# Exercises: Inheritance

Problems for exercise and homework for the [Python OOP Course @SoftUni](https://softuni.bg/courses/python-oop). Submit your solutions in the SoftUni judge system at <https://judge.softuni.bg/Contests/1941>

## Person

You are asked to model an application for storing data about people. You should be able to have a Person and a Child. The child derives from the person. Every person has public attributes **name** and **age**. Your task is to model the application.

Create a **Child** class that inherits **Person** and has the same constructor definition. However, do not copy the code from the **Person** class - **reuse the Person class's constructor**.

Submit in judge a **zip file** named **project**, containing a separate file for each of the classes.

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| person = Person("Peter", 25)  child = Child("Peter Junior", 5)  print(person.name)  print(person.age)  print(child.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_) | Peter  25  Person |

## Zoo

Create a zoo which contains the following classes: 

and submit in judge a **zip file**, containing a separate file for each of the classes using the structure shown below:

Follow the diagram and create all of the classes. **Each** of them, except the **Animal** class, should **inherit** from **another** **class**. The Animal class should have private attribute name - string and **getter** for the name.

Every class should have constructor, which accepts one parameter: **name**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| mammal = Mammal("Stella")  print(mammal.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(mammal.name)  print(mammal.\_Animal\_\_name)  lizard = Lizard("John")  print(lizard.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(lizard.name)  print(lizard.\_Animal\_\_name) | Animal  Stella  Stella  Reptile  John  John |

## Players and Monsters

Your task is to create the following game hierarchy:



and submit in judge a **zip file**, containing a separate file for each of the classes using the structure shown below:



Create a class **Hero**. It should contain the following attributes:

* **username - string**
* **level - int**

Override the **\_\_repr\_\_()** method of the base class so it returns: **"{name} of type {class\_name} has level {level}"**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| hero = Hero("H", 4)  print(hero.username)  print(hero.level)  print(str(hero))  elf = Elf("E", 4)  print(str(elf))  print(elf.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(elf.username)  print(elf.level) | H  4  H of type Hero has level 4  E of type Elf has level 4  Hero  E  4 |

## Need for Speed

Create the following **hierarchy** with the following **classes**:



and submit in judge a **zip file**, containing a separate file for each of the classes using the structure shown below:



Create a base class Vehicle. It should contain the following attributes:

* **DEFAULT\_FUEL\_CONSUMPTION - float (constant)**
* **fuel\_consumption - float**
* **fuel - float**
* **horse\_power - int**
* A public constructor which accepts (fuel, horse\_power) and **set** the **default fuel consumption** on the attribute fuel\_consumption

The class should have the following methods:

* **drive(kilometers)**
  + The **drive** method should have a functionality to reduce the **fuel** based on the travelled kilometers and fuel consumption. Keep in mind that you can drive the vehicle only if you have enough fuel to finish the driving.

The default fuel consumption for **Vehicle** is 1.25.Some of the classes have different default fuel consumption:

* **SportCar - DEFAULT\_FUEL\_CONSUMPTION = 10**
* **RaceMotorcycle - DEFAULT\_FUEL\_CONSUMPTION = 8**
* **Car - DEFAULT\_FUEL\_CONSUMPTION = 3**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| vehicle = Vehicle(50, 150)  print(Vehicle.DEFAULT\_FUEL\_CONSUMPTION)  print(vehicle.fuel)  print(vehicle.horse\_power)  print(vehicle.fuel\_consumption)  vehicle.drive(100)  print(vehicle.fuel)  family\_car = FamilyCar(150, 150)  family\_car.drive(50)  print(family\_car.fuel)  family\_car.drive(50)  print(family\_car.fuel)  print(family\_car.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_) | 1.25  50  150  1.25  50  0.0  0.0  Car |

## Restaurant

Create a **restaurant** with the following classes and hierarchy:



and submit in judge a **zip file**, containing a separate file for each of the classes using the structure shown below:



The **Product** class should have the following **attributes** and subsequent **getters**:

* **name - string**
* **price - float**

**Beverage** and **Food** classes are **products**.

The **Beverage** class should have the following **attributes** and subsequent **getters**:

* **name - string**
* **price - float**
* **milliliters - float**

The Food class should have the following **attributes** and subsequent **getters**:

* **name - string**
* **price - float**
* **grams - float**

**HotBeverage** and **ColdBeverage** are **beverages** and they accept the **following parameters upon initialization**: **name, price, milliliters**

**Coffee** and **Tea** are **hot beverages** and they accept the **following parameters upon initialization**:   
**name, price, milliliters**

The **Coffee** class should have the following **additional** **attributes** and subsequent **getters**:

* **MILLILITERS = 50 (constant)**
* **PRICE = 3.50 (constant)**
* **caffeine - float**

MainDish, Dessert and Starter are **food**. They all accept the following parameters upon initialization: **name, price, grams**.

Dessert should accept **one more parameter** in its constructor:

* **calories - float**

Create a **getter** for the attribute **calories**.

Make **Salmon**, **Soup** and **Cake** inherit **MainDish**, **Starter** and **Dessert** classes **respectively**.

A **Cake** must have the following attributes upon initialization:

* **GRAMS = 250 (constant)**
* **CALORIES = 1000 (constant)**
* **PRICE = 5 (constant)**

A **Salmon** should have the following attributes upon initialization:

* **GRAMS = 22 (constant)**

**Note: All of the attributes need to be private!**

|  |  |
| --- | --- |
| **Test Code** | **Output** |
| product = Product("coffee", 2.5)  print(product.\_\_class\_\_.\_\_name\_\_)  print(product.name)  print(product.price)  beverage = Beverage("coffee", 2.5, 50)  print(beverage.\_\_class\_\_.\_\_name\_\_)  print(beverage.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(beverage.name)  print(beverage.price)  print(beverage.milliliters)  soup = Soup("fish soup", 9.90, 230)  print(soup.\_\_class\_\_.\_\_name\_\_)  print(soup.\_\_class\_\_.\_\_bases\_\_[0].\_\_name\_\_)  print(soup.name)  print(soup.price)  print(soup.grams) | Product  coffee  2.5  Beverage  Product  coffee  2.5  50  Soup  Starter  fish soup  9.90  230 |