# Exercises: OOP

Problems for exercises and homework for the ["TS Advanced" course @ Software University](https://softuni.bg/trainings/2696/typescript-advanced-december-2019).

## Define a Class Person

**NOTE**: Create a **Person** class that has **readonly** properties **name** and **age**. They are both taken from the constructor.

Define a class **Person** with **readonly** field for **name** and **age**.

### Bonus\*

Try to create a few objects of type Person:

|  |  |
| --- | --- |
| **Name** | **Age** |
| Ivan | 20 |
| Todor | 35 |
| Peter | 14 |

## Family Members

Create a **Family** class with **public property family** which must be set to empty array through the constructor.

The class must support **2 methods:** a method for adding members to the family array – **addNewMember(member)** which is **tuple** of **string and number** which represents the name and the age of each member, and a method **oldestMember()** which returns the oldest member in the family in the following format: **“The oldest member is {name} and is {age} years old.”**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| let member = new Family();  member.addNewMember([‘Ivan’, 13]);  member.addNewMember([‘Todor, 45]);  member.addNewMember([‘Georgi’, 37]);  member.addNewMember([‘Viktor’, 30]);  console.log(member.oldestMember()); | The oldest member is Todor and is 45 years old. |

## Racing

Create an interface **Racing** which declares three functions:

1. **createRacer(name, fuel, position)**. The name is a **string** and the fuel and position are **numbers.**
2. **consumeFuel(usedFuel, name).** The used fuel is a **number** and the name is a **string**
3. **overtake(gained, lost).** The gained property is the name of the racer which gained a position and the lost – the name of the driver who lost position.

Create a class **Circuit** which implements the **Racing** interface. The class should have a public property **racers** which is set to **empty array through the constructor**. Implement the following functionality of the functions declared in the interface and create a new one – **showPositions()**:

1. **createRacer(name, fuel, position)**. Create an object with the parameters passed in the method in the following format: **{ name, fuel, position }** and add it to **the racers array**
2. **consumeFuel(usedFuel, name).** Find the racer with the name passed in the method and subtract the used fuel from the property fuel you created in the previous method. If the fuel is **equal or less than 0** create an **additional property** to the driver – **isDNF** and set its value to **“true”**
3. **overtake(gained, lost)** If both racers that are passed to the method are **valid** (are in the racers array), **not disqualified** and the racer who gains the positions **is not ahead** of the one losing it you should increase the value of the position property of the racer who lost and decrease the value to the one who gained. Furthermore, if the racer who lost position is already in last place you should return the following message: “**${racerName} is on the back of the grid!”**
4. **showPositions().** You should sort the racers in ascending order by their position and display them in the following order without the disqualified drivers:
   * **{racerName} finished on {position} position with {fuel} l remaining**
   * **{racerName} finished on {position} position with {fuel} l remaining**
   * **...**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| let race = new Circuit();  race.createRacer('LH', 100, 1);  race.createRacer('VB', 100, 2);  race.createRacer('DR', 100, 3);  race.consumeFuel(92, 'LH');  race.consumeFuel(87, 'VB');  race.consumeFuel(89, 'DR');  race.overtake('LH', 'VB');  race.overtake('DR', 'VB');  console.log(race.overtake('LH', 'VB'));  console.log(race.showPositions()); | VB is on the back of the grid!  LH finished on 1 position with 8 l remaining  DR finished on 2 position with 11 l remaining  VB finished on 3 position with 13 l remaining |

## Car Details

Define two classes **Car** and **Engine.**

**Car** has the following properties:

* **Model**
* **Weight**
* **Color**

Engine has the following properties:

* **Model**
* **Power**
* **Displacement**
* **Efficiency**

A Car’s **weight** and **color** and its Engine’s **displacement** and **efficiency** are **optional**.

Create a class **Details** that combines the data from both **Car** and **Engine** classes.

Your task is to **print** all the data about the **cars** in the format defined bellow. Every indent is a tab.

Vehicle model: {CarModel}  
Engine model: {EngineModel}  
 Power: {EnginePower}  
 Displacement: {EngineDisplacement}  
 Efficiency: {EngineEfficiency}  
Weight: {CarWeight}  
Color: {CarColor}

If any of the optional fields is missing, print the info in the following format, where again the indent is a tab:

Your task is to **print** all the data about the **cars** in the format defined bellow. Every indent is a tab.

**Vehicle model: {CarModel}  
Engine model: {EngineModel}  
 Power: {EnginePower}**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| let s = new Details(new Car('C class', 'MB177', 3982, 'red'), new Engine('MB177', 510, 3982, 'A')); | Vehicle Model: C class  Engine model: MB177  Power: 510  Displacement: 3982  Efficiency: A  Weight: 3982  Color: red |
| let s = new Details(new Car('C class'), new Engine('MB177', 510)); | Vehicle Model: C class  Engine model: MB177  Power: 510 |

## Parking Lot

Create an **abstract** class **Activities** which declares three **abstract** **methods**:

1. **parkVehicle(plate)**. The plate is of a **string** type.
2. **payForStay(hours, rate, plate).** The hours and rate parameters are **numbers** and the plate is a **string**.
3. **leaveTheParking(plate).** The plate is of a **string** type.

Create a class **ParkingLot** which extends the **Activities** class. The class should have a **public properties capacity**, which is set through the constructor, **revenue** – a number set to 0 and **parkedVehicles** – an empty array. Implement the following functionality of the methods declared in the abstract and create a new one – **overview()**:

1. **parkVehicle(plate)**. If there is enough capacity for the new vehicle push in into the **parkedVehicles** array with an object in the following format: **{ plate, didPay }** Please note that the **didPay** must be originally set to **false**! Otherwise, throw a new error **“No more available spaces in the parking lot!”**
2. **payForStay(hours, rate, plate).** Check if the vehicle is parked and if it is you should increase the revenue with the derivative of rate and hours and change the **didPay** to **true**. However, if the vehicle is not parked, throw a new error **“There is no such vehicle parked in the parking lot!”**
3. **leaveTheParking(plate).** Check if the vehicle is parked and it paid for the stay. If so remove it from the **parkedVehicles** array.
4. **overview() -** You should display the revenue and the remaining parked vehicles in the parking lot, separated by comma and space, in the following format:
   * **The current revenue of the parking lot is {revenue} and {parkedVehicles} are parked**

### Examples

|  |  |
| --- | --- |
| **Input** | **Output** |
| let p = new ParkingLot(5);  p.parkVehicle('CA1111CA');  p.parkVehicle('CA2222CA');  p.parkVehicle('CA3333CA');  p.parkVehicle('CA4444CA');  p.parkVehicle('CA5555CA');  p.payForStay(4, 2, 'CA1111CA');  p.payForStay(2, 2, 'CA2222CA');  p.payForStay(7, 2, 'CA4444CA');  p.leaveTheParking('CA1111CA');  p.leaveTheParking('CA4444CA');  console.log(p.overview()); | The current revenue of the parking lot is 26 and CA2222CA, CA3333CA, CA5555CA are parked! |