# Exercises: Defining Classes and Methods

This document defines the **exercise assignments** for the "[PHP Fundamentals" Course @ Software University](https://softuni.bg/courses/php-basics/) . Please submit your solutions (source code) of all below described problems in Judge.

## Define a class Person

Define a class **Person** with fields for **name** and **age**.

### Note

Add the following code and submit it to Judge.

|  |
| --- |
| $person = new Person();  echo(count(get\_object\_vars($person))); |

The output on the console should be **2**. If you defined the class correctly, the test should pass.

### Bonus\*

Try to create a few objects of type Person:

|  |  |
| --- | --- |
| **Name** | **Age** |
| Pesho | 20 |
| Gosho | 18 |
| Stamat | 43 |

## Creating Constructor

Add constructor to the **Person** class from the last task:

1. It should accept a string for the name and an integer for the age and should produce a person with the given name and age.

Add the following code to your main method and submit it to Judge.

|  |
| --- |
| public function \_\_construct(string $name, int $age) {  $this->name = $name;  $this->age = $age;  echo $this->name . " " . $this->age;  } |

If you defined the constructors correctly, the test should pass.

The **input** comes in two rows. On the first row you will receive the name. On the second row you will be given the age.

The **output** should be printed to the console as single line - name and age separated with one space

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho  20 | Pesho 20 |
| Gosho  18 | Gosho 18 |
| Stamat  43 | Stamat 43 |

## Opinion Poll

Using the Person class, write a program that reads from the console **N** lines of personal information and then prints all people whose **age** is **more than 30** years, **sorted in alphabetical order**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  Pesho 12  Stamat 31  Ivan 48 | Ivan - 48  Stamat - 31 |
| 5  Nikolai 33  Yordan 88  Tosho 22  Lyubo 44  Stanislav 11 | Lyubo - 44  Nikolai - 33  Yordan - 88 |

## Company Roster

Define a class **Employee** that holds the following information: **name, salary, position, department, email** and **age**. The **name, salary**, **position** and **department** are **mandatory** while the rest are **optional**.

Your task is to write a program which takes **N** lines of employees from the console and calculates the department with the highest average salary and prints for each employee in that department his **name, salary, email and age** – **sorted by salary in descending order**. If an employee **doesn’t have** an **email** – in place of that field you should print “**n/a**” instead, if he doesn’t have an **age** – print “**-1**” instead. The **salary** should be printed to **two decimal places** after the seperator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  Pesho 120.00 Dev Development [pesho@abv.bg](mailto:pesho@abv.bg) 28  Toncho 333.33 Manager Marketing 33  Ivan 840.20 ProjectLeader Development [ivan@ivan.com](mailto:ivan@ivan.com)  Gosho 0.20 Freeloader Nowhere 18 | Highest Average Salary: Development  Ivan 840.20 [ivan@ivan.com](mailto:ivan@ivan.com) -1  Pesho 120.00 [pesho@abv.bg](mailto:pesho@abv.bg) 28 |
| 6  Stanimir 496.37 Temp Coding [stancho@yahoo.com](mailto:stancho@yahoo.com)  Yovcho 610.13 Manager Sales  Toshko 609.99 Manager Sales [toshko@abv.bg](mailto:toshko@abv.bg) 44  Venci 0.02 Director BeerDrinking [beer@beer.br](mailto:beer@beer.br) 23  Andrei 700.00 Director Coding  Popeye 13.3333 Sailor SpinachGroup [popeye@pop.ey](mailto:popeye@pop.ey) | Highest Average Salary: Sales  Yovcho 610.13 n/a -1  Toshko 609.99 [toshko@abv.bg](mailto:toshko@abv.bg) 44 |

## \*Speed Racing

Your task is to implement a program that keeps track of cars and their fuel and supports methods for moving the cars. Define a class **Car** that keeps track of a car’s **Model, fuel amount, fuel cost for 1 kilometer** and **distance traveled**. A Car’s Model is **unique** - there will never be 2 cars with the same model.

On the first line of the input you will receive a number **N** – the number of cars you need to track, on each of the next **N** lines you will receive information for a car in the following format “<**Model> <FuelAmount> <FuelCostFor1km>**”, all **cars start at 0 kilometers traveled**.

