Exercises: Polymorphism

You can check your solutions in **Judge system**: https://judge.softuni.bg/Contests/3167/Polymorphism

1. 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:

```
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:

```
public int Add(int a, int b)
{
    return a + b;
}

public double Add(double a, double b, double c)
{
    return a + b + c;
}

public decimal Add(decimal a, decimal b, decimal c)
{
    return a + b + c;
}
```

2. 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:

```
Animal cat = new Cat("Pesho", "Whiskas");
Animal dog = new Dog("Gosho", "Meat");

Console.WriteLine(cat.ExplainSelf());
Console.WriteLine(dog.ExplainSelf());
```

Examples

Output
I am Pesho and my fovourite food is Whiskas MEEOW I am Gosho and my fovourite food is Meat DJAAF

Solution

```
public abstract class Animal
{
    2 references
    public string Name { get; protected set; }

    2 references
    public string FavoriteFood { get; protected set; }

    0 references
    protected Animal(string name, string favoriteFood)
    {
        this.Name = name;
        this.FavoriteFood = favoriteFood;
    }

    0 references
    public virtual string ExplainSelf()
    {
        return $"I am {this.Name} and my fovourite food is {this.FavoriteFood}";

    }
}
```

```
public class Cat : Animal
{
    0 references
    public Cat(string name, string favouriteFood) : base(name, favouriteFood)
    {
    }

