# Homework: OOP Fundamentals Part II

# Abstraction and Polymorphism

This document defines the homework assignments for ["PHP Web Development Basic" Course @ Software University](https://softuni.bg/trainings/1746/php-web-developmentbasics-september-2017). Please submit ……………………………………………….???

## Problem 1. 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}
* Dive 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 |

## Problem 2. 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 |

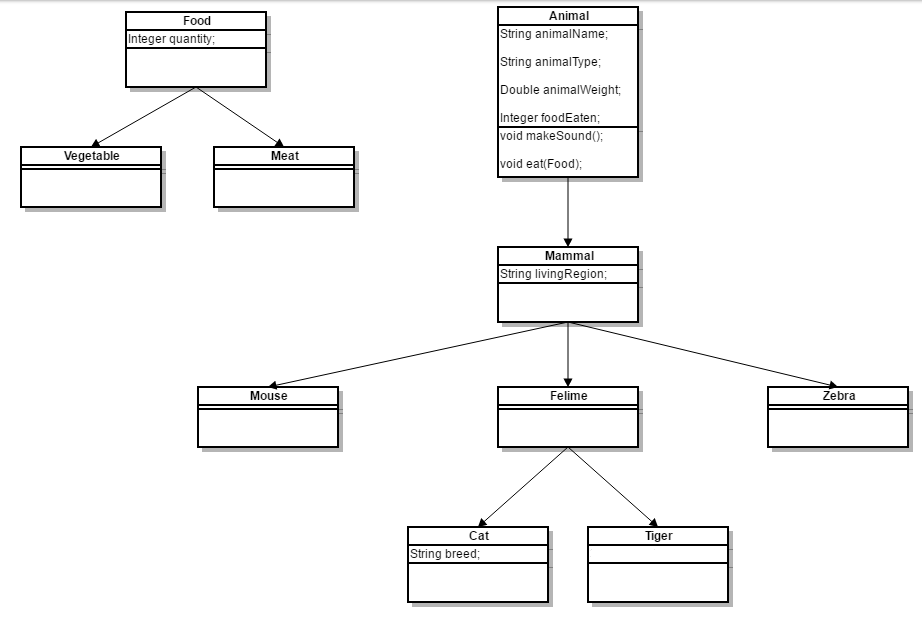
## Poblem 3. Wild Farm Hierarchy

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] |

## Problem 4. Family Tree (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.**

## Problem 5. 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?**

## Problem 6. 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

### Zero Test #1

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

### Zero Test #2

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