After the **N** lines until the command “**End**” is received, you will receive a commands in the following format “**Drive <CarModel> <amountOfKm>**”, implement a method in the **Car** class to calculate whether or not a car can move that distance, if it can the car’s **fuel amount** should be **reduced** by the amount of used fuel and its **distance traveled** should be increased by the amount of kilometers traveled, otherwise the car should not move (Its fuel amount and distance traveled should stay the same) and you should print on the console “**Insufficient fuel for the drive**”. After the “**End**” command is received, print each car and its current fuel amount and distance traveled in the format “**<Model> <fuelAmount> <distanceTraveled>**”, where the fuel amount should be printed to **two decimal places** after the separator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  AudiA4 23 0.3  BMW-M2 45 0.42  Drive BMW-M2 56  Drive AudiA4 5  Drive AudiA4 13  End | AudiA4 17.60 18  BMW-M2 21.48 56 |
| 3  AudiA4 18 0.34  BMW-M2 33 0.41  Ferrari-488Spider 50 0.47  Drive Ferrari-488Spider 97  Drive Ferrari-488Spider 35  Drive AudiA4 85  Drive AudiA4 50  End | Insufficient fuel for the drive  Insufficient fuel for the drive  AudiA4 1.00 50  BMW-M2 33.00 0  Ferrari-488Spider 4.41 97 |

## \*Raw data

You are the owner of a courier company and want to make a system for tracking your cars and their cargo. Define a class **Car** that holds information about **model, engine, cargo** and a **collection of exactly 4 tires**. The engine, cargo and tire **should be separate classes**, create a constructor that receives all information about the Car and creates and initializes its inner components (engine, cargo and tires).

On the first line of input you will receive a number **N** - the amount of cars you have, on each of the next **N** lines you will receive information about a car in the format “**<Model> <EngineSpeed> <EnginePower> <CargoWeight> <CargoType> <Tire1Pressure> <Tire1Age> <Tire2Pressure> <Tire2Age> <Tire3Pressure> <Tire3Age> <Tire4Pressure> <Tire4Age>”** where the speed, power, weight and tire age are **integers**, tire pressure is a **double.**

After the **N** lines you will receive a single line with one of 2 commands “**fragile**” or “**flamable**” , if the command is “**fragile**” print all cars whose **Cargo Type is “fragile”** with a tire whose **pressure is** **< 1**, if the command is “**flamable**” print all cars whose **Cargo Type is “flamable”** and have **Engine Power > 250**. The cars should be printed in order of appearing in the input.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  ChevroletAstro 200 180 1000 fragile 1.3 1 1.5 2 1.4 2 1.7 4  Citroen2CV 190 165 1200 fragile 0.9 3 0.85 2 0.95 2 1.1 1  fragile | Citroen2CV |
| 4  ChevroletExpress 215 255 1200 flamable 2.5 1 2.4 2 2.7 1 2.8 1  ChevroletAstro 210 230 1000 flamable 2 1 1.9 2 1.7 3 2.1 1  DaciaDokker 230 275 1400 flamable 2.2 1 2.3 1 2.4 1 2 1  Citroen2CV 190 165 1200 fragile 0.8 3 0.85 2 0.7 5 0.95 2  flamable | ChevroletExpress  DaciaDokker |

## Car Salesman

Define two classes **Car** and **Engine.** A **Car** has a **model, engine, weight** and **color**. An Engine has **model**, **power, displacement** and **efficiency**. A Car’s **weight** and **color** and its Engine’s **displacements** and **efficiency** are **optional**.

On the first line you will read a number **N** which will specify how many lines of engines you will receive, on each of the next **N** lines you will receive information about an **Engine** in the following format “<**Model> <Power> <Displacement> <Efficiency>**”. After the lines with engines, on the next line you will receive a number **M** – specifying the number of Cars that will follow, on each of the next **M** lines information about a **Car** will follow in the following format “<**Model> <Engine> <Weight> <Color>**”, where the engine in the format will be the **model of an existing** **Engine**. When creating the object for a **Car**, you should keep a **reference to the real engine** in it, instead of just the engine’s model, note that the optional properties **might be missing** from the formats.

Your task is to print each car (in the order you received them) and its information in the format defined bellow, if any of the optional fields has not been given print “**n/a**” in its place instead:

**<CarModel>:  
 <EngineModel>:  
 Power: <EnginePower>  
 Displacement: <EngineDisplacement>  
 Efficiency: <EngineEfficiency>  
 Weight: <CarWeight>  
 Color: <CarColor>**

