# Exercise: OOP Advanced Interfaces

You can check your solutions here: <https://judge.softuni.bg/Contests/472/Interfaces-and-Abstraction-Exercise>.

## Define an Interface Person

Define an interface **Person** with two methods that should be implemented by a class: setName() and setAge().

Define a class **Citizen** which implements **Person** and has a constructor which takes a **string** name and an **int** age and uses the methods given by the interface. Write the methods and add a magic method \_\_toString() which willl print the name and age of the person (example: Jackson, 35).

Create an instance of the class and use the magic method \_\_toString() to print the name and age of the person.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter  25 | Peter  25 |

## Multiple Implementation

Using the code from the previous task, define an interface **Identifiable** with a method setId(). In the class it will set a property called **Id** which is **string**.

Define an interface **Birthable** with a method setBirthdate(). In the class it will set a property **birthDate** which is **string**.

Implement setId() and setBirthdate() in the **Citizen** class. Rewrite the Citizen constructor to accept the new parameters. Rewrite \_\_toString() to output the persons Id and date of birth.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter  25  7706126789  12.06.1977 | Peter  25  7706126789  12.06.1977 |

### 2.1. Implementing a Method in Two Interfaces

Add a new method setIdBirtdate() which combines the functionality of setId() and setBirthdate().

In Identifiable it is defined like public function setIdBirtdate(int $Id, string $birthDate) and in Birthable it is public function setIdBirtdate(string $birthDate, int $Id). Is it possible to create a method which will implement both interfaces? What would you have to do if it is not possible?

## 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 (CLI)**, 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 |

### 3.1 Static Property and Static Method in Ferrari

Create a static property called **objNum** of integer type which will increase every time you make an instance of a Ferrari. You should change your constructor to use self::$objNum.

Create a static method called forUrl(string $str, string $rep="-") which replaces all appearances of space (" ") with another character and makes all characters small. Use your static method to prepare the output.

In the new output print the number of the instantiated class.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Bat Giorgi | 488-Spider/Brakes!/Zadu6avam sA!/bat-giorgi/ inst. 1 |
| Dinko | 488-Spider/Brakes!/Zadu6avam sA!/dinko/ inst. 2 |

## Telephony

You have a business - **manufacturing cell phones**. But you have no software developers, so you call your friends and ask them to help you create a cell phone software. They agree and you start working on the project. The project consists of one main **model - a Smartphone**. Each of your smartphones should have functionalities of **calling other phones** and **browsing in the world wide web.**

Your friends are very busy, so you decide to write the code on your own. Here is the mandatory assignment:

You should have a **model** - **Smartphone** and two separate functionalities which your smartphone has - to **call other phones** and to **browse in the world wide web**. You should end up with **one class** and **two interfaces**.

### Input

The input comes from the console (**CLI**). It will hold two lines:

* **First line**: **phone numbers** to call (String), separated by spaces.
* **Second line: sites** to visit (String), separated by spaces.

### Output

* First **call all numbers** in the order of input then **browse all sites** in order of input
* The functionality of calling phones is printing on the console the number which are being called in the format:

**Calling... <number>**

* The functionality of the browser should print on the console the site in format:

**Browsing: <site>!**

* If there is a number in the input of the URLs, print: **"****Invalid URL!"** and continue printing the rest of the URLs.
* If there is a character different from a digit in a number, print: **"Invalid number!"** and continue to the next number.

### Constraints

* Each site's URL should consist only of letters and symbols (**No digits are allowed** in the URL address)

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0882134215 0882134333 08992134215 0558123 3333 1  http://softuni.bg http://youtube.com http://www.g00gle.com | Calling... 0882134215  Calling... 0882134333  Calling... 08992134215  Calling... 0558123  Calling... 3333  Calling... 1  Browsing: http://softuni.bg!  Browsing: http://youtube.com!  Invalid URL! |

## Border Control

It’s the future, you’re the ruler of a totalitarian dystopian society inhabited by **citizens** and **robots**, since you’re afraid of rebellions you decide to implement strict control of who enters your city. Your soldiers check the **Id**s of everyone who enters and leaves. Define an **Interface** which should be implemented by both **citizens and robots.**

### Input

You will receive from the console an unknown amount of lines until the command “**End**” is received, on each line there will be the information for **either a citizen or a robot** who tries to enter your city in the format **“<name> <age> <id>**” for citizens and “**<model> <id>**” for robots. After the end command on the next line you will receive a single number representing **the last digits of fake ids**, all citizens or robots whose **Id** ends with the specified digits must be detained.

### Output

The output of your program should consist of all detained **Id**s each on a separate line (print in the same order of appearance).

