**Exercises: Polymorphism**

Problems for exercises and homework for the ["C# OOP" course @ SoftUni".](https://softuni.bg/trainings/3585/csharp-oop-february-2022)

You can check your solutions here: <https://judge.softuni.org/Contests/1504/Polymorphism-Exercise>

**1. Vehicles**

Create 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 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 |  | Car travelled 9 km | | | |  |  |
|  | Truck 100 0.9 |  | Car needs refueling | | | |  |  |
| 4 | |  | Truck travelled 10 km | | | |  |  |
|  | Drive Car 9 |  | Car: 54.20 | | | |  |  |
|  | Drive Car 30 |  | Truck: 75.00 | | | |  |  |
|  | Refuel Car 50 |  |  |  |  |  |  |  |
|  | Drive Truck 10 |  |  |  |  |  |  |  |
|  | Car 30.4 0.4 |  | Car needs refueling | | | |  |  |
|  | Truck 99.34 0.9 |  | Car travelled 13.5 km | | | |  |  |
| 5 | |  | Truck needs refueling | | | |  |  |
|  | Drive Car 500 |  | Car: 113.05 | | | |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  | © SoftUni – [about.softuni.bg.](https://about.softuni.bg/) Copyrighted document. Unauthorized copy, reproduction or use is not permitted. | | | | | |  |
|  |  | Follow us: | |  |  | | Page 1 of 6 |  |



Drive Car 13.5 Refuel Truck 10.300 Drive Truck 56.2 Refuel Car 100.2

Truck: 109.13

**2. 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, fuel in the tank"** and **do not add any fuel** in the vehicle’s its **tank capacity**, **create** it but start with an **empty tank**.

print on the console **"Cannot fit {fuel amount}** tank. If you try to **create** a vehicle with **more fuel** than

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 on the bus**, the air-conditioner is **turned off** and **does not increase** the fuel consumption.

Finally, add **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}"**



© SoftUni – [about.softuni.bg.](https://about.softuni.bg/) Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us:  Page 2 of 6

**Example**

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

**3. Raiding**

Your task is to create a class hierarchy like the one 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** number 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](https://www.c-sharpcorner.com/article/factory-method-design-pattern-in-c-sharp/) Design pattern to instantiate the classes.



© SoftUni – [about.softuni.bg.](https://about.softuni.bg/) Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us:  Page 3 of 6

**Constraints**

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

**Example**

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

**4. Wild Farm**

Your task is to create a **class hierarchy** like the one **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"**



© SoftUni – [about.softuni.bg.](https://about.softuni.bg/) Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us:  Page 4 of 6

* **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 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 Sammy | 1.1 Home Persian | Meow |
| Vegetable 4 | | Cat [Sammy, Persian, 2.3, Home, 4] |
| End |  |  |
|  |  |  |
| Tiger Rex | 167.7 Asia Bengal | ROAR!!! |
|  |  |  |



© SoftUni – [about.softuni.bg.](https://about.softuni.bg/) Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us:  Page 5 of 6

|  |  |
| --- | --- |
| Vegetable 1 | Tiger does not eat Vegetable! |
| Dog Tommy 500 Street | Woof! |
| Vegetable 150 | Dog does not eat Vegetable! |
| End | Tiger [Rex, Bengal, 167.7, Asia, 0] |
|  | Dog [Tommy, 500, Street, 0] |
|  |  |
| Mouse Jerry 0.5 Anywhere | Squeak |
| Fruit 1000 | Hoot Hoot |
| Owl Tom 2.5 30 | Mouse [Jerry, 100.5, Anywhere, 1000] |
| Meat 5 | Owl [Tom, 30, 3.75, 5] |
| End |  |
|  |  |



© SoftUni – [about.softuni.bg.](https://about.softuni.bg/) Copyrighted document. Unauthorized copy, reproduction or use is not permitted.

Follow us:  Page 6 of 6