### Bonus\*

Define the magic method \_\_toString() to have a reusable way of displaying the objects.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  V8-101 220 50  V4-33 140 28 B  3  FordFocus V4-33 1300 Silver  FordMustang V8-101  VolkswagenGolf V4-33 Orange | FordFocus:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: 1300  Color: Silver  FordMustang:  V8-101:  Power: 220  Displacement: 50  Efficiency: n/a  Weight: n/a  Color: n/a  VolkswagenGolf:  V4-33:  Power: 140  Displacement: 28  Efficiency: B  Weight: n/a  Color: Orange |
| 4  DSL-10 280 B  V7-55 200 35  DSL-13 305 55 A+  V7-54 190 30 D  4  FordMondeo DSL-13 Purple  VolkswagenPolo V7-54 1200 Yellow  VolkswagenPassat DSL-10 1375 Blue  FordFusion DSL-13 | FordMondeo:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: Purple  VolkswagenPolo:  V7-54:  Power: 190  Displacement: 30  Efficiency: D  Weight: 1200  Color: Yellow  VolkswagenPassat:  DSL-10:  Power: 280  Displacement: n/a  Efficiency: B  Weight: 1375  Color: Blue  FordFusion:  DSL-13:  Power: 305  Displacement: 55  Efficiency: A+  Weight: n/a  Color: n/a |

## Pokemon Trainer

You wanna be the very best pokemon trainer, like no one ever was, so you set out to catch pokemon. Define a class **Trainer** and a class **Pokemon**. Trainers have a **name**, **number of badges** and a **collection of pokemon**, **Pokemon** have a **name**, an **element** and **health**, all values are **mandatory**. Every Trainer **starts with 0 badges**.

From the console you will receive an unknown number of lines until you receive the command “**Tournament**”, each line will carry information about a pokemon and the trainer who caught it in the format “<**TrainerName> <PokemonName> <PokemonElement> <PokemonHealth>”** where **TrainerName** is the name of the Trainer who caught the pokemon, names are **unique** there cannot be 2 trainers with the same name. After receiving the command “**Tournament**” an unknown number of lines containing one of three elements “**Fire**”, “**Water**”, “**Electricity**” will follow until the command “**End**” is received. For every command you must check if a trainer has atleast 1 pokemon with the given element, if he does he receives 1 badge, otherwise all his pokemon **lose 10 health**, if a pokemon falls **to 0 or less health he dies** and must be deleted from the trainer’s collection. After the command “**End**” is received you should print all trainers **sorted by the amount of badges they have in descending order** (if two trainers have the same amount of badges they should be sorted by order of appearance in the input)in the format “<**TrainerName> <Badges> <NumberOfPokemon>**”.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Pesho Charizard Fire 100  Gosho Squirtle Water 38  Pesho Pikachu Electricity 10  Tournament  Fire  Electricity  End | Pesho 2 2  Gosho 0 1 |
| Stamat Blastoise Water 18  Nasko Pikachu Electricity 22  Jicata Kadabra Psychic 90  Tournament  Fire  Electricity  Fire  End | Nasko 1 1  Stamat 0 0  Jicata 0 1 |

## Google

Google is always watching you, so it should come as no surprise that they know everything about you (even your pokemon collection), since you’re really good at writing classes Google asked you to design a Class that can hold all the information they need for people.

From the console you will receive an unkown amount of lines until the command “**End**” is read, on each of those lines there will be information about a person in one of the following formats:

* “**<Name> company <companyName> <department> <salary>**”
* “**<Name> pokemon <pokemonName> <pokemonType>”**
* “**<Name> parents <parentName> <parentBirthday>**”
* “**<Name> children <childName> <childBirthday>**”
* “**<Name> car <carModel> <carSpeed>**”

You should structure all information about a person in a class with nested subclasses. People’s names are **unique** - there won’t be 2 people with the same name, a person can also have **only 1** **company** and **car**, but can have **multiple** **parents, chidlren** and **pokemon**. After the command “**End**” is received on the next line you will receive a single name, you should print all information about that person. Note that information can change during the input, for instance if we receive multiple lines which specify a person’s company, only the **last one** should be the one remembered. The salary must be formated to **two decimal places** after the seperator.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| PeshoPeshev company PeshInc Management 1000.00  TonchoTonchev car Trabant 30  PeshoPeshev pokemon Pikachu Electricity  PeshoPeshev parents PoshoPeshev 22/02/1920  TonchoTonchev pokemon Electrode Electricity  End  TonchoTonchev | TonchoTonchev  Company:  Car:  Trabant 30  Pokemon:  Electrode Electricity  Parents:  Children: |
| JelioJelev pokemon Onyx Rock  JelioJelev parents JeleJelev 13/03/1933  GoshoGoshev pokemon Moltres Fire  JelioJelev company JeleInc Jelior 777.77  JelioJelev children PudingJelev 01/01/2001  StamatStamatov pokemon Blastoise Water  JelioJelev car AudiA4 180  JelioJelev pokemon Charizard Fire  End  JelioJelev | JelioJelev  Company:  JeleInc Jelior 777.77  Car:  AudiA4 180  Pokemon:  Onyx Rock  Charizard Fire  Parents:  JeleJelev 13/03/1933  Children:  PudingJelev 01/01/2001 |

### Bonus\*

Define the magic method \_\_toString() to have a reusable way of displaying the objects.

