# Homework: OOP Advanced Interfaces

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 ……………………………………………….???

## 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** |
| $myCitizen = new Citizen("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 **integer**.

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** |
| $myCitizen = new Citizen("Peter", 25, 7706126789, "12.06.1977"); | Peter, 25, ID = 770612678, 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 |

## 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\*

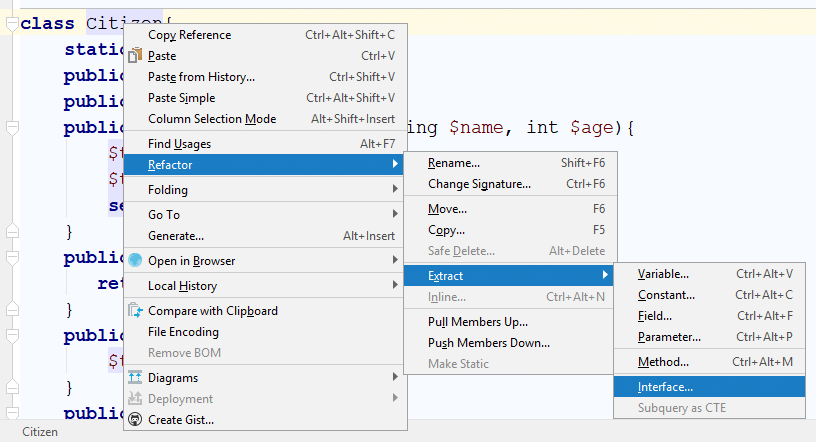
Create the following class hierarchy:

* **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 |

## Problem 9. 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.