Лабораторна робота 3

Абстрактні класи

Виконала:
студентка групи МІТ-31
Півторак Каріна
Варіант -1

Мета: ознайомитися із поняттям наслідування в ооп, навчитися створювати класи та об’єкти з наслідуванням у пайтон

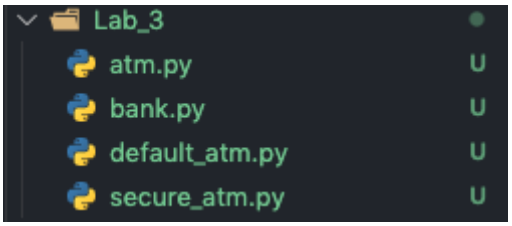
Завдання:

6	Створити абстрактний клас Банкомат, який може заблокуватись/розблокуватись, видати гроші та поповнитись інкасаторами. Створити два похідні класи: Звичайний_банкомат, Захищений_банкомат. Звичайний_банкомат видає гроші при введенні пін-коду і блокується, якщо коштів нема. Захищений_банкомат крім пін-коду просить ввести код-підтвердження з sms і може бути заблокований при повторному введенні неправильного коду-підтвердження. Створити клас-контейнер Банк, який містить декілька банкоматів різного виду, вивести інформацію про кожен банкомат. *Основна програма повинна демонструвати весь вказаний функціонал!!!*

Репозиторій: https://github.com/KariSpace/python-oop-labs.git

Хід виконання лабораторної роботи

Для зручності створюємо окремий файл під кожен клас



Створюємо абстрактний клас у якому створюємо три загальних метода - `blockATM`, `unblockATM`, `callTheIncasator`, що будуть однакові для усіх класів. що будуть наслідувати абстрактний. І один абстрактний клас, який є обов’язковим для усіх класів, що будуть наслідувати ATM

atm.py:

```
from abc import ABC, abstractmethod


class ATM(ABC):


    @abstractmethod
    def __init__(self, money):
        self.blocked = False
        self.volume = 5000
        self.money = money
    # общие методы, который будут использовать все наследники этого класса
    def blockATM(self):
        self.blocked = True
```

```
        print("ATM was blocked")

    def unblockATM(self):
        self.blocked = False
        print("ATM unblocked")

    def callTheIncasator(self):
        self.money = self.volume
        self.unblockATM()
    # абстрактные методы, которые будет необходимо переопределять для каждого
подкласса
    @abstractmethod
    def givemoney(self):
        pass
```

Ми не можемо створювати об’єкти абстрактного класу, а лише використовувати його для наслідування іншими класами

default_atm.py

```
from atm import ATM

bankAccounts = {
                "user1" : {"pin" : 5530, "savings" : 3600},
                "user2" : {"pin" : 8747, "savings" : 900},
                "user3" : {"pin" : 4055, "savings" : 12500},
               }

class Default_ATM(ATM):

    def __init__(self, money):
        self.blocked = False
        self.volume = 10000
        self.money = money
    def givemoney(self, moneyWithdraw):
        if(not self.blocked):
            print("userAccount?")
            usr = input()
            userAccount = bankAccounts.get(usr, False)
            if(userAccount):
                print("pin? --> ")
                pin = int(input())
                if(userAccount.get("pin", False) == pin):
```

```
                    if(userAccount.get("savings", False) >= moneyWithdraw):

                        if(self.money  >= moneyWithdraw):
                            self.money -= moneyWithdraw
                            print("Success")
                            bankAccounts[usr]["savings"] = userAccount["savings"]
- moneyWithdraw
                            print(bankAccounts[usr]["savings"])
                        else:
                            print("Insufficient funds")
                            self.blockATM()

                    else:
                        print("Not enought")
                else:
                    print("wrong pin")
            else:
                print("wrong user")
        else:
            print("ATM blocked, call incasators")

atm1 = Default_ATM(10)
atm1.givemoney(200)
atm1.givemoney(200)
atm1.callTheIncasator()
atm1.givemoney(200)
```

Назначаємо словник словників bankAccounts, який виступає у ролі бази даних банкових акаунтів. Клас Default_ATM настідує клас ATM, у якому переназначений абстрактный метод givemoney на власний.

