# Python OOP: Exam Preparation

**Submit your solutions in the SoftUni** [**Judge system**](https://judge.softuni.org/Contests/4116/Python-OOP-Exam-5-August-2023)



*We are a new bank committed to providing personalized financial services to our customers. Our primary focus is building strong relationships and understanding each client's unique needs. We specialize in offering competitive loans to students and adults, with flexible terms and competitive interest rates. Our streamlined application process ensures a hassle-free experience for our customers. With a dedicated team of professionals, we aim to deliver exceptional service and support to help our clients achieve their financial goals.*

You will be provided with a **skeleton** that includes all the folders and files that you will need.

***Note: You are not allowed to change the folder and file structure and change their names!***

A screenshot of a computer

Description automatically generated with medium confidence

**Judge Upload**

For the **first two problems**, create a **zip** file with the **project** **folder** and **upload it** to the judge system.

For the **last problem**, create a **zip** file with the **test folder** and **upload it** to the judge system.

You do not need to include **in the zip file** your **venv**, **.idea**, **pycache**, and **\_\_MACOSX** (for Mac users), so you do not exceed **the maximum allowed size** of **16.00 KB**.

# Structure (Problem 1) and Functionality (Problem 2)

Our task is to implement the **structure and functionality** of all the classes (properties, methods, inheritance, abstraction, etc.)

You are **free to add additional attributes** (instance attributes, class attributes, methods, dunder methods, etc.) to simplify your code and increase readability as long as it does not change the project's final result in accordance with its requirements so that the program works properly.

## Class BaseLoan

In the **base\_loan.py** file, the class **BaseLoan** should be implemented. It is a **base class** for any **type of loan,** and it **should not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **interest\_rate:** float
  + The value represents the **interest rate of the loan**.
* **amount:** float
  + The value represents the **amount of the loan**.

### Methods

#### \_\_init\_\_(interest\_rate: float, amount: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### increase\_interest\_rate()

* Method **increases the loan’s interest rate**. Keep in mind that **each type of loan** implements the method **differently**.

## Class StudentLoan

In the **student\_loan.py** file, the class **StudentLoan** should be implemented. A student loan is a **type of loan**. Each student loan has **an interest rate of 1.5 percent and an amount of 2000.0 EUR**.

### Methods

#### \_\_init\_\_()

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### increase\_interest\_rate()

* The method **increases** the **interest rate** by **0.2 percent**.

## Class MortgageLoan

In the **mortgage\_loan.py** file, the class **MortgageLoan** should be implemented. A mortgage loan is a **type of loan**. Each mortgage loan has **an interest rate of 3.5 percent and an amount of 50000.0 EUR**.

### Methods

#### \_\_init\_\_()

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### increase\_interest\_rate()

* The method **increases** the **interest rate** by **0.5 percent**.

## Class BaseClient

In the **base\_client.py** file, the class **BaseClient** should be implemented. It is a **base class** for any **type of client,** and it **should not be able to be instantiated**.

### Structure

The class should have the following attributes:

* **name:** str
  + The value represents the **name of the client**.
  + If the name is **an empty string or contains only white spaces**, raise a ValueError with the message: **"Client name cannot be empty!"**
* **client\_id:** str
  + The value represents the **id** **number** **of a client**. It should contain **exactly 10 symbols**.
  + If the client’s id **is not 10 symbols long**, raise a ValueError with the message: **"Client ID should be 10 symbols long!"**
* **income:** float
  + The value represents the **income** **of a client**.
  + If the client’s income **is less than or equal to 0.0**, raise a ValueError with the message: **"Income must be greater than zero!"**
* **interest: float**
  + **The value represents the client’s interest.**
* **loans:** list
  + Empty list that **will contain loans (objects)** each client has.

### Methods

#### \_\_init\_\_(name: str, client\_id: str, income: float, interest: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### increase\_clients\_interest()

* **Increases** the **client’s interest.** Keep in mind that **each type of client** implements the method **differently**.

## Class Student

In the **student.py** file, the class **Student** should be implemented. The student is a **type of client**. Each student has an **initial interest of 2.0 percent**.

### Methods

#### \_\_init\_\_(name: str, client\_id: str, income: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### increase\_clients\_interest()

* The method **increases** the client’s **interest** **by 1.0 percent**.

## Class Adult

In the **adult.py** file, the class **Adult** should be implemented. The adult is a **type of client**. Each adult has an **initial interest of** **4.0 percent**.

### Methods

#### \_\_init\_\_(name: str, client\_id: str, income: float)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### increase\_clients\_interest()

* The method **increases** the client’s **interest** **by 2.0 percent**.

## Class BankApp

In the **bank\_app.py** file, the class **BankApp** should be implemented. It will contain the functionality of the project.

### Structure

The class should have the following attributes:

* **capacity: int**
  + The **number** of **clients** а **Bank** **can have.**
* **loans: list**
  + Empty list that **will contain all loans** (objects) that are created.
* **clients: list**
  + Empty list that **will contain all clients** (objects) that are created.

### Methods

#### \_\_init\_\_(capacity: int)

* In the **\_\_init\_\_** method, all the needed attributes must be set.

#### add\_loan(loan\_type: str)

The method **creates** a loan of the given type and **adds** it to the **loans** collection.

* If the loan’s type is not valid, raise an **Exception** with the following message:

**"Invalid loan type!"**

* Otherwise, **create** the loan, **add** it to the loans list, and **return** the following message:

**"{loan\_type} was successfully added."**

* **Valid types** of loans are: **"StudentLoan"** and **"MortgageLoan"**

#### add\_client(client\_type: str, client\_name: str, client\_id: str, income: float)

The method **creates** a client of the given type and **adds** them to the **clients** collection.   
All clients’ **IDs** will be **unique**.

