# Lab: Interfaces and Abstraction

This document defines the **exercise assignments** for the "[PHP Web Development Basics" Course @ Software University](https://softuni.bg/trainings/1746/php-web-developmentbasics-september-2017) .

## Basic Abstraction and Polymorphism

## 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 |

## Basic Use of Interfaces

## Circle and Area interface

## One Class and One Interface

Build a simple class Circle and an interface like this:

|  |
| --- |
| inteface  Area |
| **getSurface()** |

|  |
| --- |
| **Circle** |
| **radius** |

You should have one class called **Circle** with one property **radius.** Define an interface called **Area** which calculates the area of the circle on the basis of its radius. It should implement one method **getSurface()** which returns the area of the circle. For the calculation look at: http://mathworld.wolfram.com/Circle.html

**Create one instance of Circle and use the implemented method to calculate a circle with radius 10 mm.**

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCircle with radius 10 mm. | Circle, radius = 10 mm, area = … mm |

## Circle and rectangle and Area Interface

## Two Classes and One Interface

Now we extend the previous task with one more class: Rectangle. The rectangle doesn’t have a radius but **width** and **height**. So its area is calculated in a different way. But the **Area interface** should be applied to it also. Implement the method **getArea()** in the Rectangle class. What is different in the method now?

Create an instance of a Rectangle $myRec and calculate the area of a specific rectangle as given in the Input /Output example

|  |  |
| --- | --- |
| **Input** | **Output** |
| $mRec with width = 15 mm and height = 35 mm | Rectangle, width = 15mm, height = 35 mm, area = … mm |

Your hierarchy of classes and interfaces should look like this:

|  |
| --- |
| inteface  Area |
| **getSurface()** |

|  |
| --- |
| **Circle** |
| **radius** |

|  |
| --- |
| Rectangle |
| **width**  **height** |

## Part II. Extended Use of Interfaces

## 2.1. Circle and Area + Circumference Interfaces

## One Class and Two Interfaces

A class can implement more than one interface. Let’s get back to our Circle. Besides an area, a circle can have a circumference (the length of the line of the circle which is designated by c). See the picture bellow.

Now, let the **class Circle** implement the interface **Area** and implement the interface **Circumference**. Area defines a method **getSurface()** and Circumference defines a method **getCircumference().**

How would the two methods look for a Circle? Calculate a circle’s surface/area and circumference. See the input /output table.

For the calculation of circumference look at: <http://mathworld.wolfram.com/Circle.html>

You should be able to use the class like this:

|  |
| --- |
| inteface  Area |
| **getSurface()** |

|  |
| --- |
| interface  Circumference |
| **getCircumference()** |

|  |
| --- |
| **Circle** |
| **radius** |

|  |  |
| --- | --- |
| **Input** | **Output** |
| $myCircle with radius 10 mm. | Circle with radius = 10 mm:  Area = … mm  Circumference = …. mm |

**How many interfaces can a class implement?**

## Part III. Advanced Use of Interfaces

## 3.1. Hardware Devices and Their Interfaces

## Multiple Interfaces and Multiple Implementations

Imagine that you work in an office and you have to **model** the behaviour of the **hardware devices** in the office. Different hardware devices have different interfaces or ways they connect to people that use them. What can they be:

1. Interface Keyboard with method **pressKey()** and **changeStatus()**
2. Interface Mouse with methods **move()** and **click()**
3. Interface TouchPad with methods **moveFinger()** and **click()**;

Of course there can be much more interfaces than this. But **different devices implement different interfaces**. Some of the devices may implement **all, some or none** of these. So you have at your office the following classes of devices:

1.Class DesktopComputer, 2.Class Laptop, 3.Class Tablet, 4.Class MobilePhone

But these classes are children of the following **abstract classes**: **Computer** and **Mobile**. There is no abstract class for Printer. So DesktopComputer and Laptop inherit abstract Computer. Tablet and MobilePhone inherit Mobile.

See the chart how classes and abstract classes and interfaces are connected. The **orange lines** represent implementations of interfaces. **Blue lines** represent **inheritance**.

|  |
| --- |
| **Abstract class**  **Computer** |
| **processor**  **ram** |

|  |
| --- |
| **Abstract class**  **Mobile** |
| **operator**  **canCall (true / false)**  **battery(%)** |

|  |
| --- |
| **DesktopComputer** |
| **keyboardConnected** |

|  |
| --- |
| inteface  Keyboard |
| **pressKey()**  **changeStatus()** |

|  |
| --- |
| **Laptop** |
| **battery (%)** |

|  |
| --- |
| interface  Mouse |
| **click()**  **move()** |

|  |
| --- |
| **Tablet** |
| **stdResolution** |

|  |
| --- |
| interface  TouchScreen |
| **moveFinger()**  **clickFinger()**  **writeText()** |

|  |
| --- |
| **MobilePhone** |
| **size** |

Create the class hierarchy. If you do the lab at home then create every class in a separate file. Now in the particular office we have:

**One desktop computer - $d, two laptops - $l1 and $l2, one tablet - $t, three mobile phones - $m1, $m2, $m3.**

**How many interfaces** can a class implement and **how many abstract classes** can it inherit**?** (Discuss if you are in a group)

## 3.2. Notebook++

In the office you have a new device which is a **notebook** by which you can also make **phone calls**. It has a touchscreen and can have also a keyboard and a mouse attached to it. It has the characteristics of a laptop and of a tablet.

1. **What abstract class** can Notebook++ extend?
2. **Which interfaces** should Notebook++ implement?
3. Create an instance in the variable **$n**
4. Introduce in the hierarchy a property called **writtenText** and write the two methods **pressKey()** and **writeText()** to **put the text data** in the property writtenTextand **return** it to the caller**. Where** should the property **be put and why?** (Give different options)

## 3.2. The Mouse

Write the methods **click()** and **move()** for a desktop computer and a laptop. **Where** would you put the bodies of the methods? The click method can have a **leftClick** (true|false) and **rightClick** (true|false) arguments. The **move()** method receives as arguments: **currentX** and **currentY** coordinates (in pixels) and **offesetX, offsetY** (in pixels). It returns the **newX**, **newY** coordinates.