Запускаємо код:

```
(base) Kari:lab_1 kari$ /usr/local/bin/python3 /Users/kari/Desktop/Kari_Labs/OOP/Lab_1/Lab_3/atm.py
userAccount?
user1
pin? -->
5530
how much you want withdraw?
Insufficient funds
ATM was blocked
ATM blocked, call incasators
ATM unblocked
userAccount?
user2
pin? -->
8747
how much you want withdraw?
Success
700
(base) Kari:lab_1 kari$
```

Спочатку у банкоматі не достатньо для зняття коштів, тому він блокується. При повторній спробі зняти кошти банкомат заблокований. При виклику інкасаторів банкомат розблоковується та поповнюється до свого максимального об'єму та розблоковується, тому ми можемо успішно зняти гроші(при корректно введених даних).

secure_atm.py:

```
from atm import ATM
import random
bankAccounts = {
               "user1" : {"pin" : 5530, "savings" : 3600},
               "user2" : {"pin" : 8747, "savings" : 900},
               "user3" : {"pin" : 4055, "savings" : 12500},
              }

class Secure_ATM(ATM):

   def __init__(self, money):
       self.blocked = False
       self.volume = 50000
       self.money = money
    def givemoney(self, moneyWithdraw):
       if(not self.blocked):
           print("userAccount?")
           usr = input()
           userAccount = bankAccounts.get(usr, False)
           if(userAccount):
               counter = 0
               while(not self.blocked):

                   if(counter == 2):
                       self.blockATM()
                       break

                   print("pin? --> ")
                   pin = int(input())
                   if(userAccount.get("pin", False) == pin):
                       if(userAccount.get("savings", False) >=
moneyWithdraw):

                           number = random.randint(1000,9999)
                           print(f'Your sms code: {number}')
```

```
                            code = int(input("code? --> "))

                            if(str(number) == str(code)):
                                if(self.money  >= moneyWithdraw):
                                    self.money -= moneyWithdraw
                                    print("Success")
                                    bankAccounts[usr]["savings"] =
userAccount["savings"] - moneyWithdraw
                                    print(bankAccounts[usr]["savings"])
                                    break
                                else:
                                    print("Insufficient funds")
                                    self.blockATM()
                                    break
                            else:
                                 print("wrong code")
                        else:
                            print("Not enought")
                    else:
                        counter += 1
                        print("wrong pin")
            else:
                print("wrong user")
        else:
            print("ATM blocked, call incasators")

atm1 = Secure_ATM(9000)
atm1.givemoney(200)
atm1.givemoney(200)
atm1.callTheIncasator()
atm1.givemoney(200)
```

Запускаємо код:

Намагаємося зняти гроші, двічі вводимо неправильний пін код, банкомат блокується. Викликаємо інкасаторів, банкомат поповнюється і розблоковується. Вводимо правильний пін, отримуємо смс. Якщо введений код з смс правильний, гроші зняті

```
(base) Kari:lab_1 kari$ /usr/local/bin/python3 /Users/kari/Desktop/Kari_Labs/OOP/Lab_1/Lab_3/secure_atm.py
userAccount?
user1
pin? -->
35436
wrong pin
pin? -->
3534
wrong pin
ATM was blocked
ATM blocked, call incasators
ATM unblocked
userAccount?
user2
pin? -->
8747
Your sms code: 1881
code? --> 1881
Success
700
(base) Kari:lab_1 kari$
```

Написати клас-контейнер, у який можна додавати, з нього можна видаляти, у ньому можна заміняти екземпляри класів, які наслідують абстрактний клас. У основній програмі обов'язково необхідно продемонструвати однотипну обробку екземплярів у контейнері з використання циклу.

Створюємо клас-контейнер:

```
class MainBank:
    def __init__(self, atm=None):
        if (not atm):
            self.atm = []
```

При ініціалізації перевіряємо, чи існує `atm`, і, якщо ні, то створюємо список:
`self.atm = []`

Реалізовуємо методи класу. Для цього використовуємо одну із концепцій ооп(поліморфізм) і перевантажуємо магічні методи:

```
    def __setitem__(self, key):
        self.atm.append(key)

    def __delitem__(self, key):
        self.atm.remove(key)
```

```
    def __len__(self):
        return len(self.atm)

    def __iter__(self):
        iter_atm = iter(self.atm)
        return iter_atm

    def __contains__(self, item):
        return True if item in self.atm else False
```

Створюємо кілька екземеплярів класів:

```
atm1 = Secure_ATM(9000)
atm2 = Default_ATM(200)
atm3 = Default_ATM(100)
```

Створюємо екземпляр класу банк:

```
bank = MainBank()
```

Додаєм екземпляри класу в контейнер:

```
print(isinstance(bank, collections.abc.Container))
bank.__setitem__(atm1)
bank.__setitem__(atm2)
bank.__setitem__(atm3)
```

Обробка екземпярів класу з використанням циклу:

```
for i in range(bank.__len__()):
    print(bank.atm[i])
```

Висновок: Було створено три класса, кожен з яких наслідує попередній. Екземпляри цих класів мають як свої власні методи, так і ті, що були унаслідувані від попереднього. Також, створений клас-контейнер, у якому можна додавати і видаляти екземпляри класів, що наслідують абстрактний