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho 22 9010101122  MK-13 558833251  MK-12 33283122  End  122 | 9010101122  33283122 |
| Toncho 31 7801211340  Penka 29 8007181534  IV-228 999999  Stamat 54 3401018380  KKK-666 80808080  End  340 | 7801211340 |

## Birthday Celebrations

It is a well known fact that people celebrate birthdays, it is also known that some people also celebrate their pets birthdays. Extend the program from your last task to add **birthdates** to citizens and include a class **Pet**, pets have a **name** and a **birthdate**. Encompass repeated functionality into interfaces and implement them in your classes.

You will receive from the console an unknown amount of lines until the command “**End**” is received, each line will contain information in one of the following formats **“Citizen <name> <age> <id> <birthdate>**” for citizens, “**Robot** **<model> <id>**” for robots or “**Pet <name> <birthdate>**” for pets. After the end command on the next line you will receive a single number representing **a specific year**, your task is to print all birthdates (of both citizens and pets) in that year in the format **day/month/year** (print in order of appearance).

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Citizen Pesho 22 9010101122 10/10/1990  Pet Sharo 13/11/2005  Robot MK-13 558833251  End  1990 | 10/10/1990 |
| Citizen Stamat 16 0041018380 01/01/2000  Robot MK-10 12345678  Robot PP-09 00000001  Pet Topcho 24/12/2000  Pet Kosmat 12/06/2002  End  2000 | 01/01/2000  24/12/2000 |
| Robot VV-XYZ 11213141  Citizen Penka 35 7903210713 21/03/1979  Citizen Kane 40 7409073566 07/09/1974  End  1975 | <no output> |

## Food Shortage

Your totalitarian dystopian society suffers a shortage of food, so many rebels appear. Extend the code from your previous task with new functionality to solve this task.

Define a **class Rebel** which has a **name**, **age** and **group** (string)**,** names are **unique -** there will never be 2 Rebels/Citizens or a Rebel and Citizen with the same name**.**

Define an **interface Buyer** which defines a method **BuyFood()** and an integer property **food** (in the classes implementing the interface).

Implement the **Buyer** interface in the **Citizen** and **Rebel** class, both Rebels and Citizens **start with 0 food**, when a Rebel buys food his **Food** increases by **5**, when a Citizen buys food his **Food** increases by **10**.

On the first line of the input you will receive an integer **N** - the number of people, on each of the next **N** lines you will receive information in one of the following formats “**<name> <age> <id> <birthdate>**” for a Citizen or “**<name> <age><group>**” for a Rebel. After the **N** lines until the command “**End**” is received, you will receive names of people who bought food, each on a new line. Note that not all names may be valid, in case of an incorrect name - nothing should happen.

On the only line of output you should print the total amount of food purchased.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  Pesho 25 8904041303 04/04/1989  Stancho 27 WildMonkeys  Pesho  Gosho  Pesho  End | 20 units food |
| 4  Stamat 23 TheSwarm  Toncho 44 7308185527 18/08/1973  Joro 31 Terrorists  Penka 27 881222212 22/12/1988  Jiraf  Joro  Jiraf  Joro  Stamat  Penka  End | 25 units food |

## Military Elite\*

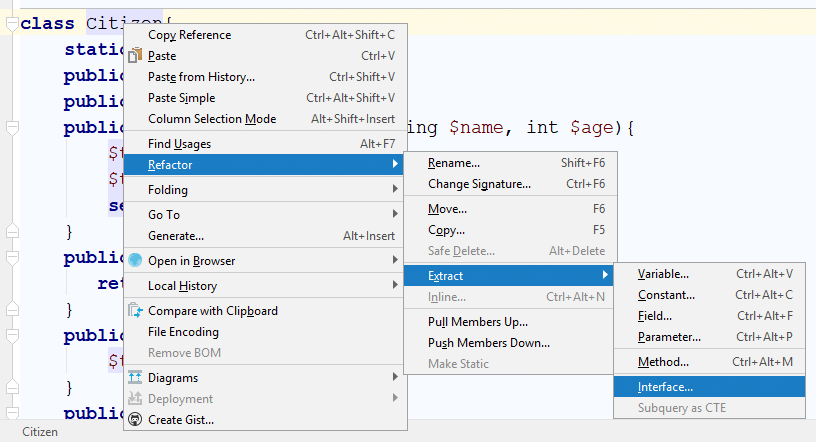
Create the following class hierarchy:

* **Soldier** – general class for soldiers, holding **id**, **first name** and **last name.**
  + **Private** – lowest base soldier type, holding the field **salary**(double).
    - **LeutenantGeneral** – holds a set of **Privates** under his command.
    - **SpecialisedSoldier –** general class for all specialised soldiers – holds the **corps** of the soldier.
      * **Engineer** – holds a set of **repairs**. A **repair** holds a **part name** and **hours worked**(int).
      * **Commando** – holds a set of **missions**. A mission holds **code name** and a **state** (***inProgress*** or ***Finished***). A mission can be finished through the method **CompleteMission()**.
  + **Spy** – holds the **code number** of the spy.