    4 references
    public override string ExplainSelf()
    {
        return base.ExplainSelf() + Environment.NewLine + "MEEOW";
    }
}
```

3. Shapes

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

Create a class hierarchy, starting with **abstract** class **Shape**:

- **Abstract methods:**
  - **CalculatePerimeter(): doulbe**
  - **CalculateArea(): double**
- **Virtual methods:**
  - **Draw(): string**

Extend the **Shape** class with two children:

- **Rectangle**
- **Circle**

Each of them need to have:

- **Fields:**
  - **height and width for Rectangle**
  - **radius for Circle**
- **Encapsulation for these fields**
- **A public constructor**
- **Concrete methods for calculations (perimeter and area)**
- **Override methods for drawing**

4. 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.** Its 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 it's **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

5. Vehicles Extension

Use your solution of the **previous** task for the starting point and add more functionality. Add a new vehicle – **Bus**. Add to every **vehicle** a new property – **tank capacity**. A vehicle cannot **start with** or **refuel above** its **tank capacity**.

If you **try to put more fuel** in the tank than the **available space,** print on the console **"Cannot fit {fuel amount} fuel in the tank"** and **do not add any fuel** in the vehicle's tank. If you try to **create** a vehicle with **more fuel** than its **tank capacity**, **create** it but start with an **empty tank**.

Add a **new command** for the bus. You can **drive** the **bus with or without people**. **With people**, the **air-conditioner is turned on** and its **fuel consumption** per kilometer is **increased by 1.4 liters**. If there are **no people in the bus**, the air-conditioner is **turned off** and **does not increase** the fuel consumption.

Finally, add a **validation** for the **amount** of **fuel** given to the **Refuel command** – if it is 0 or negative, print **"Fuel must be a positive number"**.

Input

- On the **first three lines** you will receive information about the vehicles in the format:
  - `"Vehicle {initial fuel quantity} {liters per km} {tank capacity}"`
- **On the fourth line – the number of commands N that will be given on the next N lines**
- **On the next N lines – commands in format:**
  - `"Drive Car {distance}"`
  - `"Drive Truck {distance}"`
  - `"Drive Bus {distance}"`
  - `"DriveEmpty Bus {distance}"`
  - `"Refuel Car {liters}"`
  - `"Refuel Truck {liters}"`
  - `"Refuel Bus {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"`
- **If you try to refuel with an amount ≤ 0** print:
  - `"Fuel must be a positive number"`
- **If the given fuel cannot fit in the tank, print:**
  - `"Cannot fit {fuel amount} fuel in the tank"`
- **After the End command, print the remaining fuel for all vehicles, rounded to 2 digits after the floating point in the format:**
  - `"Car: {liters}"`
  - `"Truck: {liters}"`
  - `"Bus: {liters}"`

Example

Input	Output
Car 30 0.04 70 Truck 100 0.5 300 Bus 40 0.3 150 8 Refuel Car -10 Refuel Truck 0 Refuel Car 10 Refuel Car 300 Drive Bus 10 Refuel Bus 1000 DriveEmpty Bus 100 Refuel Truck 1000	Fuel must be a positive number Fuel must be a positive number Cannot fit 300 fuel in the tank Bus travelled 10 km Cannot fit 1000 fuel in the tank Bus needs refueling Cannot fit 1000 fuel in the tank Car: 40.00 Truck: 100.00 Bus: 23.00

6. Raiding

Your task is to create a class hierarchy like the described below. The **BaseHero** class should be abstract.

- **`BaseHero – string Name, int Power, string CastAbility()`**

- **Druid – power = 80**
- **Paladin – power = 100**
- **Rogue – power = 80**
- **Warrior – power = 100**

Each hero should override the **CastAbility()** method:

Druid – "{Type} – {Name} healed for {Power}"

Paladin – "{Type} – {Name} healed for {Power}"

Rogue – "{Type} – {Name} hit for {Power} damage"

Warrior – "{Type} – {Name} hit for {Power} damage"

Now use the classes you created to form a raid group and defeat a boss. You will receive an integer **N** from the console. On the next lines you will receive **{heroName}** and **{heroType}** until you create **N** amount of heroes. If the hero type is invalid print: **"Invalid hero!"** and don't add it to the raid group. After the raid group is formed you will receive an integer from the console which will be the boss's power. Then each of the heroes in the raid group should cast his ability once. You should sum the power of all of the heroes and if the total power is greater or equal to the boss's power you have defeated him and you should print:

"Victory!"

Else print:

"Defeat..."

Bonus*

Use the Factory Design pattern to instantiate the classes.

Constraints

You need to create heroes until you have **N** amount of **valid** heroes.

Example

Input	Output
3 Mike Paladin Josh Druid Scott Warrior 250	Paladin - Mike healed for 100 Druid - Josh healed for 80 Warrior - Scott hit for 100 damage Victory!
2 Mike Warrior Tom Rogue 200	Warrior - Mike hit for 100 damage Rogue - Tom hit for 80 damage Defeat...

7. Wild Farm

Your task is to create a **class hierarchy** like the **described below**. The **Animal**, **Bird**, **Mammal**, **Feline** and **Food** classes should be **abstract**. Override the method **ToString()**.

- `Food – int Quantity;`
  - `Vegetable;`
  - `Fruit;`
  - `Meat;`
  - `Seeds;`
- `Animal – string Name, double Weight, int FoodEaten;`
  - `Bird – double WingSize;`
    - `Owl;`
    - `Hen;`
  - `Mammal – string LivingRegion;`
    - `Mouse;`
    - `Dog;`
    - `Feline – string Breed;`
      - `Cat;`
      - `Tiger;`

All **animals** should also have the **ability** to ask for food by **producing** a **sound**.

- `Owl – "Hoot Hoot";`
- `Hen – "Cluck";`
- `Mouse – "Squeak";`
- `Dog – "Woof!";`
- `Cat – "Meow";`
- `Tiger – "ROAR!!!";`

Now use the **classes that** you have created to **instantiate** some **animals** and **feed them**. Input should be read from the console. Every **even** line (starting from 0) will **contain information** about an **animal** in the following format:

- `Felines - "{Type} {Name} {Weight} {LivingRegion} {Breed}";`
- `Birds - "{Type} {Name} {Weight} {WingSize}";`
- `Mice and Dogs - "{Type} {Name} {Weight} {LivingRegion}";`

On the **odd** lines, you will receive **information** about a piece of **food** that you should **give** to that **animal**. The line will consist of a **FoodType** and **quantity**, separated by a whitespace.

Animals will only eat a certain type of food, as follows:

- **Hens** eat **everything**;
- **Mice** eat **vegetables** and **fruits**;
- **Cats** eat **vegetables** and **meat**;
- **Tigers**, **Dogs** and **Owls** eat **only meat**;

If you try to give an animal a different type of food, it will not eat it and you should print:

- `"{AnimalType} does not eat {FoodType}!"`

The **weight** of an **animal** will **increase** with **every piece** of **food** it **eats**, as follows:

- **Hen – 0.35;**
- **Owl – 0.25;**
- **Mouse – 0.10;**
- **Cat – 0.30;**
- **Dog – 0.40;**
- **Tiger – 1.00;**

Override the **ToString()** method to print the information about an animal in the formats:

- **Birds – "{AnimalType} [{AnimalName}, {WingSize}, {AnimalWeight}, {FoodEaten}]"**
- **Felines – "{AnimalType} [{AnimalName}, {Breed}, {AnimalWeight}, {AnimalLivingRegion}, {FoodEaten}]"**
- **Mice and Dogs – "{AnimalType} [{AnimalName}, {AnimalWeight}, {AnimalLivingRegion}, {FoodEaten}]"**

After you have read the **information** about the **animal** and the **food**, the **animal** will **produce a sound** (**print** it on the **console**). Next, you should **try** to **feed** it. After receiving the "**End**" command, **print** information about **every animal** in **order** of **input**.

Example

Input	Output
Cat Pesho 1.1 Home Persian Vegetable 4 End	Meow Cat [Pesho, Persian, 2.3, Home, 4]
Tiger Typcho 167.7 Asia Bengal Vegetable 1 Dog Doncho 500 Street Vegetable 150 End	ROAR!!! Tiger does not eat Vegetable! Woof! Dog does not eat Vegetable! Tiger [Typcho, Bengal, 167.7, Asia, 0] Dog [Doncho, 500, Street, 0]
Mouse Jerry 0.5 Anywhere Fruit 1000 Owl Toncho 2.5 30 Meat 5 End	Squeak Hoot Hoot Mouse [Jerry, 100.5, Anywhere, 1000] Owl [Toncho, 30, 3.75, 5]