**Домашня робота №52 ООП. Створення класів. Продовження роботи №51**

1. Додамо до класу BankAccount класовий метод "find\_accounts\_by\_owner", який приймає ім'я власника і повертає список об'єктів "BankAccount", які належать даному власнику.

   @classmethod  
    def find\_accounts\_by\_owner(cls, owner\_name):  
        matching\_accounts = []  
        for account in cls.accounts:  
            if account.owner\_name == owner\_name:  
                matching\_accounts.append(account)  
        return matching\_accounts

Як бачимо, потрібно додати атрибут класу accounts, що міститиме список всіх створених рахунків. Створення рахунку із вже існуючим номером забороняється.

Протестувати виконання класового методу на самому класі і на деякому його екземплярі.

1. Додати ще один класовий метод "get\_average\_balance" до класу "BankAccount", який повертає середній баланс коштів всіх об'єктів "BankAccount", створених до цього моменту.
2. Змінимо клас "BankAccount" так, щоб він мав підтримку рахунків в різних валютах. Для цього використаємо клас "Money" для представлення суми та валюти, а атрибут balance буде представлений як екземпляр класу Money. Зробіть модифікацію існуючих методів у класу з огляду на присвоєння self.balance = Money(balance, currency). Наприклад, модифікуйте метод transfer так, щоб переказ був можливий, якщо валюти рахунків однакові.

class Money:  
    def \_\_init\_\_(self, amount, currency):  
        self.amount = amount  
        self.currency = currency  
      
    def \_\_str\_\_(self):  
        return f"{self.amount} {self.currency}"  
      
class BankAccount:  
    accounts = []  
      
    def \_\_init\_\_(self, account\_number, balance, owner\_name, currency):  
        self.account\_number = account\_number  
        self.balance = Money(balance, currency)  
        self.owner\_name = owner\_name  
        BankAccount.accounts.append(self)

1. В \_\_init\_\_ видозмініть рядок BankAccount.accounts.append(self) так, щоб не використовувати ім’я BankAccount на пряму, а отримати його через self.
2. Якщо валюти не співпадають при трансфері реалізуйте переведення по деякому курсі і зарахування коштів отримувачу. Створіть новий метод " transfer\_funds(self, target\_account, amount)". У класі "BankAccount" для цього додамо деяку структуру даних (подумайте як реалізувати) "\_\_exchange\_rate", що містять курси обміну між валютами.

*Пояснення до реалізації пункту 14*

До класу "BankAccount" додайте приватний атрибут-поле класу "\_\_exchange\_rate" як словник та заповніть його за допомогою класового методу create\_ exchange\_rate. "\_\_exchange\_rate" має містити курси обміну між валютами, наприклад до гривні. Можна скористатись даними із API у вигляді json, що можна програмно завантажити модулем request за посиланням <https://bank.gov.ua/NBUStatService/v1/statdirectory/exchange?json>

 @classmethod  
    def create\_ exchange\_rate(cls):

# prepare data from API  
        [cls.\_\_exchange\_rate](https://obj.new/)  = # data from API

При трансфері євро у долари метод " transfer\_funds(self, target\_account, amount)", наприклад, має спочатку перевести євро у гривні, а потім гривні у долари. Передбачити трансфер, якщо є курси валют у словнику exchange\_rate.

1. \*Добавте функціонал, який буде при створені банківського рахунку як об’єкта зберігатись його стан у папці data у вигляді деякого текстового файла із назвою account\_number, що міститиме інформацію про баланс і власника. У разі зміни стану атрибутів об’єкта потребує зміни і відповідний файл.
2. Реалізуйте власний метод для видалення банківського рахунку за номером як об’єкта, так і відповідного файла у папці data.