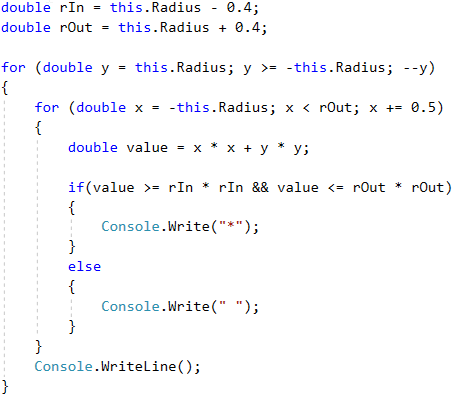
|  |
| --- |
| StartUp.cs |
| var radius = int.Parse(Console.ReadLine());  IDrawable circle = new Circle(radius);  var width = int.Parse(Console.ReadLine());  var height = int.Parse(Console.ReadLine());  IDrawable rect = new Rectangle(width, height);  circle.Draw();  rect.Draw(); |

### Examples

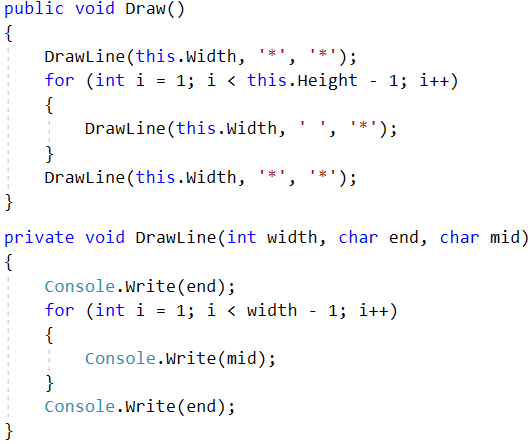
|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  4  5 | \*\*\*\*\*\*\*  \*\* \*\*  \*\* \*\*  \* \*  \*\* \*\*  \*\* \*\*  \*\*\*\*\*\*\*  \*\*\*\*  \* \*  \* \*  \* \*  \*\*\*\* |

### Solution

The algorithm for drawing a circle is:



The algorithm for drawing a rectangle is:



## 11. Cars

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

Build a hierarchy of interfaces and classes:



Your hierarchy must be used with this code:

|  |
| --- |
| StartUp.cs |
| ICar seat = new Seat("Leon", "Grey");  ICar tesla = new Tesla("Model 3", "Red", 2);  Console.WriteLine(seat.ToString());  Console.WriteLine(tesla.ToString()); |

### Examples

|  |
| --- |
| **Output** |
| Grey Seat Leon  Engine start  Breaaak!  Red Tesla Model 3 with 2 Batteries Engine start  Breaaak! |

## 12. Ferrari

Model an application which contains a **class Ferrari** and an **interface**. Your task is simple, you have a **car - Ferrari**, its model is **"****488-Spider"** and it has a **driver**. Your Ferrari should have functionality to **use brakes** and **push the gas pedal**. When the **brakes** are pushed down **print "****Brakes!"**, and when the **gas pedal** is pushed down - **"****Zadu6avam sA!"**. As you may have guessed this functionality is typical for all cars, so you should **implement an interface** to describe it.

Your task is to **create a Ferrari** and **set the driver's name** to the passed one in the input. After that, print the info. Take a look at the Examples to understand the task better.

### Input

On the **single input line**, you will be given the **driver's name**.

### Output

On the **single output line**, print the model, the messages from the brakes and gas pedal methods and the driver's name. In the following format:

<**model**>/<**brakes**>/<**gas** **pedal**>/<**driver's** **name**>

### Constraints

The input will always be valid, no need to check it explicitly! The Driver's name may contain any ASCII characters.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bat Giorgi | 488-Spider/Brakes!/Zadu6avam sA!/Bat Giorgi |
| Dinko | 488-Spider/Brakes!/Zadu6avam sA!/Dinko |

# Part IV: Interfaces & Abstraction

## 13. MathOperation

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

Create a class **MathOperations**, which should have 3 times method Add(). Method Add() has to be invoked with:

* Add(int, int): **int**
* Add(double, double, double): **double**
* Add(decimal, decimal, decimal): **decimal**

You should be able to use the class like this:

|  |
| --- |
| StartUp.cs |
| public static void Main()  {  MathOperations mo = new MathOperations();  Console.WriteLine(mo.Add(2, 3));  Console.WriteLine(mo.Add(2.2, 3.3, 5.5));  Console.WriteLine(mo.Add(2.2m, 3.3m, 4.4m));  } |

### Examples

|  |
| --- |
| Output |
| 5  11  9.9 |

### Solution

Created MathOperation class should look like this:



## 14. Animals

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

Create a class Animal, which holds two fields:

* name: string
* favouriteFood: string

Animal has one virtual method ExplainSelf()**: string.**You should add two new classes - **Cat** and **Dog. Override** the ExplainSelf() method by adding concrete animal sound on a new line. (Look at examples below)

You should be able to use the class like this:

|  |
| --- |
| StartUp.cs |
| Animal cat = new Cat("Pesho", "Whiskas");  Animal dog = new Dog("Gosho", "Meat");  Console.WriteLine(cat.ExplainSelf());  Console.WriteLine(dog.ExplainSelf()); |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
|  | I am Pesho and my fovourite food is Whiskas  MEEOW  I am Gosho and my fovourite food is Meat  DJAAF |

### Solution





## 15. Vehicles

Write a program that models 2 vehicles (a **Car** and a **Truck**) and simulates **driving** and **refueling** them. **Car** and **truck** both have **fuel quantity**, **fuel consumption** **in liters** **per km** and can be **driven a given distance** and **refueled with a given amount of fuel.** It’s summer, so both vehicles use air conditioners and their **fuel consumption** per km is **increased** by **0.9** liters for the **car** and with **1.6** liters for the **truck**. Also, the **truck** has a tiny hole in its tank and when its **refueled** it keeps only **95%** of the given **fuel**. The **car** has no problems and adds **all the given fuel to its tank.** If a vehicle cannot travel the given distance, its fuel does not change.

### **Input**

* On the first line – information about the car in the format: "Car {fuel quantity} {liters per km}"
* On the second line – info about the truck in the format: "Truck {fuel quantity} {liters per km}"
* On the third line – the number of commands N that will be given on the next N lines
* On the next N lines – commands in the format:
* "Drive Car {distance}"
* "Drive Truck {distance}"
* "Refuel Car {liters}"
* "Refuel Truck {liters}"

### **Output**

* After each Drive command, if there was enough fuel, print on the console a message in the format:
* "Car/Truck travelled {distance} km"
* If there was not enough fuel, print: "Car/Truck needs refueling"
* After the End command, print the remaining fuel for both the car and the truck, rounded to 2 digits after the floating point in the format:
* **"Car: {liters}"**
* **"Truck: {liters}"**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Car 15 0.3  Truck 100 0.9  4  Drive Car 9  Drive Car 30  Refuel Car 50  Drive Truck 10 | Car travelled 9 km  Car needs refueling  Truck travelled 10 km  Car: 54.20  Truck: 75.00 |
| Car 30.4 0.4  Truck 99.34 0.9  5  Drive Car 500  Drive Car 13.5  Refuel Truck 10.300  Drive Truck 56.2  Refuel Car 100.2 | Car needs refueling  Car travelled 13.5 km  Truck needs refueling  Car: 113.05  Truck: 109.13 |