Extract **interfaces** for each class. (e.g. **ISoldier**, **IPrivate**, **ILeutenantGeneral**, etc. Each class should implement its respective interface. Validate the input where necessary (corps, mission state) - input should match **exactly** one of the required values, otherwise it should be treated as **invalid**. In case of an **invalid** **mission** **state** only the mission should be skipped.

### Note On Extraction

In the PhpStorm IDE you can easily **refactor** a class by putting your cursor at the **class name** (in the example Citizen) and clicking the right mouse button like this:



You will receive from the console an unknown amount of lines containing information about soldiers until the command “**End**” is received. The information will be in one of the following formats:

* Private: “**Private <id> <firstName> <lastName> <salary>**”
* LeutenantGeneral: “**LeutenantGeneral <id> <firstName> <lastName> <salary> <private1Id> <private2Id> … <privateNId>**” where privateXId will **always** be an **Id** of a private already received through the input.
* Engineer: “**Engineer <id> <firstName> <lastName> <salary> <corps> <repair1Part> <repair1Hours> … <repairNPart> <repairNHours>**” where repairXPart is the name of a repaired part and repairXHours the hours it took to repair it (the two parameters will always come paired).
* Commando: “**Commando <id> <firstName> <lastName> <salary> <corps> <mission1CodeName> <mission1state> … <missionNCodeName> <missionNstate>**” a missions code name, description and state will always come together.
* Spy: “**Spy <id> <firstName> <lastName> <codeNumber>**”

Define magic method \_\_**toString()** in all classes to print detailed information about the objects.

Private:  
**Name: <firstName> <lastName> Id: <id> Salary: <salary>**

Spy:  
**Name: <firstName> <lastName> Id: <id>  
Code Number: <codeNumber>**

LeutenantGeneral:  
**Name: <firstName> <lastName> Id: <id> Salary: <salary>  
Privates:  
 <private1Id>  
 <private2Id>  
 …  
 <privateNId>**

Engineer:  
**Name: <firstName> <lastName> Id: <id> Salary: <salary>  
Corps: <corps>  
Repairs:  
 <repair1X>  
 <repair2X>  
 …  
 <repairNX>**

Commando:  
**Name: <firstName> <lastName> Id: <id> Salary: <salary>  
Corps: <corps>  
Missions:  
 <mission1X>  
 <mission2X>  
 …  
 <missionNX>**

Repair:  
**Part Name: <partName> Hours Worked: <hoursWorked>**

Mission:  
**Code Name: <codeName> State: <state>**

**NOTE:** Salary should be printed rounded to **two decimal places** after the separator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Private 1 Pesho Peshev 22.22 Commando 13 Stamat Stamov 13.1 Airforces  Private 222 Toncho Tonchev 80.08  LeutenantGeneral 3 Joro Jorev 100 222 1  End | Name: Pesho Peshev Id: 1 Salary: 22.22  Name: Stamat Stamov Id: 13 Salary: 13.10  Corps: Airforces  Missions:  Name: Toncho Tonchev Id: 222 Salary: 80.08  Joro Jorev Id: 3 Salary: 100.00  Privates:  Name: Toncho Tonchev Id: 222 Salary: 80.08  Name: Pesho Peshev Id: 1 Salary: 22.22 |
| Engineer 7 Pencho Penchev 12.23 Marines Boat 2 Crane 17  Commando 19 Penka Ivanova 150.15 Airforces HairyFoot finished Freedom inProgress  End | Name: Pencho Penchev Id: 7 Salary: 12.23  Corps: Marines  Repairs:  Part Name: Boat Hours Worked: 2  Part Name: Crane Hours Worked: 17  Name: Penka Ivanova Id: 19 Salary: 150.15  Corps: Airforces Missions:  Code Name: HairyFoot State: finished  Code Name: Freedom State: inProgress |

## \*Collection Hierarchy

Create 3 different string collections – **AddCollection**, **AddRemoveCollection** and **MyList**.

The **AddCollection** should have:

* Only a single method **Add** which adds an item to the **end** of the collection.

