# 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**.

## 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**

## 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}"**

## 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**

## 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 getters for each of them:

* **name – string**
* **price – float**

**Beverage** and **Food** classes are products. The **Beverage** class should have the following attributes and getter for the milliliters:

* **name – string**
* **price – float**
* **milliliters – float**

The Food class should have the following attributes and getter for the grams:

* **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. The **Coffee** class should have the following additional attributes and getter for the caffeine:

* **COFFEE\_MILLILITERS = 50 (constant)**
* **COFFEE\_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**

Crate 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:

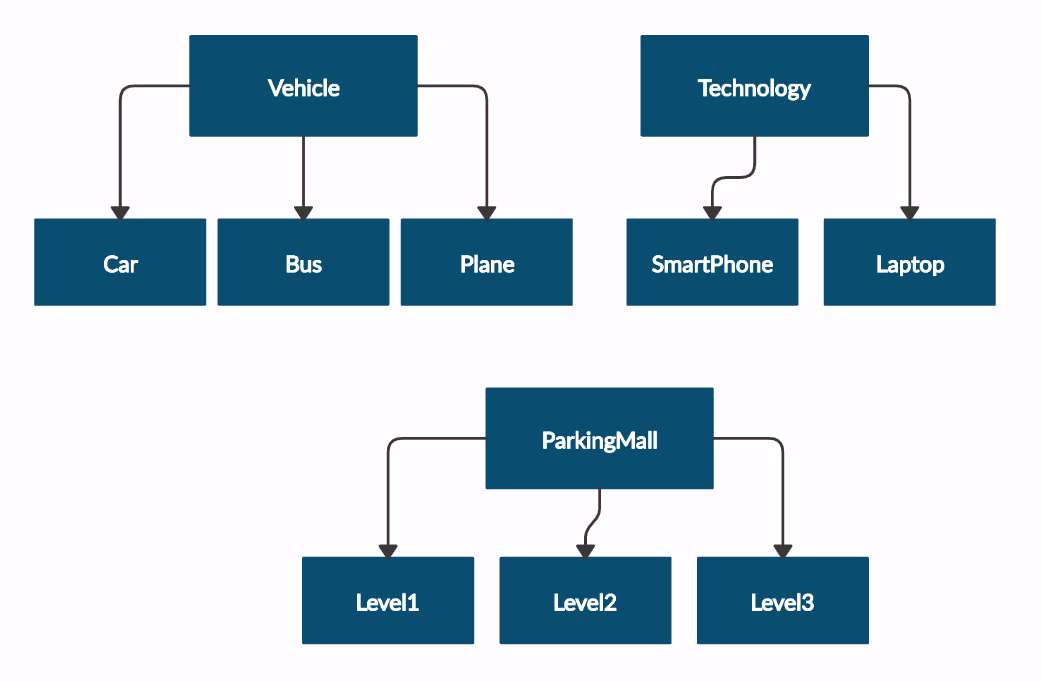
* **CAKE\_GRAMS = 250 (constant)**
* **CAKE\_CALORIES = 1000 (constant)**
* **CAKE\_PRICE = 5 (constant)**

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

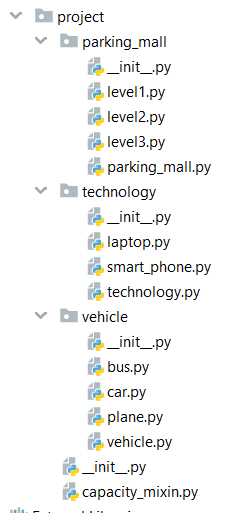
* **SALMON\_GRAMS = 22 (constant)**

## Mix it

Create the classes shown below with the following hierarchy:



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



Class **Vehicle** will get **available\_seats: int** upon initialization. Classes **Car**, **Bus** and **Plane** will inherit class **Vehicle**.

The **Car** class should have the following attributes:

* **available\_seats** **– int**
* **fuel\_tank – int**
* **fuel\_consumption – float**
* **fuel – float**

Create getter and setter for attribute **fuel** and validate the fuel not to exceed the fuel capacity.

The class should have the following methods:

* **drive(distance)** – check if you have enough fuel to travel the given distance.Reduce the fuel if you've managed to drive the car and return the following message **"** **We've enjoyed the travel!".**
* **refuel(liters) –** check if you have enough space in the tank to take the given liters.Increase the fuel in the tank and return the liters available or return **"Capacity reached!".** To do this inherit **CapacityMixin** with method named **get\_capacity(capacity, amount)** which will return the message above if the amount provided is bigger than the capacity, otherwise - the difference between the capacity and the given amount.

The **Bus** class should have the following attributes:

* **available\_seats** **– int**
* **ticket\_price – float**
* **tickets\_sold = 0 at the beginning**

The class should have the following methods:

* **get\_ticket(tickets\_count)** – use method **get\_capacity** from **CapacityMixin** to check if there are still seats available in the bus and track the current number of tickets sold.
* **get\_total\_profit() –** return the profit from the sold tickets

The **Plane** class should have the following attributes:

* **available\_seats** **– int**
* **rows – int**
* **seats\_per\_row – int**
* **seats\_available – empty dict**

The class should have method **buy\_tickets(row\_number, tickets\_count)**:

* check if the given row number is valid and return the following message if it's not: **"There is no row {row\_number} in the plane!"**
* check if you can sell the given tickets count in the desired row using the method **get\_capacity** from **CapacityMixin** and return **tickets\_count** if the number of tickets is available for sale in this row. Don't forget to update the seats\_available dictionary after selling some tickets!
* If the number of tickets for the row number are not enough (smaller than the **tickets\_count**) return the following message: **"Not enough tickets on row {row\_number}!"**

Class **Technology** will get **memory: float** and **memory\_taken: float** upon initialization. Classes **Laptop** and **SmarhPhone** will inherit class **Technology**.

The **Laptop** class should have the following attributes:

* **memory – float**
* **memory\_taken – float**

The class should have method **install\_software(software, software\_memory)**:

* check if you have memory to install the software and return the memory left after the install if sucessfully, otherwise return **"You don't have enough space for {software}!"**

The **SmartPhone** class should have the following attributes:

* **memory – float**
* **memory\_taken – float**

The class should have method **install\_apps(app, app\_memory)**:

* check if you have memory to install the app and return the memory left after the install if sucessfully, otherwise return **"You don't have enough space for {app}!"**

Class **ParkingMall** will get **parking\_lots: int** upon initialization.

The class should have method **check\_availability()**:

* using the **get\_capacity** method from **CapacityMixin** class check if there are any slots available in the level and return **"Parking lots available: {self.parking\_lots} "**. If there are no places available in this level return **"There are no more parking lots!"**

Classes **Level1, Level2** and **Level3** will inherit class **ParkingMall**. **Level1** has 150 parking lots, **Level2** has 100 parking lotsand **Level3** has 80 parking.