* **First**, check if the **client type** is valid, and if **not** raise an **Exception** with the following message:

**"Invalid client type!"**

* **Then**, check if there is available **bank** **capacity,** and ifnot **return** the following message:

**"Not enough bank capacity."**

* Otherwise, **create** the client, **add** it to the clients list, and **return** the following message:

**"{client\_type} was successfully added."**

* **Valid types** of clients are: **"Student"** and **"Adult"**.

#### grant\_loan(loan\_type: str, client\_id: str)

The method **adds the loan** of the given type to the **client’s loans** collection. Both **loan** and **client** will **always exist**.

* **First**, check if the loan **can be granted** to the client. The **student client** can get **ONLY** a **student type of loan** and the **adult client** can get **ONLY** a **mortgage type of loan**. In case of a mismatch, **raise an Exception** with the following message:

**"Inappropriate loan type!"**

* If the loan can be **granted successfully** to the client, **remove** it from the **bank's loan collection**, and **add it** to the **client’s loan collection**. **Return** the following message:

**"Successfully granted {loan\_type} to {client\_name} with ID {client\_id}."**

* Take the **first loan** of the given **type** from the collection.

#### remove\_client(client\_id: str)

The method **removes the client** with the given **ID** from the **bank**.

* **First**, check if there is a client with the given **ID** in the **client’s collection**. If not, **raise an Exception** with thefollowing message:

**"No such client!"**

* **Then**, check if the client **has loans.** If so, **raise an Exception** with thefollowing message:

**"The client has loans! Removal is impossible!"**

* If the client can be **removed successfully**, **remove** them from the bank, and **return** the following message: **"Successfully removed {client\_name} with ID {client\_id}."**

#### increase\_loan\_interest(loan\_type: str)

The method **increases** the **interest rates** for **all loans** of the **given type** that are in the **bank’s loan collection**. The loan type will be one of the **valid types** (**StudentLoan** or **MortgageLoan**). When all rates for the given loan type are successfully changed (**hint**: use increase\_interest\_rate() method), **return** the following message:

**"Successfully changed {number\_of\_changed\_loans} loans."**

* Loans that are **already granted to clients** should **not** **be affected**.

#### increase\_clients\_interest(min\_rate: float)

The method **increases** the **interest rates** for **all clients** that are in the **bank’s client collection** who currently have an **interest rate less than the min\_rate value**. When rates are successfully changed (**hint**: use increase\_clients\_interest() method), **return** the following message:

**"Number of clients affected: {changed\_client\_rates\_number}."**

#### get\_statistics()

Returns information about the **bank’s loans** and its **clients**. Each string is on a **new line**.

"Active Clients: {total\_clients\_count}

Total Income: {total\_clients\_income}

Granted Loans: {loans\_count\_granted\_to\_clients}, Total Sum: {granted\_sum}

Available Loans: {loans\_count\_not\_granted}, Total Sum: {not\_granted\_sum}

Average Client Interest Rate: {avg\_client\_interest\_rate}"

* All sums and rates should be formatted to the 2nd decimal place.
* Average Client Interest Rate refers to the property interest each client has.
* Avoid ZeroDivisionError

#### Examples

|  |
| --- |
| **Input** |
| **bank = BankApp(3)**  **print(bank.add\_loan('StudentLoan'))**  **print(bank.add\_loan('MortgageLoan'))**  **print(bank.add\_loan('StudentLoan'))**  **print(bank.add\_loan('MortgageLoan'))**  **print(bank.add\_client('Student', 'Peter Simmons', '1234567891', 500))**  **print(bank.add\_client('Adult', 'Samantha Peters', '1234567000', 1000))**  **print(bank.add\_client('Student', 'Simon Mann', '1234567999', 700))**  **print(bank.add\_client('Student', 'Tammy Smith', '1234567555', 700))**  **print(bank.grant\_loan('StudentLoan', '1234567891'))**  **print(bank.grant\_loan('MortgageLoan', '1234567000'))**  **print(bank.grant\_loan('MortgageLoan', '1234567000'))**  **print(bank.remove\_client('1234567999'))**  **print(bank.increase\_loan\_interest('StudentLoan'))**  **print(bank.increase\_loan\_interest('MortgageLoan'))**  **print(bank.increase\_clients\_interest(1.2))**  **print(bank.increase\_clients\_interest(3.5))**  **print(bank.get\_statistics())** |
| **Output** |
| **StudentLoan was successfully added.**  **MortgageLoan was successfully added.**  **StudentLoan was successfully added.**  **MortgageLoan was successfully added.**  **Student was successfully added.**  **Adult was successfully added.**  **Student was successfully added.**  **Not enough bank capacity.**  **Successfully granted StudentLoan to Peter Simmons with ID 1234567891.**  **Successfully granted MortgageLoan to Samantha Peters with ID 1234567000.**  **Successfully granted MortgageLoan to Samantha Peters with ID 1234567000.**  **Successfully removed Simon Mann with ID 1234567999.**  **Successfully changed 1 loans.**  **Successfully changed 0 loans.**  **Number of clients affected: 0.**  **Number of clients affected: 1.**  **Active Clients: 2**  **Total Income: 1500.00**  **Granted Loans: 3, Total Sum: 102000.00**  **Available Loans: 1, Total Sum: 2000.00**  **Average Client Interest Rate: 3.50** |

# Task 3: Unit Tests (100 points)

You will **be provided with another skeleton** for this problem. **Open** the **new skeleton** as a **new project** and write tests for the **SecondHandCar** class. The class will have some methods, fields, and one constructor, all of them working properly. You are **NOT ALLOWED** to change anything in the class code. Cover the whole class with unit tests to make sure that the class is working as intended. Submit **only the test** folder.