The **AddRemoveCollection** should have:

* An **Add** method– which adds an item to the **start** of the collection.
* A **Remove** method which removes the **last** item in the collection.

The **MyList** collection should have:

* An **Add** method which adds an item to the **start** of the collection.
* A **Remove** method which removes the **first** element in the collection.
* A **Used** property which displays the amount of elements currently in the collection.

Create interfaces which define the collections functionality, think how to model the relations between interfaces to reuse code. Add an extra bit of functionality to the methods in the custom collections, **add** methods should return the index in which the item was added, **remove** methods should return the item that was removed.

Your task is to create a single copy of your collections, after which on the first input line you will receive a random amount of strings in a single line separated by spaces - the elements you have to add to each of your collections. For each of your collections write a single line in the output that holds the results of all **Add operations** separated by spaces (check the examples to better understand the format). On the second input line you will receive a single number - the amount of **Remove operations** you have to call on each collection. In the same manner as with the Add operations for each collection (except the AddCollection), print a line with the results of each Remove operation separated by spaces.

### Input

The input comes from the console. It will hold two lines:

* The first line will contain a random amount of strings separated by spaces - the elements you have to **Add** to each of your collections.
* The second line will contain a single number - the amount of **Remove** operations.

### Output

The output will consist of 5 lines:

* The first line contains the results of all **Add** operations on the **AddCollection** separated by spaces.
* The second line contains the results of all **Add** operations on the **AddRemoveCollection** separated by spaces.
* The third line contains the result of all **Add** operations on the **MyList** collection separated by spaces.
* The fourth line contains the result of all **Remove** operations on the **AddRemoveCollection** separated by spaces.
* The fifth line contains the result of all **Remove** operations on the **MyList** collection separated by spaces.

### Constraints

* All collections should have a **length of 100.**
* There will never be **more than 100** add operations.
* The number of remove operations will never be more than the amount of add operations.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| banichka boza tutmanik  3 | 0 1 2  0 0 0  0 0 0  banichka boza tutmanik  tutmanik boza banichka |
| one two three four five six seven  4 | 0 1 2 3 4 5 6  0 0 0 0 0 0 0  0 0 0 0 0 0 0  one two three four  seven six five four |

### Hint

Create an interface hierarchy representing the collections.

## MooD 3

You are an owner of the most epic video game of the world - **3 MooD**. Your employees have gone on summer vacation. But there is a problem in the application and you are on your own. So the problem is how to store the information for the players. The best approach to you, seems to be, storing them in **GameObjects**.

In your game, there are two types of characters - **Demon** and **Archangel**. All characters in the game have:

* **username**
* **hashedPassword**
* **level**
* **special** **points**.

The **main difference** between the Demon and the Archangel is that the **Demon has energy** (as special points) and the **Archangel has mana** (as special points). Your task is to model the application.

When you receive the username and the character type, you should generate the hashed password by the formulas below:

* For a **Demon**: **username length \* 217**
* For an **Archangel**: **(username characters in reversed order) + (username length \* 21)**

Your task is to print the info as it is written in the Output.

### Input

The input consists of **single line**. First, you will get the username of a player. The second parameter is its character type. The next two parameters are his mana / energy points and his level. Format:

<**username**> | <**character type**> | <**special points**> | <**level**>

### Output

Print the info on two lines, for a single entry (player) in the format:

<”**username”>** | **<”hashed password”>** -> <**character type**>

<**special points \* level**>

### Constraints

* **Username** – alphabetical letters (**Latin**), no more than 10 characters and you do not need to check it explicitly.
* **Character** **type** – String, Demon or Archangel, no need to check it explicitly.
* **Special points (Mana)** – a valid Integer, no need to check it explicitly, print as integer without decimal separator and trailing zeros.
* **Special points (Energy)** – a valid Double, no need to check it explicitly, round up and print one digit after the decimal separator.
* **Level** – a valid Integer, no need to check it explicitly.

### Example

|  |  |
| --- | --- |
| **Input** | **Output** |
| **"KoHaH"** | **Demon** | **100.0** | **100** | **""KoHaH"" | "1519" -> Demon**  **10000.0** |
| **"Akasha" | Archangel | 5 | 100** | **""Akasha"" | ""ahsakA"168" -> Archangel**  **500** |

### Note

Implement **interface**, holding the **main functionalities of** **all characters**. Create an **abstract class** to hold all the same characteristics of the characters.