## Method Says Hello!

You will receive the person name as an input. Write a class **Person** that only has a name and a **method**. The method should describe a greeting by the person, returning a String "<Person name> says Hello!". Print the result of the method call.

### Note

Code sample.

|  |
| --- |
| $name = trim(fgets(STDIN));  $person = new Person($name);  $fields = count(get\_object\_vars($person));  $methods = count(get\_class\_methods($person)); **if** ($fields!= 1 || $methods != 1) {  **throw new** Exception(**"Too many fields or methods"**); } |

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Peter | Peter says "Hello"! |

## Oldest Family Member

Create class **Person** with fields **name** and **age**. Create a class **Family**. The class should have **list of people**, method for adding members (**void addMember(Person member)**) and a method returning the oldest family member(**Person getOldestMember())**. Write a program that reads name and age for **N** people and **adds them to the family**. Then **print** the **name** and **age** of the oldest member.

If you’ve defined the class correctly, the test should pass.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 3  Pesho 3  Gosho 4  Annie 5 | Annie 5 |  | 5  Steve 10  Christopher 15  Annie 4  Ivan 35  Maria 34 | Ivan 35 |

## Last Digit Name

Write a class **Number** that will hold an integer number. Write a **method** in the class that returns the **English name** of the last digit of the given number. Write a program that reads an integer and prints the returned value from this method.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 1024 | four |  | 512 | two |

## Number in Reversed Order

Write a class **DecimalNumber** that has a method that **prints all its digits** in **reversed order**.

### Examples

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Input** | **Output** |  | **Input** | **Output** |
| 256 | 652 |  | 1.12 | 21.1 |

## Fibonacci Numbers

Define a class **Fibonacci**. It should keep a **list** of all **Fibonacci numbers** starting from **0, 1 until Nth** number in the sequence. Write a **method** in the Fibonacci class that receives as parameters **start position** and **end position** and returns **part of the sequence** starting from **start position (inclusive)** until **end position (exclusive)**.   
**ArrayList<Long> getNumbersInRange(int startPosition, int endPosition)**.

Write a program that reads **start position** and **end position** and prints the **Fibonacci numbers in that range**.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 0  6 | 0, 1, 1, 2, 3, 5 |
| 6  7 | 8 |
| 17  20 | 1597, 2584, 4181 |

## Car

Create a class **Car**. Every car has a **speed (km/h), fuel (liters)** and **fuel economy (L/100km)** (given in the same order on the first line). They should be stored in the class. Your task is to create a program which executes one of the commands:

* **Travel <distance>** – makes the car travel the specified <distance>

If you are given a distance which you don't have enough fuel to travel, just go as far as you can.

* **Refuel <liters>** – refuels the car with the specified <fuel>
* **Distance** – gets the total travel distance
* **Time** – get the total travel time
* **Fuel** – gets the remaining fuel
* **END** – stops the program

### Output

Print the total distance traveled, total travel time and fuel left at the end of the trip as in the Example below.

### Note

Round values up to one decimal digit after the decimal point, applies for **kilometers** and **liters**.

Show only minutes, discarding the seconds. For Example 2 minutes 40 seconds and 2 minutes 20 seconds all become 2 minutes.

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 100 20 20  Travel 100  Distance  Time  Fuel  END | Total distance: 100.0 kilometers  Total time: 1 hours and 0 minutes  Fuel left: 0.0 liters |

## Date Modifier

Create a class **DateModifier** which stores the difference of the days between two Dates. It should have a method which takes two String parameters representing a date as Strings and **calculates the difference in the days between them.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1992 05 31  2016 06 17 | 8783 |
| 2016 05 31  2016 04 19 | 42 |

### Hint

Use the **DateTime** class.

## \*Print People

Create a class **Person**. Every person should have name, age and occupation. Your task is to create the class and read some people, while adding them to an array. Sort them by age and print them in the given format. **Define magic method \_\_toString() and use custom sorting function for arrays.**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| Gosho 22 Dentist  Mimi 13 Student  END | Mimi - age: 13, occupation: Student  Gosho - age: 22, occupation: Dentist |