## Vehicles

Write a program that models 2 vehicles (**Car** and **Truck**) and will be able to simulate **driving** and **refueling** them. **Car** and **truck** both have **fuel quantity**, **fuel consumption** **in liters** **per km** and can be **driven given distance** and **refueled with given liters.** But in the summer both vehicles use air conditioner 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 his tank and when it gets **refueled** it gets only **95%** of given **fuel**. The **car** has no problems when refueling and adds **all given fuel to its tank.** If vehicle cannot travel given distance its fuel does not change.

Input

* On the first line - information about the car in format {Car {fuel quantity} {liters per km}}
* On the second line – info about the truck in format {Truck {fuel quantity} {liters per km}}
* On third line - 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}
* Refuel Car {liters}
* Refuel Truck {liters}

Output

After each Drive command print whether the Car/Truck was able to travel given distance in format if it’s successful. Print the distance with all digits after the decimal separator except trailing zeros.

Car/Truck travelled {distance} km

Or if it is not:

Car/Truck needs refueling

Finally print the remaining fuel for both car and truck rounded 2 digits after floating point in format:

Car: {liters}

Truck: {liters}

### Example

|  |  |
| --- | --- |
| **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 |

## Extending Vehicles

Use your solution of the previous task for starting point and add more functionality. Add new vehicle – **Bus**. Now every vehicle has **tank capacity** and fuel quantity **cannot fall** **below 0** (If fuel quantity become less than 0 **print** on the console **“Fuel must be a positive number”**).

The **car** and the **bus** **cannot be filled** with fuel **more than their tank capacity**. If you **try to put more fuel** in the tank than the **available space,** print on the console **“Cannot fit fuel in tank”** and **do not add any fuel** in vehicles tank.

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

### Input

* On the first three lines you will receive information about the vehicles in format:

**Vehicle {initial fuel quantity} {liters per km} {tank capacity}**

* On fourth line - 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 print whether the Car/Truck/Bus was able to travel given distance in format if it’s successful:

Car/Truck/Bus travelled {distance} km

* Or if it is not:

Car/Truck/Bus needs refueling

* If given fuel is **≤ 0** print **“Fuel must be a positive number”.**
* If given fuel cannot fit in car or bus tank print “Cannot fit fuel in tank”
* Finally print the remaining fuel for car, truck and bus rounded 2 digits after floating point in 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 | Cannot fit fuel in tank  Bus travelled 10 km  Cannot fit fuel in tank  Bus needs refueling  Car: 30.00  Truck: 1050.00  Bus: 23.00 |

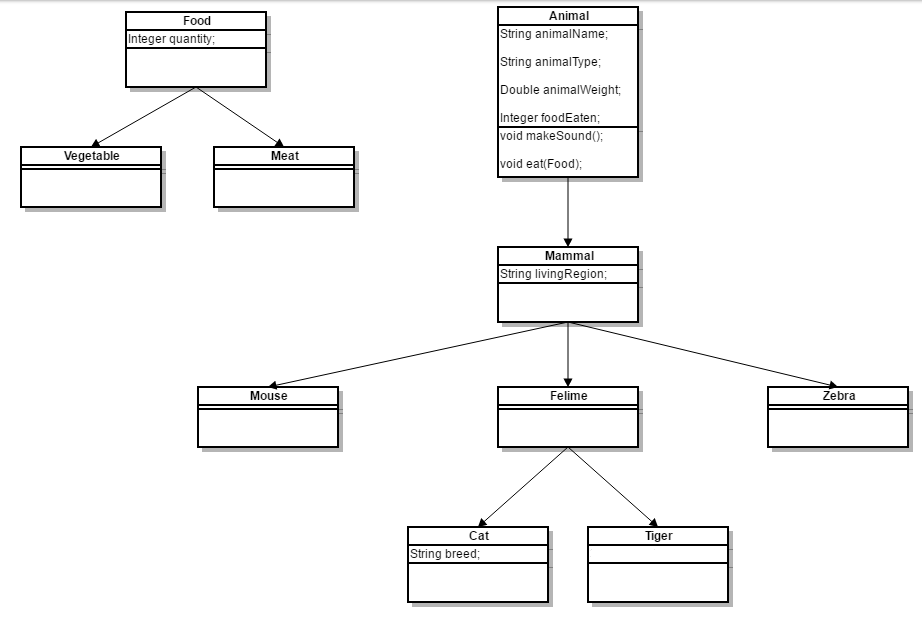
## Wild Farm Hierarchy

Your task is to create a class hierarchy like the picture below. All the classes except Vegetable, Meat, Mouse, Tiger, Cat & Zebra should be abstract.

Input should be read from the console. Every **even** line will contain information about the Animal in following format:

**{AnimalType} {AnimalName} {AnimalWeight} {AnimalLivingRegion} [{CatBreed}** *= Only if its cat***]**

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



You should build the logic to determine if the animal is going to eat the provided food. The Mouse and Zebra should check if the food is a Vegetable. If it is they will eat it. Otherwise you should print a message in the format:

**{AnimalType} are not eating that type of food!**

**Cats** eat **any** kind of food, but **Tigers** accept **only Meat**. If **Vegetable** is provided to a **tiger** message like the one above should be printed on the console.

**{AnimalType} [{AnimalName}, {CatBreed}, {AnimalWeight}, {AnimalLivingRegion}, {FoodEaten}]**

Print all **AnimalWeight** with no trailing zeroes after the decimal separator.

After you read information about the Animal and Food then invoke **makeSound** method of the current animal and then feed it. At the end print the whole object and proceed reading information about the next animal/food. The input will continue until you receive “**End**” for animal information.

|  |  |
| --- | --- |
| **Input** | **Output** |
| Cat Gray 1.1 Home Persian  Vegetable 4  End | Meowwww  Cat[Gray, Persian, 1.1, Home, 4] |
| Tiger Typcho 167.7 Asia  Vegetable 1  End | ROAAR!!!  Tigers are not eating that type of food!  Tiger[Typcho, 167.7, Asia, 0] |
| Zebra Doncho 500 Africa  Vegetable 150  End | Zs  Zebra[Doncho, 500, Africa, 150] |
| Mouse Jerry 0.5 Anywhere  Vegetable 0  End | SQUEEEAAAK!  Mouse[Jerry, 0.5, Anywhere, 0] |

## (Father, Sun and GrandSun)

Do you remember the problem about Person? Now your purpose is to create a hierarchy of classes based on Person.

Define a class **Father** with properties **yearBirth, yearDead** and **name**. Then define a class called **Son** which **inherits Person**. Define a class that is called **GrandSon** which inherits **Son**.

Write a method called **getTimeLived()** in Person which should be inherited in all child classes. Write an overwritten method called **getGenerationNum()** which returns 1 for Person, 2 for Sun and 3 for GrandSon. It’s person is to show us the level in which the person is.

You have to create a hierarchy of objects that corresponds to a particular family tree.

### Input

The input is hard coded as an array of objects.

### Output

You should print an **HTML table** in three levels by your design – the person, sons and grandsons:

**1. Person:** James Strong, **1970 – 1940**, lived **30** years

**2. Sons:**

Name, from- to, lived …

Name, from- to, lived …

**Average Lifespan: 43 years**

**3. Grandsons: …**

Name, from- to, lived …

Name, from- to, lived …

**Average Lifespan: 51 years**

See **step 4** for the particular input data!

### Note

Your classes' names **MUST** be the same as the names shown above!!!

|  |
| --- |
| **Sample Code** |
| **class** Father {    *// 1. Add the Properties  // 2. Add the Constructor  // 3. Add the Methods* } |

### Step 1. Define the Properties of a Father

Define the **yearBirth, yearDead** and **name** properties of a Person. Ensure that they can only be **changed by the class itself or its descendants** (pick the most appropriate access modifier).

### Step 2. Define the Constructor

Define a constructor that accepts **name, yearBirth and yearDeath** arguments.

|  |
| --- |
| **Sample Code** |
| **function** \_\_construct(string $name, int $yearBirth, int $yearDeath){  $**this**->setName($name);  // More todo } |

### Step 3. Perform Validations

After you have created the **properties** (e.g. **name** and **age**). Next step is to **perform validations** for each one. The **getter should return the corresponding property’s value** and the **setter should validate** the input data before setting it. Do this for each property. The name should be a string at least 3 characters long.

### Step 4. Create a Hierarchy of Objects

Create a hierarchy that corresponds to the Following **family tree**:

1.James Strong lived from 1873 to 1923. He had two sons:

1.1. Peter Strong lived from 1894 only some 34 years and died

1.3. Andrew Strong lived from 1899 to 1970 and was a blessed person whom everybody loved

Andrew Strong had 3 daughters and **3 sons** as follows:

2.1. Jackson Strong was born when Andrew was 28 years old and lived 65 years.

2.2. Martin Strong was born in 1927 and died in 1967

2.3. Gregory Strong was born in 1931 and lived for 69 years

Use the **Father, Son and GrandSon classes** to create objects for each person. Use an **array of objects** to hold the hierarchy not simple variables.

### Hint

Use the overwritten method in each class to filter persons by their level in hierarchy.

### Step 6. Calculate the Average Timespan

You can create a separate function and pass all persons/objects from one generation which will calculate the average timespan. Another way is to use the method **getTimeLived()** of each class and find the average**. Choose the more memory efficient solution.**

## Abstract Class Person

Go back **to problem 4** and create an **abstract class Person**. Define **getTimeLived()** and **getGenerationNum()** as **abstract methods.** Now redefine the Father class to extend the abstract Person.

### Step 2. Do Not Define GetGenerationNum() in GrandSon

**What error do you get when this method is not implemented? Why do you get it?**

### Step 3. Do Not Define Person As Abstract

Let Person be not defined abstract but let **getTimeLived()** be defined as an **abstract method/function? Can you then create an instance of Person? Why not? Is Person still abstract?**

## Mass Effect Engine\*\*

The galaxy is a dangerous place - the vast void is full of starships which endlessly fight one another.

There are several **star systems** in the galaxy - Artemis Tau, Serpent Nebula, Hades Gamma and Kepler Verge. **Starships** can travel from one star system to another (if they are neighbouring) and **fight** **other starships**. A starship has **health**, **shields**, **damage**, **fuel** and shoots a specific **projectile** when attacking another ship. There are 3 types of starships - **Frigates**, **Cruisers** and **Dreadnoughts,** and each one respectively has different characteristics.

Your task is to implement the **project structure**, each of the **ships, projectiles, enahcements** and a few **commands**.

### Star Systems

**Artemis-Tau** has neighbors with the following fuel consumption:

* **Serpent-Nebula** – 50
* **Kepler-Verge** – 120

**Serpent-Nebula** has neighbors with the following fuel consumption:

* **Artemis-Tau** – 50
* **Hades-Gamma** – 360

**Hades-Gamma** has neighbors with the following fuel consumption:

* **Serpent-Nebula** – 360
* **Kepler-Verge** – 145

**Kepler-Verge** has neighbors with the following fuel consumption:

* **Hades-Gamma** – 145
* **Artemis-Tau** – 120

### Commands

Implement the following commands:

* **create {shipType} {shipName} {starSystem} {enhancement1 enhancements2 ...}** - creates a new ship with the given name in the given star system. Optionally, an arbitrary number of enhancements can follow in the input string - they should all be **added** to the ship through the **AddEnhancement** method and their **effects applied**.
  + Success message: **Created {shipType} {shipName}**
* **attack {attackerShip} {targetShip} -** forces a ship to attack another. The attacking ship should **produce a projectile** and it should be given to the target ship to **respond** to it (study the methods of the **IStarship** and **IProjectile** interfaces to get a better idea).
  + Success message: "**{attackerShip} attacked {targetShip}"**If the ship is destroyed as a result, print "**{targetShip} has been destroyed**" as well.
* **plot-jump {shipName} {starSystem}** - changes the location of the given ship to the new star system.
  + Success message:"**{shipName} jumped from {startLocation} to {destinationLocation}**"
* **status-report {shipName}** - displays information about the ship in the following format:

|  |  |
| --- | --- |
| **If health > 0** | **If health <= 0** |
| **--{shipName} - {shipType}**  **-Location: {locationName}**  **-Health: {health}**  **-Shields: {shields}**  **-Damage: {damage}**  **-Fuel: {fuel}**  **-Enhancements: {enh1, enh2, ...}** | **--{shipName} - {shipType}**  **(Destroyed)** |

If there are no enhancements, print "**N/A**".

If the ship is a **frigate** (and not destroyed), it should also display the number of projectiles fired so far in the format:

|  |
| --- |
| **-Projectiles fired: {count}** |

* **over** - stops the game engine

### Ships

There are 3 types of ships:

* **Frigate** - has start **health 60**, **shields 50**, **damage 30** and **fuel 220**. Shoots a **ShieldReaver** with damage equal to its **own damage**.
* **Cruiser** - has start **health 100**, **shields 100**, **damage 50** and **fuel 300**. Shoots a **PenetrationShell** with damage equal to its **own damage**.
* **Dreadnought** - has start **health 200**, **shields 300**, **damage 150** and **fuel 700**. Shoots a **Laser** with damage equal to **half its shields + own damage**. Responds to an attack by raising its shields by **50** before the attack and removes them after it.

### Enhancements

Enhancements give a ship bonuses. Enhancement effect (shields, fuel and damage) should be applied to the ship when added through the **AddEnhancement** method.

* **ThanixCannon** - gives a ship bonus **50 damage**.
* **KineticBarrier** - gives a ship bonus **100 shields**.
* **ExtendedFuelCells** - gives a ship bonus **200 fuel**.

### Projectiles

* **PenetrationShell** - removes **health** from the ship equal to the projectile's **damage**.
* **ShieldReaver** - removes **health** from the ship equal to **the projectile's damage**. It also removes **shields** from the ship equal to **2x the projectile's** **damage**.
* **Laser** - removes **shields** from the ship equal to the projectile's **damage**. If the damage is more than the ship's shields, it also takes health equal to the damage left. (e.g. **50 shields** and **100 health** - a laser of **80 damage** would remove **50** shields and **30** health, resulting in **0 shields** and **70 health** for the ship).

### Validations

Perform the following **validations**:

|  |  |
| --- | --- |
| **Exceptional conditions** | **Messages** |
| If a destroyed ship attacks/is attacked or tries to travel to another star system | Ship is destroyed |
| Creating a ship with a name that already exists | Ship with such name already exists |
| Attacking a ship that is not in the current star system | No such ship in star system |
| Attempting to travel to the same star system | Ship is already in {starSystemName} |

### Extending the Engine

Add and implement the following command:

* **system-report** **{starSystem}** - displays information about all ships in the given **star system** in the following format:

|  |
| --- |
| **Intact ships:**  **{information about ship\_1}**  **{information about ship\_2}**  **Destroyed ships:**  **{information about ship\_3}** |

Intact ships should be sorted by **health** in descending order and by **shields** as secondary criteria (again in descending order). Destroyed ships should ordered alphabetically by **name** in ascending order.

### Additional Notes

* The star systems will always be the same (Artemis-Tau, Serpent-Nebula, Hades-Gamma and Kepler-Verge).
* A ship's **health** and **shields** should not fall below **0**.
* The ship's fuel should be rounded to **1 digit** after the decimal point.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| create Frigate Normandy Serpent-Nebula ThanixCannon  create Dreadnought DestinyAscension Serpent-Nebula KineticBarrier  attack Normandy DestinyAscension  attack Normandy DestinyAscension  attack Normandy DestinyAscension  status-report DestinyAscension  plot-jump Normandy Artemis-Tau  status-report Normandy  over | Created Frigate Normandy  Created Dreadnought DestinyAscension  Normandy attacked DestinyAscension  Normandy attacked DestinyAscension  Normandy attacked DestinyAscension  DestinyAscension has been destroyed  --DestinyAscension - Dreadnought  (Destroyed)  Normandy jumped from Serpent-Nebula to Artemis-Tau  --Normandy - Frigate  -Location: Artemis-Tau  -Health: 60  -Shields: 50  -Damage: 80  -Fuel: 170.0  -Enhancements: ThanixCannon  -Projectiles fired: 3 |

|  |  |
| --- | --- |
| **Input** | **Output** |
| create Frigate Normandy Hades-Gamma  create Cruiser Alarei Hades-Gamma  create Dreadnought Sovereign Kepler-Verge  status-report Alarei  attack Normandy Alarei  status-report Alarei  status-report Normandy  attack Alarei Normandy  status-report Normandy  plot-jump Sovereign Hades-Gamma  attack Sovereign Alarei  attack Normandy Sovereign  system-report Hades-Gamma  system-report Artemis-Tau  over | Created Frigate Normandy  Created Cruiser Alarei  Created Dreadnought Sovereign  --Alarei - Cruiser  -Location: Hades-Gamma  -Health: 100  -Shields: 100  -Damage: 50  -Fuel: 300.0  -Enhancements: N/A  Normandy attacked Alarei  --Alarei - Cruiser  -Location: Hades-Gamma  -Health: 70  -Shields: 40  -Damage: 50  -Fuel: 300.0  -Enhancements: N/A  --Normandy - Frigate  -Location: Hades-Gamma  -Health: 60  -Shields: 50  -Damage: 30  -Fuel: 220.0  -Enhancements: N/A  -Projectiles fired: 1  Alarei attacked Normandy  --Normandy - Frigate  -Location: Hades-Gamma  -Health: 10  -Shields: 50  -Damage: 30  -Fuel: 220.0  -Enhancements: N/A  -Projectiles fired: 1  Sovereign jumped from Kepler-Verge to Hades-Gamma  Sovereign attacked Alarei  Alarei has been destroyed  Normandy attacked Sovereign  Intact ships:  --Sovereign - Dreadnought  -Location: Hades-Gamma  -Health: 170  -Shields: 240  -Damage: 150  -Fuel: 555.0  -Enhancements: N/A  --Normandy - Frigate  -Location: Hades-Gamma  -Health: 10  -Shields: 50  -Damage: 30  -Fuel: 220.0  -Enhancements: N/A  -Projectiles fired: 2  Destroyed ships:  --Alarei - Cruiser  (Destroyed)  Intact ships:  N/A  Destroyed ships:  N/A |